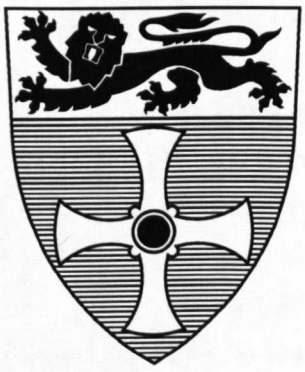
UNIVERSITY OF
NEWCASTLE



The Effect of Diverse Development Goals on Computer–Based System Dependability[1]

Anthony Thomas Lawrie

Spring 2006

[1]Presented to the School of Computing Science at The University of Newcastle upon Tyne. In part fulfilment of the requirements for a degree of a Doctorate of Philosophy.

Acknowledgments

I would like to thank, first and foremost, my PhD supervisor Professor Clifford Jones for his invaluable guidance in producing this thesis. Secondly, I would also like to thank my family, friends and colleagues for their encouragement during this time. Thirdly, I would like to acknowledge the funding sponsors EPSRC for providing the practical financial means to conduct this work. Fourthly, I would like to take this opportunity of thanking my reviewers, Professor Brian Randell and Professor Bev Littlewood for their helpful advice and comments that improved the quality of this thesis. Last, but not least, I would also like to formally recognise the positive research environment afforded to me by the Centre for Software Reliability department, within the school of Computing Science, at Newcastle University where I was based throughout this period.

Abstract

Society's increasing dependence upon software control and information processing provision has demanded comparable increases in software dependability. While the existing software dependability approach has resulted in significant improvements, its focus is heavily aimed towards achieving software dependability via redundant fault–tolerant mechanisms built into the software artifact to provide error–control in the presence of activated faults. Less emphasis appears to have been placed upon how software dependability can also be promoted through a fault–avoidance approach in the software creation process by incorporating human redundancy and diversity. In this thesis, a process intervention which can potentially improve fault–avoidance is considered. This involves the setting of diverse development goals within important generic computer–based system contexts in order to increase detection of potentially harmful assumptions which can result in subtle systemic conflicts that can undermine the dependability of the resultant artifact during the early development phases of requirements, specification and design. A search theoretic simulation model is progressed and developed to capture some of the important dynamics involved. The eventual outputs of the simulation model indicate that increased fault coverage and sensitivity can be obtained through the setting of diverse development goals during the early phases of software development.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Focus of the Thesis

As society becomes increasingly dependent upon information technology and information processing provision there has been an increasing concern about the trustworthiness of software. Such concerns have given rise to the need to improve the dependability in the service that software systems deliver. In response to such concerns, the dependability community has been focused upon increasing the dependability of software systems through the introduction of redundant computation and structure in the software artifact to improve the software system's resilience to residual design and implementation faults that are considered inevitable in any *real-world* complex software system. Whilst such approaches have resulted in significant increases in software dependability, the ever–increasing pervasion of information technology systems has resulted in a recognition that such approaches have limitations. It is therefore now becoming increasingly apparent that a wider and more encompassing view of computer–based system dependability is needed that includes the influencing role played by humans and how they can both compromise and improve the dependability of such computer systems is needed.

Subsequently, whilst there has been a reasonable amount of effort focused on how humans and human error can compromise the dependability of computer systems during operation, there has been comparatively much less focus upon how humans, in the creation process, can be better resourced and deployed to improve the dependability of the created software artifact.

The focus of this thesis is to consider how a socio–technical process intervention utilising diverse goal setting in the context of a computer–based system perspective can help improve the dependability of the creation process through the increased dependability means of fault–avoidance.

1.2 Nature of the Thesis

This thesis is interdisciplinary in nature. In no small part this has been directly influenced by the interdisciplinary approach of a large multi–university site EPSRC

research programme that the author has been involved in from late September 2000 until the present day, which has been focused upon exploring how the fields of Psychology, Computer Science, Mathematics and Sociology can help improve the dependability of computer–based systems.

1.3 Overview of the Thesis

To help the reader obtain an understanding of the thesis structure, this section provides a brief overview of the chapters.

1.3.1 Chapter 2

Chapter 2 focuses upon the dependability of the software artifact and covers the existing dependability framework that has been progressed and advanced over recent decades. The framework provides a high–level guidance on: the desired attributes of the software artifact; the threats that undermine such attributes; and the means by which the threats can be controlled and the attributes promoted. The approach taken by the dependability community is to accept that, in any *real–world* software development, residual faults are inevitable. Therefore the primary focus of the dependability community is to rationalise how redundancy, in the form of additional computation and structure, can be introduced into the software artifact to control the activation of residual faults during operational execution, from resulting in judgements of delivered service failures by the user.

1.3.2 Chapter 3

Chapter 3 focuses upon the dependability of the software creation process through increased fault–avoidance. So far, less emphasis has been placed upon how to improve the dependability of the software creation process, by the dependability community. In this chapter a dependable process view is discussed, that attempts to define what attributes, threats and means would such a mature and predictable software creation process possess. It is argued that to achieve such a dependable software creation process would require an integrated understanding of the many

process dynamics. The role of process redundancy and diversity is then discussed along the process dimensions of: the software task; human resources; and process technology. Finally, two measures of the benefits of justifying the inclusion of such process redundancy and diversity are briefly covered.

1.3.3 Chapter 4

Chapter 4 focuses on the need for a more holistic computer–based system conception. In this computer–based system view, the boundaries of the system are extended to include not only the technical computer system, but also the interfacing roles, responsibilities, and motivations of humans as a subsystem. With this system conception dependability is viewed as a super ordinate high–level system goal that requires a greater bottom–up holistic view that integrates the different meanings and purpose ascriptions that different context stakeholders will make during the creation process. The inability to achieve such an holistic view can not only result in faults and vulnerabilities, but can also lead to technically dependable systems failing through non-technical aspects. For these reasons, the computer–based system conception incorporates important context classifications of: the utility context; the engineering context; the deployment context; and the evolution context to help achieve such a holistic dependability perspective.

1.3.4 Chapter 5

Chapter 5 follows on from chapter 4 to provide a case study of faults and vulnerabilities from a computer–based system perspective, based upon factual problems reported over many years in the Automatic Teller Machine (ATM) domain. A total of nine issues that resulted in faults and vulnerabilities are analysed and characterised in terms of a computer–based system perspective of dependability.

1.3.5 Chapter 6

Chapter 6 focuses on assumptions. Assumptions are a necessary reasoning and communication mechanism in many decision–making situations. However, particularly in the software creation process, flawed assumptions represent an im-

portant fault–phenomenology that can undermine both the dependability of the creation process and the eventual created artifact. This chapter therefore reviews some of the known literature on assumptions in order to get a better understanding of their nature and what factors influence their occurrence.

1.3.6 Chapter 7

Chapter 7 focuses upon the role of purpose and function ascription. Since differing ascriptions of purpose and meaning can result in non–technical judgements of dependability failure from a computer–based system perspective, it is necessary to understand the underlying philosophy and empirical evidence of how purpose and goal–setting affects human performance. It can be seen from the literature that goal–setting offers a dual complementary approach of instituting diversity into the creation process and promoting a more holistic conception required by a computer–based system viewpoint.

1.3.7 Chapter 8

Chapter 8 focuses upon pulling many of the issues of the previous chapters into a rich discussion of how a combination of diverse goal–setting and diverse computer–based system contexts can help detect assumptions that can undermine the dependability of a computer–based system when dependability is viewed as a holistic super ordinate goal. The chapter discusses the nature of non–functional attributes and the existing ways in which these are considered during software development. The chapter then goes on to introduce and justify an existing non–functional analysis notation along with its merits and differences for graphical illustration of how the proposed Goal–Diversity process intervention can help identify harmful assumptions within a computer–based system conception. This adapted notation then reuses the nine issues raised from the Automatic Teller Machine case study of chapter 5 as an illustration.

1.3.8 Chapter 9

Chapter 9 focuses at drawing upon two longstanding and complimentary areas of knowledge and literature from Search Theory and Software Inspections as a preliminary guide to the important concepts, terminology, and processes involved in searching and detection type tasks. These concepts, terminology, and processes are then used in Chapter 10 to help inform and structure a design rationale for a search simulation model.

1.3.9 Chapter 10

Chapter 10 focuses upon the design, implementation, configuration, and simulation of a search simulation model. It does so by first utilising the notions, terminologies and conceptions from both the areas of Search Theory and Software Inspections covered in chapter 9. These concepts, terminology and processes are then discussed and justified into a design rationale that is structured upon a basis of desired simulation goals, modelling scope, and modelling detail. The design rationale employs a visual diagrammatic technique known as influence diagrams to help illuminate and explain the many related design decisions discussed and justified in determining the simulation models design. The chapter then goes on to discuss the verification, validation and implementation issues before providing a justification for the configuration of the simulation experiments. Lastly, the results of the simulation experiments are discussed, analysed and statistically evaluated.

1.3.10 Chapter 11

Chapter 11 provides a summary of the work contained in this thesis along with recognised limitations of the work and potential future areas of work that this thesis stimulates.

Chapter 2

Dependable Software Artifacts

2.1 Chapter Introduction

Society's increasing dependence upon computerised software control and information processing has resulted in an increasing demand for the dependability of software in safety–critical and mission–critical applications in recent years. Dependability can be promoted in both the creation process via a means of fault–avoidance and by including additional redundancy directly into the created artifact via a fault–tolerance approach.

This chapter primarily focuses upon the fault–tolerant approach to increase the the dependability of the created artifact.

2.2 The Dependability Framework

The dependability approach to constructing software artifacts makes the assumption that, no matter how professional and disciplined the development process is, any *real–world* software system will contain residual defects that can compromise the trust and confidence a user will be prepared to place in the service it delivers.

In order to facilitate a systematic approach to the dependable construction of software artifacts, the dependability community has refined and progressed a conceptual framework [1, 2, 3]. At the highest conceptual level, this includes the three categories of: a) the threats that undermine promotion of these system attributes; b) an integrated view of required system attributes; c) the essential means to the promotion of these system attributes. These are illustrated in figure 2.1 on the following page and are presented and discussed in turn in the subsections that follow.

2.2.1 Threats to Dependability

The threats to dependability represent, at the most general level, a three phase causality chain that captures the failure pathology of software artifacts. A **fault** is the adjudged or assumed cause of an **error** state when the fault becomes activated

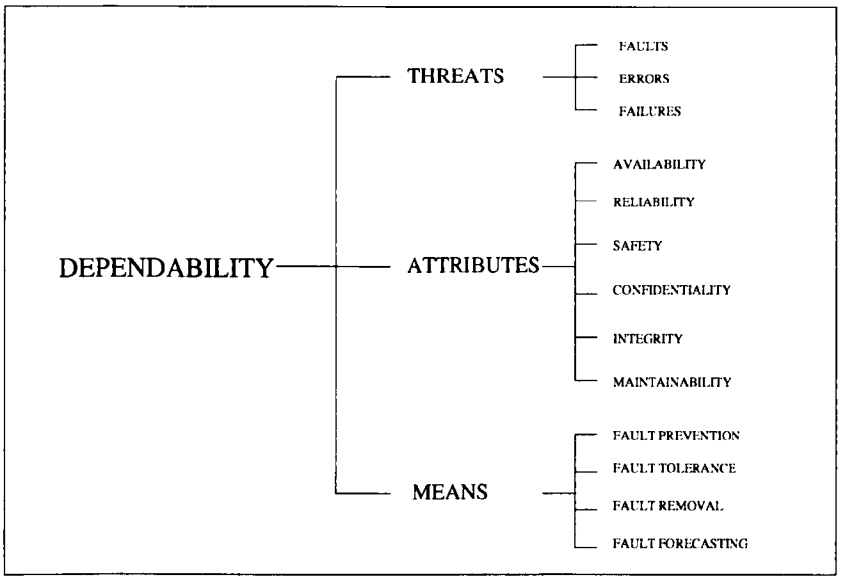


Figure 2.1: The Dependability Tree [source: [1]: p 5]

by some combination of computation input conditions (activation pattern). However, the presence of a fault in a software artifact does not necessarily result in an error state, this only occurs when input conditions result in its activation. A fault that is present in the artifact, but is non–active is a **dormant** fault. **Failure** occurs when the fault is activated and creates an error state which then permeates through the service interface and results in a judgement of incorrect service delivery. A failure is thus a visible event occurence that results in a transition from correct to incorrect service delivery. The presence of an error state (due to an activated fault) may not necessarily result in a failure transition from correct to incorrect service delivery, as this may be due to: a) some control or containment of the error state before it permeates the service interface; or b) the error state permeates through the service interface but is not detected or judged to result in a transition from correct to incorrect service. An error state that goes undetected or judged to not result in a failure is a **latent** error.

It should be noted that the labelling of the threats is particularly important for two reasons [3]. Firstly, the fault labelling represents a finality of causal considerations. No further explanation is sought to ask such questions as *"Why or how did*

the fault occur in the software artifact?" This is to prevent endless retracing or recursion to explain the presence of faults in the software artifact. Secondly, the inclusion of error between fault and failure in the causality chain is important to emphasise the possibility of intervention that can be enacted in the form of dependability *means* to forecast, prevent, remove, and tolerate the fault and ensure that its activation does not necessarily result in a system failure transition.

2.2.2 Attributes of Dependability

Dependability is an integrative system concept that requires the promotion of important non-functional system properties to be achieved — if the service a software artifact provides is to be trustworthy. These include: a) **availability** — the readiness for correct service delivery; b) **reliability** — continuity of correct service delivery; c) **safety** — the absence of catastrophic consequences of incorrect service delivery on the user(s) or the system environment; d) **confidentiality** — the absence of unauthorised disclosure of information; e) **integrity** — the absence of improper software artifact state alterations; and f) **maintainability** — the ability to undergo repair, change, and alterations.

It is important to point out that other important secondary software artifact properties are often a combination of the six primary attributes. For example, **security** is argued to be the synergy of availability + confidentiality + integrity. Another example is **performability** which often relates to the fact that a software artifact delivers more than one mode of service. Performability captures a measure of service delivery quality in terms of timeliness of delivery that may range from full capacity to emergency service delivery. Furthermore, the priority placed upon any one (or more) of the six primary or secondary system attributes can vary from one particular application or domain to another depending upon the criticality of service demands expected from the artifact.

2.2.3 Means by which Dependability is Attained

In order to help ensure attainment of important dependability attributes a number of techniques can be employed. These are: 1) **FAULT PREVENTION** — how to

prevent the occurrence or introduction of faults into the software artifact. Essentially, this *dependability means* relates to the professionalism, discipline, or quality of the creation process of the software artifact. In particular, to process technology — in the form of development assistance tools, development methods, and development techniques (e.g. structured programming, information hiding, modularisation, etc, etc); 2) **FAULT TOLERANCE** — how to deliver correct service in the presence of faults. This *dependability means* is provided by a combination of error detection and error recovery. **Error Detection** first initiates an error signal within the system. Two classes of error detection are possible. The first is *concurrent error detection* — which occurs during actual service delivery. The second is *preemptive error detection* — which takes place when service delivery is (temporarily) suspended and checks the software artifact for latent error states and dormant faults. **Error Recovery** is then used to transform the detected error state(s) into a state that is not considered to be erroneous. This can be achieved in three ways: a) *Rollback* — the recovery operation returns the detected error state back to a prior saved error–free state. The saved error–free state is often referred to as a checkpoint; b) *Compensation* — the erroneous state possesses sufficient redundancy to enable error elimination; and c) *Rollforward* — a state substitution occurs where a new error–free state is substituted for the previously detected error state; 3) **FAULT REMOVAL** — how to reduce the number or severity of faults. This *dependability means* relates to both the creation process and operational lifetime of the software artifact. During the creation process, fault removal involves three steps; a) **Verification** is the process of ensuring that the software artifact functions according to verification conditions. If verification fails *Fault Diagnoses* occurs. This involves determining the fault(s) that resulted in verification failure. Once the offending fault(s) are determined *Fault Correction* takes place to remove the fault(s) from the software artifact. After correction it is appropriate to reiterate a verification phase known as *Regression Verification* to ensure that the first verification phase has not resulted in introducing new fault(s) that can cause verification to fail again. Verification techniques can be categorised by whether they result in dynamically exercising the software artifact or not. *Static Verification* occurs without dynamic execution of the software artifact (e.g. inspections, walk–throughs, model checking, or theorem proving, etc). *Dynamic Verification*

involves exercising the software artifact. This can either involve symbolic inputs or actual inputs being supplied; b) **Validation** — relates to determining the adequacy or appropriateness of the specification. Specification fault(s) relate to detection of any behavioural situation where the software artifact will not perform its required function in a complete, consistent, or correct manner; c) **Fault Maintenance** — relates to the fault removal process during the operational lifetime of the software artifact. Two approaches are recognised: a) *Corrective Maintenance* — aimed at removing fault(s) that have been reported; and b) *Preventive Maintenance* — is focused on detecting and removing faults before they cause error states during usual operational use; 4) **FAULT FORECASTING** — involves a forecast evaluation of the dependability requirements of the software artifact concerning the fault potential for fault occurrence or fault activation. This *dependability means* has two main facets: a) **Qualitative or Ordinal Evaluation** — which includes the identification, classification, and ranking of potential failure modes along with the environmental conditions that can result in system failures; and b) **Quantitative or Probabilistic Evaluation** — which includes the forecast evaluation in terms of fault probabilities of activation or occurrence. The probabilistic evaluations attempt to capture the degree to which the attributes of dependability have been satisfied. These are then considered measures of dependability. Examples include: a) reliability — mean time to failure (MTTF); and b) maintainability — mean time to change (MTTC).

2.3 The Dependability Process

Before discussing specific examples of how software artifacts can be made more dependable, it is important to distinguish the software creation process from the created software artifact — in terms of the dependability framework discussed in section 2.2. A view of a dependable process incorporating: a) the creation process; b) the static software artifact; c) the dynamic execution of the software artifact; and d) the operational process, is illustrated in figure 2.2. This view explicitly incorporates into these process phases the threats to dependability (i.e. faults, errors, and failures) along with the essential means by which dependability can be attained or improved. In doing so, it provides a greater insight into where

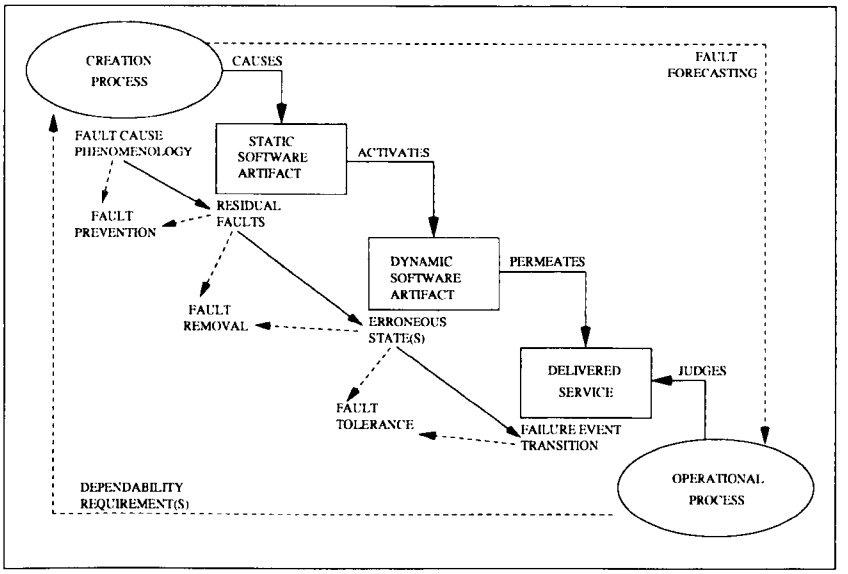


Figure 2.2: The Dependability Process

each ***threat*** (or impairment) and ***means*** most importantly relates — in terms of whether the dependability ***means*** and ***threat*** belong to the creation process, the created artifact, or a hybrid of both in some way.

2.3.1 The Software Creation Process

It can be seen from figure 2.2 that the means of fault prevention is solely the responsibility of the creation process. By this it is meant that in order to prevent a fault ever being introduced into the process, it must either never occur, or its potential occurrence is detected and prevented prior to creation. This may be achieved through human detection, some form of process technology (i.e. development tools, methods, or techniques) or some interaction of both. It can be seen that fault forecasting is solely the responsibility of the creation process — in terms of leveraging human experience and knowledge to understand the operational demands and constraints to accurately predict the failure modes possible and dependability requirements demanded in that particular domain.[1] The effectiveness of how well this is done will have a major bearing upon the types of redundancy mechanisms that are deemed most feasible, effective, and advantageous to incorporate into the

[1]This could also be facilitated by process technology

software artifact (see section 2.4) [4].

As mentioned in the dependability framework in section 2.2, the fault cause phenomenology is strictly a concern of the creation process — in terms of process approaches to the prevention of such causes occurring.[2] However, the issue of residual faults being introduced into the artifact also reveals that fault removal is also an important responsibility of the creation process — in terms of detecting and correcting them at some prior development stage (i.e. inspection or testing etc) in the software artifact before it is placed into operational use. This of course mandates consideration to both the static and dynamic software artifact and involves detection and removal of dormant faults and error states.

2.3.2 The Created Software Artifact

It can be seen from the dependability process view in figure 2.2 that the means of fault tolerance is the sole responsibility of the created software artifact if residual faults become oblivious to both human and process technology detection in the creation process. In such a situation either the residual fault (once activated into an error state) will need to be controlled before it permeates the service delivery interface, or the risk of judgements of failure by the operational domain may result. In order to achieve this, additional redundant mechanisms will need to be incorporated directly into the created software artifact (see section 2.4). However, the means of fault removal is often also an important software artifact responsibility of fault tolerance — during operational execution usage. This is often referred to as **Fault Handling**. Fault handling is geared towards preventing located residual faults in the static artifact, once activated, from ever being activated again. It involves four stages [3]: 1) *Fault diagnoses* — which detects and records the causes of the error state(s) in terms of both artifact location and type; 2) *Fault isolation* — which conducts logical exclusion of the residual fault(s) in the software artifact from any possibility of further participation in service delivery; 3) *System reconfiguration* — which can switch into action redundant mechanisms

[2]The dependability framework does go into more detail on the generic classification of potential faults that may be introduced — such as accidental, malicious, interaction faults etc, but does not hypothesize how these are caused or overlooked etc

or reassign processing tasks among other non-failed components; and 4) *System reinitialisation* — that can check, update, and record a new configuration along with updates to internal data structures (i.e. databases, tables, records etc).

2.3.3 Process and/or Artifact Responsibility Issues

It can be seen therefore, that a delineation can be made between dependability responsibilities for promoting software dependability in both the software creation process and the created software artifact. Fault forecasting and fault prevention are sole considerations and responsibilities of the software creation process — along with issues of fault causality phenomena. Fault tolerance is the sole consideration and responsibility of the created software artifact — along with error state control to prevent activated residual faults permeating through the service interface.

However, an overlapping of responsibilities is shared between the creation process and created artifact when we consider issues of a) the dependability means of fault removal; and b) responsibility for detecting faults. In the creation process, responsibility lies with detecting and removing faults prior to delivery of the system, whereas fault tolerance mechanisms, in the created artifact, are often responsible for handling residual faults that have eluded detection (either by human, process technology, or both) in the creation process. It should be further noted, that although fault forecasting is essentially the responsibility of the creation process, its impacts (good and bad) — in terms of how effective it is can propagate through to have a corresponding consequential effect upon the effectiveness of the fault tolerance mechanisms incorporated into the created artifact.

Finally, for the benefit of simplicity in discussing broader issues of redundancy introduced in the creation process of chapter 3, a strict delineation of responsibilities between the creation process and the created artifact will be represented by two labels. To capture the fault forecasting, fault prevention, fault removal responsibilities of the creation process — along with its responsibility for considerations of fault cause phenomenology and detection of residual faults, the label of soft-

ware **FAULT AVOIDANCE** will be used.[3] To capture the fault toleration and fault removal responsibilities of the created artifact — along with threats of residual fault, error, and failure control, the label of software **FAULT TOLERANCE** will be used.[4]

2.4 Software Artifact Redundancy

Having discussed and established the differing and overlapping responsibilities of the creation process and the created artifact in terms of a view of the dependability process in section 2.3, this section will expand on the issues and techniques involved in achieving software dependability via fault tolerance in the created artifact. In this section artifact redundancy concerns: a) the addition of system functionality — over–and–above that required to provide required functionality needed to deliver correct service in normal operating conditions; and b) additional system structure — to facilitate other creation process or operational process (i.e. maintenance) activities. A broader discussion on redundancy and related issues of how software dependability can be promoted via a fault avoidance approach in the creation process will be covered in chapter 3.

This section will first discuss fault tolerance from a broader system control approach by considering fault tolerance as system error control in subsection 2.4.1. Next broader issues of software artifact redundancy will be discussed in subsection 2.4.2. Finally, some limitations of software artifact redundancy will be discussed in subsection 2.4.3.

2.4.1 Software Error Control

In order to provide successful fault tolerance, the software artifact must be capable of intervening between fault activation and permeation of the error state through

[3]This is similar to labelling by the dependability community — such as definitions given by Laprie in [1, 2]. However, in this thesis the process consideration of fault forecasting and fault cause phenomenology are also included in the term.

[4]This labelling is likely to be less controversial and is more in-keeping with traditional views held by the dependability community. In this label issues of fault removal are interpreted as fault handling techniques which are subsumed under the label of fault tolerance

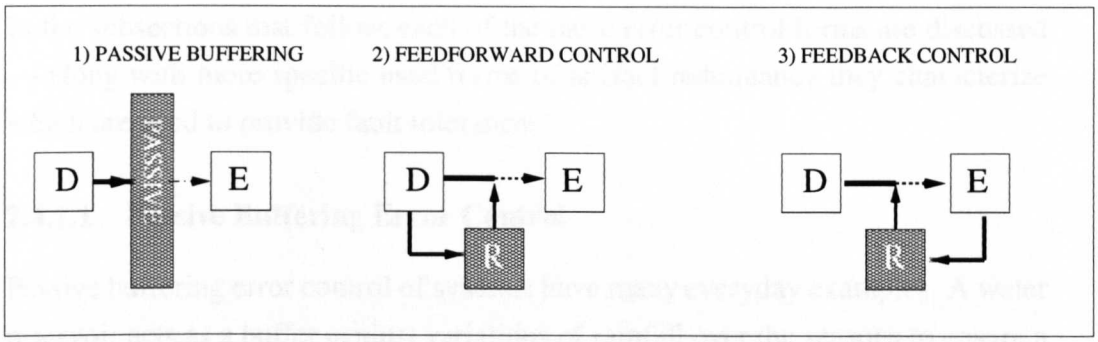


Figure 2.3: Three Forms of System Control [source [5]: p 14]

the service interface causing a potential failure transition from correct to incorrect service delivery. In essence the error state(s), once detected, must be controlled. Consideration of broader system control from systems theory approaches show that there are three ways in which a system can be regulated [5]. These are: a) buffering control; b) feed-forward control; and c) feedback control. A visual depiction is provided in figure 2.3. In each form of regulation control, the effect of disturbance (labelled D) on the essential variables (labelled E) is reduced by either a passive buffer or by an active regulator (labelled R). [5]

To return to issues of software artifact redundancy — in view of these three basic forms of system control, it can be seen from figure 2.3 that in the absence of disturbance (equivalent to faults in the software artifact) the additions of the passive buffer or active regulators (shaded in figure 2.3) are not necessary (hence the term *"Redundant"*). However, in the presence of residual faults in the artifact, which, with real world software artifacts, is the more realistic case, the passive buffer(s) or active regulators are no longer unnecessary but in fact critical to ensuring the system's stability (equivalent to ensuring continued correct service delivery). Artifact redundancy is therefore only redundant (classical meaning) to the extent of artifact functionality required to provide service delivery in the absence of non-active residual artifact faults (of whatever type).

[5] It is noted by Heylighen and Joslyn [5] that systemic disturbance can originate externally or internally. However, these external/internal influences can be combined and abstracted away to represent either external or internal disturbance.

In the subsections that follow, each of the basic error control forms are discussed — along with more specific used forms of artifact redundancy they characterize which are used to provide fault tolerance.

2.4.1.1 Passive Buffering Error Control

Passive buffering error control of systems have many everyday examples. A water reservoir acts as a buffer against variations of rainfall over the seasons to ensure a stable supply of water — in spite of seasonal variations. In manufacturing, an organisation may often deliberately build–up and store dormant stocks of produced goods to absorb variations in demand (i.e. sales) and supply (i.e. production) in order to ensure a reliable (or stable) supply of goods — even if there are problems in purchasing raw materials/components or the production process. In both these examples, and generally with passive buffering error control, variations can be tolerated without any direct intervention being necessary (hence the qualification of the term *"passive"*).

A specific example of such error control with fault tolerance is Triple Modular Redundancy (TMR) architectures (also called n-version redundancy) [4]. A visual illustration is provided in figure 2.4.[6] With TMR designs, each of the three channels are independently created diverse implementations (i.e. white box data structures and logics) of the same functionality (i.e. black box functionality) and to ensure absolute separation of failure are best operated upon individual computer processing units and associated hardware (if required). Input data is passed to each of the three independent and diversely implemented module channels. The output from each independent module channel is then compared by a voting module (sometimes called an adjudicator). If all three channels produce the same data outputs a very high level of confidence can be placed in the correct computation of the output data.

[6]While not shown in the figure, it should be noted that in many TMR schemes the voters are also triplicated to increase fault–tolerance, and independent TMR modules vote across and compare from redundant voters also.

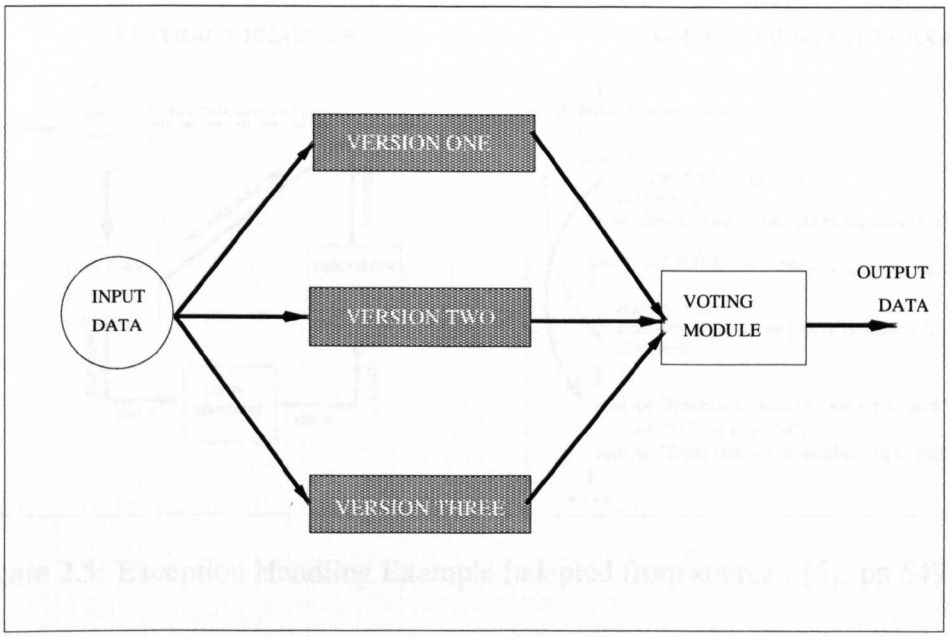


Figure 2.4: Triple Modular Redundancy [adapted from source: [4]]

If, on the other hand, there is a disagreement between the independent module channel's data output — where one module channel produces a different computation result from the other two module channels, then a *'majority–rules'* protocol is enforced by the voting module and it accepts the computational output data of the two agreeing module channels. This is a clear example of how TMR is a specific software artifact form of passive buffering, as sufficient computational redundancy exists to absorb erroneous computation in one module channel without any direct or active intervention being required (in the fault tolerance literature; this is often referred to as fault–masking [3]). Fault–masking approaches to fault tolerance can be particularly important when the criticality (or safety) of the system has performability or timeliness of control response requirements. For example fly–by–wire aircraft are not only safety critical systems (by applicational definition), but the timeliness of control responses are also safety critical — due to the speed at which aircraft travel.[7] In such applicational domains fault masking, like TMR, with the ability to provide sufficient computational redundancy to absorb

[7]Even the delay of a fraction of a second in active error regulation at super sonic speeds could increase the potential for a collision or accident under emergency or combat conditions.

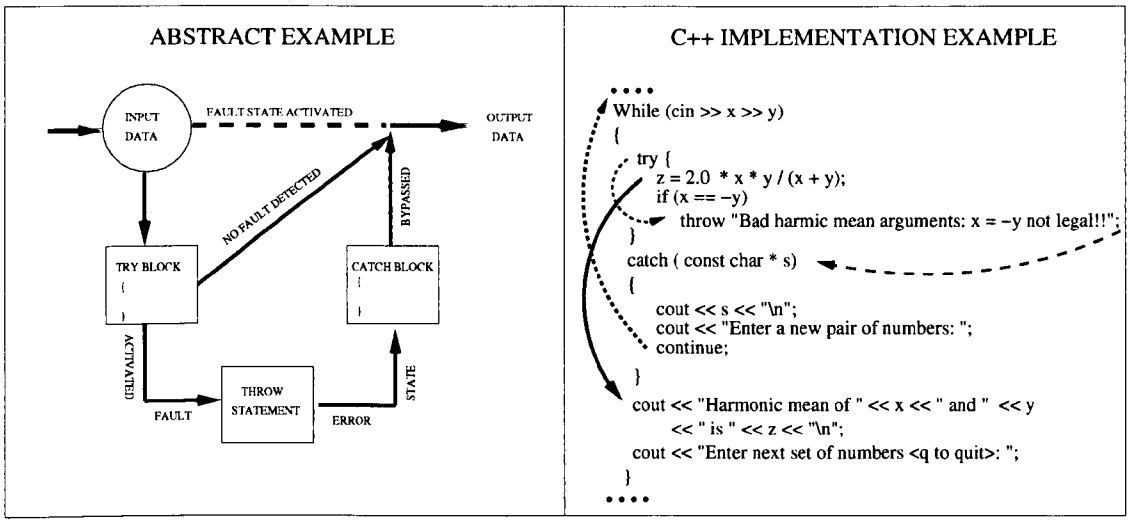


Figure 2.5: Exception Handling Example [adapted from source : [6]: pp 649-60]

error states without direct regulation intervention, helps ensure timeliness of error control without losing critical response time.

2.4.1.2 Feed-Forward Error Control

The concept of system error control via feedforward control relies upon prior knowledge of the disturbance in order to anticipate its effects ***before*** the disturbance actually destabilizes the system. Such approaches are often referred to as *"anticipatory–control"* and place a great demand upon the system to collect information and knowledge of its internal states and external environmental conditions before such disturbance can cause a serious deviation from its intended stable state ranges. Feed-forward control therefore presumes some prior system goal as an important facet of effective control — as without it, the system would not know what to consider and categorise as disturbance. Feed-forward control is therefore limited to controlling only prior known internal/external disturbance, with known or anticipated effects it can have upon the system's stability.

A good example of such an error control approach within the fault tolerance literature is exception handling [7]. Since studies have shown that as much as 66% of all software crashes can result from failures that could have been con-

trolled/prevented by exception handling [8], exception handling error control represents an important contribution to software artifact robustness from predictable software faults — such as wrong data type input formats, dividing–by–zero, reading past end–of–file etc. Furthermore, it does so in a much more structured and modular fashion than ad–hoc error code checking or defensive programming techniques. This aids other dependability properties also, such as maintainability (i.e. decoupling and code comprehension) and reliability, as a more structured error control approach helps reduce all round complexity of the software artifact and its consequent implications for potential side–effects (cf subsection 2.4.3.2).

In the abstract example illustrated in figure 2.5, it can be seen how feed-forward control is enacted. First some data input or computation is placed within the *"try block"*. Once data computations are placed within this block, this informs the system that the statements, data, or computations may result in exceptions occurring. If no exceptions are raised by the data or computations, then control passes out of the try blocks, ignores the *"catch block"*, and control flow progresses as normal. However, in the event that input data or computations do raise or *'throw'* an exception (due to the activation of an anticipated residual fault in the software artifact) then control passes to the *"catch block"*. The catch block then bypasses the error state and outputs an appropriate error message (and possibly passes control to either retrieve from the error state or returns control back to re–input data). This is in contrast to when no exception handling is performed (indicated by the straight dashed portion of the line in the abstract example of figure 2.5) which would result in the dormant fault state becoming activated into an error state.

In the C++ implementation example in figure 2.5 a specific computation of the harmonic mean of two numbers is illustrated. The harmonic mean of two numbers is defined as the inverse of the average of the inverses and represents mathematically the formula: $\frac{2xy}{x+y}$. With this formula it can be seen that when y is the negative of x the computation will result in a division by zero error. This error can be bypassed by utilizing exception handling and placing the computation into a try block and throwing an exception when x = -y. This then invokes the handler of

the catch block which outputs an appropriate error message before passing control flow back to the beginning of the loop for re–entry of input data — thereby bypassing the error state. The lines shown in the C++ implementation example of figure 2.5 indicate how control flow progresses. The solid line shows how control flow passes from the harmonic mean computation to the end of loop output messages when no activated fault is anticipated. The dashed lines, on the other hand, show how control flow passes from the try block once an exception is thrown to the catch block and then back to the start of the loop.

With the progress of programming paradigms (e.g. object oriented), program languages now offer sophisticated inclusion for exception handling — including customised exception handling classes to relate exception handling to accommodate for specific applicational domain conditions [cf.[9]: pp 907-920]. However, as indicated earlier, feed-forward control relies heavily upon fault–forecasting to anticipate the types of faults that can occur — whether internal computation or custom application domain based. Exception handling fault tolerance therefore, as in many other aspects of software engineering [10], places great demands upon individual human ability, experience, and domain knowledge acquisition to ensure effective error control.

2.4.1.3 Feedback Error Control

The concept of feedback control is based upon compensating the system ***after*** the disturbance has resulted in a serious deviation from the system's intended stable state ranges. Therefore, like feed-forward control, feedback control is active control that also relies upon knowledge of a desired goal state in order to take action. However, unlike feed-forward control, feedback control is not reliant upon high levels of knowledge and information of internal states and external conditions in order to function. Instead, it relies upon information directly from the disturbance effects to enact regulation. A well known example used is that of a thermostat which records the temperature of the room. When the room temperature deviates beyond the thermostat's desired goal range temperature (i.e. this low–end deviation is representative of disturbance) then the thermostat switches on the heating.

Again, when the temperature deviates beyond the thermostat's desired goal range (i.e. high–end disturbance) the thermostat switches off the heating. Because feedback control is reliant upon information from the actual disturbance experienced, feedback control is often referred directly to as *"error–controlled regulation"* [cf. [5]].

Feedback control, however, has its own disadvantages. Firstly, it is only as good as the sensitivity of the regulator to detect disturbance quickly enough to enact active regulation before the system becomes irretrievably unstable (i.e. very serious deviation(s) from desired stable goal state ranges). Senge [11] provides a cruel example of where feedback control can fail due to a lack of disturbance sensitivity in the feedback regulation mechanism — which he entitles "THE PARABLE OF THE BOILED FROG." cf. pp 22-23. He notes that if you place a frog into a pan of boiling water, then the frog will immediately try and escape. However, if, instead, you place a frog into a pan of water at room temperature and don't scare the frog, then it will stay in the pan. If the pan is already on a heat source and is gently turned on, then, as the temperature approaches 70 to 80 degrees, the frog is very likely to stay in the pan — and may even give indications and signs that the frog is quite enjoying being in that temperature of water. But as the temperature of the water rises above this, the frog will become groggier, until it is so weakened that it will eventually become unable to climb out of the pan. Senge notes that although there is nothing physically restraining the frog, it will actually sit there in the pan and boil. Senge points out the reason why this can happen, stating [p. 22]:

> *"The frog's internal apparatus for sensing threats to survival is geared to sudden changes in its environment, not to slow, gradual changes."*

Secondly, the impact of system disturbance may be so great that feedback control is completely inappropriate for this form of systemic disturbance. For example, in general system considerations of feedback control, Heylighen and Joslyn [5] note that:

> *"...if you see someone pointing a gun in your direction, you would*

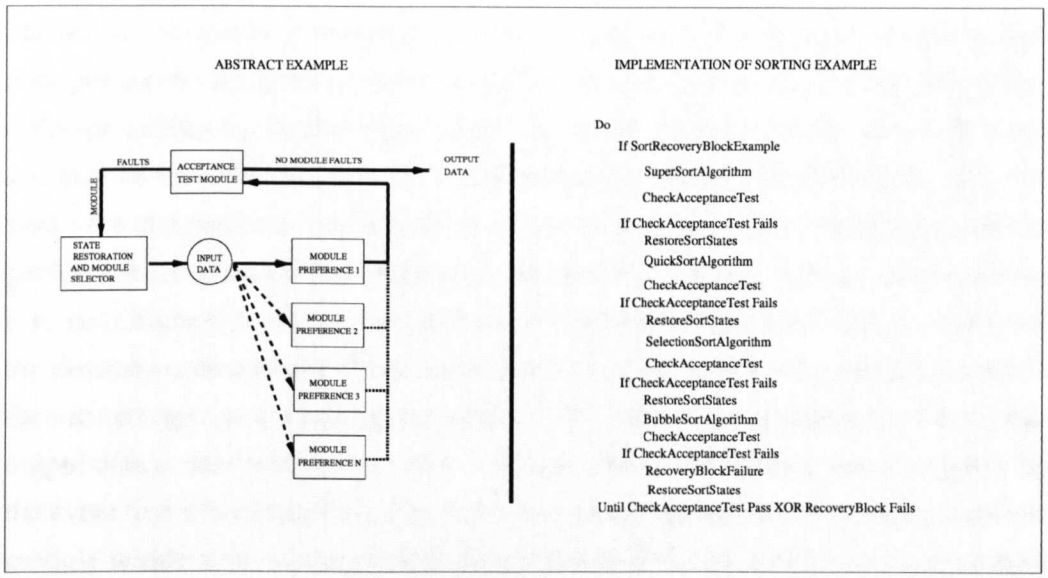


Figure 2.6: Recovery Block Example [adapted from source : [12]: pp 410-13]

> *better try and move out of the line of fire immediately, instead of waiting until you feel the bullet making contact with your skin."* [8]

Therefore, although, unlike feed-forward control, feedback control can be very useful in controlling unanticipated system disturbance. The nature of the disturbance, however, must be of a recoverable nature for that particular system, in order to enact this form of error control successfully.

A good example of feedback error control in the fault tolerance literature is the recovery block architecture [cf. [13, 4, 7, 12]]. An abstract example is illustrated in figure 2.6. It can be seen that, like the TMR architecture in subsection 2.4.1.1, it utilizes multiple module channels. However, unlike TMR, these are not activated in parallel with an adjudicator module to provide fault–masking, instead, they are activated in a sequence hierarchy — often based upon a performability/reliability preference protocol. For instance, input data is first passed to the module channel with the highest preference. The output data from this (highest preferred) module channel is then checked against some general acceptance criteria (more often

[8] In this example, the damage a bullet can cause is clearly anticipated and (often) unrecoverable from, therefore, feed-forward control is a much more attractive control mechanism to employ.

known as *"acceptance checking"*) in the acceptance test module. If the output data passes the acceptance test criteria, the output data is used within the wider software artifact for further processing, etc. In the event, however, that it fails the acceptance test criteria, control is then passed to another module that: first, restores the computation states back to a state before the highest preference module performed its (presumed erroneous) computations; second, selects an alternative (i.e. next highest preference on performability/reliability etc criteria) to carry out the desired computations. The output data from this (next best) module channel is then tested against the acceptance criteria. If it passes the acceptance criteria, the output data is used within the wider software artifact for further processing etc. In the event that this output data also fails, then, once again, control passes to another module which restores the computation states and selects a different computation module channel (third best choice). Once again the input data is then passed to this particular module channel for computation processing. The output data is then checked against acceptance criteria and if it passes the output data from this module channel is used within the wider software artifact for processing. This cycle continues through however many recovery block channels are included (in figure 2.6 four recovery block channels are illustrated) until either one passes the acceptance test criteria, or all recovery block channels have failed.

In figure 2.6 a pseudo–code implementation example of a recovery block architecture is also given. In this case, the recovery blocks provide different sorting algorithms. The overall recovery block is enclosed within a "Do–Until" loop which will iterate at least once (assuming no faults in the actual acceptance test module and state restorer module). In this example selection of alternative recovery block module channels is handled by the sequence logic — instead of a separate activated module. The highest preference sorting algorithm is the *'SuperSortAlgorithm'* (probably because of its performance in sorting data very quickly). Providing this algorithm results in no dormant fault state activation that fails the *'CheckAcceptanceTest'* module, its sorted output data will be utilized in the wider program (not shown in figure 2.6). If this sorted output data does fail, then the sequence logic will result in the *'RestoreSortState'* module being called to provide the initial computation states prior to the *'SuperSortAlgorithm's'* execution. The

'QuickSortAlgorithm' will then be used to sort the data. If the *'QuickSortAlgorithm'* passes the criteria in the *'CheckAcceptanceTest'* module, its sorted output data will be utilized in the wider program. If it fails, then the sequence logic will result in the *'RestoreSortState'* module being called to re–initialise the data to that which it was prior to the *'QuickSortAlgorithm'*. The *'SelectionSortAlgorithm'* will then be used to sort the data. If the *'SelectionSortAlgorithm'* passes the criteria in the *'CheckAcceptanceTest'* module, its sorted output data will be utilized in the wider program. If, however, it should also fail to pass the acceptance criteria, the sequence logic will, once again, result in the *'RestoreSortState'* module being called to re–initialise the data to that which it was prior to the *'SelectionSortAlgorithm'* module being executed on the data. Finally, the *'BubbleSortAlgorithm'* will be used to sort the data. If the *'BubbleSortAlgorithm'* data outputs passes the *'CheckAcceptanceTest's'* criteria, then the *'BubbleSortAlgorthim's'* output data will be utilised in the wider program. If it fails, however, with this four block recovery example in figure 2.6 on page 35, then there is no more redundant module channels to enact feedback recovery. In this situation the flow of control passes to the *'RestoreSortState'* and records that the overall recovery block error control mechanism's status has failed. Depending upon a) the overall criticality of service delivery of the software artifact; and b) the specific criticality of the data sorting operation in the program on service delivery, the recovery block failure status may just result in a failure report outputted to the user(s), or some other software fault tolerance mechanisms may be employed to handle this particular recovery block failure situation.

A number of issues are raised by this recovery block example — in relation to feedback error control and redundancy in software artifacts. Firstly, it is possible to see how a particularly critical component of the recovery block mechanism is the sensitivity of the acceptance test criteria to detect when a error state occurs in any one of the executed sorting module channels. Failure to be sensitive enough to all the possible error states that could occur in a sorting algorithm will quickly result in a false positive acceptance of the (erroneous) data sorting output and this could, if undetected elsewhere in the software artifact, lead to an accumulation and propagation of fault, error, failure in the wider system artifact — right up to

the service interface potentially resulting in judgements of incorrect service delivery (i.e. system failure). Secondly, sequential, instead of parallel execution, along with the active feedback nature of the recovery block architecture, means that, unlike fault–masking, it is less desirable for systems and applications where timeliness of error control are critically important — as the recovery block approach involves carrying–out acceptance checking, restoration of data states, and re–execution of another algorithm increases processing overhead. Thirdly, an important feature of the recovery block approach to achieving fault tolerance allows a gradual degradation of service delivery. In the fault-tolerance literature this is often referred to as a failure mode of *"graceful"* degradation of service [1], whereby, in the presence of faults, a system fails in a gradual, predictable, and controllable manner. This can be seen from the example of different sort algorithms in figure 2.6, each one lower down in the preference hierarchy performs (or sorts data) in a less and less efficient manner (assuming large amounts of data are involved). Finally, the recovery block architecture, unlike exception handling, allows for the toleration of unanticipated residual faults in the software artifact.

2.4.1.4 More Sophisticated Error Control

The three basic forms of error control can be combined to produce more sophisticated error control when the system in question is more complex. A good example of combined feed-forward and feedback control in software fault tolerance is Co–ordinated Atomic Actions (CAA) [4]. CAA fault tolerance architecture is often used to co–ordinate error recovery control along multiple independent and concurrent processing threads of required computation. When a dormant fault state is activated and an error state is subsequently detected in one (or more) of the concurrent processing threads, internal control intervention can be enacted in two ways. Firstly, feed-forward error control may be attempted by bypassing the anticipated error state (much like exception handling) and substituting the error state with a future non–error state in a single processing thread, providing it can be coordinated at some future check point with the other concurrent processing threads. Secondly, if feed-forward error control is not possible (i.e. the fault was unanticipated, etc), then feedback error control will be attempted. This in-

volves coordinating all of the independent and concurrent processing threads by employing backward recovery to a past check point state and allowing concurrent processing to continue from there.

These control theoretic explanations of established fault–tolerant mechanisms, whilst useful for situating the software fault–tolerant literature within a broader category of system control theory, are not part of the normal terminology used within computing science. Therefore it is appropriate to highlight the comparisons between the two:-

- **Fault Masking** relates to the broader system theory of buffering control whereby possible causes of disturbance of the system–of–interest can be automatically prevented without any direct active control by the system taking place. In achieving software fault tolerance this is exemplified by such mechanisms as triple modular redundancy that can automatically filter out any computation fault by a voting adjudicator;

- **Forward Recovery** relates to the broader system theory of feedforward control whereby prior knowledge of possible causes of disturbance of the system–of–interest and its environment allow anticpatory control to prevent the cause of disturbance before it is experienced by the system. In achieving software fault–tolerance this is exemplified by exception handling techniques and other such mechanisms (i.e. defensive programming) that can anticipate such error states before computation takes place and substitute these for error–free states;

- **Backward Recovery** relates to the broader system theory of feedback control whereby knowledge of possible causes of disturbance of the system–of–interest and it's environment are not possible in advance, but only subsequent detection of a disturbance once it has occurred within the system. In achieving software fault–tolerance this is exemplified by such mechanisms as recovery blocks which can only detect a fault once it has been computed through some acceptance criteria and then must restore the system before providing some alternative processing module.

As we have seen with the case of coordinated atomic actions, more sophisticated fault–tolerant mechanisms may incorporate more than one of the general system control approaches (i.e. feedforward and feedback control) in providing greater dependability of the system.

2.4.2 Broader Software Artifact Redundancy Issues

Whilst the examples of buffering, feed-forward, and feedback error control represent well known usages of software artifact redundancy in the fault tolerance literature, a number of other classifications of fault tolerance and goals of software artifact redundancy also need mentioning. In the subsections that follow, an important classification and related software artifact redundancy issues will be discussed in subsection 2.4.2.1. In subsection 2.4.2.2 some broader purposes of software artifact redundancy, and how they contribute to the promotion of dependable artifacts, will also be considered.

2.4.2.1 Computation Redundancy Classification(s)

It can be seen from the software redundancy examples in section 2.4.1 that, broadly, redundancy can be categorised into either [14]: a) Multi–version redundancy — such as the examples of TMR and Recovery Blocks; or b) Single–version redundancy — such as exception handling. Each of these categories present their own issues and implications for employing software artifact redundancy, and these are covered below.

Multi–Version Redundancy

Multi–version fault tolerance is essentially the incorporation of two or more variants of a software algorithm that is either executed in sequence (i.e. Recovery Blocks) or in parallel (i.e. Triple Modular Redundancy) [14]. The underpinning justification for multiple versions of the same required computation(s), in a single software artifact, is that should one variant fail, then at least one (and possibly more) variant(s) will be able to continue computation and provide the necessary outputs.

While multiple version redundancy pre–dates its usage in software fault tolerance in improving the reliability of computer hardware via replication of hardware components [15], physical replication of hardware components demonstrate truly independent (or random) failure behaviour [16]. Therefore, its usage in achieving software fault tolerance is essentially based upon the assumption that if different individuals or teams develop individual versions in isolation then these algorithmic variants will also, like replicated hardware components, fail independently (or randomly) ensuring that, through the independence law of probability theory, the overall reliability of the multi–version computation function will be the product of the individual reliability of each algorithmic variant [17]. This assumption with software, however, was later exposed to be flawed since, although multiple algorithmic variants do provide reliability gains over any single algorithmic variant [18], a number of studies [cf. [19, 20, 21]] have shown that the assumption of truly independent failure between multiple software variant versions is limited in its ability to provide truly independent failure to tolerate design faults. This limitation can result in dependent failures due to multiple versions containing common faults whereby all (or some) of the algorithmic variants can fail simultaneously (e.g. two (or more) independent TMR module channels produce the same (erroneous) data output(s)).

A final consideration with multi–version redundancy, is the additional development cost involved in producing multiple algorithmic variants for certain computations in the software artifact [14]. Even for safety–critical software artifacts, complete development duplication (i.e. requirements, specification, design, coding, testing, etc) would prove to be extremely costly, however, studies have indicated that the cost of developing two versions is not equal to twice the costs [22]. Furthermore, it is often found that even in safety–critical software, only a subset of the software system's functioning or computations are considered sufficiently safety critical to justify multiple–version redundancy. In these circumstances, development costs can be reduced by only applying diverse development for those parts of the software.[9]

[9]Although it should be noted that other uniform or homogeneous software artifact(s) — such as a common requirements document, specification, architecture, etc do increase the likelihood of

Single–Version Redundancy

As the term implies, in contrast to multi–version redundancy, single–version redundancy is focused on improving a given piece of software's ability to detect the presence of residual faults [14]. Therefore, one of the primary purposes of including additional redundancy in a single version is to enhance fault state sensitivity during computation. This can be achieved in a number of ways [23]:-

- **Reverse–checking.** Providing that the computation function is *'transparent'* [9] — in that no information loss occurs between input data and required output data, then an alternative computation can double check that the used computation has not produced erroneous output by comparing computed outputs with actual inputs provided to the computation function. For example, the denomination algorithm for converting required total cash amounts at an Automatic Teller Machine (ATM) into available denomination amounts (e.g. £20, £10, £5 amounts) represents a transparent function — where no information loss occurs, as the actual input total cash amount can be reproduced by multiplying the denomination types by the total number of each denomination of each type and then summing.[10]

- **Check–Digits.** Check digits may often be used in single version redundancy to detect transcription and transposition errors in important data and computations during software execution. A longstanding example of check-summing is MODULUS 11, which will detect all transcription and transposition errors — as well as 91% of random errors [24]. MODULUS 11 is performed as the following example shows: a) Original code number = 4214; b) Multiply each digit by the weights 5432 giving (4x5) = 20, (2x4) = 8, (1x3) = 3, (4x2) = 8; c) Sum the products, giving 20 + 8 + 3 + 8 =39; d) Divide by modulus 11, giving 39 mod 11 = 3 remainder 6; e) Subtract

common mode failure between subsequent algorithmic variants in the eventual software produced.

[10]For example, the actual data input amount required to be dispensed is (say) £100.00. And the the denomination algorithm computes output amounts of two £20 notes, four £10 notes, and four £5 notes. The computed outputs can be compared by reverse checking computation with the actual inputs by the computation (2x20) + (4x10) + (4x5) = 100 to ensure that they are of equal value. In the event that they do not equal each other, then some error has occurred in the denomination algorithm.

the remainder, giving 11 - 6 = 5; f) 5 now becomes the check digit which is then added as redundant data to the end of the code number 4214 to be stored giving 42145. The check digit is then used to detect if any corruption to the code occurs through faulty future computation(s). This is carried out as follows and the checking result of the number should yield zero. 42145 is calculated (from the least significant digit) by the the weighting 54321, giving (4x5), (2x4), (1x3), (4x2), (5x1). This gives the sum 20 + 8 + 3 + 8 + 5 = 44. This is then divided by mod 11, giving the remainder 0 — demonstrating to a high level of confidence that no fault corruption during computation(s) has occurred.

- **Assertion Checking.** Assertions are additional conditional software statements coded into the software artifact to define what should always be true about the computation states concerning some particular function [12]. Assertions are often placed:-

  - At the beginning of a function and are often called preconditions. They define what the are the allowable state ranges in procedure or function parameters;
  - At the end of a function or procedure and are often called postconditions. These assertions determine what are the allowable exit state ranges that can result from the computation(s) within the function or procedure;
  - Within a control loop. Such assertions define an allowable and invariant state range before, and after each iteration of the control loop;
  - At the head of classes in object oriented languages. These are sometimes referred to as class invariants, and define what are allowable state ranges before and after any public method calls of the class.

- An example of a class assertion upon a stack class is provided by Bell [[12]: p. 401]. With a stack structure data can be added to the stack by calling the method *push*, and taken off the stack structure by calling the method *pop*. Assuming the stack structure is of a fixed size defined by a constant named

capacity, and the number of data items at any time placed onto the stack is defined by the variable *count*, it is possible to increase the class's fault state sensitivity to be able to detect activated faults (i.e. error state(s)) by employing the following class assertions:-

- In the stack class, the class invariants can be defined as: 1) ASSERT (COUNT >= 0); and 2) ASSERT (CAPACITY >= COUNT). Assertion number one ensures that any value *count* has must be greater than or equal to zero, meaning that *count* can never be a negative number as a negative amount of data items on the stack is a nonsense and such a value would indicate some error state situation. Assertion number two captures the fact that because the stack is of a fixed size (defined by the constant *capacity*), the number of data items placed onto the stack (defined by *count*) must never exceed this fixed size.
- Individual procedure based assertions for calling each of the class methods *push* can also be defined as: 1) as a precondition to calling the method the assertion, ASSERT (COUNT < CAPACITY) is used; and 2) as a postcondition of the method the assertion, ASSERT (COUNT' = COUNT + 1) is used. The precondition assertion number one ensures that the method *push* is not invoked unless the size of the data items presently on the stack is, at least, one less than the maximum number of data items that can be placed onto the stack. The postcondition assertion on exit from the push method, ensures that for every call of the push method, only one extra data item can be placed onto the stack.[11] Anything other than this is considered an error state.

The examples provided above are not intended to be a complete coverage of how single–version redundancy can be used to increase error detection, instead, they are used as a subset of examples to emphasise how extra system structure can improve fault state computation sensitivity. In fact, often, multi–version redundancy approaches may often need to incorporate some aspects of single–version

[11]For example, if through some fault, a single method call could place two data items onto the stack when the size of count = capacity, or count = capacity-1, then this would result in the maximum size of the stack being exceeded.

redundancy in some parts of the overall architecture to ensure sufficient fault state sensitivity in such components responsible for check points (i.e. CAA) and acceptance checking (i.e. Recovery Blocks), etc. There is always the concern, however, of how adding extra software redundancy to the artifact can result in increasing overall software complexity (with its potential for side–effectual fault causation) and how this could mitigate the overall performability of the software artifact during operational usage [14]. These are facets of dependability requirement decisions that have to be taken into consideration and compared with the expected dependability benefits of employing such redundancy in the artifact.

2.4.2.2 Structural Redundancy Issues

It can be seen from subsection 2.4.2.1 on computational redundancy classifications that the categories of multiple and single version redundancy were primarily focused on introducing redundancy to either duplicate computational functionality, or to provide additional redundancy to increase fault state sensitivity to check computation. Therefore, both of these forms of redundancy were concerned directly with contributing fault toleration during computation. However, other software artifact redundancy incorporated into the software artifact is concerned with promoting fault control or promoting certain dependability attributes without being directly involved in the computational functionality of the software artifact. In this regard, such redundancy is directly related to either: a) preventing, protecting, and/or containing software faults via stronger system structure; or b) promoting other important dependability attributes through improving the system structure. These issues of structural redundancy are discussed in the subsections that follow.

Improving Security of Data

Under normalisation rules in relational databases, it has long been the goal of database design to remove redundant storage of data by applying a number of normal forms (i.e. first normal form, second normal form ... etc). This removes repeating groups of data, etc — with many–to–many correspondence. Whilst this is generally regarded as good relational database design practice, the dependability attributes of security can be promoted by violating this practice and ensuring

1) IMPROVING INTERPRETATION	2) IMPROVING COMPREHENSION

A) BAD EXAMPLE
`if ( ! (block_id < actblks) || !(block_id >= unblocks));`
B) GOOD EXAMPLE :
`if ( (block_id >= actblks) || (block_id < unblocks));`
	A) BAD EXAMPLE
	`*x += ( *xp= ( 2 * k < (n-m) ? c[k+1] : d[k--]));`
	B) GOOD EXAMPLE :
	`if ( 2 * k < n-m)`
	`  *xp = c[k+1];`
	`else`
	`  *xp = d[k—];`
	`*x += *xp;`

Figure 2.7: Redundant Structure and Comprehension [source: [25]: pp. 6–7]

that highly sensitive data (e.g. a bank customers address details, personal identification number (for credit cards and ATMs), and bank card number, etc, which could all be used to commit fraud or card cloning) are placed into a separate table with increased permission access restrictions. In this case, usual rules of normalisation are dispensed with as such data and information will result in repeating groups of data (i.e. one–to–one correspondence), however, by doing so, the attribute of security (i.e. availability, confidentiality, and integrity of the sensitive information and data) is promoted through introducing redundant software structure to increase protection and prevention from malicious unauthorised access.

Improving Reuse and Maintainability of The Artifact

Since many real world software artifacts must be maintained and evolved, an important dependability attribute is maintainability of the software artifact — in terms of corrective, adaptive, and enhancement changes that become necessary. System structure, in this regard plays a big part in how easily a software artifact can be corrected, altered, and improved [cf. [26, 27, 28]]. In this regard, system structure, in terms of its degree of information hiding, coupling, and cohesion, has not only a part to play in protection against unintended side–effects that can cause faults, but also how easily the software artifact can be reused and its logical structure comprehended:-

1. **Improving reuse and extension:** Riel [29] highlights that introducing redundant artifact structure(s) — in the form of decoupling containers in

an object oriented language promotes ease–of–reuse and future extension through ensuring that necessary class methods retain message passing autonomy;

2. **Improving understanding:** Kerningham and Pike [25] provide numerous examples of how incorporating (essentially) redundant structure(s) into the software artifact can greatly improve understandability of the code. In figure 2.7 two of their examples are illustrated. The first example 1) improving interpretation illustrates how conditional logic can be improved by using positive forms of conditional logic that is more clearly and natural to interpret. Kerningham and Pike note that in the top example (i.e. A) that the conditional logic uses a negative form which is always more difficult and less intuitive to interpret correctly, whereas, in the second example (i.e. B) the logics have been transformed into positive conditional forms that are more intuitive and natural to interpret. The second example 2) improving comprehension is a clear indication of how additional redundant structure(s) can greatly aid comprehension of the code. In the top example (i.e. A) the code is obfuscated within the rich C syntax where all the conditionals, operators, and statements are crammed into (what at first glance looks like) one line statement. Whereas, in the lower example (i.e. B) the structure of the code is spread over several lines and explicitly uses the more normal *if–else* conditional statements which reveals that the processing required is actually a number of different statements and, because of these (essentially) redundant coding structures, now becomes far easier to understand.

The examples provided here were not intended to be exhaustive, but to illustrate that the incorporation of additional redundant structuring of the software artifact can greatly improve both fault toleration — in terms of fault prevention, protection, and containment of faults, as well as facilitate and promote other important dependability attributes of, for example, security and maintainability.

2.4.3 Limitations of Software Artifact Redundancy

Whilst there is little doubt that the introduction of redundancy into the software artifact to improve fault tolerance has significantly increased the dependability of software systems, software redundancy — in the form of fault tolerance, also has limitations that are worth mentioning. These are briefly discussed in the subsections that follow.

2.4.3.1 Limitations of Error Control

If we look deeper into the essential three forms of system regulation, upon which, in one form or another, all fault tolerant mechanisms are based, it is possible to highlight a number of restrictions. Firstly, in terms of passive buffering error control — such as triple modular redundancy (TMR) fault tolerant schemes, it is possible to see that such control is really only effective against purely random system disturbance. This has introduced controversial claims that such mechanisms of TMR, whilst effective against accidental development faults, are likely to be much less effective against intelligent and informed malicious faults — such as those that undermine the security of a software system [30]. Secondly, in terms of feedback error control mechanisms, there are three main concerns for such fault tolerant mechanisms — such as recovery blocks:-

1. **Failure Prohibitive.** As mentioned earlier in subsection 2.4.1.3, one of the main problems of feedback error control is that the disturbance must be of a type which the system can recover from. For example, in the Therac–25 accidents [31], even if there had been some feedback error control fault tolerance (i.e. Recovery Blocks, etc) that detected lethal doses of radiotherapy doses after they had been administered and then recovered to prescribe the correct doses, it would have been of no real value, as it is not possible to de–administer a lethal dose after it has taken place. This is an example of system deviation of service that is unrecoverable from (in terms of the patient). With such heavily safety critical systems, it is preferable that they have fail–silent failure modes [1]. However, contrast this serious and unrecoverable failure with that of a wrong dispensation of cash from an automatic teller machine (ATM). Here, feedback error control is feasible, as

the ATM, after dispensing the wrong cash and detecting so, can either a) make–up the short–fall immediately; or b) inform the customer that it (the ATM) is aware of the short–fall cash dispensation and assure the customer that they will only be debited for that reduced amount. In this case, the unanticipated error can be recovered from in a satisfactory manner.

2. **Time Prohibitive.** Even when the system disturbance is of a type that the system can recover from, there is still the issue of whether there sufficient time, in the wider system the feedback error control regulates, to enact feedback error control? For example, in many real–time computer control domains, such as Air Traffic Control Systems (ATCS), a residual and unanticipated fault, error, failure chain propagation occurrence that could (at least temporarily) wrongly direct, stack, or queue waiting aircraft (perhaps travelling at speeds of 200 plus mph) may not allow the fault tolerant system sufficient time to detect, restore, redirect, process, and test acceptance of another (say) recovery block alternative and then provide the correct(ed) airspace co–ordinates before a mid–air collision occurred. As mentioned earlier in subsection 2.4.1.1 passive buffering error control (such as TMR) that provides sufficient parallel computation redundancy to (inactively) mask–out such an error state is more appropriate for time/safety critical software control systems.

3. **State Prohibitive.** In order to provide feedback error control the regulation component of the system must have sufficient state representation to be able to enact such recovery of unanticipated residual fault activations. For example, if due to incompleteness in the original software artifact specification, an ATM's controlling embedded software did not provide for a physical and digital state representation of the amount of cash present in the physical cash magazines, then once the cash magazines have been emptied (or insufficient cash left to fulfil a customers cash request) then there is no possibility of detecting or recovering from such an unanticipated residual fault activation — as the up–stream fault phenomenology results in undermining detection of the system deviation.

It can be seen from these three examples, that they present some limitations and concerns of feedback error control. In order for these issues to have become more amenable to feedback error control it would have been necessary to enact prior detection at a deeper level within the software hierarchy (e.g. prior detection of lethal dosage administering data indicators, etc). However, as mentioned earlier, with feed-forward error control in subsection 2.4.1.2 this places a heavier reliance upon the effectiveness of the fault–forecasting means in the creation process. To an extent, this also would defeat the purpose and value of feedback error control, as its value and contribution, in a broader and general system control sense, is used to control unanticipated system disturbance (or unanticipated residual faults in the software artifact) and other error control approaches — such as feed-forward control, would be more appropriate once such disturbance had become more anticipated.

2.4.3.2 Increasing Artifact Complexity

As has already been indicated, there is a trade–off between incorporating additional computational or structural redundancy into the software artifact and increasing the overall complexity and its consequential potential for also increasing more residual faults. Therefore, the expected benefits of increased dependability and associated development costs involved with a particular application by employing certain fault tolerance mechanisms must also be compared with the potential it presents for increases in systemic complexity and its consequential possible effects on undermining the achievement of dependability status required.

2.5 Chapter Summary

This chapter has considered the many ways in which software dependability can be improved via a fault–tolerant approach. The existing dependability framework provides an encompassing generic framework for capturing the desired goals to be achieved; the means by which these goals can be attained; and the threats to undermining software dependability. A crucial issue in achieving fault–tolerance is the introduction of computational and structural redundancy to aid continuance of

correct service delivery in the presence of faults. Essentially, fault–tolerant mechanisms can be categorised in a broader system theory sense as error–control. This chapter has discussed three ways to achieve error–control, namely: passive buffering; feedforward control; and feedback control. Each error–control approach has its own advantages and weaknesses which need careful consideration and knowledgeable analysis of the particular failure modes, domain criticality, and operational demands if dependability requirements are to be correctly unearthed. In the next chapter, dependability will be sought in terms of the creation process with a focus upon the fault–avoidance means of promoting software dependability.

Chapter 3

Dependable Software Processes

3.1 Chapter Introduction

Whereas chapter 2 focused upon improving software dependability via a fault–tolerant approach, this chapter considers how software dependability can be improved via a fault–avoidance approach in the creation process to prevent, detect, remove, and forecast faults.

The chapter first considers some major problems that the software development process suffers from before progressing to a view of a dependable software creation process that is in keeping with the existing approach adopted by the dependability community.

3.2 Problems in The Software Development Process

There are many longstanding issues that surround the software process. Software projects have proved to be amongst the most difficult projects to manage — with many projects being abandoned, delivered over–schedule, delivered over–budget. In addition, even if software is delivered on time and budget, the software system may not provide the expected benefits envisaged, or suffer from rejection by its intended users. It is possible to list, broadly, the types of failure that the software process can experience as follows:-

- **Process Failure:-**
  - Software delivered over planned schedule (i.e. project management failure);
  - Software delivered over planned budget (i.e. project management failure);
  - Software abandoned due to economic infeasibility (i.e. economic failure);
- **System Failure:-**

- Delivered software does not provide envisaged strategic benefits (i.e. expectation failure);
  - * Delivered software was delivered too late to exploit the potential benefits (i.e. opportunity failure):[1]
  - * The optimism placed in delivery of the software was misplaced (i.e. conceptual failure);[2]
- Delivered software does not provide the essential functionality required (i.e. functionality failure);
- Delivered software is rejected by the intended users (i.e. deployment failure);
- Delivered software is abandoned due to the infeasibility of the technology to be created (i.e. technology failure);[3]

While process failure and system failure are separated, in the listing above, the view taken in this chapter is that all of these failures are inherently connected with the software process, since, as the proverb goes – *"product always follows process"*. Furthermore, it should be noted that these failure types are not isolated, but often have a cyclic causality dynamic (i.e. technical complexity or infeasibility is likely to cause schedule/budget failure, which in turn can result in economic failure, etc). Issues surrounding such failures as expectation failures and deployment failures require an expansion of the overall system boundaries — concerning the software creation process, these are introduced and discussed in more detail in chapter 4. In this section, directly associated complexities — concerning the software creation process, and how it can increase the potential for such failures

[1]This is not to be confused with a budget overrun, as the software system may have been delivered to planned schedule, but (for instance) a competitor had already seized the advantage before the software system could be delivered;

[2]This is not to be confused with some form of functionality failure or deployment failure, as the essential envisaged functionality may have been delivered as required, and the intended users enthusiastic about its usage, but the overall beliefs in its strategic advantages were misunderstood or poorly conceived.

[3]This is not to be confused with the dependability consequences of technical failure to deliver the service once in operation, instead it is referring to misunderstanding the sheer technical complexity involved in creating the system that results in failure to deliver the software system. A good example was the U.S. Governments' *"Star Wars"* project in the 1980s cf. [32].

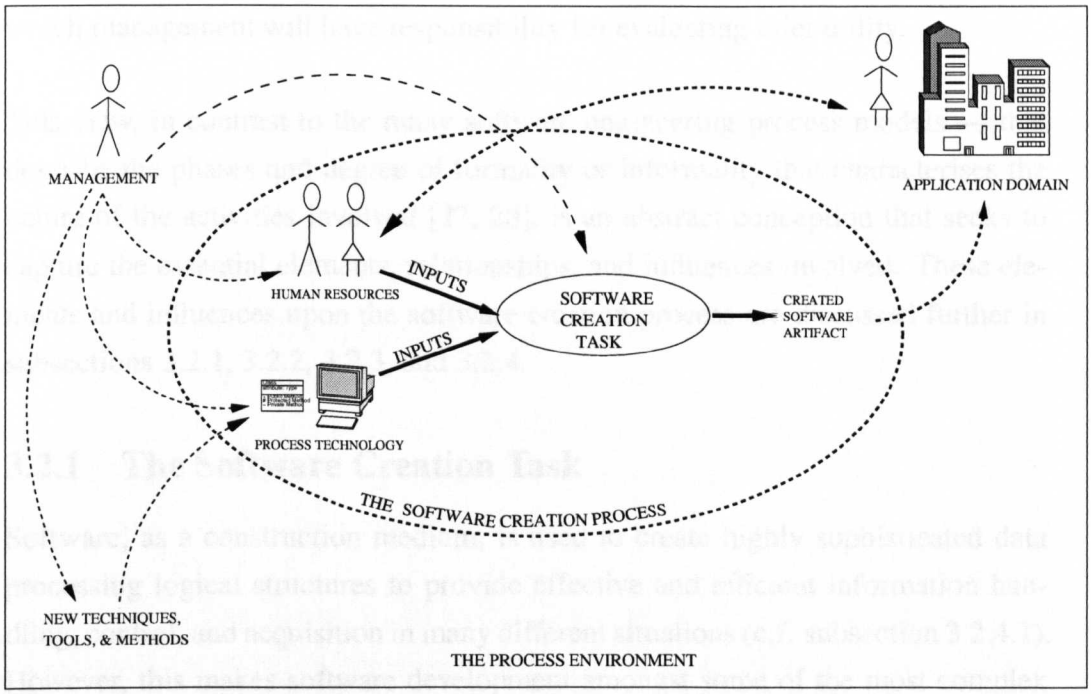


Figure 3.1: Abstract View of The Software Process

will be discussed. The essential areas covered are illustrated in figure 3.1. The diagram captures the essential entities and relationships involved in the software process. The entities are: a) management; b) human development resources; c) the process technology (e.g. tools, methods, techniques and programming languages); and d) the applicational domain. The relationships are represented by the dashed arrows. It can be seen that management has responsibilities for such entities as the human resources, process technology and overall planning, coordinating, and controlling these entities (as resources) in the software creation task. The double headed dashed arrow between human development resources and the applicational domain represents the respective responsibilities for eliciting and refining requirements. In a more scoping manner, it can be seen that the software creation task (dashed elipse) encompases the entities of human resources and process technology as inputs to the software creation task that produces the eventual software artifact. Furthermore, new process technology is (like the applicational domain) considered as an environmental influence that indirectly influences the software creation process as new techniques, methods, etc become available —

which management will have responsibility for evaluating thier utility.

This view, in contrast to the many software engineering process models — that describe the phases and degree of formality or informality that characterises the nature of the activities involved [27, 28], is an abstract conception that seeks to capture the essential elements, relationships, and influences involved. These elements and influences upon the software creation process are discussed further in subsections 3.2.1, 3.2.2, 3.2.3, and 3.2.4.

3.2.1 The Software Creation Task

Software, as a construction medium, is used to create highly sophisticated data processing logical structures to provide effective and efficient information handling, control, and acquisition in many different situations (c.f. subsection 3.2.4.1). However, this makes software development amongst some of the most complex systems a person or group of people could ever attempt to construct [33, 34]. Complexity, however, is a vague and overused term in many areas and software complexity can be also interpreted in many different ways, such as the number of operations an algorithm performs, the space/time complexity (i.e. as denoted by O notation), or the number of routes possible through a program. Here, to provide one specific type or measure of software complexity, it is possible from considerations of software testing to highlight the huge data space complexity of software by showing that even a small software program can possess extraordinary large numbers of possible data states. Firstly, it should be noted that with many other construction mediums, the complexity of a structure (such as, for example a building) only increases linearly with the size of the structure being created. One of the distinctive aspects to the nature of software, is that although, like other construction mediums, its complexity also increases with its size, this can do so exponentially. A piece of software ten thousand lines of code long, is not necessarily ten times more complex than a piece of software one thousand lines long — its complexity, in terms of possible achievable data states, could be up to (say) a hundred times more complex. In fact even in relatively small software programs, the data state complexity can be enormous. This is clearly indicated by Pressman [27]

who indicates that a hundred line program can be written in the C language that contains only two nested loops that executes (a maximum) of only twenty times each contains a combinatorial data state complexity of 10^{14}. Pressman puts the software complexity, and exhaustive testing infeasibility, of this trivial program into perspective, by noting [p. 470]:

> *"...assume that some magical test processor ("magical" because no such test processor exists) has been developed for exhaustive testing. The processor can develop a test case, execute it, and evaluate the result in one millisecond. Working 24 hours a day, 365 days a year, the test processor would have to continually work for 3,170 years to exhaustively test the program."*

Real–world software systems are vastly more complex that this trivial example. When this complexity is combined with the human potential to make mistakes, slips, and oversights, in the software creation task, it is not suprising that software development is an inherently error prone activity. This not only effects the dependability of the eventual delivered artifact (and therefore the necessity of fault–tolerance, covered in chapter 2), but can also play havoc upon issues of managing and controlling the actual development work (cf. subsection 3.2.4.2).

Another problem involved with software is that it is intangible by nature [35]. The only physical form software really takes, is as a pattern of high or low voltages stored on temporary or permanent magnetic computer hardware [15]. As a result, there is an unusual representation, interpretation, and communication problem — whereby the various phases of the software creation process (i.e. requirements, design, etc) need to capture, characterise, and communicate the required behaviour of the software artifact in many different ways and levels of abstraction — suitable for that particular phase. This causes a number of difficulties. First, interpretation can be highly error–prone, whereby interpretations of one representative form at a certain process phase and abstraction level can become incomplete or misinterpreted — this can not only directly result in residual faults that, subsequently, compromise the dependability of the software artifact during operational usage, but the reworking, revising, and corrective maintenance it can

cause presents serious management problems and project risks. Second, the extra workload introduced in representation can extend project budgets and schedules. Whilst professional software engineering wisdom advocates careful technical documentation and traceability in the software development process, there is often a serious separation between theory and practice [12]. Not only do developers often not document their work, but when they do, it can sometimes be conducted after the development work for that phase has already been performed [32], or done in a hurried and incomplete fashion. Additionally, as project budgets and schedules come under increasing pressure to meet milestones and delivery targets, there is also the temptation amongst management to view the documentation work as of secondary importance. Finally, in contrast to many other engineering disciplines, the intangibility of software makes the necessary measurement for planning and control over the creation process extremely difficult. With other engineering disciplines (i.e. civil engineering), even when lower levels of formal project planning and tracking of progress are introduced [36], management can often retain acceptably high levels of control over the project via regular visual inspection of the work.[4]

3.2.2 Human Resources

As figure 3.1 indicates, within the software creation process, human resources are one of the two essential inputs to the actual software creation task. As Constantine emphasizes [[37]: p. 17]:

"Good software comes from people. So does bad software."

[4]While doing my masters thesis I had the opportunity of reviewing a number of different projects. Most were I.T. related, but one was a building refurbishment project. While interviewing the project manager on this project I was surprised to find both the level of informality in planning and controlling the project and yet the overall satisfaction of the levels of process control claimed by the project manager. After commenting about this to him, he informed me that in tracking the progress and exercising overall control of the project he operated a policy of *"control–by–walking–around"* the building three to four times every day. He recited numerous occasions of how this allowed him to assess the progress of the work and detect and prevent problems early before they had time to have a serious impact. In contrast, although there was often more attempts made to formally plan and track the I.T. projects, most of the project managers commented that they felt heavily reliant upon only individual progress reporting by the developers involved — which often proved overly optimistic and frequently contributed to missed milestones and delivery overruns.

This comment actually highlights an ongoing issue regarding the variability of human performance — regarding software development. Amongst programmers there have been many studies on productivity that have found enormous performance variability measures upon code production between individual programmers. Studies of over three hundred software organisations, conducted by Demarco and Lister [38], led them to conclude that, over a sample of any software programmers using any productivity metric, the following productivity performances tend to prevail [p. 45]:

- Count on the best people outperforming the worst by approximately a ratio of 10:1;
- Count on the best performer being approximately 2.5 times better than a median performer;
- Count on the half that are better–than–median performers out–performing the other half by more than a ratio of 2:1;

However, large individual performance variability not only occurs in the programming task. Brooks [34] has long argued, from his experiences of managing the production of the IBM 360 operating system in the late 1960s, that, during the design phase, to help ensure the conceptual integrity of the software, the architecture should be the product of (at most) one or two talented software designers. In fact, to accommodate such human performance variances, and help ensure the eventual quality of the product in software development, Baker [39] and Baker and Mills [40] proposed a custom—based team composition especially for software development, characterised as *"The Chief Programmer Team"* . With this team composition, only (at most) two talented designers were allowed to take on the design and coding roles, whilst other team members fulfilled supportive roles of administration, documentation, etc. The rationale being that, in terms of both software quality and productivity, two talented designers/programmers can produce better software faster, than a group of designers/programmers of mixed ability. This, however, goes against conventional view of the benefits of team work, and suggests that the nature of the software creation task is not additive, in nature.

Social Psychologists have identified that, in group work, there are essentially four types of task natures that can be identified [cf. [41]]:-

1. **Additive Type Tasks:** in which the contributions of each group member can be predictably combined into an overall group performance. Examples include a) brick laying; b) moving a heavy object; and c) selling a product, etc. In all these examples, the group's output is determined by the sum of the individual efforts;

2. **Conjunctive Type Tasks:** in this case, the group's final product is largely determined by the weakest individual performance(s) of the group. A good example includes a mountaineering group. In this situation, the overall group can only progress as fast as its slowest group member(s). This is an example of the *"weakest link"* effect;

3. **Disjunctive Type Tasks:** with this type of task, the overall group's performance is largely dependent upon the performance of the strongest or most competent individual group member(s). With this type of task, it is not only necessary for the most competent group member to identify a viable solution, but he/she must also be able to convince the other group members of its viability. In a survey of group dynamics, by Hill [42], it was revealed that often: a) the verifiability characteristics of a particular task; b) socialising effects — such as pressure for conformity; and c) individual assertiveness (i.e. emergent leaders), often play an important influence in the ability of the most competent group member being able to attain solution acceptance, by the rest of the group. This is an example of the *"strongest link"* effect;

4. **Compensatory Type Tasks:** in this case the contributions of the individual group members are averaged together to form a single group outcome. The limitation here is that it is only feasible for tasks that can be reduced to an average — such as forecasting, estimating, etc. In such task situations, the benefit of averaging the overall group efforts is that optimistic predictions are off–set by pessimistic predictions and the average prediction or estimation tends to be more accurate than any single prediction of a group member.

With regards to the nature of the software task, such findings and team compositions indicate that software development is either a) a conjunctive type task — i.e. lower end performers limit overall group performance; or b) a disjunctive type task — i.e. team composition interventions need to take place to permit the best performers to reduce errors, and increase quality and productivity.

The intangible nature of software also appears to have negative group performance effects. Brooks [43] first illustrated that, unlike many other types of engineering projects, the nature of software development in group-work cannot be factored out to accelerate project work to expedite project schedules or reduce fixed projects costs by increasing the manpower [cf. [36]]. He noted that the software task places a heavy reliance upon learning and interpersonal communication to co–ordinate the work between developers. Once extra manpower is added to an existing project, exponential increases in communication overhead actually results in slowing down activities on productive tasks. This was also reinforced by Gordon and Lamb's [44] analysis of such production losses in analysing the effects of the *'learning–curve'*.[5] However, they argued that such slowing down effects are only temporary learning effect delays that include: a) "coming up to speed" on the task; b) acquisition of specific knowledge; c) having to teach/train other (new) group members; and d) the need for task(s) coordination and communication. After a time, these learning–curve effects diminish and the group will start performing at a greater collective productivity level. Gordon and Lamb [44] therefore argued for adding extra developers early on in the project before schedule acceleration was required — in order to accommodate for such learning effects before project acceleration was needed.

3.2.3 Process Technology

Due to the essential complexity, error–prone nature, and intangibility of software, there is an increasing need placed upon assistive process technology — in the

[5]Gordon and Lamb note that the 'learning–curve' relates to either a) the acquisition of essential task skills; or b) the acquisition of specific knowledge for a particular task. In referring to Brooks Law of software projects, they are concerned primarily with the latter (i.e. the acquisition of specific knowledge required).

form of tools, methods, and techniques. However, there are a number of concerns surrounding: a) the suitability of process technology; and b) the effectiveness of process technology. These are briefly considered in the subsections below.

3.2.3.1 Suitability of Process Technology

An indication of software engineering process immaturity is that it suffers from a saturation of tools, methods, and techniques — each of which proposes to be the answer to developing high quality software in a predictable manner. While, many of these make certain improvements and have certain strengths and weaknesses, appraising their suitability — in terms of advantages and disadvantages for a particular software development system and application domain is very difficult and can be the cause of problems in the software development process. Unfortunately, management often believe that greater productivity, quality, and project control can be achieved simply by the employment of some new tool, method, or technique. For instance, from his experience in managing large information system (IS) projects, Hallows [45] argues against the temptation of management buying and employing new and revolutionary process technology (i.e. CASE tools, methods, and techniques) as the learning overhead they present, under tight schedule/budgeting duress, is often not accommodated for explicitly within the project planning scope and can result in serious project delays and artifact defects being introduced. Such issues, concerning process technology, have began to suggest that instead of improving developer productivity, software quality, and project control — process technology is potentially becoming subtle causalities of failure of software projects [46].

3.2.3.2 Effectiveness of Process Technology

The traditional approach to software engineering has been to concentrate primarily upon achieving fault–avoidance in the software development process through increasing the sophistication of tools, methods, and techniques that both guide and constrain the developer from introducing faults into the software artifact or which improve detection and removal. While, by comparison to earlier generation ad–hoc approaches [47], these have no doubt raised the level of both software

engineering and software dependability,[6] this progress is more than matched by the pace of technological advancement and associated commercial drivers that demand software controlled systems that involve unprecedented increases of application novelty and technical complexity [cf. [48, 49]].

An over reliance upon process technology in software engineering has been criticised on three fronts. Firstly, it has been argued that is has never (or is ever likely to have) solved the fundamental problems that software engineering has always presented [34]. Secondly, improved methods, tools, and techniques, can actually act as drivers themselves to the development of vastly more complex software controlled systems — cancelling out any process support technology gains originally expected [48]. Finally, an over emphasis of systematic tools, methods, and techniques in the development process has been criticised for motivating a *"one–size–fits–all"* solution–orientated paradigm that can often stifle rigorous problem analysis in many cases [50].

3.2.4 The Process Environment

The software development process does not take place in an environmental vacuum, direct process environmental influences upon the software development process include the management of the process and the complexity of the application domain into which the eventual software artifact will be deployed.

Other, less direct influences, such as the wider organisational structure and culture can also impact (positively or negatively) upon the dependability of the software process [cf. [51, 52, 53]], however, such influences are considered to belong to the realms of organisational development (OD) and are considered out of scope of this thesis.[7]

[6] This is particularly true of improved language design, formal approaches, computer–assisted software engineering (CASE) tools, and integrated development and debugging environments [cf. [12, 27, 28, 48]]

[7] This is a practical consideration, not a principle one, and the reader should not infer that the wider organisational structure and culture of the software engineering organisation are considered unimportant.

3.2.4.1 Application Domain

Software is the ultimate isomorphic machine [54]. By this, it is meant that software functionality can be created in a universal manner to emulate or simulate almost any behaviour required. This means that its context of application extends way beyond the limits usually imposed by other engineering disciplines such as civil or mechanical engineering, etc that have definite applicational limitations. In the early years of business computing, software was often used to automate well–established existing systems — such as centralised payroll systems, etc [55]. With the combination of increases of processing power and reduced costs of hardware, over the last twenty years or so, the value that can be realised from incorporating software control or processing of information in ever more novel applications has ensured that software systems have become increasingly complex and ubiquitous in society.

This facet places a heightened dependence upon software engineers to fully understand the particular nature of the application domain in the software creation process. This phase has often been referred to as the requirements engineering phase. Jackson [35] notes that natural language, if not carefully used, can introduce many interpretations that can result in erroneous definition(s) of application requirements. Furthermore, he notes that there are essentially two types of application requirements:-

1. **Domain Requirements:** these refer to indicative properties that a particular domain possesses — irrespective of the additional behavioural requirements that the software system will be designed to provide;

2. **Software Requirements:** these refer to the optative properties that the software system, itself, is to provide.

Failure to elicit and capture these requirements can result in a) incomplete requirements documents; b) inconsistencies being introduced; c) incorrectness — in the required behaviour of the software system; and d) ambiguities due to different emphasis and/or interpretations of the desired software behaviour. All of these

requirements failures can seriously undermine the eventual dependability of the deployed software system.

Advances in process technology, particularly precise formalism and modelling of requirements specifications [cf. [56, 57, 48]], with their increases of precise semantics, and validating proof techniques, have helped reduce ambiguities, inconsistencies, and incorrectness of requirements. However, such process technology, is not a panacea for solving all requirements problems, as such heavily mathematical modelling approaches are still vulnerable to incompleteness concerns through assumptions made upon the scope of the application's requirements. For such reasons, the value of, user–centered development, iterative prototyping and domain expertise in the requirements phases are also considered to be crucial [cf. [58, 55, 10]].

3.2.4.2 Management Issues

Management can be defined in many ways. In a broad interpretation, management involves many activities relating not only to the structure and decision–making of the organisation (i.e. strategic, tactical, and operational) [59], but also in *'softer'* terms — relating to the political and cultural climate of the organisation [60, 61]. Additionally, vague definitions often exist between the sociological and task differences of what distinguishes management from leadership. Kotter [62] argues that the essential differences between leadership and management relate to dealing with complexity versus dealing with change. He further divides this distinction into three dimensions, as follows:-

- **Direction vs Planning:** Direction is a leadership responsibility which involves gathering large amounts of information to assess patterns, relationships, and linkages in order to provide a vision to explain things. By contrast, planning is a management responsibility designed to establish order and produce predictable and orderly results;

- **Aligning people vs Organising or Co–ordinating people:** Alignment of people is a leadership responsibility and involves ensuring that everyone moves towards common goals of the organisation — in times of change.

Organising or coordination of people is a management responsibility and relates to ensuring that established plans can be resourced as precisely and efficiently as possible. Typically, this involves complex decision–making relating to the process structure, task structure, matching of skills to tasks, and training;

- **Motivating People vs Controlling:** Human Motivation is a deep psychological and social psychological area of study [cf. [63]]. Nevertheless, leadership plays an important role. It is not sufficient to provide directive visions and collective alignment, in times of change, without also ensuring enthusiastic interest and support and preventing barriers to change and resistance. Likewise, in terms of management, it is not sufficient to establish effective plans and suitable organisation of resources without effective monitoring and tracking mechanisms to detect when deviations from established plans, target, and goals occur.

In the context of this thesis, although there are many issues and arguments for how such managerial aspects of culture, organisational structure, political climate, and leadership can improve or undermine the effectiveness of the software creation process [cf. [37, 38, 64]], this subsection focuses upon direct managerial influences relating to process control, namely: a) planning; b) organising or coordination; and c) monitoring or tracking. Drawing upon issues and problems already raised in this section, with the software creation process, these essential management elements are considered in terms of how the nature of the software creation process can result in undermining their effectiveness in achieving process control. These are briefly discussed in subsections below.

3.2.4.3 Planning

Planning, is predictive, by nature. It relates to acquiring as much relevant information and data to deconstruct and order the work so that its progression can be reconstructed into an orderly scheme of work so that relevant tasks can be sequenced and appropriate types and levels of resources applied. Two particularly important aspects concern: a) in order to realistically assess the entire scope of

work to begin with; and b) estimate the amount of duration and costs involved in the performance of required tasks. Only by achieving a sufficiently accurate assessment of both of these aspects can a realistic and useful plan be determined. Two particular problems that the software creation task presents — relating to these two essential planning features, concerns: c) the problematical nature of ensuring adequate definition of the requirements scope — in its totality. Any omissions, inconsistencies, etc fundamentally undermine the integrity of the plan — resulting in unrealistic deadlines, budgets, and resourcing levels; and d) estimating task performance is another critical aspect of achieving a realistic plan. However, as discussed in subsection 3.2.2, the individual variability of software developers can lead to grossly optimistic task performance assessments — resulting in planned resource allocation levels being seriously inadequate.

3.2.4.4 Coordinating

Coordinating tasks, people, and process technology is essentially an interventionist managerial activity which will frequently occur in order to keep a project on–track. It is often a response to problems experienced during the project — such as unexpected staffing shortfalls, lack of adequate staff skills, budgeting or scheduling overrun predictions, etc. In order for process coordination to be effective, predictable effects of human performance, training, and the effectiveness of employed process technology is needed to be present. However, as has been discussed in previous subsections, the suitability/effectiveness of process technology, the variability of individual human performance, the unpredictable additive nature of collective or collaborative group performance, etc seriously undermines the manager's ability to achieve predictable effects.

3.2.4.5 Tracking

Project tracking (or monitoring) of project progress is essentially a goal–orientated activity. It, first and foremost, depends heavily upon the initial integrity (in terms of realism) of the original planning phase — as this is used to establish the time–scales, costs, and resourcing projections, against which, project progress will be tracked. An unrealistic or inadequate overall scheme of work will result in

early beliefs of serious problems with schedule overruns, budget overruns, resource allocation shortfalls, and missed milestone stages. Additional problems that the software creation task presents are: a) that of invisibility of work undertaken by developers. This firstly, seriously restricts the sensitivity of the tracking mechanism to adequately represent the extent of work to be progressed. Without such visible *'yard–sticks'* the project manager will be heavily dependent upon reported progress by the individual developers which can result in deviant practices that report either overly optimistic progress assessments, or deliberately incorrect progress reports.[8] Both of these can result in progress tracking becoming ineffective and preventing the project manager from taking remedial action — until serious (and potentially unrecoverable) schedule/budget/resourcing problems have become manifest; b) the complexity, novelty, and error–prone nature of the software task can all combine to result in project progress tracking losing sight of what constitutes progress. This occurs when faults and errors or requirements incompleteness/inconsistencies[9] etc result in so much and frequent reworking of tasks and phases (i.e. specification, design, coding, testing) that the whole shape and structure of the project renders the initial planning and (subsequently) tracking meaningless. As a consequence, a serious loss of overall process control results.

3.3 A Dependable Process View

In section 3.2 some fundamental problems associated with the software creation process were considered. To provide a more holistic perspective of these many problems and issues figure 3.2 shows them in terms of how these dynamics influence one another within the software creation process. The diagram is a graphical summary of section 3.2 and shows how the creation process and its process environmental elements illustrated in figure 3.1 interact and influence the overall dependability of the creation process. The lines reflect the many interacting dynamic influences that are implicit in the descriptive text of section 3. Further

[8]Such reporting progress dependency problems are synonymous with such issues as the SNAFU principle, and Parkinson's Law : *"A task will spend ninety nine percent of it's time ninety nine percent complete."*

[9]These could be also called faults — but they may not be the type that are so easily detected or can be tolerated.

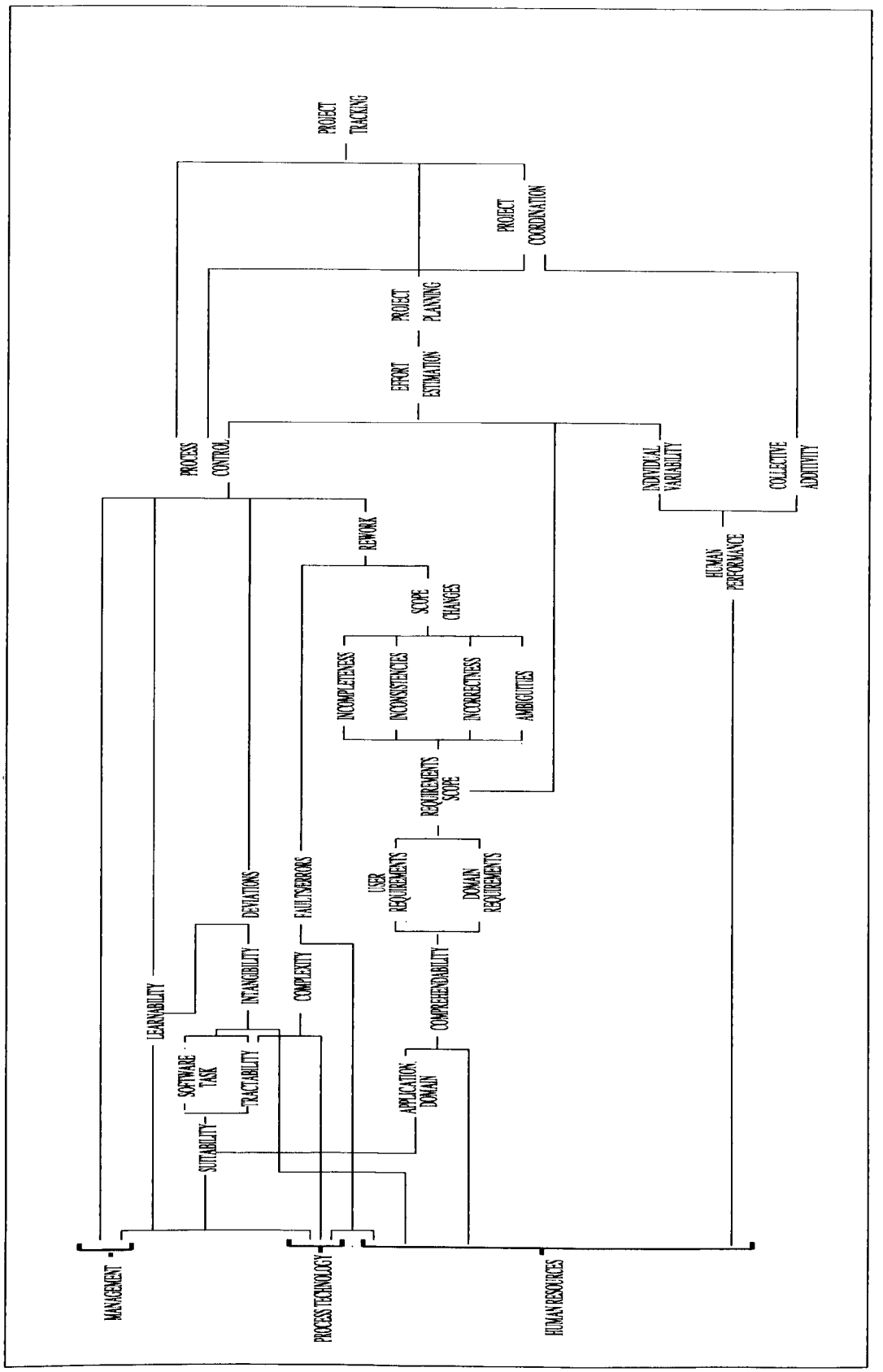


Figure 3.2: Process Dynamics

examples of such subtle process influences are given and explained in this section as a dependable process view is progressed.

From this perspective it is possible to see that many of the essential elements of the abstract view of the software creation process in figure 3.1 interact in complex cause/effect relationships which can improve or undermine the dependability of the process.

For example, improving comprehension of the application domain would not only help improve the dependability of the produced artifact — in technical terms, it would also greatly improve the overall process control of the process in two fundamental ways. Firstly, it would help reduce unnecessary rework when omissions, ambiguities, or inconsistencies were detected later in the downstream phases of the development process. Secondly, it would ensure that the estimation and planning of the work initially was based upon a more complete and consistent requirements set.

Figure 3.2 of an initial set of process dynamics, therefore, allows a consideration of what inter–related factors would need to be addressed in order to achieve a more mature and dependable software creation process. The idea of improving the maturity of the software creation process is not new in any way, however, The Software Engineering Institute at The Carnegie Mellon University has progressed a structured software development process improvement initiative called *"The Capability Maturity Model"* (CMM) for over a decade [cf. [65]]. The CMM approach to improving the maturity of the software creation process is for the software development organisation to gradually increase its process maturity in stages, each of which provides the foundations for the next level or stage upward. Overall, the CMM approach provides five progress stages or levels:-

1. Initial CMM — Level 1: The software process is characterised as being ad–hoc, and occasionally chaotic. Few process phases are defined, and success largely depends upon individual effort and heroics;

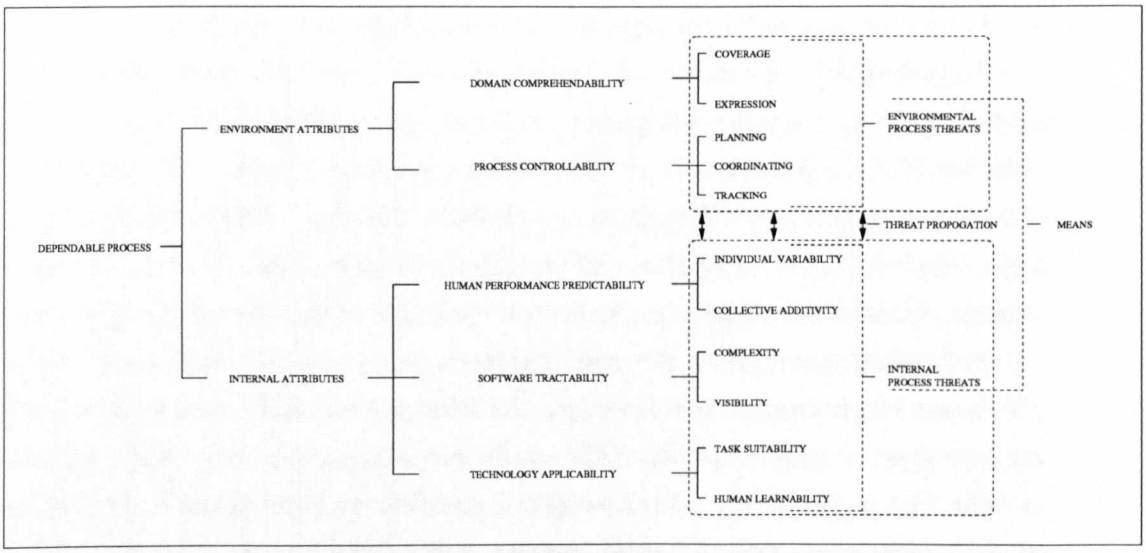


Figure 3.3: Attributes of A Dependable Process

2. Repeatable CMM — Level 2: Basic project management processes are established to track budgets, schedules, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications;

3. Defined CMM — Level 3: The software process for both management and engineering activities is documented, standardised, and is integrated to support the standard software process for the organisation. Projects apply a customised and approved version of the organisation's processes for creating and maintaining software;

4. Managed CMM — Level 4: Detailed measures of the software process and associated quality of software are provided and collected. The software process and the products produced are quantitatively appraised and controlled;

5. Optimising CMM — Level 5: The process is continually improved — aided by quantitative feedback in the process and from piloting new and innovative technologies and suggestions for improvement.

While empirical studies into CMM process improvement initiatives have been positive for improving the maturity of the software process and software product

quality — especially for organisations improving their processes from level 1 to level 3 [66], however, the CMM approach is not without its problems and critics that claim there is little evidence for indicating that such an approach actually results in higher quality software artifacts. Firstly, introducing a CMM initiative requires high levels of strategic management support if the initiative is to be successful and prevent resistance to changes in the work place [66]. Secondly, some have argued that the CMM approach introduces too much unnecessary bureaucracy and quality software is more reliant upon the quality and skills of the individuals involved than notions of better organised and supported processes [67]. Thirdly, there is the assumption that all the different capabilities in each level are achievable without other capabilities at higher levels. For instance, at CMM — Level 2, there is the assumption that rigorous formal project management can be implemented to improve the process in isolation from detailed measurement of the work (i.e. CMM level — 4). However, as has been discussed in section 3.2 (and illustrated in the dynamics of figure 3.2), effective process control — in terms of planning, coordinating, and tracking has antecedent task, human resource, and process technology requirements. Without improvements in these, it could be argued, merely introducing a formal project management approach will be likely to be inadequate to achieve effective overall process control.

The view taken in this chapter section is that improving the maturity and dependability of the process requires an holistic, not hierarchical, approach whereby multiple process attributes, and relationships between each other, must be understood and improved. Drawing upon issues and problems raised in section 3.2 and resulting process dynamics in figure 3.2 subsections 3.3.1, 3.3.2, and 3.3.3 employ a dependability approach to provide an initial view of the attributes, threats, and means by which to improve the dependability of the process. This view is illustrated in figure 3.3.

3.3.1 Process Attributes

In figure 3.3 a tree view of a set of dependable process attributes is provided — based upon the issues raised in section 3.2. These are separated further into: a)

process environment attributes — that introduce a direct influence upon the dependability of the software process; and b) internal process attributes — in terms of the essential inputs of the software process, and the actual transformational characteristics presented by the software task. It should also be noted that, as the process dynamics of figure 3.2 exemplified, cyclic cause/effect relationships exist between the environmental and internal process attributes (e.g. the influence upon human performance variability upon planning and coordinating etc). These could be referred to as inter–process dynamics. Furthermore, within each of these categories cause/effect relationships also exist (e.g. unsuitable methods, tools, or techniques for a given software task and its consequent potential for increasing the occurrence of faults, or insufficient domain knowledge resulting in incomplete or inconsistent requirements definition(s), etc). In subsections 3.3.1.1 and 3.3.1.2 these environmental and internal process attributes are further discussed.

3.3.1.1 Environmental Process Attributes

In this thesis the process attribute of domain comprehendability is essentially consistent with the definition provided by the dependability community in [68] [p. 29] and relates to the *"...representativeness of situations to which the computer system is subjected during its analysis compared to the actual situations that the computer system will be confronted with during its operational lifetime"*. With regard to domain comprehendability, it can be seen that unless sufficient coverage is provided, this can not only result directly in undermining the eventual dependability of the software artifact(s), through omissions, but also can directly result in lack of understanding through such omissions that lead to incomplete coherence of requirements demands and lead to ambiguities, inconsistencies, and incorrectness. Even if complete coverage is achieved[10] the requirements may be misunderstood, over/under emphasised, or mis–communicated in some way — leading to erroneous interpretations. Therefore, expression of covered requirements in a clear, unambiguous, and consistent manner is also critical to achieve correct comprehension of an application domain.

[10]Which, with any real world complex application domain, is unlikely.

Process controllability is also a critical attribute of a mature and dependable process, as, without process control, the resulting chaotic effects can easily result in increasing fault introduction in the eventual software artifact(s) (i.e. over—emphasis upon over–time working cf. [69] and its potential for increasing human error via fatigue and monotony) or reducing the creation process's ability to detect and remove faults introduced (i.e. through cutting or omitting important development phases such as validation, verification, etc to expedite project schedules, reduce project budgets, or meet phased delivery milestones etc). As section 3.2 and the process dynamics in figure 3.2, highlighted, process control is achieved via: a) effective planning to arrive at a realistic baseline of work scope, work sequencing, and effort allocation; b) effective coordination to intervene with predictable effects when problems are identified; and c) tracking or monitoring of work, in order to detect early deviations from overall project goals of time, cost and quality, so that effective remedial action can be enacted.

Therefore, these two environmental process attributes are defined as follows:-

- **Domain Comprehendability:** The ability to adequately cover and clearly express relevant domain, software, and user requirements in a complete, consistent, and correct manner;
- **Process Controllability:** The ability to produce accurate project plans and monitoring mechanisms, and enact corrective project coordination in a effective and efficient manner.

As stated earlier, however, while these are important environmental process attributes, they also rely upon other internal process attributes being accomplished, in order to be feasibly achievable.

3.3.1.2 Internal Process Attributes

To begin with, human performance predictability, it has already been discussed in section 3.2 that both uncertainties surrounding the individual variability and collaborative additivity can result in reducing process control — through inadequate effort estimation during planning and uncertain performance effects with

teams during project co–ordination. This can be seen as an instance of propagation effects via inter–process dynamics between the process environment and the creation process. However, as subsection 3.2.2 indicated, there are additional internal process performance concerns about human resources during collective team decision—making regarding sociological influences that can undermine collaborative performance. Furthermore, as indicated in the process dynamics in figure 3.2, on an individual performance level, there are also the concerns of how faults and errors are introduced with other internal process attributes — such as the complexity and invisibility of the software task, and the suitability of process technology for a software task. Therefore, human performance predictability is an important attribute towards both undermining and improving the dependability of the overall process.

The software task is also a major process factor that has characteristics of complexity and intangibility, that, combined, can directly undermine both the dependability of the resultant artifact (in terms of residual faults) and also seriously destabilise the software process control. Furthermore, as the process dynamics in figure 3.2 indicate, the nature of the software task can also combine with other internal process factors of process technology and/or human resources to further complicate this situation (e.g. developers not familiar with process technology, etc). Therefore, reducing the complexity of the task and increasing the software tasks' visibility is an important single and interrelated factor for improving both the dependability of the software artifact and the software creation process.

Process technology, in the form of the many tools, methods, and techniques that can be applied has, for a long time, been perceived as a fundamental process factor that improves both the dependability of the resultant artifact and raise the maturity and dependability of the creation process. Like the other internal process attributes of human resource performance and the essential nature of the software task, it can have propagation effects via inter–process dynamics (i.e. unsuitability of use for a particular application domain can result in increasing incompleteness, inconsistencies, etc). However, as shown in figure 3.2, it also has important internal interaction dynamics within the software process — such as improving/reducing

the complexity/intangibility of the software task or aiding/undermining individual or collaborative human performance. As a consequence, process technology, and its usage as an input to the creation process, also plays an important part in both improving the dependability of the resultant software artifact and increasing the maturity and dependability of the creation process.

These three internal process attributes can therefore be defined as follows:-

- **Human Performance Predictability:** The ability to forecast both individual and collaborative task performances in an accurate measurable manner;
- **Software Tractability:** The ability to employ, as process inputs, human resources and/or processes technology to reduce task complexity and increase development work visibility in an effective and efficient manner;
- **Technology Applicability:** The ability to appraise the suitability and determine the learnability of process tools, methods, and techniques in the context of a given application domain and software task in an efficient and effective manner.

Again, attainment (or at least improvement) in the state of these internal process attributes, influences, and is influenced by, attainment (or improvement) of the environmental process attributes mentioned earlier. [11]

3.3.2 Process Threats

Threats to achieving a dependable process, as indicated in figure 3.3, are divided into three categories of: a) Environmental Process Threats; b) Internal Process Threats; and c) Threat Propagation. These are briefly discussed in subsections 3.3.2.1, 3.3.1.2, and 3.3.2.3.

[11] For example, if no attempt is made to formally control a *real–world* complex development process, then this will undermine, in turn, the achievement and the benefit of those achievements, of the other internal process attributes. The result (most likely) is both a less dependable software artifact, and a less dependable process.

As discussed earlier in chapter 2, the view taken here is that the creation process is concerned with promoting dependability in the software artifact through a fault–avoidance approach. The existing dependability approach stops at the level of a fault in the fault–error–failure chain, and is not concerned with the many and varied ways of how faults occur (i.e. fault phenomenology). However, in considering the creation process, the view taken here is that greater understanding and responsibility of the fault phenomenologies is crucial in both promoting the attributes of a dependable process and improving the means by which fault–avoidance, in the software artifact(s), can be achieved. Therefore, in the subsections that follow, issues relating to how, in an undependable creation process, faults can be introduced into the software artifact are considered.

3.3.2.1 Environmental Process Threats

Environmental threats to achieving a dependable process relate to the direct influences of the process environment upon: a) management; and b) the application domain has upon both the dependability of the creation process and the dependability of the resulting software artifact — in terms of how these can cause fault introduction or ameliorate the creation processes' ability to subsequently detect and remove faults introduced. Two obvious threats are: a) violations and workarounds of legitimate software activities and phases in the creation process — by management deviancy prioritising economic or timescale factors over software artifact quality factors; and b) completeness, consistency, and correctness issues — due to the novelty of the application domain. However, these are only suggested as some of the main direct process environment fault phenomenology influences that can undermine both process and artifact dependability.

3.3.2.2 Internal Process Threats

Internal process threats to achieving a dependable process relate to problems that occur inside the creation process — namely, the inputs of human resourcing and process technology as well as the nature of the software task that can undermine the dependability of the process and introduce faults or undermine subsequent detection and removal of introduced faults. The possible fault phenomenologies

that can occur are many and varied and can result from one internal process factor source or through some combined relationship influences between two or all three of them. Some obvious examples include: a) insufficiently trained or skilled developers; b) inappropriate application of process technology for a particular software creation task; and c) the increase of human error (i.e. slips and lapses) due to insufficient experience with the process technology employed. The last example indicates how faults can occur from a combination of internal process factors. Again, these examples are by no means intended to be complete, but to merely indicate how the dependability of the process and the dependability of the resultant software artifact can be undermined through fault phenomenologies that occur through one or more internal process factors.

3.3.2.3 Threat Propagation

Threats can also propagate between the two levels of the process environment and the creation process. This is where a) environmental influences act as a negative causal influence upon the other internal creation process factors that increases the potential for fault introductions or undermine the ability of the process to subsequently detect and remove faults introduced. Some examples of this type of propagation were given in subsection 3.3.2.1; or b) where the internal process factors (i.e. human resources, etc) act as negative causal effects upon the process environment which then, in turn, results in negative causal influence back upon the creation process factors. An example would be inadequately covering all of the requirements, which then, invalidates the planning, which then results in a loss of process control which in turn results in omissions of important process activities and phases being properly conducted. Consequently, this can ultimately result in both introducing faults into the software artifact and undermining subsequent detection and removal following introduction. This would indicate that the underlying cause/effect relationships are cyclic, in nature, whereby feedback effects throughout the levels of the process result in negative reinforcement of any failure(s) of the creation processes' factors of management, application domain, human resources, process technology, and the software creation task(s).

3.3.3 Process Means

It was highlighted in chapter 2 that the view taken is that the creation process is responsible for the fault–avoidance means of : a) fault–prevention; b) fault–removal; and c) fault forecasting. From the process dependability issues already discussed in this section, it can be appreciated that in order to promote the maturity and dependability of the process, and also increase the dependability of the eventual software artifact, relies heavily upon greater understanding of the fault phenomenology.

Firstly, in terms of the means of fault prevention, in order to preclude faults being introduced in the software artifact, a greater understanding of how human resources and/or process technology can be developed and employed in such a way as to preclude fault introduction. Secondly, in terms of the means of fault–removal, in order to remove faults already introduced into the software artifact, it is first necessary, in order to remove faults, to understand how to human resources and/or process technology can be employed and developed to increase the sensitivity to detecting such faults. Following detection, it is then important to understand how human resources and/or process technology can be employed and developed in such a way so as to preclude or prevent further introduction of faults during the corrective activities of removing detected faults. Finally, in terms of fault–forecasting it is important to understand how human resources and process technology can be employed and developed in such a way as to increase coverage and prediction of the types of fault, errors, and failures that the eventual software artifact could be subjected to, during its operational lifetime, in order to both gain a greater domain understanding and rationalise what fault–tolerant mechanisms will be most suitable to be employed.

3.4 Process Redundancy and Diversity

Chapter 2 discussed how dependability can be achieved in the software artifact through the inclusion of additional computational and structural redundancy. Whilst replication and duplication redundancy is a well established method for improving

the dependability of physical structures, it was also mentioned that toleration of faults, in the software artifact, against design faults, required the introduction of diversity also. However, apart from the problems of achieving random failure and independence in software design, the nature of some design faults[12] are caused directly through erroneous human interpretations and judgements during requirements engineering and specification. This category of design fault often results in inadequate state behaviour representation – in terms of not providing the necessary state, logic, or functional behaviour which means the system is less amenable to software fault-tolerant approaches and relies upon improving software dependability through the means of fault-avoidance in the process that creates the software system [70]. Such intolerable design faults may often propagate into errors of inadequate software control over the system being controlled and emerge as failures of user expectations about how the system should behave. If this occurs, it will directly undermine the attributes of dependability upon which judgements of user confidence and system trustworthiness are ultimately based[13] [71, 4, 1].

This raises the question of how can human redundancy — in the form of human diversity, in the development process, be employed in as effective manner as in using it in the artifact to avoid and detect the introduction of such faults. In this

[12]Flawed assumptions made in the requirements engineering, specification, and conceptual design phases will result in errors-of-omission or errors-of-commission that result in the absence of desirable data, function, and/or structural representation. For example, an assumption is made that: *"There is always cash in the ATM."* This will result in the state of the physical cash in the cash magazine(s) not being represented in the software control of the ATM. If the ATM dispenses all of its cash, then the next customer to use the ATM will receive no cash (or be short-changed) but will still have their account debited for the full amount. This is due to the assumption there is no consideration of such a situation to begin with and therefore no provision for data, logic conditions or functional behaviour to accommodate it. Such a design fault is not amenable to recovery or treatment during operational execution and will result in judgements of failure by the user even though any comparison of the ATM specification and implemented system reveal isomorphic behaviour (i.e. correctness).

[13]For example, a design assumption that, *"Up to a final commit stage, the user should always be allowed to cancel the initiated ATM transaction without data change side-effects"* (i.e. data alteration), results in the (usual) banking security policy of only allowing three PIN entry violations before reclamation of card becoming violated, since a fraudster with a cloned or stolen card can now potentially enumerate all possible 10^4 PIN combinations by cancelling the unauthorised transaction after two failed attempts. This is again due to the absence of some desirable persistent data state and functional computation that records and retrieves the number of previous failed PIN attempts independent of the current ATM transaction.

section, a number of approaches that incorporate the employment of human redundancy and diversity will be discussed. However, before doing so, it is important to distinguish between notions of classical and engineering interpretations of redundancy. The classical meaning of redundancy in normal usage refers to someone or something that is no longer required, needed, or wanted.[14] In contrast, the meaning of the term redundancy used in engineering contexts refers to the provision of additional components in a system, over and above the minimum required to perform the function, for the purpose of achieving reliability or robustness of the system.[15] The two interpretations bring into consideration whether only the minimum function of something is required or whether other properties of something are considered sufficiently important to justify unproductive functional additions. This comparison clearly indicates that the engineering interpretation is congruent with the classical definition in terms of recognising that components will be functionally unproductive, but departs from that classical definition in as much as that the functionally unproductive additions are critical in contributing to some other desired purpose, property or output (i.e. reliability) of the system. This is made explicit in Lewin and Noaks' ([15]: pp 413) discussion of the purpose of engineering redundancy in contrast to the classical definition for achieving greater computer system reliability, noting:

> *"...enhancing the reliability of a computer system is to use the principle of redundancy by duplicating various parts or functions of the system. Note that the additional equipment is redundant only when considered in the sense of providing the* ***basic*** *system requirements: it is, of course, essential if increased reliability is required."*

In this section the issue of process diversity will be discussed in terms of the essential elements of the software process from section 3.2: a) the software creation task in section 3.2.1; b) the human resources in section 3.2.2; and c) process technology in section 3.2.3. These will be discussed in terms of how human redundancy and diversity can promote the dependability attributes of the process discussed earlier in section 3.3.

[14]This is the definition provided by the Oxford Dictionary.

[15]This is the definition provided by the Oxford Computing Dictionary.

3.4.1 Fault–Avoidance and Fault–Tolerance

So far, from chapter 2, the discussion may have appeared to present fault-avoidance and fault-tolerance as contrasting approaches to achieving increased software dependability. This is not actually the case as both approaches are required in promoting software dependability, since, intuitively, the fewer residual faults a fault-tolerant mechanism must detect, treat, and recover from, the less potential there is for a residual fault occurrence which the mechanism cannot tolerate. Therefore, although a quite strict delineation of dependability responsibilities between the creation process and created artifact was provided in section 2.3.3 of chapter 3, the creation process can also be viewed as a particular system–of–interest in its own right (i.e. a system that creates another system). When viewed in this way the question of: *"How can the system that creates another system be made more fault–tolerant in avoiding the introduction of faults into the created artifact?"* can be posed. In this respect we can see that a creating system also requires process redundancy to prevent, detect, and remove faults in order to tolerate the presence of imperfect management, human resources, and process technology within the creation process. Finally, improvements of fault-avoidance in the process improves the reliability of the individual redundant fault-tolerant components employed [15] — thereby directly raising the overall dependability of software synergistically through both fault-avoidance and fault-tolerant means.

In the following subsections the issue of how human diversity via human redundancy has been used to promote fault–avoidance will be discussed.

3.4.1.1 The Software Creation Task

It has already been presented in chapter 2 how providing redundant diverse designs and implementations of required artifact functionality can increase the overall dependability of the eventual software artifact via design diversity and forced diversity. It can be seen, therefore, that such approaches can be focused at also improving the fault–tolerance in the software artifact, not fault–avoidance in the actual creation process.

3.4.1.2 Human Resource Redundancy

Although, it appears, that the software engineering community has placed significantly less emphasis upon the role of human diversity in the software process a number of exceptions do however exist. One notable exception is the consideration of many individual, collaborative, and organisational error vulnerabilities in the requirements engineering phase to promote fault-avoidance from an interdisciplinary perspective of psychology and computer science [72]. Another long-standing example is Weinberg's advocacy of open and informal code reviewing to improve fault detection and correction at the coding phase in order to help avoid coding faults into the eventual deployed product [64]. Finally, the recent rise of open-source software development (OSSD) is another example where greater emphasis has been placed upon the role of humans, rather than process technology, to achieve fault-avoidance through large scale deployment of human resources for massive code reviewing [73]. OSSD has directly motivated others to consider and experiment with the increased fault-avoidance benefits on other process phases (i.e. requirements engineering) to reduce security specification vulnerabilities and incompleteness through employing a similar parallel approach of human resources [74].

However, many of the approaches make the assumption that employing multiple human resources for a given task will provide sufficient levels of diversity to explore the requirements space and/or subsequently detect faults introduced. A number of approaches to achieving process and product diversity through the use of human-redundancy currently exist in the software engineering and dependability communities and each of them are based upon some assumption about the diversity of humans and the manner in which achieving diversity in the process can be achieved, as follows:

1. **Natural Diversity:** This approach is primarily used to achieve process-diversity for fault-avoidance (examples are: [74, 73, 64, 75]). The approach makes the assumption that functionally redundant human resources are sufficiently diverse to promote a dependable process through the means of the extra fault-avoidance they provide.

2. **Forced Diversity:** This approach is primarily used to achieve product-diversity for fault-tolerance. Unlike the natural diversity approach, it does not accept the assumption that functionally redundant human resources are sufficiently diverse. Therefore, in its simplest form (often referred to as *design-diversity*) interaction and collaboration between functionally redundant human resources is prevented to preclude any design copying or solution influencing that could result in common faults in diverse channels or algorithms (examples are: [7, 76]). In its more sophisticated form (often referred to as *forced-diversity*) developers will also be forced to use different process support means (i.e. tools, methods, and techniques and sometimes different specification representations and designs) to further reduce the likelihood that by using the same process support technology the same common faults could result in diverse channels or algorithms (examples are: [77, 78, 79]). However, some forced–diversity approaches are more creation process orientated. One such example is provided by Littlewood et al [16] who demonstrate through both statistical modelling and statisically analysed empirical data that there are strong arguments for the fault detection benefits of factoring out fault detection project effort over a range of diverse fault detection techniques (e.g. inspection, testing etc) than using all of the project effort utilising only one fault detection technique.

3. **Composed Diversity:** This approach has been little used to improve fault-avoidance directly in promoting dependability (exceptions are: [80, 81]) and has essentially been the preserve of psychology and social psychology to study human diversity performance effects on group problem-solving (examples are: [82, 42, 83, 84, 85, 86, 87]). Again the assumption that functionally redundant human resources are sufficiently diverse to promote a dependable process through the means of greater fault-avoidance is rejected. However, unlike forced-diversity, the approach is to influence diversity in the performers (not the task) by composing teams/groups of individuals who are intrinsically diverse on some pre-tested human dimension (e.g. personalty, cultural background, attitude, values, intelligence, etc).

It can be seen from these categories that while adding redundant human resources can improve the levels of human diversity, increases in human diversity through human redundancy can be achieved by direct intervention in the creation process.

3.4.1.3 Process Technology Redundancy

Process technology is also another dimension of the creation process which can improve fault avoidance through diverse application of tools, methods, and techniques. A good example of utilising diverse process technology during development was experimented by Kelly et al [77]. In this research they wanted to explore the fault–avoidance benefits of employing three diverse formal specification methods (using Estelle, LOTOS, and SDL) which were then *'back–to–back'* tested to compare for coincident faults. Kelly et al found that the redundant (and diverse) application of multiple formal specification languages found 25 of the total 40 (62.5%) faults early in the development life–cycle at the end of the specification stage. Furthermore, during back–to–back testing phases, the diversity introduced into the specification stages avoided any co–incidental or common–mode faults.

3.4.2 Justifying Process Redundancy

As discussed in the previous section on process–redundancy, for human–redundancy to be viable, the nature of the task must be additive in nature or capable of being made more additive through some form of process intervention. With *natural diversity* the assumption is made that the task is naturally additive,[16] while the other two approaches do not accept this assumption and intervene in the process in an attempt to increase the additivity of the task.[17]

Another consideration with respect to employing human redundancy to gain greater diversity in the development process is to attempt to determine the contribution

[16]By *'additive'* it is meant that the human resources are (to a satisfactory level) considered factorable in the performance they collectively produce on a given task. An example wood be brick laying as defined in section 3.2.2 by social psychological studies of group performance.

[17]Forced diversity uses social interaction and process support technology constraints to stimulate diversity, and composed diversity uses intrinsic performer characteristics constraints to generate diversity.

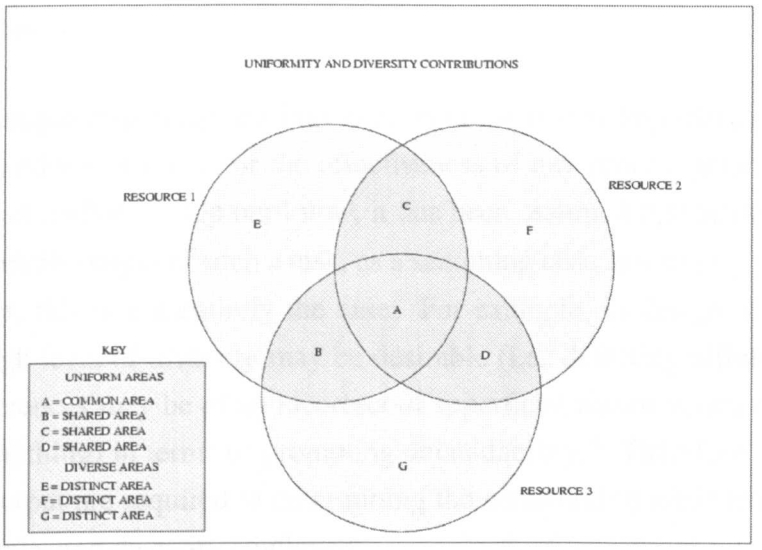


Figure 3.4: Uniformity and Diversity Contributions

that diversity makes, since, while diversity within the task itself is always required – when employing human-redundancy, the desirable outcomes from the task may consider diverse output not to be additive.[18] This distinction between the task and the desirable output of the task is necessary in determining the additive contribution that is made by employing functionally unproductive human resources in promoting dependability. For example, consider the set diagram in figure 3.4.

If the task nature is a searching task, then the desirable output contributions by the three resources employed in the diagram is generation or identification of different things. In this case, the more things that lie in the areas of *E, F, & G* the better (i.e. they increase the additivity between the resources as a performance of the group of resources). However, if the desirable output from employing human diversity is uniformity (such as a calculation task), then the more things that lie in the area *A* the better (i.e. confidence in the correctness of the calculation is additive in this

[18]For example, if the task nature is a *searching task*, then the desirable and additive output from the task is generation and recognition of different things (i.e. diverse requirement consideration, diverse design criteria generation, or diverse fault detection). However, if the task was a complex calculation task to compare outputs then although the task nature still requires diversity, the desirable output from the task is corroborated answers – not diverse answers. Therefore, in beginning to propose some form of measurement of diversity, it is important to make a distinction between the nature of the task and the desirable additive output required from the task.

type of task).[19]

This example does raise one final consideration that is important in determining the additivity of the task or the effectiveness of any process intervention to improve task additivity. Up until now, it has been assumed that achieving diversity as a desirable output of such a task, as a searching task, is ultimately a good thing. However, this is not entirely the case. For example, in design or code reviewing a high level of diversity may be desirable (i.e. detecting different faults) but the differences may be of an incorrect or superficial nature which contributes little (if anything) in terms of promoting dependability.[20] Therefore, two measures of the output are required in determining the contribution made from any human redundancy and diversity employed:

1. **Productivity effect.** This is a measure of the desirable additive output from the task. So in a searching type task this would be a measure of difference;

2. **Quality effect.** This is a measure in purely value or fitness-for-purpose in terms of promoting dependability. No consideration to difference is made — just its contribution in promoting dependability.

The two measures combined provide a good indication of how effective human redundancy has been in promoting dependability on a task, or how effective some process intervention has been in promoting dependability through employing human-redundancy on a task.[21]

[19]To give a more traditional example, consider a TMR scheme: it has three components (resources, 1, 2 & 3) two of which at any one time are functionally redundant or unproductive (in the classical sense), but essential in masking out faults if a safety-critical system function is not to fail (i.e. dependability contribution). The nature of the task they provide requires computational diversity (achieved through process intervention to stimulate greater potential of achieving computational diversity). The desired additive output, however, does not require difference but corroborating (and preferably identical) data. Using the set diagram, the most desired additive output is area *A* while any of the shared areas *B, C or D* are also desirable for fault-masking through consensus voting by the adjudicator. If areas *E, F and G* result then drastic alternative actions are required.

[20]Worst case would be the identification of false-negative faults during the inspection/review that could result in needless rework, time, and cost. Therefore, some qualitative measure is also necessary.

[21]It should be noted though that a low productivity effect means that the redundant human resources employed were contributing little and in terms of their dependability contributions were

3.5 Chapter Summary

In this chapter the issue of promoting software dependability via a fault–avoidance in the software creation process has been considered. Software development suffers from a number of ongoing problems that can undermine the dependability of the created artifact. These essential problems have been condensed down into a number of essential process and process environment factors. The view taken, in this chapter, is that a dependable software artifact is dependent upon a mature and dependable creation process. An approach to achieving a dependable process, that is consistent with the existing dependability framework, has been presented and discussed. It has been argued that, improving the dependability of the process, requires an holistic and integrated view that captures the goals that must be achieved in order for a creation process to be considered dependable. The threats mandate, unlike the existing dependability framework, a greater understanding of the fault phenomenology — in order to better understand how process technology can be effective, and how it should be resourced so that faults can be prevented or faults introduced can be better detected and removed. Finally, the issue of process redundancy to improve fault–avoidance was introduced. It was discussed how process redundancy means employing extra effort, that, while not functionally productive, are considered crucial in promoting other dependability properties through some means of achieving greater levels of fault-avoidance in the process or fault-tolerance within the product. For process redundancy to be justified, however, the nature of the task must either be sufficiently additive to extra effort or be capable of being made sufficiently additive through some form of direct process intervention. The purpose of employing process redundancy, in the process, is to gain greater levels of diversity to improve the fault–tolerant nature of the creation process through increasing fault–forecasting, fault-prevention and detection (i.e. fault–avoidance via human and/or technology diversity) and thereby increase the dependability of the software artifact produced. A number of approaches already exist to achieve human diversity in the process. These either accept that the nature of a task is sufficiently additive or attempt to make it more additive through direct

classically redundant. A low productivity effect may only be justified on safety-critical systems, as even though little diversity is actually achieved, the extra diversity may have resulted in the avoidance of a fault that could have resulted in a high consequence failure.

process intervention. Determining the additivity of process diversity requires two measures that 1) measure the degree of diversity achieved — in terms of a productivity effect; and 2) measure the value or fitness-for-purpose of the contributions made — in terms of a quality effect. These combined measures are suggested as an indication of how effective the process redundancy involved has been in promoting dependability through fault–avoidance.

Chapter 4

Computer–Based Systems

4.1 Chapter Introduction

Chapter 3 discussed the many issues involved in the software creation process along with a view of characteristics of a dependable software process. In this chapter the broader system considerations involving a computer–based system (CBS) viewpoint will be introduced and discussed.

The chapter first examines in section 4.2 what is meant by a system view by illuminating the many established issues involved in analysing and perceiving systems. Next, in section 4.3, a wider computer system conception of a computer as a computer–based system is provided using the many system analysis issues from section 4.2.

4.2 System View

Before discussing the issue of a computer–based system, it is necessary to consider what is typically meant by the term *"system"*, as loose reference and usage of the term is widespread and has often resulted in some computer scientists avoiding the term altogether to prevent vagueness [cf. [50]]. A systems viewpoint is a generic conception that attempts to explain and understand the workings of some natural or artificial entity of interest. It is sometimes loosely applied to refer to a legal system, an ecological system, an economic system, or a computer system etc. However, implicitly in all these usages and references, the term *"system"* implies some degree of organisation (i.e. as a verb to *organise*). Therefore, we can list the characteristics of a system as follows:

1. **System Environment.** Unless the system is closed from its environment, the environment of the system will often exert indirect influences upon the system — forcing the system to respond and/or adapt to changes. The nature of the environment is an important facet in systems theory. Cooke and Slack [59] highlight that there are two aspects to consider with respect to the environment of a system. Firstly, is the system environment complex or simple? Simple environments are those that exert only a few well known

disturbances and influences upon the system. By contrast, a complex environment will exert a large number of relatively unknown disturbances and influences. Secondly, is the system environment static or dynamic? A static environment is one that is largely stable and unvarying. Whereas, a dynamic environment will demand much larger influences upon the system to adapt and change. Together, these two system aspects introduce a 2 x 2 matrix in which to characterise a system's environment: Static/Simple — where the perceived uncertainty surrounding the system is very low; Dynamic/Simple — where the perceived uncertainty surrounding the system is moderately high; Static/Complex — where the perceived uncertainty is moderately low; and Dynamic/Complex — where the perceived uncertainty is very high.

2. **System Boundary.** Defining the system boundary is critical to determining, communicating and understanding the system–of–interest. Two different people may often view the same entity at different levels of abstraction — through viewing one person's entire system as only a part of their system viewpoint. Sometimes two system viewpoints may overlap — and one person will only perceive part of the other's system view, while both are oblivious to certain aspects of the other person's system viewpoint.

3. **System Inputs and Outputs.** Unless the system is closed to its environment the system will receive inputs and deposit outputs back into its environment. The manner, effectiveness, and efficiency, by which the system converts these inputs into outputs will largely be determined by its purpose, and influence the system's ability to survive within that environment.

4. **System Interfaces.** Systems will often be composed of other parts or subsystems when viewed from a particular level of abstraction. Within the perceived system boundary, the manner in which these parts of subsystems communicate information is via a system interface. If these interfaces are altered or corrupted, then the system may be quite different in nature or fail to fulfill its intended purpose.

5. **System Control.** Systems that posses survivability characteristics are those

systems that can protect their identity. To achieve this quality requires effective control subsystems as a part of the system's internal organisation. Chapter 2 discussed the three essential means of achieving control in a system using either passive buffering, feed-forward, or feedback control. In essence, this involves blocking, anticipating, or correcting samples of inputs from either the environment or at the internal system interfaces.

6. **System Emergence.** Emergence, while a definitive attribute of a system, is also a deep and somewhat controversial system aspect. Fundamentally, it is the view of what properties the entire system possesses,[1] but which no individual part or subsystem of the system possesses. It is this that gives a system view an holistic attribute. Ashby [54] reveals that a system may be considered to have emergent properties because the parts or subsystems are either not completely understood, or do not reveal sufficient knowledge about their behaviour under study. In this sense, the creator or observer of the system in question only has a homomorphic understanding of the system's possible state space when subsystems are coupled or observed at that level.[2][3] The essential point that Ashby wished to make was that while a system viewed at a certain level will contain behavioural properties not possessed by the parts or subsystems, these are only emergent (or unanticipated) system behaviours if the creator or observer of the system at that particular level cannot discriminate the subsystem parts and interface linkages that result in those behaviours. Otherwise, if an omniscient view of the state space and interface linkages is provided, then there will be no unantic-

[1] This of course depends often upon how the system boundaries are set.

[2] Ashby gives the examples of a) the gases Ammonia and Hydrogen Chloride — both are gases, but when mixed form a solid — a property not possessed by either; b) Carbon, Hydrogen, and Oxygen are all tasteless, but the compound of sugar possesses a taste — which none of the elements do; c) The twenty amino–acids in a bacterium have no "self—reproducing" property of there own. Yet the bacterium they make–up possesses this property.

[3] A point that highlights how a particular system or view of a system may seem to possess emergent properties when there is incomplete knowledge of the subsystems is also given by Ashby with the property of elastic, which for years confounded chemical scientists as to why elastic had a stretching quality, when the molecules that made it up possessed no extension properties at all. It was later realised that during the composition process the individual molecules jostled tightly for position resulting in each molecule taking a length that was less than its maximal length.

> ipated behavioural properties of the system as a whole.[4] Other references to emergence are often considered in terms of *"self–organisation"* [cf. [88]]. In self–organising systems, emergence is defined as a survivability property, not based upon some top–down command and control subsystem, but instead based upon a non–hierarchical bottom up emergent order that employs local rules and interactions to represent the important macro–level emergent properties critical for a system to maintain its identity and adapt to its environment.[5]

In the context of this thesis, the view of emergence relevant to a dependability perspective (and particularly the wider view of a computer–based system dependability) relates to the potential for unexpected behaviours of the system (that will be judged to be undesirable by another judging system) that, in essence, results from an inability to completely comprehend and/or construct the system to prevent such unanticipated and undesirable behaviour.

An additional aspect of a system view is that systems exhibit goal–directed behaviour [5].[6] This purposive view, as discussed above, often is influenced by the particular viewer or creator of the system and connected to the system boundaries that are set. Furthermore, while a system may have a primary macro–level purpose — as a super–ordinate type goal, it can also have many secondary micro–level purposes — as sub–ordinate goals. The latter is often referred to as a system having *latent* functionalities. Merton [89] makes a clear distinction between *manifest* and *latent* functionalities of systems to remove the confusions often introduced in the

[4]It is important to point out that emergence in this context is viewed as something inexplicable about the system when observing an existing (say natural) system, or is considered an undesirable system behaviour that is unintentional and defeats or undermines the purpose of the system when creating a (say artificial) system. However, in either case, it can still be ascribed to an incomplete understanding of the system's total state space.

[5]Johnson [88] provides examples of harvest ant colonies where the local chemical rules and physical interactions between individual ants accumulate, in a representational manner, to ensure that sufficient ants are involved in the colonies critical activities of nest building, food foraging, nest cleaning, nest defence, etc activities. This order is achieved without any top–down command and control hierarchy being present in the colony.

[6]This is a controversial aspect with natural systems that are often given material and efficient causal explanations [cf. chapter 7]. However, it is an obvious aspect when considering artificial systems.

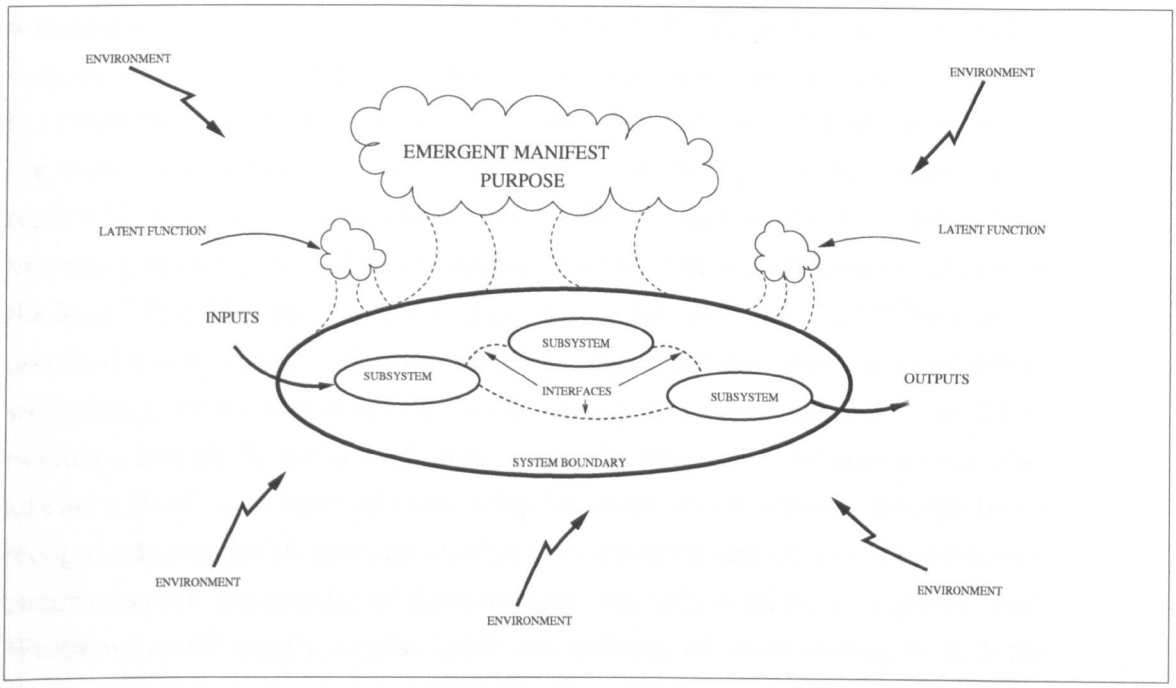


Figure 4.1: System View

sociological literature to distinguish between conscious intentional actions and motivations and unconscious, unintentional actions and motivations that are often found in explanations of entities, processes, or system views. Checkland and Wilson [90] provide two good examples and distinctions between manifest and latent functional explanations, when considering human–activity systems, from a sociological perspective, as follows [p. 53]:-

> *"...various tribal rain–making ceremonies do not in fact serve their manifest — in this case, meteorological — function; but they do serve the latent function of reinforcing the group identity. Nearer home, the purchase of an expensive car may serve not only the manifest purpose of providing a means of transport but also the latent function of declaring, claiming or reaffirming social status."*

It is important to note here, that in the first case the primary or manifest purpose was actually completely ineffective, while in the second case both manifest and latent functions of the system (i.e. car) are present. Moreover, purpose of even

a simple activity can often be very subjective, relevant, and representational to only an individual's self interest. A good example of this is provided by Drucker [91] with the story of the three stone–cutters. Drucker takes the scenario of asking three stone–cutters individually what they are doing. The first stone–cutter replies *"I'm making a living cutting stone."* The second stone-cutter replies *"I'm practising the skilled art of stone–cutting, of which I'm one of the most talented in the land."* The third stone–cutter replies *"I'm building a Cathedral."* The point to note in this story is that each stone–cutter ascribed a different purpose to what they were doing. To the first stone–cutter the activity was merely an end in itself (i.e. making a living). To the second stone–cutter the purpose of the activity was also an end in itself — in terms of reinforcing his sense of self–identity through being recognised amongst his peers as a highly talented stone–cutter. To the third stone–cutter however, the activity of stone–cutting was only a means to a greater end. Whilst we could simply ascribe latent functionality of stone–cutting to both the first and second stone–cutter, and manifest functionality to the third stone–cutter, this story also highlights the potential for conflict to emerge between manifest and latent functionalities of entities, activities, or systems. For instance, optimising stone-cutting by, for instance, introducing some machine to semi–automate the stone–cutting process to improve productivity and/or quality would quickly bring conflict with the first and second stone–cutters — as the first would potentially perceive it as a threat to his livelihood, while the second may perceive it as a personal attack upon his status and sense of social identity. However, the third stone-cutter would most likely be content and agreeable with the new situation — as such activity optimising is in perfect alignment with the purpose ascription he attaches to stone–cutting.

Figure 4.1 provides a visual depiction of the system view issues provided in this section.

4.3 Computer–Based System View

Before providing a specific computer–based system view, it is important to stress that perceiving some aspect of the world as a system is very different from claim-

ing it is a system. Checkland and Scholes [92] emphasise this, stating:-

> *"....it is perfectly legitimate for an investigator to say "I will treat education provision as if it were a system," but that is very different from saying that it is a system...Claiming to think about the world as if it were a system can be helpful. But this is a very different stance from arguing that the world is a system, a position that pretends to knowledge that no human being can have."*

This does not imply that a systems view always has no substance — as a car or a computer are physical entities that can be touched and objectively observed. Only that it is often a matter of subjective interest what different individuals choose to select and emphasise as a system view.

Having stated this explicitly, the following subsections provide the system aspects of interest in determining a computer–based system view.

4.3.1 A Holistic Perspective

As discussed in section 4.2, the single most influencing aspect of determining a system viewpoint is in the definition of the system boundaries. Instead of considering a computer system as primarily a technical construction of an artifact and associated activities — as chapters 2 and 3 have focused upon, the system view proposed here is to expand the system boundary outward to include not only the technical system, but also the human system with which it must interface with. By the term *'human subsystem'* it is meant to not only encompass the role of people involved in the direct creation process, but also a wider inclusion of the important influences of people involved in its strategic and operational exploitation. In this system viewpoint both the technical computer system and the human system are considered subsystems of interest at a higher level of system conception. In principle, this is what is to be interpreted by the term *"Computer–Based System"*.[7]

[7]This system view is consistent with that defined by the DIRC research programme cf. www.dirc.org.uk

As hardware, network, and software technology advances at an increasingly fast rate, it is becoming ever more necessary to gain a more complete and inclusive systemic view of information technology. It has been traditional for professionals to specialise in certain areas (i.e. hardware, networks, software) and build computer systems in relative isolation. As computer systems undertake more and more control of complex and novel information processing situations that can affect everyday lives, the weaknesses of such specialist and isolated approaches are becoming ever more apparent. This view has long been reinforced and advised by Neumann [93] noting that long–standing failures of the Hubble telescope, the ARPANET collapse, and the AT&T long–distance slow down where all directly attributable to *"...a lack of suitable systems perspective on the part of the developers, administrators, users, etc."* Furthermore, Neumann advises that process technology alone is insufficient to provide such a holistic perspective where different dependability attribute requirements (i.e. reliability, integrity, security, etc) need an integrative representation — in order to appraise their particular importance in a given application domain and identify conflicts that can result later in the form of obscure system flaws.

4.3.2 The Generic CBS Contexts

In this section I introduce the notion of four generic computer–based system contexts–of–interest that have been interpreted from a variety of information sources that are cited within this section. However, such an inter–related computer–based notion is different in the way that this relates the usual major influences and relationships to be found. As this section and future chapters (e.g. chapter 5 and chapter 8) will show this can be useful in representing and discussing a wider, holistic, and integrative view when considering computer–based systems.

This section provides an initial system viewpoint of the important subsystem contexts that can often affect the success and failure of a computer system. In chapter 3 it was noted that a number of ways in which the creation process can be judged to have failed would be expanded upon in this chapter and this would require a wider computer–based system perspective. It was briefly highlighted in chapter 2 that

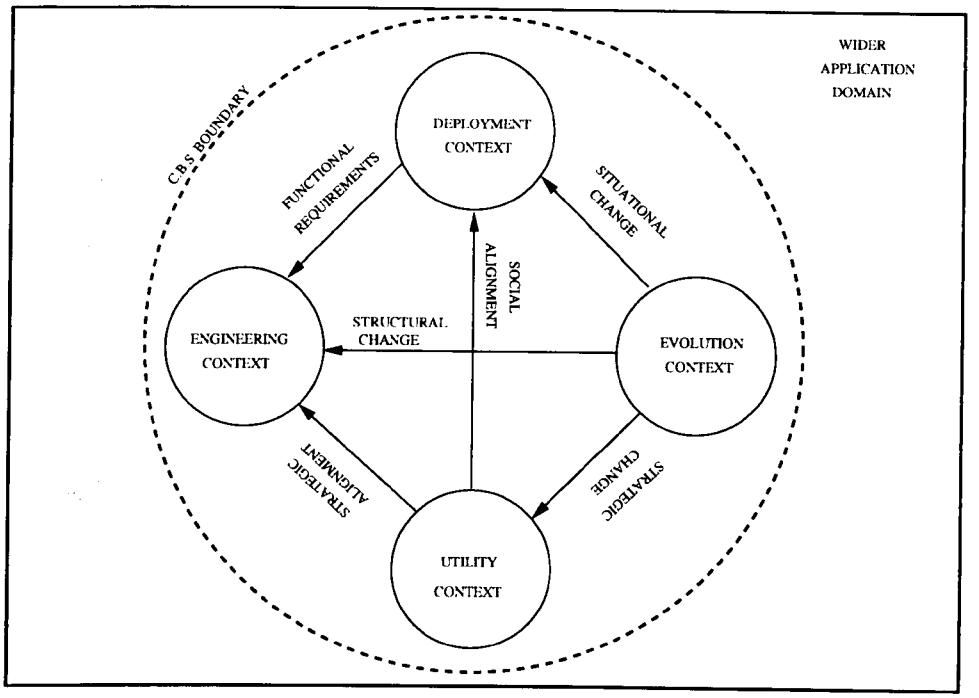


Figure 4.2: Generic CBS Contexts

a number of other judgements of failure — such as a) expectation failure; and b) conceptual failure require a wider computer–based system explanation. However, it should be pointed out, that such a wider computer–based system view should not be considered a panacea for discussing, thinking about, and developing all computer systems, as some computer system developments may not require such a system view. This is particularly true of *'systems'* software and *'middle-ware'* — where the domain is much more limited to, essentially, technical considerations.[8] Therefore, such a computer–based system view is more associated with *'applications'* software. With applications software, however, a number of non–technical considerations — can often be responsible for judgements of failure. The generic computer–based contexts, discussed below and illustrated in figure 4.2, can often be viewed as *'sources'* of interest from which such judgements may be made. The actual technical computer system development is an assumed technical system that influences (and is influenced by) all four of the generic computer–based

[8]Good examples of *'system'* software are operating systems, and peripherals. Examples of *'middleware'* software are web server and network communication software.

system contexts discussed.

4.3.2.1 The Utility Context

This computer–based system context of interest embodies the high–level, and overriding, justification for the computer system's existence to begin with. From the late 1970's to present day, the advances of information processing technology, along with the ongoing reductions in the cost of such technology — has resulted in information technology occupying an increasingly strategic role within many organisations [94]. Information technology (IT) has been continually recognised as a strategic organisational resource which can be exploited to provide a given organisation with competitive advantage. A number of strategic analysis frameworks have been developed to aid in the the strategic planning of organisational information systems, the most widely known and enduring of which has been [cf. [95, 96]]: a) the *"value–chain"* model for assessing an organisations' primary and secondary functions — and how IT can be used to improve the organisations' effectiveness and efficiency; b) associated and adapted with other longstanding strategic analysis frameworks — such as the *"five–forces model"* and the *"generic positioning model"* to determine how the information content of a service or product can add value and act as competition barriers.

What all the above strategic issues of IT imply, is that there will always be some high–level strategic value or utility rationale for justifying a computer–system's existence to begin with. The nature of the organisation itself will often have a fundamental influence in determining what this strategic justification is. For instance, in a commercial organisation this will often be closely connected with cost savings, quality of service improvement, greater market penetration projections, etc, whilst in a non–commercial organisation, such strategic IT value or utility considerations may well have wider macro sociological, economic, or political objectives to satisfy. The term of importance here is *"satisfaction"*, as strategic planning and analysis considerations are, by definition, of a predictive and forecastive decision–making nature [cf.][59]], emphasising utility or value judgements that reflect forecasts and predictions of strategic importance benefits envis-

aged with the system. In this regard, it is possible to make clearer what was meant in chapter 3 — regarding considerations of expectation failure judgements. For example, in order for the envisaged IT system to satisfy the strategic utility and value judgements made, the expectations (i.e. actual predictions) of strategic utility and value must be matched by subsequently deployed strategic performance of the IT system. It can be appreciated that, expectation failure occurs, when either the original expectations are unrealistic in some way, or the expectations were realistic but subsequent performance of the deployed IT system during operational deployment was less than the potential that the system could have achieved. Both of these will result in dissatisfaction with the strategic utility or value that was expected initially. Amongst the major strategic problems that have compromised satisfaction with IT investments, by organisations, has been a lack of alignment between strategic plans (i.e. business etc — depending upon the organisation type), the information systems necessary to support such plans, and the suitability of the actual IT systems deployed and developed to produce such information [cf. [96]]. Analysis of such expectation failures have often been *'rooted'* in non–technical problems — such as inadequate strategic linkage and representation of technical know–how at the board level, etc [cf. [94]].

4.3.2.2 The Deployment Context

If the utility context represents strategic value considerations and judgements, then the deployment context reflects the *'lived–in'* day–to–day judgements and experiences of how well the deployed computer system supports the many individual's own occupational, career, and work requirements. Failure of the IT system to be effective in aiding people to perform their work commitments and responsibilities can quickly result in a perfectly capable (i.e. technically) IT system not fulfilling its envisaged strategic benefits. Again, this highlights that, within a wider, and holistic, computer–based system viewpoint, judgements of failure may result from non–technical problems. A very good example of such non–technical deployment failure, is given by Lynne–Markus and Keil [97]. The computer system's strategic aims (i.e. utility) was to reduce the cost inefficiencies and improve customer service through reducing computer parts configuration errors when creating and

fulfilling customer orders. But despite the computer–systems technical capability to produce error–free configurations more dependably than the average sales personnel, two usage surveys carried out in 1986 and 1989 revealed that only, at best, 25%, and, at worst, 10% of the sales personnel reported using the computer–system. Despite redesign attempts that significantly improved the usability of the computer system, usage did not improve. It was finally revealed, that, although the computer system was technically well designed, the computer systems' underlying organisational design conception failed because of the non–technical issues of being fundamentally mismatched with the responsibilities, roles, and motivations of its intended users — the sales–force personnel. As Lynne–Markus and Keil stress, the system failed because of [p.14]:-

> *"**No Motivation:** Sales reps were simply not motivated to produce error–free configurations...CompuSys's organisational structure and reward systems...did not give any incentive to the sales department to prevent configuration errors but rewarded reps on the basis of their sales volume....Furthermore, the sales reps believed that verifying the accuracy of the configurations they specified on the sales orders was not their responsibility. Although the redeployment effort removed many barriers to the use of CONFIG, it did nothing to attack the users' basic lack of motivation or incentives, without which they were highly unlikely to use CONFIG...**Disincentives:** The sales reps actually had disincentives to using CONFIG; it made it harder for them to do what they had the motivation to do — make sales. Its developers had the goal of optimizing the configuration process. But, from the sales reps' perspective, creating the configuration is actually a subprocess: it's only a means to the end of the true goal — getting the customer a price quote in the course of making a sale."*

This example of a conflict of notions of the *'true'* purpose of the computer system resonates with the earlier example of Drucker in section 4.2 and the scenario of the three stone–cutters. The manifest strategic goal of the computer system (i.e. cost efficiencies and customer service) conceived within the utility context–of–interest, fundamentally conflicted with *'other'* (say) latent deployment sales goals of the

sales department personnel — in the way they perceived their responsibilities and roles in connection with the computer system. Such a system purpose conflict is not an isolated occurrence either — as Kirby [98] reveals in his analysis of a failed Information System (IS) project called IRIS. IRIS's strategic utility or value rationale was as an Integrated Requisitioning Information System to re–engineer the requisitioning, purchasing, receiving and disbursement of employee reporting processes. It was expected that streamlining business functions and eliminating reporting inefficiencies would result in reduced costs and increased profits — this justified the system's existence. Furthermore, IRIS also had competitive advantage expectations, as it was believed that IRIS would bring suppliers on–line, and thereby increase their dependence upon, and commitment to, the organisation.

However, it quickly became apparent to the managers, employees, and intended users of the computer system that it would radically alter their roles, responsibilities, and work content of their jobs. Furthermore, many began to see that, if not in the immediate short–term, then in the longer–term, IRIS could threaten their livelihoods within the organisation. Eventually, they viewed IRIS as headquarter's way of increasing their domination and control of their workforce and automating and reducing the richness of their present roles, responsibilities, and jobs. Kirby highlights that the two views of IRIS (i.e. utility context vs. deployment context) was underpinned by the attribution of different meaning interpretations to the same set of design rationales and computer system issues, stating that such differing perspectives about the same things can [pp. 210–11]:

> *"....be thought of as a set of underlying assumptions and beliefs about the way in which the world operates. Depending upon the assumptions adopted in analysing IRIS, different conclusions can be reached. Any one perspective supplies only a partial view of the world, and no perspective can hope to explain everything...In each case, the individuals involved saw the same events as management, but attributed different meanings to them....management saw things from one perspective, that of economic rationality and cost–benefit analysis, whereas others ascribed meanings such as self–preservation, domination, and power acquisition...management made the mistake of assuming that*

> *everyone saw things from the same rational, economic perspective. It never occurred to them that the same events could be interpreted so differently...The best way to recognize that people are ascribing different meanings to events is to be aware of the possibility of it happening."*

From these examples we can see that, in terms of a wider computer–based system view, it is necessary to be aware that different meanings, views, assumptions, and beliefs from different contexts of interest can result in non–technical judgements of failure in spite of the computer systems *'actual'* technical capability — due to different views of the computer–systems perceived purpose. Furthermore, as briefly discussed in chapter 3, such conflicts can result in a deployment failure — a situation, as described above, where a technically capable computer system fails for non–technical organisational reasons like organisational mismatches, informal rejection, or covert rejection due to conflicts of meaning and purpose between the two contexts–of–interest.

Finally, just as the strategic utility must be in alignment with the eventual actual design rationale and infrastructure of the computer system to avoid expectation failure, the strategic utility must also be in alignment with deployment issues of latent purpose ascription — if its expected strategic performance is not to be undermined by non–technical deployment problems. This is illustrated as a context interfacing issue between the utility and deployment contexts in figure 4.2 on page 99.

4.3.2.3 The Engineering Context

The engineering context issues have already, largely, been covered in chapter 3. Essentially, as discussed there, the engineering issues relate to the technical construction of the computer system and its supporting infrastructure to both realize the strategic utility expectations and ensure that the eventual deployed computer system results in a system that the users are motivated in using. However, as issues of deployment discussed in subsection 4.3.2.2 indicate, although ergonomic and

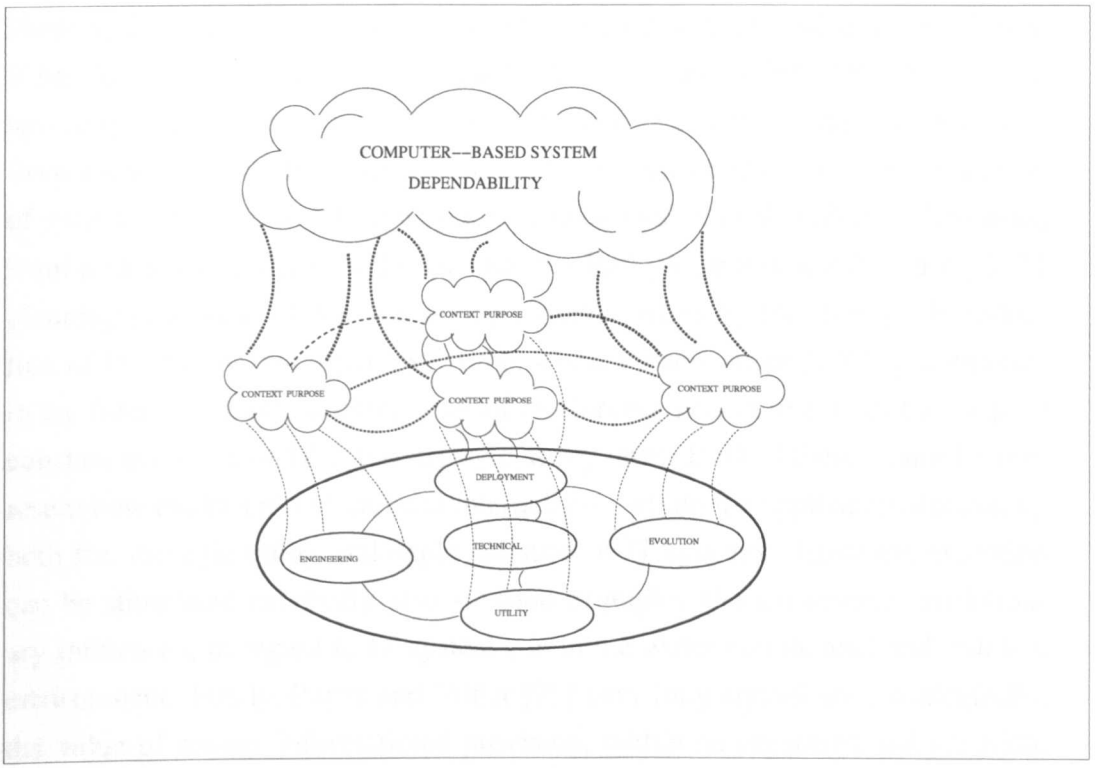


Figure 4.3: Emergent CBS Dependability

usability factors can greatly improve the likelihood of the deployed system being used [cf. [55]] through such approaches as User–Centred–Design (USD), this alone will not guarantee adoption if there exists fundamental conflicts of notions of purpose and meaning, etc due to a lack of representation of important sociological and organisational issues. This requires greater coverage of the strategic and deployment situation than is usually associated with elicitation of functional requirements found in many software engineering texts [cf. [27, 28]], and, as already discussed in subsection 4.3.1, requires a much greater holistic contextual integration — if such aspects of strategic alignment, sociological alignment, and functional requirements are to be more completely covered and considered.

4.3.2.4 The Evolution Context

All open systems must change and adapt to environmental pressures if they are to persist their identity over time. A computer system is no different in this respect.

Pressure for change can result from both internal and external sources. Firstly, it has long been recognised as a law by Lehman and Belady [99] that IT system usage stimulates users to request and require more functionality. Therefore, from a deployment perspective, IT system introduction invokes a reinforcing set of system dynamics which continually promotes situational evolution. Secondly, from a strategic utility or value perspective Earl [96] notes that the strategic IT planning process is also a reinforcing dynamic whereby the strategic introduction of IT systems can often, over time, stimulate new strategic IT opportunities in the future — again invoking a set of reinforcing system dynamics that require constant evolution of IT at the strategic utility level. Both of these examples represent how evolution is stimulated internally — within the application domain by both the strategic utility and deployed state of IT systems. However, evolution can be stimulated externally also — good examples of such external evolutionary influences, in regard to IT systems, is in the wider commercial and political environment. Firstly, Porter and Miller [95] have long argued how, strategically, the value of greater informational provision, within an organisational situation, can result in creating whole new ways by which competition is defined — in extreme cases whole new industry sectors can be created where the strategic use of IT systems has become a prerequisite for competition. Secondly, many public organisations are often influenced by government political strategies that mandate fundamental utility or deployment changes to new and pre–existing IT systems — this is particularly true of the National Health Service (NHS) in the UK where the government actually provides periodic IT strategies at both the national and local level.

Lastly, the engineering context is tasked with the problems associated with evolving IT systems. Research has highlighted the extent of such problems of changing, correcting, and adapting existing IT systems, it reveals that as much as 80% of the total lifetime costs associated with IT expenditure can be used up in the evolution. maintenance, and adaptation of IT systems [cf. [100, 101, 12]]. Failure to do so, however, can often result in IT systems losing their usefulness and becoming legacy systems, these are systems that the organisation still relies upon for their strategic and operational viability, but which can no longer be evolved [cf.

[101, 28]].

4.4 Chapter Summary

In this chapter the view of dependability has been extended to consider what dependability means in the wider view of a computer–based system — where the boundaries of the system–of–interest are widened to include the interfacing roles, responsibilities, and motivations of the human subsystem — in its many forms (i.e. strategic, engineering, deployment, and evolution). It can be seen from figure 4.3 that dependability is an emergent super–ordinate goal which is dependent upon an emphasis of a bottom–up and integrated view of the contexts–of–interest that ascribe often different and subjective notions of what constitutes the meaning and purpose of the computer system. Identifying and reconciling these subjective context–of–interest views is crucial, as, within a wider holistic computer–based system conception it can be seen that judgements of failure, satisfaction, and success, with the computer system, can be determined by non–technical aspects of the system. If these differing views are not identified then different interpretations, meanings, assumptions, and purpose ascriptions can be fundamentally in conflict with each other. When this occurs, as has been discussed, the overall perceived dependability of the computer–based system can often be compromised.

In the next chapter a more specific example of applying a wider computer–based system view of dependability is provided with an analysis of the Automatic Teller Machine domain.

Chapter 5

ATM Case–Study

5.1 Chapter Introduction

In this chapter the domain of the Automatic Teller Machine (ATM) is used to emphasise a computer–based system view of dependability issues. The Automatic Teller Machine is chosen as it is a well exposed domain — spanning almost 25 years of wide-scale deployment. This chapter raises and discusses a number of failures and vulnerabilities that have been reported over this period. The chapter ultises these issues to show how ATMs can be more broadly conceptualised as a computer–based system and by taking this view demonstrates the need for a broadening of the system–of–interest to help understand the fault phenomenology of harmful assumptions that can result in computer–based system failures.

5.2 ATM Contexts

In the following subsections a variety of failures of ATM's will be discussed. These are based upon a wide number of literature sources — some going back to the early/mid 1980s. The purpose is not to provide a complete coverage, but to illuminate some varied ways in which ATM's can fail or be exploited. To be consistent with chapter 4 the subsections will be divided into the computer–based system contexts–of interest of:-

- The **Utility** Context — in terms of the strategic IT strategy issues that have influenced such failures and vulnerabilities;
- The **Engineering** Context — in terms of the IT development issues that have resulted in failures and vulnerabilities;
- The **Deployment** Context — in terms of operational IT issues that have resulted in failures and vulnerabilities;
- The **Evolution** Context — in terms of the changing and environmental issues that have resulted in failures and vulnerabilities.

It should be noted, however, that each failure or vulnerability, while originating from a particular context, often manifests in another context. As a consequence,

each context often refers to related ATM contexts in which the consequences of failures and vulnerabilities manifest.

5.2.1 The Utility Context

This subsection relates to the strategic IT policy issues and oversights that can result in failures and vulnerabilities becoming manifest in other contexts. While ATMs have offered banks and financial institutions advantages, these have often been short-lived as ATMs represent a purely technological differentiation approach to achieving competitive advantage that has been easily replicated by competing banks and financial institutions [55]. In real terms, the ongoing ubiquity of ATMs is due to their indispensable industry influence for banks and financial institutions to maintain competitive market standing with their competition [102].

ATMs were among the first commercial systems to utilize information transfer protection via data encryption [103]. Some believe that the 56 bit encryption algorithms, often used in ATMs were deliberately limited for national security reasons [104], or economic reasons — rather than purely technological reasons [cf. [105]]. Nevertheless, banking ATM security policies often employ hardware encryption modules of the Intel 8751H, Intel 8752H, or AMD 9761H standard for deriving and verifying PINs in *off-line* ATM designs[1] [106]. With later *on–line* ATM designs[2] a more sophisticated form of data encryption utilized a Unique Key Per Transaction (UKPT) format which signifactly reduced access by unscrupulous third-party's either internally (i.e. banking staff, maintainers, etc) or externally (i.e. line-tapping) as the terminal hardware UKPT hardware module translates the PIN information into an irreversible internal form so the customers PIN never appears again, either within the ATM or central accounting banking systems in an unencrypted format [107].

[1]Off-line ATMs formed most of the earlier ATMs used. These designs did not have *'live'* network links to the banking account systems or credit clearing houses. Consequently, Personal Identification Numbers (PINs) and Daily withdrawal limits had to be enforced through the ATM card with only periodic batch-style updating between the ATM and the banking account systems.

[2]On-line ATM designs have a *'live'* direct link with the banking account systems. Consequently, PIN, card, and Daily Withdrawal Limits can be directly verified in (practically) real–time with a bank customers account information.

Encryption Policy – Issue 1. *A particular dependability issue that can arise from such banking encryption policy decisions is where technically uniformed decisions subsequently result in vulnerabilities being introduced via the Engineering context. In this specific example the preference for utilising proprietary software encryption over professionally developed hardware encryption modules has been particularly criticised, for three reasons [cf. [103, 104]]: a) Unintentional software faults can seriously undermine the sophistication of the data encryption — resulting in them being more readily translated; b) The bank's programmers are presented with an opportunity to introduce malicious code for: i) 'back-door' access; ii) corrupt the algorithm and keys so it produces a vastly reduced set of encrypted codes; or iii) use their knowledge of the encryption algorithm and PIN Verification Values (PVV with off-line ATMs) maliciously to gain illegal unauthorised access to bank customers' accounts. In such a situation, because PINs offer only a small set of combinations (i.e. usually 4 digits — so* 10^4 *combinations), all that is required by the 'knowledgeable' programmer is to tabulate the data into 'look-up' tables. This method of attack is usually referred to as data–inversion; and c) Systemic flaws can be introduced into the encryption algorithm unwittingly — due to inexperience, that then allows the encryption code to be subsequently compromised. For instance, Clough [104] argued that proprietary encryption methods are unlikely to have been subjected to professional independent reviewing — which only a handful of people outside the National Security Agency (NSA) in the USA can really provide.*

The serious dependability implications, of such a decision, are exemplified by Anderson [108] where he reveals how proprietary software encryption was compromised by software programmers being able to extract encryption keys that resulted in a sustained fraud in a large London clearing house during 1985-86.

One of the most controversial issues in ATM banking policies is the view taken by the banks and financial institutions (i.e. credit companies) that card and PIN authorisation is impervious to making unauthorised withdrawals at ATMs without carelessness or collusion by the card and PIN holder [103, 104]. Carelessness means that the card and PIN holder has been neglectful in their responsibility to prevent a third–party gaining access to their cards and PINs. Examples include writing down their PINs on the cards itself [cf. [109]]; telling their PINs to some third–party (i.e. friends and family members), or allowing some third–party to use their cards or PINs to make ATM withdrawals on their behalf. Collusion refers to ATM card–holders deliberately attempting to defraud the banks or gain money by deception through nefarious collaborations or coalitions with a third–party (i.e. friends or family members) that allow them to make withdrawals upon their account(s) which they then claim to the banks have been made unlawfully by persons unknown.

While there are examples of carelessness and collusion by bank customers [110], there also exists a plethora of ways in which the ATM authorisation policies of separate card and PIN identification can be (and have been) undermined to allow fraudulent access and unauthorised withdrawal(s) of money from a bank customer's account(s). Such withdrawals have become known as *'Phantom Withdrawals'* [cf. [103, 104]]. Phantom withdrawals have become one of the most controversial issues surrounding ATMs. In order to maintain customer confidence, public spokesmen/women (such as the Association of Payment Clearing Systems) have continued to publicly deny the possibility of such fraud taking place [111]. As Clough [[104]: p 59] argues, banks have frequently and officially stated:-

> *"Banks do not make mistakes, and there are no such things as phantom withdrawals."*

Clough goes on to explain that the banks and financial institutions fear that if they were to admit the possibility of *phantom withdrawals* then this would open the floodgates for financial compensation to every customer who genuinely forgot making a particular ATM transaction. Secretly, however, banks and financial institutions refer to unauthorised phantom withdrawals as *"white card fraud"* [113].

Year	Cost in millions £
1997	£8.2
1998	£9.7
1999	£12.2
2000	£18.3
2001	£21.1
2002	£29.1
2003	£39.0

Table 5.1: ATM fraud in the UK [source: [112]]

The vast majority of reported ATM fraud to the UK Banking Ombudsman relates to unauthorised or *phantom withdrawal* complaints [104].

Table 5.1 shows the recent annual cost of ATM fraud in the UK over the last seven years to 2003.[3] It is clear from these figures that ATM fraud, whilst only a small percentage of wider credit card fraud [114], is clearly on the increase in the UK.

> **Authorisation Policy – Issue 2.** *The maintenance of this strategic policy view by banks and financial institutions, while to an extent understandable, also fails to recognise the subsequent deployment dependability context considerations — in terms of potential safety consequences*[4] *it presents for individual customers. While ATMs don't represent critical safety concerns like other IT/Software systems (i.e. embedded control of aircraft or nuclear power stations, etc), their usage can result in undesirable consequences — with regard to wrongful (or potentially unreliable) criminal convictions that have occurred from bank customers claiming to have been victims of phantom withdrawals. Notable cases include [cf. [108, 103]]: a) In 1985 a teenage girl was charged and convicted of obtaining money by deception by using her fathers ATM card to make an unauthorised withdrawal. She*

[3] At the time of writing, recent statistics reported for 2004 in the press estimate that this figure has further doubled in two years to approx. £85 million.

[4] This is not the usual interpretation of safety — in dependability terms of catastophic damage or loss, etc. Rather, in the context of computer–based systems in this chapter, it should be interpreted in a more general manner to mean some judgement of an undesirable consequence has occurred.

> *was advised to plead guilty by her legal representation to ensure a lighter sentence. It later transpired that no theft had ever taken place and the £40 deficit resulted from a clerical banking error which the bank then tried to cover-up; b) In two separate incidents, an elderly woman from Plymouth and a taxi driver from Great Yarmouth were both charged with attempting to obtain money by deception through ATMs . Both charges were later dropped when it was revealed that in both cases the banks security systems were unacceptably vulnerable; and c) In 1994 a police constable was charged and convicted of attempting to obtain money by deception when he complained of six phantom withdrawals being made to his building society account. The conviction went ahead despite serious concerns over the building society's quality assurance and security practice. Additionally, a number of other technical anomalies existed, including: i) the card issued by the building society was issue number 2 — without any explanation of why a second card issue had to be raised; and ii) one of the phantom withdrawals had been made from an ATM in Omagh Ireland — which the police constable had never visited. An appeal against his conviction was later successful.*

While these instances are rare they nevertheless represent a severe consequence for the individuals concerned — in terms of everyday risks of using ATMs in their deployed state. Anderson [108] captures this sentiment in his comment:-

> *...the idea that complaining about a computer error could land me in prison is beyond my tolerance limit.*

Less consequential safety risks of ATMs, related to this strategic policy issue include: a) financial loss; b) extra bank charges due to unknowingly going overdrawn through a phantom withdrawal; c) loss of bank interest that is due; and c) unavailability of cash dispensing services. It is for such consequences that in the early 1990s approximately 400 aggrieved bank customers filed law suits through J. Keith Park Solicitors against UK banks and building societies claiming that

their ATMs are in breach of contract due to their susceptibility to fraud [115]. Individual claims ranged from £90 to £13,000, and when aggregated, totaling nearly £500,000.

5.2.2 The Engineering Context

The Engineering context relates to the process by which ATMs, as artificial systems, are created. This context is responsible for providing a system to the customers' required technical specifications within cost and timescale estimates, which may be reinforced by legal contractual agreements. However, it should be noted that if the Engineering context involves an external organisation, the created artifact may also constitute a product of that organisation itself (i.e. the reader should read the scenario quote by Malan and Bedemeyer [116] from subsection on page 184 in chapter 8 for an example of this kind). This may fundamentally influence the development criteria for the artifact, as the organisation attempts to maximise its product base through modular or object–based product lines. Such issues, specifically relating to ATMs are presented in [117, 29] where the functional software design of such devices as card-readers, keypads, and printing devices are generalized so that they can be quickly used for other related products like security doors, EFTPOS equipment, etc.

Nevertheless, whether acting as an internal function of an organisation, or as an external organisation itself, the important issue that the engineering context plays is the management of the many resources, as inputs (i.e. human resources and process technology), towards the creation of an ATM. Importantly, this involves a great deal of specific and definite technical decision–making in the artifacts' creation that places an emphasis upon communication, elicitation and interpretation of large amounts of information. This is a prerequisite to providing a dependable product, which is often made much harder when providing a product for an external customer — due to the complexities of access to, and understanding of, information about another specific application domain. More precisely, the engineering context must integrate and balance issues presented from the other contexts such as strategic utility concerns (i.e. ATM banking policies already

discussed), deployment concerns (i.e. bank staff and bank customer operational issues) and evolution concerns (i.e. for example evolution issues presented by inter-banking and inter-credit merges over time as large credit card institutions like Visa emerged).

A more specific complexity of the Engineering context, is that it must consider a further complicating facet, (more specific to development of an ATM) that the software provided constitutes an embedded control component of the actual physical ATM system (like firmware mechanisms) as well as a client information processing component of the central banking accounting system and/or clearing house credit system [cf. [118, 110]]. Roberson [119] argues that a lack of sufficient knowledge representation during the engineering is the cause of many faults, omissions, and inconsistencies between software, hardware, and firmware involved in ATM designs. In particular he stressed that this significantly reduces the potential for accurate fault hypothesis, fault detection, and subsequently error handling as each area of concern is separated with often little communication and many questionable assumptions being made about another area of concern.[5] For example, the early designs of ATMs omitted to consider the potential for an ATM user to make a cash withdrawal and forget to take their cash from the cash dispensing slot [104]. As a consequence, the amount was debited to the ATM user's account with the cash often then stolen by passers-by or subsequent ATM users. This left the *absent-minded* ATM user with a financial loss for their error.

> **Human Error — Issue 3.** *Whilst the responsibility for fault of the loss from such errors and the ability to have anticipated such problems earlier in the development and deployment of ATMs could be debated, over time, it was recognised that such human-errors of this kind with ATMs are part of a broader set of known human operational pathologies that can occur [120]. The increased frequency of such human error with ATMs resulted in ATM cash dispensing slots being fitted with sensory devices, cash–retracting firmware mechanisms, and time-out software control that would retain cash forgotten by the*

[5] Such comments and views reinforce the need for a more inclusive, integrated and holistic viewpoint, as mentioned in chapter 4.

ATM user after a certain time period. It is clear from this historical fact of early ATM development that, in error–forecasting terms, the original development undertaken in the engineering of many ATMs failed to consider the human reliability deployment issues surrounding the day-to-day usage and operation of ATMs by bank customers.

While later ATM development considered human reliability issues (such as issue 3 above) in ATM deployment, subsequent ATM designs often failed to identify (what may be called here) *Opportunistic Theft* by legitimate ATM users attempting to steal money during a *bona fide* transaction interaction with an ATM.

Opportunistic Theft — Issue 4. *A good example of this is reported by Kristiansen [121]. Some customers worked out that when forgotten cash was retracted back into the ATM the whole transaction record was undone to leave the customers account unchanged. Therefore, if, instead of removing all of the cash dispensed at the dispensing slot, they removed only some of the money, by* teasing *some notes out, then, after a certain time period the rest of the money would be retracted back into the ATM and the shortfall would become untraceable to a particular ATM customer. While only a small amount of cash appears to have been lost by the banks through this security vulnerability, the potential for this security hole to be spread by 'word-of-mouth', by unscrupulous bank customers, has the potential to increase losses if it is not identified early. Also it does, in principal, reveal another example where, in dependability terms, the engineering context failed to anticipate the deployment security vulnerabilities involved in the dispensing of cash, and its subsequent retraction functionality — following some human error.*

Some security flaws in ATMs can be particularly obscure. One in particular is due to the (quite justifiable) human interaction principle of allowing a computer system user to cancel a current computer interaction at any time during the transaction stage. With regard to ATMs this allowance has proved important in pre-

venting customers entering erroneous data, losing their cards, or leaving an ATM exposed to subsequent theft or fraud, as the following two cases exemplify:

1. Colville [122] reports on a situation where because the hardware keypad of the ATM was not registering the input data, or was *'bouncing'* the input data twice, it would be a desirable feature to cancel the whole transaction with that ATM and use another rather than risk losing your card during PIN entry, withdraw too much money, or transfer too little, etc;

2. Chiasson [123] reports on a personal experience where a state inconsistency fault between the software and hardware device (i.e. card-reader) would have allowed a subsequent ATM user or passer-by access to his account if the ATM had not allowed him to cancel the entire transaction and restore a consistent state between the software and hardware.

While both of these examples clearly indicate the dependability advantages of providing transaction undo features with ATM designs, Naggs [113] reveals a subtle flaw in allowing an ATM user to completely undo the entire transaction **before** the ATM user has satisfactorily completed the authentication process by both inserting the card and entering the correct PIN. Naggs inserted his card and entered two (known) incorrect PIN numbers before cancelling and repeating the process for a second time. In his own words he noted:

> *"If you reinsert the card into the ATM it does not remember your previously failed attempts."*

Such a security flaw clearly compromises the PIN protection offered with ATMs. It is normal for ATMs to only allow the user three attempts at correctly entering the PIN. With a four digit PIN (which is most usual) this would give any fraudulent person only a 1 in 3333.33 (recurring) chance — even if they had access to a customer's ATM card or had the ability to clone that ATM card (10^4 = 10,000 possible combinations divided by 3 PIN attempts). However, this flaw allows many more than just three attempts and when it is combined with other fraudulent techniques such as covert observation or surreptitious *'shoulder–surfing'* to (at least)

gain partial knowledge of an ATM user's PIN, then it clearly makes unauthorised or phantom-withdrawals much easier to achieve.[6] For instance, assuming that a fraudulent third-party had the ability to get access to a bank customer's ATM card, or had the ability to create a cloned version of that ATM card, then with partial knowledge of that same customer's PIN (via covert surveillance or *"shoulder–surfing"*) he/she could:

- Gain access with a maximum of 4 reinsertions of the card and PIN when only one of the four digits were unknown e.g. users PIN is 4965 so if 1 unknown, ?965, or 4?65, or 49?5, or 496?;
- Gain access with a maximum of 49 reinsertions of the card and PIN when only two of the four digits were unknown e.g. users PIN is 4965 so if two unknown — such as ??65 would range from 0065....9965. With this particular pin (i.e. 4965) it would only require 22.5 reinsertions (or 45 continuous PIN enumerations).

The combination of this security flaw initiated from within the engineering context, along with fraudulent techniques to gain (at least) partial PIN knowledge from within the deployment context, clearly undermines strategic issues such the authorisation policies expected to be in force. When the two techniques are combined, it makes ATM theft and fraud much easier.

> **Obscure Security Flaw Conflicts — Issue 5.** *The specific problem that makes this issue harder to detect is that there is a conflict between what constitutes, generally in computer–system development, as good usability and interaction and the specific application requirements of what constitutes a dependable ATM deployment. In this case it would require the engineering context to integrate and emphasise the bank's authorisation policy in force together with anticipating the shortcomings above in order to identify that every ATM user must authorise*

[6] The latest covert observation *'scams'* is to insert a minature pin-hole type camera to the top underside of the ATM, allowing fraudsters to then remotely (and covertly) observe a particular ATM customer's PIN entry.

> *themselves with a correct card and PIN entry* ***before*** *a complete transaction undo feature can be initiated. Otherwise any incorrect PIN entries should be retained in the ATM/banking system as a permanent (i.e. memorised) state across different ATM authorisation attempts if the integrity of the PIN protection is to be maintained.*

As discussed earlier, a specific issue of ATM designs includes the problem of how embedded control software used to co–ordinate the inputting, processing, and outputting of data (e.g PIN data, withdrawal amounts, etc) and physical artifacts (e.g. ATM cards, money, print receipts, etc) interfaces with the hardware and physical firmware mechanisms that make–up an ATM. The danger in this case, perhaps emerges from the flexible or malleable nature of software, in contrast with fixed and definite physical constraints imposed by hardware and firmware mechanisms with which it is composed. As the following two examples indicate, when these components are considered in isolation, the overall reliability of the ATM system can be seriously undermined.

In the first example:-

> **Interaction Consistency and Completeness — Issue 6.** *Robertson [119] reports on the reliability problems that can result from the flexibility that is offered by the embedded software control becoming* out–of–step *with the more definite physical constraints of hardware devices and firmware mechanisms. In this case Robertson wanted to use an ATM to make a large withdrawal of \$700.*[7] *After entering this amount to be withdrawn he noticed that the ATM seemed to delay in outputting his cash at the dispensing slot and eventually made some strange noises and appeared to be struggling to do so. Eventually it only outputted \$220 in eleven \$ bills. Subsequent investigations by Robertson with his bank revealed that he had been debited for the full amount of \$700 and that the problem had occurred because the*

[7]It should be noted here that it is implicit from the Robertson's report that this large amount of \$700 appears to be within his ATM Daily Withdrawal Limit (DWL) allowed by his banking institution.

ATM's cash magazine had run out of the larger money denomination of $50 bills. The ATM had attempted to fulfill the withdrawal request by outputting thirty five $20 bills — which was more than could be passed, at one time, through the cash dispensing slot.

The second example is similar to Issue 6 above, but relates to a common and more general complaint about ATMs:-

State Representation Completeness — Issue 7. *This is when an ATM user's account is debited for an entered withdrawal amount, but the cash dispenser dispenses (at best) less money than requested and debited, or, (at worst) no money at all. Amongst the many possible causes of such ATM failures is the possibility that the embedded software control does not represent the physical state of the internal cash magazines that contain the physical money to be outputted (cf. [124] for an example of when this issue was explicitly included into a formal ATM specification in VDM). For a variety of usage, operational and/or maintenance reasons the replenishing of these internal money magazines may be delayed or prevented.*[8] *If the physical state of the cash magazines is not accurately represented by the embedded software control, and the ATM's money magazine becomes empty, or beneath a level that can fulfill a particular ATM user's withdrawal request, then the ATM user will be debited without having their withdrawal request satisfactorily fulfilled.*[9] *A good indication of how ATM manufacturers perform differently in identifying such embedded control software issues is reported by Minow [125]. Minow reveals that in comparing IBM and Diebold ATMs in the 1980s, many complaints were made against IBM designed ATMs dispensing incorrect amounts*

[8]For instance, there may be unusual peak–demand usage for ATM services — such as just before a holiday period. Alternatively, there may be a temporary shortage of money or certain denominations at the bank, or remotely located ATMs dependent upon outsourced organisations for re–stocking, may be delayed for a variety of logistical reasons (such bad weather, traffic–jams, etc)

[9]*The best or worst case may well be dependent upon which, and how many, money denomination magazine(s) actually go empty or low (i.e. £20, £10, or £5).*

> *to ATM users. However, Diebold were considered much more reliable in this regard as they were capable of determining how much money existed in the internal magazines. In situations where there existed less money in the magazines than would fulfill an ATM user's withdrawal request, the ATM would dispense what money was left in the cash magazine and only debit the customer to that (lesser) amount.*

While both of these issues above relate to how inconsistencies arise between interfaces and separation–of–concerns of embedded software control, that control/coordinate physical devices such has hardware components and firmware mechanisms from the engineering context, they also reveal that faults and vulnerabilities can result from other CBS context concerns. Issue 6 above reflects a need to recognise that while software can be designed with generality and flexibility or later enhanced to evolve the ATM system to meet future utility demands, this must not exceed the physical constraints into which it is embedded.[10] The problems reported in Issue 6 above therefore relate to a need to consider the evolutionary context (in terms of functional flexibility/generality and adaptation) in a holistic sense that includes recognition of the more restrictive (and less flexible) hardware and firmware devices and mechanisms of the ATM — into which the software becomes embedded.[11] Issue 7 is a more local context fault that occurs largely within the engineering context but becomes self-evident (sooner or later) once the ATM is deployed.[12]

5.2.3 The Deployment Context

The deployment context involves the many day–to–day operational issues and influences the ATM system (as a whole) is subjected to as its envisaged utility and

[10]For example, Daily Withdrawal Limits (DWL) vary from bank to bank, customer to customer, and account type to account type. Not only this, but as the spendable value of money progressively reduces over time, banks and financial institutions operating ATM networks will need to extend DWL.

[11]It is obvious from Issue 6 that enhancements of firmware mechanisms (internal cash transfer and counting mechanisms) and hardware devices (size of cash dispensing slot) would also have needed to have been enhanced if this problem were to be avoided.

[12]By *"self-evident"* it is meant that sooner or later, for a variety of reasons, the internal cash magazines will become insufficient to fulfill an ATM user's withdrawal request or become empty.

value is exploited over its lifetime. This includes such situational operational issues as organisational practices and procedures of the bank, within which the ATM is situated, and ATM bank customer issues (some of which have already been discussed in earlier subsections). A particular facet of ATM deployments is that they physically interface directly into the wider public environment. In fact, this is at the heart of the intended strategic utility of ATMs — that they can provide convenient 24–hour access to primary banking services of cash–withdrawals, account balances, and payment transfers, etc for banking customers. As a result, it has been an ongoing strategic policy of banks and financial institutions to offer more and more access to such services in more remote/diverse geographical locations [cf. [126, 102]]. However, this also makes ATMs, ATM users, and confidential bank customer information more vulnerable to attack, theft, and fraud.

From a computer–system perspective, ATM deployments are also *'walk–up–and–use' (WUAU)* systems — meaning that greater consideration must be given to the usability of such systems in terms of intuitiveness, learnability, and simplicity, as ATM users will be provided with little, if any, prior training in how to operate them. High levels of usability across a wide public user–base is easier to state than achieve as the survey of 1,530 ATM users by Rogers et al [127] revealed. They found that across many age groups improvements in training and design are required to make ATMs more user-friendly, useful, and attract more patronage. With regards to training they argue that:-

- Existing pamphlets on ATM usage are insufficient;
- Older users would be more willing to use ATMs if some prior training were provided;
- In particular, as more complex ATM features are added to the design of ATMs (e.g. account transfers, money depositing, bill payments, and ticketing acquisition, etc), training should be focused on these more sophisticated functions;
- The transfer of training knowledge over time and across different ATM designs was also highlighted as a particular area in which to focus future train-

ing programmes.

With ATM design improvements, Rogers et al distinguish the hardware and firmware mechanisation with the label *"surface–level"* improvements, while referring to embedded software as *"conceptual–level"* improvements. With surface–level improvements they advised that:-

- Better (anti–glare) screens, bigger text, and better physical alignment of keypad and screen buttons would improve usability;
- Hardware and firmware mechanisms were too slow in functioning, and future design should focus upon improving the turnaround times at ATMs;
- Greater thought should be given to the physical environmental context that ATMs are deployed into. A particular example stated was better location in respect to the sun to reduce glare and improve readability of on screen information.

With regards to conceptual–level improvements, Rogers et al advise:-

- Older users expressed concerns about remembering transactions made that raise challenges to future ATM designers;
- Future ATM design should attempt to reduce emphasis on technology and become more user–centred — in terms of the personal and interactive nature of ATMs.

While all of these recommendations are of value in improving future dependability of ATMs, the general public (which ATM users make up) present such a heterogeneous set that it is rarely possible to accommodate every different user's cognitive model, learnability style, or intuition — influenced by distinct knowledge, bias, personality, or past experience etc. This problem is indicated explicitly by Delvin [128] from a situational information theoretic standpoint. Delvin sought to highlight that it is very difficult (and impossible – in terms of all eventualities) to anticipate how each different individual will respond in a certain interaction (whether

human–to–human or human–to–machine). He retold, as an example, the experience his wife had one day while interacting with an ATM. His wife wished to deposit a cheque for $100 and make a withdrawal of $50 directly on that deposit transaction.[13] The interaction proceeded as follows:-

1. She inserted her card and PIN;
2. ATM asked her which service was required;
3. She selected "Make a Deposit";
4. ATM responded "Do you want cash back from your deposit?";
5. She answered "Yes";
6. ATM responded with the command "Enter amount";
7. She entered $50;
8. ATM responded with the command "Enter amount you want to withdraw".

At this point Delvin noted that his wife realised she had assumed that step 6 meant enter the amount she wished to withdraw. At this point she cancelled the whole transaction,[14] removed her card and tried again. The point Delvin made from this, from a situational information theoretic perspective, is that he had used the same ATM *"transaction–splitting"* features without any trouble for some time. Delvin [pp. 108-109] himself explains why his wife, and not he, had experienced such a problem:-

> *"As a mathematician and computer scientist, I automatically viewed the preplanned, artificially staged "conversation" with the machine as a machine transaction, and as a result I had always taken the machine's instruction "Enter Amount" to refer to the first decision point in the process, namely my choice to make a deposit. In technical*

[13] A feature sometimes called *"Transaction–Splitting"* with ATMs.

[14] Note how, in comparison to the issues raised earlier with Obscure Security Flaws (issue 5) that again cancelling an entire transaction is a useful error–recovery option. However, note also that in this particular case, it wa a post–authorisation, not a pre–authorisation, transaction cancellation.

terms, I assumed (correctly) that the machine operated using a simple model of what computer scientists call a queue — first–in–first–out. My wife, on the other hand, viewed the transaction as a highly constrained human interaction, where reference would generally be to the thing mentioned last. Since the instruction "Enter amount" came immediately after reference to wanting cash back she took the instruction to refer to the amount of cash she wanted back. The model she was assuming (incorrectly) was that of a computer scientist's stack —- last–in–first–out."

The above issues taken together clearly show that the ATM deployment context presents many dependability issues that were not easily anticipated, or that can be easily designed for — without unearthing such *'hidden'* assumptions. The following issues echo some of these points, although to prevent needless repetition with CBS context issues already discussed (i.e. engineering context and utility context), the focus will be upon more evolutionary context issues.

5.2.4 Specific ATM Environment Adaptation

As already discussed earlier, an ongoing strategic policy of banks and financial institutions has been to promote 24–hour availability of basic banking services which has resulted in the deployment of higher numbers of ATM services in increasingly diverse and remote locations. While this no doubt offers greater convenience for banking customers — especially in countries, states and provinces where bank opening is restrictive [129], it is also possible to recognise that there is a potential conflict, in dependability terms, between attributes of availability of service and security and safety of these services. These conflicts are the focus of the last two issues raised in this chapter below.

First the safety related issue:-

Environmental Adaptation — Safety Issue 8. *Safety concerns appear, from a number of research surveys, to be a general concern of many ATM users when using ATMs [cf. [127, 130]]. However, when*

ATMs are remotely located, then this general concern appears to be even more relevant — especially during off–peak times. These fears are further reinforced by reports that (both in the US and UK) usage of ATMs can leave customers vulnerable to physical attack. In the worst cases police have believed that ATM users have been even murdered for their ATM card(s) and PIN information while using ATMs [131, 114].

Secondly, the security related issue:-

Environmental Adaptation — Security Issue 9. *Security concerns also exist with the situation of remotely located ATMs. Fraudsters, who have obtained stolen ATM cards or have been able to clone ATM cards are much more likely to carefully choose which ATMs they use to make illegal or unauthorised withdrawals. It is obvious that these will most likely be remotely located ATMs during off-peak times as this allows them then to circumvent visual detection devices provided by ATMs (i.e. VCRs and CCTV) by wearing masks or disguises etc [cf. [132]].*[15] *Furthermore, sophisticated fraud, that results in the cloning of many ATM cards would require fraudsters to target remote ATMs during off-peak times as it would require the reinsertion of many ATM cards which would quickly attract suspicion by passers-by and other ATM users [cf. [133]].*

These conflicting issues, between service availability, safety, and security result in ATM dependability vulnerabilities due to the commercial policy of 24–hour remote access not reflecting how dependability is undermined (in terms of customer safety and banking security against fraud) by not considering how ATMs, and their services offered, must be appropriately adapted to the particular environment in which they are deployed and operated. Remote ATMs, such as those deployed on industrial estates, colleges/universities, supermarkets, etc experience

[15]It was recently reported in the press that 'skimming' devices fitted to the ATM card–reader device were primarily used on remote ATMs in villages, etc — as, unlike like those in towns and cities, such ATMs are not commonly under the surveillance of Close Circuit Television (CCTV).

high usage only during certain times (i.e. working hours). During off-peak times, legitimate demand for such services will be greatly reduced anyway, so they could restrict their services — such as reduced withdrawal amounts being available as in the example reported by Curry [134], or completely shutting ATM services down in the interests of customer safety and bank security which would dissuade both customers and fraudsters from using them in preference for: a) a more safer ATM for legitimate services; and b) preventing opportunities for fraud, or forcing fraudsters to use ATMs that will increase thier capture and detection.[16]

Vulnerabilities discussed in these two issues reflect an inability to consider the Evolution context, in terms of ensuring that ATM services (or service restrictions) are adapted dependably (specifically – safety and security) to their environments (both geographically and temporally in these cases).

5.3 Chapter Summary

In this chapter a more detailed and specific exemplification of a computer–based system perspective has been provided. As stated in chapter 4, a computer–based system perspective expands the boundaries of the system to encompass and consider both the technical and human systems as subsystems–of–interest. When this view is adopted both types of subsystems are viewed as interfacing systems–of–interest. This chapter has used the specific application domain of the Automatic Teller Machine (ATM) to illuminate a much more hollistic and integrative perspective of a ATMs as computer–based systems than is normally the case. This has been achieved by breaking the analysis and considerations of the ATM into four generic computer–based system contexts–of–interest. These are: a) the strategic utility context — that ecompasses the strategic value justifications for the systems existence; b) the engineering context — that is challenged with the creation of the technical system; c) the deployment context — that is directly responsi-

[16]For example, where I work the University has an ATM on-site which is very remote during out-of-hours, but would make an ideal target for a mugger or a fraudster to use. Conversely, in Consett Co. Durham there is an ATM virtually outside the town police station. If I had to use an ATM during the early hours, I would use the latter ATM. Equally, it is more probable a mugger or a fraudster, would seek to use the former ATM.

ble for its day–to–day operational requirements; and d) the evolution context — that is concerned with maintenance and future adaptation considerations. The ATM application domain was therefore analysed within this computer–based system conception as it is a long–established domain that provides interesting and insightful examples of how computer–based system dependability can become compromised. A total of nine such issues have been presented, analysed, and discussed from a computer–based system viewpoint in detail. It can be seen that many failures of computer–based systems are holistic, in nature, and a good way of improving holistic understanding of these failures is by achieving a balanced emphasis and interpreting them into the generic contexts. Such a computer–based system view illuminates how vulnerabilities, faults, errors, and failures are often a matter of how contextual interests are often over/under represented or subsequently promoted.

Chapter 6

Assumptions

6.1 Chapter Introduction

In this chapter the issue of assumptions is explained. Assumptions are often characterised as *"unstated reasons"* used for a basis of argument or activity. While, broadly, this is true, it doesn't reflect the many possible ways assumptions can occur or the underlying influences that can often give rise to them. This chapter first focuses upon reasoning and assumptions. In doing so, both formal deductive and informal inductive reasoning is discussed, as these both have interesting implications for how assumptions are used or emerge. Next the role of communication and assumptions is discussed. In communication, assumptions can result from a number of influences that can result in ambiguities and inconsistent meaning interpretations. Assumptive influences in problem solving activities are then covered. It will be indicated that effective problem solving can sometimes be undermined by artificial constraints and limits we impose upon our thinking during problem solving activities. Finally, the importance of context and assumptions is discussed. It can be seen that a contextual situation (however this is interpreted) often results in collective or shared assumptions being made. While these can result in effective collaboration, there is also the danger that unquestioned assumptions can result in erroneous actions and judgements when the context changes, overlaps, or places serious knowledge acquisition limitations upon us.

6.2 Assumptions in Reasoning

As will be seen in the various subsections in this section, reasoning provides a good insight into the nature of assumptions. Before going into more detail, however, it would be beneficial to discuss broadly the established nature of reasoning. The particular facets of reasoning covered in the following sections relate to deductive and inductive reasoning.

6.2.1 Deductive Reasoning

Deductive reasoning has its origins in ancient Greek philosophy and is essentially concerned with the truth–preserving structure(s) of human reasoning [135]. In

essence it is the validity of the conclusions that can be inferred from the premises that is of major concern. In deductive reasoning structures, it is not possible to have a true conclusion from false premises. This can be shown as follows [cf. [136]: p. 109]:

> If inflation is receding, the government's economic policies are sound.
> Inflation is receding.
> Therefore, the government's economic policies are sound.

What is important in this piece of deductive reasoning is the structure. The first two lines provide the premises — the truth validity of which, ultimately determines the validity of the conclusion. In the reasoning structure above, the structure is a form of deductive reasoning known as *Modus Ponens,* which has the structure:

> If A, Then B.
> A is true.
> Therefore B is true.

Another valid deductive reasoning structure — based along similar lines, using the same structure as above is as follows:

> If inflation is receding, the Government's economic policies are sound.
> The Government's economic policies are not sound.
> Therefore, inflation is not receding.

Again, this deductive reasoning structure ensures the validity of the conclusions — depending upon the truthfulness of the premises. In this case the conclusion that inflation is not receding can be inferred from the lack of soundness of the Government's economic policies. This form of deductive reasoning is known as *Modus Tollens* and has the structure:

> If A, then B.
> B is not true.
> Therefore, A is not true.

A number of invalid deductive reasoning inferences also exist — based upon this deductive structure. The first is commonly known as *Affirming the Consequent* as follows:

> If inflation is receding, the Government's economic policies are sound.
> The Government's economic policies are sound.
> Therefore, inflation is receding.

In this case the conclusion is not necessarily true — even if both premises (A and B) are true as inflation may or may not be receding for many other reasons quite apart from the quality of the Government's economic policies. In such cases the conclusion results in a *non sequitur* — meaning that it doesn't necessarily follow from the premises. Such an invalid reasoning has the following deductive structure:

> If A, then B.
> B is true.
> Therefore, A is true.

Another invalid deductive structure is know as *Denying the Antecedent* as follows:

> If inflation is receding, the Government's economic policies are sound.
> Inflation is not receding.
> Therefore, the Government's economic policies are not sound.

Again, in this case the conclusion that the Government's economic policy is not sound does not follow from the premises in a deductive sense since although inflation is not receding the Government's economic policies may (or may not) be sound for a wide variety of other reasons irrespective of the inflation rate. This invalid reasoning has the following deductive structure:

> If A, then B.
> A is not true.
> Therefore, B is not true.

Although there exist other deductive reasoning structures not covered here [cf. [136]: pp. 108—133] the main point is that the structure of the reasoning, in deduction, is all important in ensuring that: a) a true conclusion cannot be deduced from false premises; or b) a false conclusion cannot be deduced from true premises. Although to say that an argument is deductively valid is to say something strong and positive about its structure, as can be seen from some of the sections that follow, the validity of the premises themselves can often be an important pragmatical concern.

6.2.2 Inductive Reasoning

While deductive reasoning can inform us of the truth–preserving nature of reasoning structures, much of our reasoning in everyday situations is not so much to establish the truth of something as it is to reinforce our beliefs or provide extra information about something of interest [137]. This form of reasoning is essentially probabilistic, in nature, and reflects a reasoning process that is essentially moving from specific knowledge, based upon experience, observation, and understanding to generalisations that reinforce our belief system(s) (at whatever level) and our comprehension of the world we live in [135].

A good example of this inductive reasoning process is exemplified, within a problem–solving context, by Polya [138] using a characterisation of the famous journey of Christopher Columbus in 1492, as follows [p. 178]:

> *"As Columbus and his companions sailed westward across an unknown ocean they were cheered whenever they saw birds. They regarded a bird as a favourable sign, indicating the nearness of land. But in this they were repeatedly disappointed. They watched for other signs too. They thought that floating seaweed or low banks of cloud might indicate land, but they were again disappointed. One day, however, the signs multiplied. On Thursday, the 11th of October, 1492, they saw sandpipers, and a green reed near the ship. Those of the caravel Pinta saw a cane and a pole, and they took up another small pole which appeared to have been worked by iron; also another bit*

> *of cane, a land–plant, and a small board. The crew of the caravel Nina also saw signs of land, and a small branch covered in berries. Everyone breathed afresh and rejoiced at these signs. And in fact the next day they sighted land, the first island of a New World."*

The point Polya wished to make from this is that in unknown situations we often use past experience and knowledge in an analogous reasoning manner to interpret signs that we are making progress towards something (be it a solution to a problem, or discovering new lands). Polya goes on to show the inductive reasoning process, using the above example, with the well known 'if, then' deductive reasoning structure discussed in section 6.2.1, as follows:

> If we are approaching land, we often see birds.
> Now we see birds.
> Therefore, it becomes more credible that we are approaching land.

It can be seen, from section 6.2.1, that in a purely deductive reasoning context, this structure represents a formal fallacy of *Affirming the Consequent.* However, Polya notes that the conclusion is written in the following manner:

> If A, Then B.
> B is true.
> Therefore, A Becomes More Credible/Probable.

Polya argues, that while, within a deductive evaluation, this structure is fallacious, in a inductive/heuristic reasoning context, this structure is both fair and reasonable — providing the conclusion(s) are stated in probabilistic inductive terms and not in a formal deterministic deductive manner. Polya also notes that in using an inductive reasoning approach, the more data or more frequent the signs are then the more these reinforce the probabilistic conclusions. From a wider computer–based system standpoint it can be argued that inductive reasoning can result in faults and errors through one computer–based system context making (seemingly quite) reasonable associations that are well established and common within that particular context–of–interest in terms of past experience, training etc. Such an example was

the (seemingly quite reasonable) implicit assumption that an ATM user should always be allowed to completely undo a transaction (i.e. pre authentication undo) — as this is widely considered to be a good usability principle. This computer–based issue (i.e. Obscure security flaw conflict — issue 5) was discussed in section 5.2.2 of chapter 5. However what was overlooked in this specific issue was the security vulnerability that this introduces that allows a fraudulent enumeration of an ATM customer's PIN code. What is actually needed is for the system to either enforce authentication before the ATM user can completely undo the transaction or record any failed PIN code verification attempts between distinct ATM accesses.

6.2.3 Suppositions and Presuppositions

Suppositions are premises used for progressing a particular line of argumentation or reasoning [135]. A particular facet of suppositions is that the person making them does not necessarily believe in the premise(s) themselves. Two examples of suppositions is provided by Warburton [[135]: p. 116]:-

> *...[police inspector] "**Had the murderer entered the house by the window.** Surely we would expect to find some evidence of a forced entry."*
>
> *...[prosecuting lawyer cross–examining the defences' expert witness] "**Even if we believe that you are right that watching video nasties can trigger violence in a small percentage of viewers.** Can you be sure that they wouldn't have found other triggers if video nasties didn't exist?"*

It can be seen from both these examples that the emboldened text represents explicitly stated premises that are *"supposed"*. In neither cases does the person(s) making them necessarily believe in the truth of the premises, instead, they are used only as an informal reasoning mechanism — much like a hypothesis, in order to continue a particular avenue of argumentation, reasoning, or thinking. Because of this explicit hypothetical nature, suppositions are *'internal'* to the structure of argumentation, thinking, or reasoning and therefore they usually represent an explicitly stated form of assumption(s). Such informal reasoning mechanisms are

widespread in the software engineering literature [cf. [56, 9]] and although they can be useful reasoning mechanisms thier informalness can often turn out to be a weakness. Such an issue was discussed within chapter 5 (i.e. State representation completeness and consistency — issue 7). It can be seen that the physical cash magazines were never assumed or supposed to become empty or contain insufficient money to fulfil an ATM customer's withdrawal request, but such an informal and assumed justification can easily prove to be false if not investigated and verified.

Presuppositions are somewhat different, in that a presupposition is a proposition of another statement, which, if the presupposition is false, makes the statement (for which it is a presupposition) irrelevant, or pointless [139]. By contrast with suppositions, presuppositions tend to be, by nature, *'external'* to the argument, reasoning, or thinking process. This is exemplified by Ennis [[139]: pp. 76-77] who also stresses that in everyday usage people are particularly susceptible to accepting and becoming committed to the external and unstated assumptive nature of presuppositions, noting:-

> *"...I find myself less resistant to believing the [unstated] proposition that there is a missile gap when I am told, "The missile gap will take five years to eliminate." than when I am told, "There is a missile gap and it will take five years to eliminate.""*

In the first statement there is an increased compulsion to assert (see subsection 6.2.4) the presupposition of *"a missile gap exists"* due to this proposition being *"pre–supposed"* to be true and externalised (i.e. unstated). Whereas, in the second statement form, this proposition is explicitly asserted and internalised (i.e. stated). As a consequence, there is an increased likelihood that someone may seek to question the truthfulness or falsity of the premise part of this proposition by asking *"Is there a missile gap?"* In some cases, pre–suppositions can be used maliciously and deliberately, in order to get people to commit to a particular argumentation line or reasoning position. This is particularly the case with asking and answering questions — as questions include positive information in the form of propositions

[136]. The answerer who responds directly to a question based upon presuppositions risks committing to any (or even all) the presuppositions it is based upon. Walton [[136]: p. 29] provides the classic loaded question based upon harmful pre–suppositions for the directly agreeing respondent, as follows:-

> *"Have you stopped beating your spouse?"*

No matter which way the answerer directly responds by answering *"yes"* or *"no"* they become committed to the *"pre–supposed"* (and prejudicial) proposition that *"You have beaten your spouse"*.

Therefore, in stark contrast to suppositions, pre–suppositions contain an implicit hypothetical characteristic that tend to be *'external'* to the structure of argumentation, reasoning, answering, or thinking and can often result in people unwittingly believing in, or committing too, the unstated propositions they are based upon. This is indeed a flaw in the structure of deductive reasoning since in both the examples of correct forms (e.g. modus ponens and modus tollens) in subsection 6.2.1 the causal relationship between (to take the example given there) inflation and the soundness of the government's economic policies is presupposed. With deductive structures there need not be any causal nature to provide deductively valid argument structures. The following example demonstrates this:-

> If it is Monday, I'm a millionaire
>
> It is Monday
>
> Therefore I'm a millionaire.

It can be easily appreciated that although this structure is a deductively valid case of modus ponens there is clearly no causal relationship between the day of the week and someone being a millionaire. Such deductive type structures are ubiquitous in reasoning and implementing (i.e. IF/THEN condition branches in programming languages) software systems and presupposed completeness of subtle causal relationships between such things can often be the source of faults, errors, and failures.

6.2.4 Assertions and Beliefs

Unlike suppositions and pre–suppositions, assertions represent an explicit, yet unsupported, statement of belief by the individual making the assertion [135]. In this case, an assertion is not some externalised or implied proposition in the argumentation, reasoning, or thinking process that someone unknowingly believes in, or commits to. Rather, it is a stated explicit statement of belief in the truthfulness or validity of something.

However, simply asserting something, does not necessarily make it true in itself, as the following influences can undermine or help establish the validity of something asserted:

6.2.4.1 Beliefs

Our beliefs, while important in establishing our sense of identity with the world around us, are, at best, changing incomplete co–creations and constructions which have been heavily influenced, biased, and formed by our social, cultural, and individual experiences [140]. Therefore, even though we may genuinely believe in something we have explicitly asserted as being true or valid, the underlying beliefs upon which it is generated may be based upon incomplete knowledge or some (largely) unchallenged assumption(s) about something(s). O'Conner & McDermott [[140]: pp. 62-81] characterise human beliefs as a cognitive mental model and reveal four main ways in which humans create, maintain, and change their belief systems:-

1. ***Deletion:*** Every day our senses are inundated with massive amounts of information which we could not possibly hope to accommodate. Therefore we need to selectively filter this information which is usually dependent upon our moods, interests, values, and preoccupations. Although there is always other information which we could have chosen, our belief system is formed and maintained through the deletion of other information that accords with our (believed) notions of what is, and what is not, important to us;

2. ***Construction:*** Is the inverse of 'deletion'. We often have the compulsion to construct things that actually don't exist at all through trying to link (what we consider to be) *"...probable cause with possible effect."* Uncertainty will invariably result in *'gap'* construction, whereby we fill-in gaps so that our pre-existing beliefs are reinforced so that the world still makes sense to us;

3. ***Distortion:*** Is when we change our actual experience by increasing some aspects while reducing others. While this can be a healthy thing to do, it can also be done as a protective measure to preserve our existing values, beliefs, and conceptions about our world;

4. ***Generalisation:*** Is when we *"...take one experience and make it represent a group."* It has very close links with inductive reasoning. While it is, in many cases, a valid mental conception which allows us to learn and build knowledge, the risks are that we take unrepresentative experiences and over–generalise them to all similar situations — becoming oblivious to the extent at which we have done so. As O'Conner & McDermott warn: *"Generalisation combined with prejudice...is the basis of all racial and sexual discrimination."*

As will be seen from other sections and subsections, in this chapter, such traits that influence our beliefs (and resulting mental models) can have a direct bearing upon the assumptions we make in any given situation.

6.2.4.2 Formal Argumentation

In formal argumentation, a careful separation of assumptive beliefs and explicitly stated assertions are made. A good example of this careful separation is provided by Velleman [[141]: pp. 82-85], advising:-

"Never assert anything until you can justify it completely."

By this Velleman argues that in proof-writing techniques, based upon the logical deductive form of the conclusion, often involves transforming the problem to be solved into an equivalent, but easier one. Frequently they involve steps in which

the prover must assume, for the sake of the proof, that some statement is true without providing any justification for that assumption. Whilst this, at first, appears to conflict with the above rule that assertions must always be justified, it is again pointed out, that to assert something is different from assuming something in formal argumentation. Assertion means to claim that a particular statement is true, while the purpose of making an assumption in a proof is not to make a claim about *"what is true"*, instead it is used to help determine *"what would be true"* if the assumption involved was to be correct. This is why it is necessary to be aware that any conclusion that may be arrived at based upon an assumption may well turn out to be false if the assumption is incorrect. For instance, if during some proof to establish the truth of statement Q, someone assumes that a statement P is true, and then uses this assumption to later conclude that another statement Q is true, it would be false to believe that this is a proof of Q as they have not established that the assumption P (upon which the proof of Q is truth dependent) is true. The proof of Q would be (at best) incomplete. However, if the conclusion of the proof was a composite proof to prove P => Q then the proof is complete as the following is a well established proof strategy [cf. [141]: pp. 85–91]:-

> **To prove a conclusion of the form P => Q:**
> *Assume P is true and then prove Q.*

The following proof is a specific simple example of the form P => Q [cf. [141]: pp. 86-87]: ***a*** and ***b*** are real numbers. Prove that if $0 < a < b$ then $a^2 < b^2$:-

- Given(s): is that ***a*** and ***b*** are real numbers;
- P: is the statement $0 < a < b$;
- Q: is the statement $a^2 < b^2$;
- Prove: Conclusion is P => Q;
- Assume: $0 < a < b$ and make $a^2 < b^2$ the goal;
- Multiply both sides of inequality $a < b$ by ***a*** and ***b***;
- Gives: with ***a*** $a^2 < ab$ and with ***b*** $ab < b^2$;

- Gives $a^2 < ab < b^2$;
- Proved: Therefore, if : $0 < a < b$ then $a^2 < b^2$.

It can be seen, therefore, that, due to the analytical reasoning power offered by mathematical proofs, assertions are more than just unsupported explicit statements of belief. They are statements, the truth validity of which, should be established through formal reasoning before being employed as such statements of truth. Additionally, there is a sharp distinction made between assumptions and assertions. Assumptions are used in formal reasoning, much like suppositions are used in informal argumentation, as a reasoning mechanism which is not believed to be true but only taken as being true to pursue a specific line of argumentation (in this case mathematical). In this regard we can appreciate the benefits of formal argumentation over informal argumentation as formal argument provides a more rigorous proof mechanism to formally verify the integrity of assumptions used in reasoning about systems, once they have been identified as such. As mentioned earlier in subsection 6.2.3 (on suppositions and presuppositions) the assumed physical state of the cash magazines[1] can be more rigorously investigated using formal argumentation to ensure this is, in fact, the case. Indeed this ATM consideration was explicitly covered in a formal ATM specification using VDM [124].

6.2.5 Enthymemes or Suppressed Premises

Enthymatic reasoning is an argument with a suppressed premise. During everyday speech and interaction it is largely unnecessary to have to make explicit every premise of an argument or line of reasoning — as many people will share and understand the suppressed premises. In some cases, however, if this premise is not realised then the conclusion of the argument can often result in a *non sequitur* (which means *"It does not follow"*) [135]. Because of the suppressed premises in enthymatic reasoning they often have the structure of: a) premise, so conclusion; or b) premise, therefore conclusion. This is shown in the following two examples. First example is a full deductive reasoning structure:-

[1] State representation completeness — Issue 7. "That there will always be sufficient money in the physical cash magazines to fulfil an ATM customers withdrawal request."

All Sundays are days when I don't have to go to work.
Today is Sunday.
Therefore, I don't have to go to work today.

The second example is the same reasoning , but this time as an enthymeme:

Today is Sunday.
Therefore, I don't have to go to work today.

This second example is an enthymeme where the premise: *"All Sundays are days when I don't have to go to work"* has been left out. This is not a problem in argumentation providing that everyone understands which premises has been suppressed.

However, Warburton [135] reveals that when either the premise is not clearly understood from the context or when there are multiple possible suppressed premises that could be inferred, then, in informal argumentation terms, the enthymatic reasoning structure of: premise, so conclusion; or premise, therefore conclusion, is deemed to be spurious and is considered to be a subtle case of *non sequitur* reasoning. Two examples indicate this. The first example is a case where a person is not familiar with the particular context,[2] and would therefore be confused by the following enthymatic reasoning structure:

This cereal contains wheat.
Therefore you should not eat it.

Now if this line of reasoning was used within a group of people — many of whom were not aware of a particular person suffering from celiac[3] absorption disorder (or even worse have never even heard of such a disorder) then they are very likely to think that the person making the reasoning had made a conclusion that simply did not follow (i.e. non sequitur). They would quite naturally become inquisitive

[2]The context in this case can be thought of as a knowledge context.

[3]People affected by this disease experience damage to the villi which shorten and flatten in the lamina propia in the intestines when they consume certain foods that contain toxic amino acid sequences. Such amino acid sequences are found in wheat and barley [cf. www.celiac.com]

and perhaps ask somthing like: *"what does wheat have to do with not being able to eat the cereal?"* However, the person making the reasoning, and the absorption disorder sufferer, would clearly understand the knowledge context and therefore the suppressed premise that *"You should not eat wheat products"*.

The next example is where multiple possible suppressed premises could be inferred. In such enthymatic reasoning situations, others may believe that either the reasoning is a disguised assertion of belief (see subsection 6.2.4) or that the reasoner's claims are confusing or spurious — resulting in another subtle case of non sequitur reasoning [cf.[135] : p. 113]:-

> Boxing often causes brain damage, so it should be banned.

Several suppressed premises could be installed with this enthymatic reasoning structure: a) *"Any activity which often causes brain damage should be banned"*; b) *"Sports which often cause brain damage should be banned"*; and c) *"If boxing often causes brain damage then it should be banned"*. There is obviously more that could be listed, but the point is that, with enthymatic reasoning structures, the suppressed premise should be made explicit in situations where others cannot detect the intended premise to install and comprehension is therefore lost.

6.3 Assumptions in Communication

It should be obvious from the discussion earlier on enthymatic reasoning in subsection 6.2.5 that assumptions are important to communication — on the condition that the suppressed assumptions are clearly understood or made explicit so they can be questioned. During communication, we all have to make assumptions a lot of the time. The important aspect is to be aware of the assumptions that are being made and to ensure that they are true [135].

This is easier said that done, however, as misunderstandings in communication can have, at their root, underpinning assumptions that result in false interpretations. Such well known ambiguities where multiple interpretations are possible include [cf. [135]: pp. 9–11]:-

- ***Lexical Ambiguity:*** This is when a word with two or more potential interpretations is used in a statement so that the format in which it occurs could be comprehended in different ways. A good example, is in the case of humour when a word is used as a pun that invokes two relevant meanings in that given context. The example given by Warburton is: *"Dr. Johnson saw two women standing on their doorsteps arguing. He quipped that the two women will never be able to agree because they were both arguing from different* ***premises.***" Obviously the dual meaning of the word *'premise'* in this case has two pertinent meanings in this particular context. Awareness and appreciation of this provides the particular humour intended by Dr Johnson;

- ***Referential Ambiguity:*** This is a situation where a word is used that could be understood to refer to two or more objects. Referential ambiguities tend to happen when using a pronoun such as 'it', 'her' , 'him' and 'they'. In such pronoun usages the pronoun does not make it precisely clear what the pronoun is referring to. While in many situations the usage of such pronouns will be made clear from the context, even when not using pronouns within a context, referential ambiguity may still occur. Such a situation is also exemplified by Warburton: *"If two people are in a room and both are called John, then just walking in and saying: "There's a phone call for John" will be confusing to both of the people called John."* In such a situation other cues (i.e. looking at the John for whom the phone call is for) would be required in addition to the above statement. If, of course, the person making the statement didn't know which John it was for, or didn't know that there were two people called John in the room, then this is a situation where extra information and perhaps subsequent inquiry would be clearly required;

- ***Syntactical Ambiguity:*** This is sometimes called *amphiboly*. It occurs when the ordering of words can invoke two or more interpretations. A good example of syntactical ambiguity provided by Warburton is: *"I heard about what you got up to at work yesterday."* This statement has syntactical ambiguity in two ways: Firstly, the statement could mean either ***they*** heard what ***you*** got up to when ***you*** were at work, or ***they*** were at work when ***they***

> heard what ***you*** got up to. In this case there is ambiguity about where the two people involved were when they heard about it; Secondly, the statement is ambiguous because the order of the words makes it vague as to ***when*** the person heard about ***what*** happened or ***when*** the incident actually occurred. Was it yesterday that ***they*** heard about the person doing something, or was it something that the person did yesterday?

Although it is hard to remove all ambiguity in communication, when there is the potential for serious misinterpretations, it is better to make the intended meaning as clear as possible [135]. This is particularly true in some contexts, situations, or activities, as not ensuring consistent meanings can have disastrous results. A good example is provided by Delvin [[128]: pp. 76–79]. In December 1995, American Airlines Flight 965 from Miami to Columbia was on its final approach to Cali airport when it crashed into a nearby mountain range, killing all the 159 passengers and crew on board. When the final crash investigation report was published the following August it was obvious that the crash was not the result of some mechanical failure or the consequence of bad weather. Instead it was the result of a decision based on the meaning of information represented by the on-board computer system and another meaning being interpreted on that on-board computer information by the flight crew.

What actually happened was that the air controller at Cali instructed the crew to fly toward the nearby beacon called *"Rozo"*. This was identified on the navigational charts by the letter *"R"*. The crew entered that letter into the on-board flight management computer system and the screen presented a list of six navigational beacons. To the flight crew, such a list presents the beacons on a ranking from nearest to farthest from the plane. Therefore, the crew naturally accepted that the top ranked entry on the screen denoted by *"R"* was the *"Rozo"* beacon nearest to them. However, the air traffic control and the flight crew on-board were operating with different listings and meanings connected with the top ranked letter *"R"*. The air traffic control computer system ranked the beacons from farthest to nearest (not nearest to farthest) and the top ranked beacon denoted by *"R"* referred to the beacon *"Romeo"* in Bogota airport more than 100 miles away in a direction more

than 90^{o} off the required course. As a result, when the flight crew selected and entered the top ranked letter *"R"* into the on-board flight management system, the autopilot silently turned the plane more than 90^{o} to the left toward Bogota airport. By the time the flight crew and ground air traffic control crew realized what had gone wrong it was impossible for the aircraft to avoid crashing into the mountains.

Clearly we can see that the flight crew and air traffic control crew attributed and therefore interpreted different meanings as to which beacon was being referred to by the letter *"R"*. These meaning attributions and interpretations were underpinned by different assumptions built in the form of context conventions concerning the ranking order of the beacon listing contained within the ground crew's computer system. The on-board flight crew (in an out–of–context manner) assumed the beacon listing ranked beacons from top to bottom to represent the nearest to farthest beacons, whilst the air traffic ground control crew worked with the (known in–context valid) assumption that the beacon listing always ranks the beacons farthest to nearest.

Here then, we can see that many ambiguities can arise in communication due to one or more meanings that can be attributed to something. In such situations the criteria for choosing a particular meaning from the set of possible other meanings is usually underpinned by some assumption. It is obvious from the examples provided that the particular context (context is broadly meant e.g. physical, cultural, knowledge, etc) possesses both the potential to provide cues that can clarify and result in a consistent meaning during communication, or, conversely, obscure and result in (potentially disastrous) inconsistent meanings being interpreted during communication. These contextual influences will be returned to in section 6.5.

6.4 Assumptions in Problem–Solving

Assumptions often play an important role, also, in problem–solving. For example, Delin et al [142] wished to gain a more psychological understanding of the nature of assumptions in the context of problem–solving. To do this, Delin et al

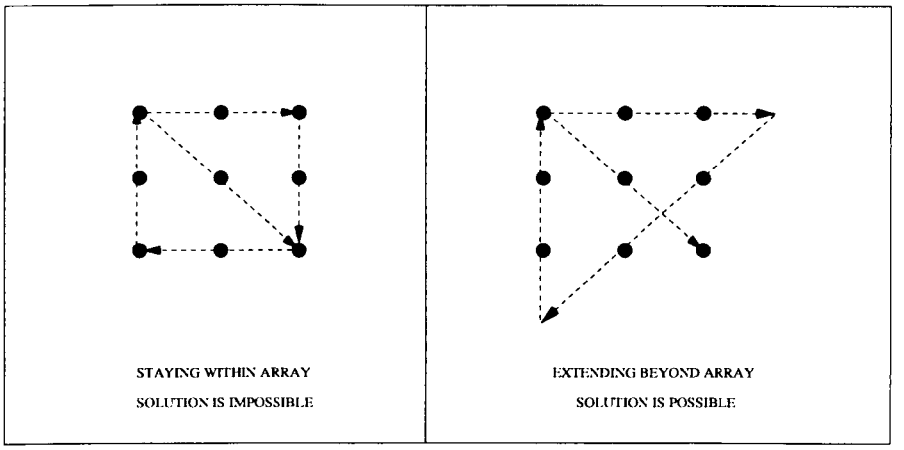


Figure 6.1: The Nine Dots Problem

deliberately chose a range of problems which people were prone to make assumptions about. Furthermore, the problems selected were impossible to solve unless the particular assumption was identified. One of the many problems chosen was (what I will call) *'the nine dots problem'* . This problem is illustrated in figure 6.1. The task is to join up all nine of the dots in the array with four straight lines while not removing the pen from the paper.

The difficulty with solving this problem is that if the person attempting to solve the problem tries to stay within the length of the dots array then the minimum number of lines required to join up all of the nine dots requires five lines (i.e. shown on the left hand side in figure 6.1). The only way the problem can be solved is to become aware of the implicit assumption that people often make when reasoning about this problem of:-

"I must stay within the area of the nine dots array."

However, Delin et al's contention was that it is widely believed that people can somehow become aware of the assumption. Over a number of controlled experiments using (what we may call) *'assumption–seeking'* problems, Delin et al split the problem solvers into two groups: the control group would not be told in advance that they need to be aware of making some assumption in solving the problem; while the experimental group was explicitly told, before starting, that they

needed to be aware of making some assumption in order to solve the problem. The results from the experiments showed no significant performance increase of the experimental group over the control group. This Delin et al took to indicate that although common usage of the term *"assumption"* leads us to believe that it is something we carry around with us in our heads while thinking, reasoning, or arguing, Delin et al argued that it is more like a constraint acting upon our particular thinking episode about something at any particular time. In their own words Delin et al state:

> *"Most assumptions tend to correspond more to the absence of conception than its presence."*

They further argue that the only way to finding such assumptions would be to examine one's own thinking to try and observe in what ways it was becoming constrained. However, the shortfall they recognise in this kind of *'meta-thinking'* or cognitive reflection approach is that in searching for such assumptions the individuals conceptual searching would be limited by the same conceptual constraints that enforced the assumption to begin with.

These views that assumptions are constraints upon the mental conception about something are consistent with DeBono's [143] long–held views about assumptions within the *'Lateral Thinking'* philosophy. With this type of thinking the purpose is to deliberately restructure an individuals mental conception about how they think about something. DeBono argues that constraints upon the mental conception about something results from long–held conventions or traditional ways about looking at something. He notes that such stereotypical views not only constrain how we may think about something but also these *cliche* mental patterns restrict our ability in even trying to re-think about them.

DeBono recognises that in problem–solving a person always has to assume certain boundary conditions. While this is a necessary thing to do, otherwise we would become ineffective and indecisive in doing so, he warns that the danger is that we often impose such boundary conditions without knowing we have done so or lazily impose them for no better reason than that it is convenient to do so. In this

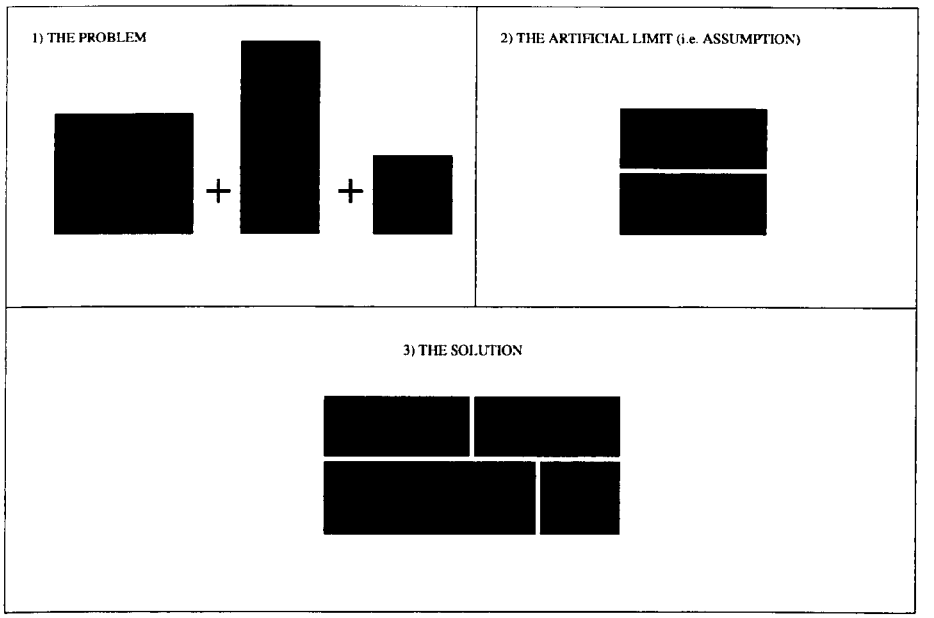


Figure 6.2: Example of an Artificial Limit (source: [143] p. 83)

respect DeBono refers to such assumptions that impose unnecessary boundaries upon our thinking as *"artificial limits."*

In figure 6.2 is a reproduced example, from DeBono [p. 83], of a problem that people often find unsolvable because they impose upon themselves an artificial conceptual limit (i.e. an implicit assumption) which prevents them from finding the solution.

The task is to arrange the shapes in (top left portion 1 in figure 6.2) to give a single well identifiable shape that is easy to describe that has only four straight sides. It is impossible to accomplish by just adding the existing shapes together, but, if instead of trying to fit the existing shapes together, the person decided to actual question and re–examine the existing shapes and consider actually splitting the large black square into two equal half's (top right portion 2 in figure 6.2) then it follows quite quickly to arrange the (now) four shapes into a simple rectangle with only four straight sides (bottom portion 3 in figure 6.2).

DeBono used this visual problem analogy to demonstrate how often a problem is impossible to solve by merely accepting the given shapes as fixed. Only by questioning that the shapes must remain fixed will the person stand any chance in solving the problem. He notes, however, that if someone was set this problem and after not being able to solve it they were told the solution, then there would more than likely be claims that this would be *'cheating'* as it would have been implicitly assumed that the given shapes could not themselves be altered. Such a situation is what DeBono characterises as an artificial limit in the mind of the person solving the problem.

In order to help identify such implicit assumptions DeBono advises that a person must take nothing as sacred and challenge the underlying assumptions by challenging the necessity of the boundary limits and the validity of the individual concepts that may underpin them. In his own words DeBono states [pp. 84–85]:

> *"As in lateral thinking in general, there is no question of attacking the assumption as being wrong. Nor is there any question of offering better alternatives. It is simply a matter of trying to restructure patterns. And by definition, assumptions are patterns which usually escape the restructuring process."*

To help ensure that assumptions do not escape the restructuring process, DeBono advices using the *"Why Technique"*. DeBono notes that while this is similar to a young child continually asking an adult *"why"* in order to understand something or gain additional knowledge, the purpose of this technique, in lateral thinking, is to ask *"why"* when the person does understand something and perhaps already knows the answer. The purpose is to deliberately restructure all possible thinking patterns to help ensure that any artificial limits (i.e. implicit assumptions) upon a person's mental conceptions about something is strongly challenged. DeBono does highlight, however, that although the technique seems easy it is much harder to perform properly as there is the natural tendency to run out of explanations or circle back on oneself and provide an answer that has already been used before. In addition there is a compulsion to just answer *"well because"* if something seems completely obvious. To avoid such problems DeBono advises that any *"why"*

questions should be directed to some particular aspect of any previous explanation rather than a general response. DeBono demonstrates this with the object of a blackboard, as follows:

- **Question:** Why are blackboards black?
  - *Answer:* So that the white chalk marks can be easily seen;
- **Question:** Why do you want to see the white chalk marks?
  - *Answer:* So that students in the class can see examples written by the teacher;
- **Question:** Why do students want to see the examples by the teacher?
  - *Answer:* So that they can better understand what the teacher is teaching them;
- etc, etc.

In each case the *"why"* question is focused upon some aspect of the previous answer.

The ultimate purpose of employing the *"why"* technique is to elicit more information. But to work, the technique should not at any point be comfortable, instead, at every stage a feeling of discomfort and tension should be felt in posing the questions and replying with answers. This is to attempt to force one to explain things and think about things in a different way. Only then is there a possibility in restructuring the thought patterns in such a manner that could unearth *'hidden'* or implicit assumptions. As an example of how implicit or *'hidden'* assumptions in computer–based system contexts–of–interest can result in flaws, vulnerabilities, and faults is exemplified in chapter 5 issue 4 (i.e. Opportunistic theft). A distinct possible contextual assumption made by the engineering context during development could have been of the nature *"Forgotten cash is the result of human error only."* Such a *'cliche'* mental pattern, in the mind of the developer, indicates an

unquestioned conventional representation that a legitimate ATM customer will either take all of the money, or in error, forget to take thier cash withdrawal and the implicit assumption acts as a constraint in considering that a legitimate ATM customer would be deviously motivated to remove only some of the dispensed cash — circumventing the retraction protection if additional account tracing and cash auditing functionality is not provided. As noted by DeBono earlier in this section, in order to break such artificial limits requires a more enquiring and challenging process that provides more information of the real influences present. Such assumptive cases vindicates the need for a more challenging computer–based system conception — that provides the potential for such implicit assumptions to be detected by interfacing different contexts–of–interest.

6.5 Assumptions in Contexts

It has already been inferred in section 6.3 that context — whether this be knowledge, culture, or some characteristic of the physical environment, etc, can result in individuals making different assumptions. In this section the role(s) that culture and knowledge plays in generating assumptions is discussed.

6.5.1 Culture

In a prolonged study that researched the national differences of cultures, Hofstede [144] likened culture to a *"software of the mind"*. By this, Hofstede didn't mean that humans are literally programmed in the same way a computer can be programmed, as humans still retain the ability to deviate from these influences and think in new ways. Rather Hofstede was using the analogy to mean that humans experience a significant amount of pre–conditioning right from their early childhood to adulthood, that hold a large influence upon how individual(s) react in certain situations. Such influences include family values, the particular neighbourhood a person grows up in, the particular school(s) they attend, the particular workplace they are employed in, and the wider community values that are formed around such institutions and influences. Hofstede also noted, from his research, that culture is always a collective phenomenon claiming [p. 5]:

> *"...it is at least partly shared with people who live or lived within the same environment, which is where it was learned. It is the collective programming of the mind which distinguishes the members of one group or category of people from another."*

Handy [60] provides additional justification for these views in his studies and experiences of organisational settings. He argued that assumptions affect, not only our institutions and organisations, but additionally, at a higher national social level, they have a significant influence upon the whole shape of our political structures, the design of our educational systems, along with the management and leadership of such institutions and organisations.

Such collective (and often unquestioned) belief structures can often result in problems and errors of judgement being made in a collective fashion. A good example of this is provided by Scholes and Johnson [145]. In the 1970s consumer goods organisations were very powerful, but lost a massive market share to grocery retail chains — which became larger and more organised. The consumer goods companies continued with the assumption that it was *'they'* that exercised a large influence over consumer buying patterns as this had been the case for many years prior. However, from the 1970s onwards, retailers become more organised, many mergers took place, and as a consequence, these organisations became much larger and exerted greater influence directly over consumer buying patterns. Both buying power and influence over the market structure passed over to the retailers and by the time consumer goods organisations realised the change many had lost their buying power and market place advantage which resulted in many going out of business or being taken over. Scholes and Johnson note, in particular, such collective organisational assumptions (i.e. *consumer goods companies have the power and influence over consumer buying patterns*) are not like explicit values of the organisation's culture, instead they are deeply held collective beliefs that are rarely talked about, made explicit, or thought to be problematic in any way.

A further point to note from the last example, and one that connects with systemic views from chapter 4, is that it highlights how the changing environment

can invalidate previously valid assumptions. A good example that exemplfies how previously viable assumptions about an ATM system can subsequently (over time) become invalidated is provided by issue 6 in chapter 5 (i.e. Interaction consistency and completeness). Here, although the assumption that the physical firmware dispensing mechanisms were capable of dispensing maximum amounts allowed by the embedded software DWL were originally viable, over time, with increases in the allowable DWL by embedded software control, the original assumptions made about the physical cash dispensing firmware limits was eventually invalidated. The result was an inability to physically dispense legitimately allowable DWL to the ATM user in certain circumstances.

From a situational information theory perspective, Delvin [128] explains how such collective problems and errors of judgement occur within contexts. Delvin provides the following formula to explain the concept of situational information theory [pp. 32–34]:

Information = Representation + Constraint

With regards information, Delvin makes a distinction between *data* and *information.* Data is simply a number of signals, signs, or symbols that contain no real information themselves. Information, on the other hand, refers to signs, signals, or symbols that contain or convey some meaning about something. Data therefore only acts as a representation for something. Because such representations can be interpreted in many ways there needs to be something else that converts a given representation into a particular interpretation. Once this representation conveys a particular interpretation then we can attribute meaning to some data. We often attribute a particular interpretation (i.e. meaning) to a given representation via systematic regularities or conventions we have long associated with particular signs, signals, and symbols (i.e. data representation). Delvin provides the following examples as cases where we attribute specific interpretations from such conventions and systematic regularities [p. 30]:

"There is systematic regularity between the existence of smoke and the existence of fire and a systematic regularity between dark clouds

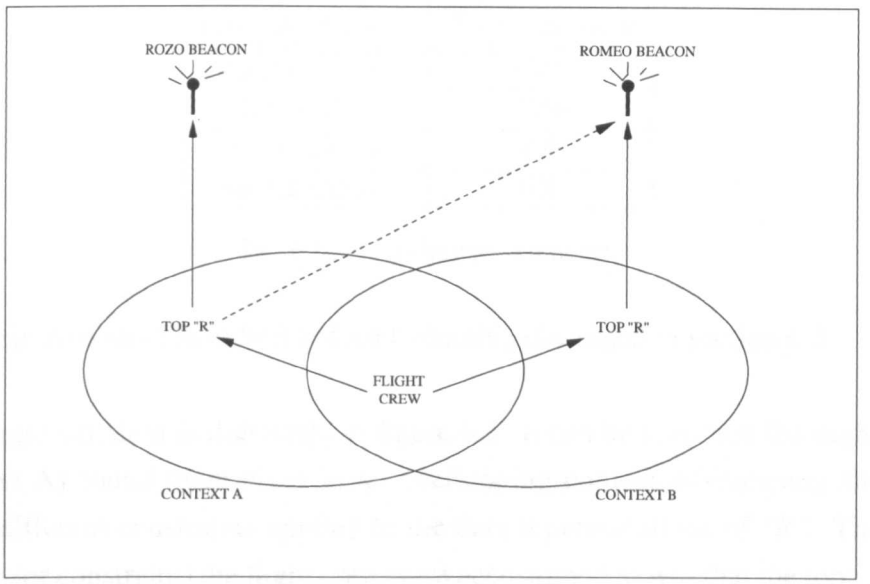


Figure 6.3: Overlapping Contexts [source [[128]: p. 78]

> *in the sky and rain. Human beings and other creatures that are able to recognize those systematic regularities can use them in order to extract information. The person who sees dark clouds can take an umbrella to work, and the animal that sees smoke on the horizon can take flight."*

Such conventions, systematic regularities, rules, guidelines, and (natural or human made) laws all constitute *constraints* that are placed upon a given data representation in order that it conveys meaning and hence information to us. However, the nature of the particular context in which the representation occurs therefore places the overriding selection of what constraints we attribute to a particular representation.

To return back to the role of Culture in generating assumptions, Delvin notes that such conventions and systematic regularities, based essentially on the same representations, can vary widely between countries, regions, organisations and institutional settings. In such situations problems and errors of judgement can occur when contexts overlap. Such an example was the tragic aircraft crash of

Possible Answer	% of Audience
Pacific Ocean	53%
Atlantic Ocean	32%
Indian Ocean	9%
Arctic Ocean	6%

Table 6.1: Audience Answers

American Airlines Flight 965 in Cali Columbia discussed in section 6.3.

This tragic situation is illustrated in figure 6.3. It can be seen that the flight crew (Context A) found themselves in an overlapping context–of–meaning situation where different constraints applied to the data representations of *"R"*. The convention (or constraint) the flight crew were accustomed to was that the top ranking letter *"R"* represented the nearest landing beacon *"Rozo"*. However, unaware to them was that the context they were now operating in meant that the top ranked letter *"R"* really referred to (shown by a dashed line) the *"Romeo"* landing beacon 100 miles away (and 90^{O} off course) in Bogota. Meanwhile, the ground crew (Context B) were operating in a local convention of representing the top ranked letter *"R"* as the farthest landing beacon *"Romeo"*. The confusion over these overlapping contexts of meaning resulted in the death of 159 people on-board. From a computer–based system viewpoint such meaning constraints that emerge from different contextual influences underlines the need to more thoroughly represent and make explicit different contextual meaning attributions as, at thier root, they are frequently underpinned by implicit assumptions. Failure to do so, as has been seen, can result in (potentially) serious flaws, vulnerabilities, faults, errors, and failures that ameliorates the overall dependability status of computer–based systems.

6.5.2 Knowledge

Limited knowledge, or limited means of knowledge acquisition can also result in collective assumptions being made. A good example of this was provided on a very popular television game-show in the UK in early April 2003. The question posed to the contestant was as follows: *"Which world ocean is 5.5 million square*

Possible Answers	Actual Answer %	Expected ad-hoc Answer %
Pacific Ocean	53%	25%
Atlantic Ocean	32%	25%
Indian Ocean	9%	25%
Arctic Ocean	6%	25%

Table 6.2: Biased vs Random Guessing

miles in size?" The possible four answers (of which one was correct) were: a) the Pacific ocean; b) the Atlantic ocean; c) the Indian ocean; or d) the Arctic ocean. Not being sure of the answer himself, the contestant decided to get the views of the live studio audience. Their responses is shown in table 6.1 on the preceding page.

The contestant selected, as his answer, the most popular studio audience answer, but lost when he was informed that the correct answer was actually the Arctic ocean. This represented the least popular studio audience answer — the view of only 6% of the live studio audience.

Now what makes this interesting is two things: a) that the vast majority (i.e. 94%) of the live studio audience incorrectly answered the question; and b) such knowledge already exists to answer the question correctly (i.e. size of the world's oceans exists in encyclopaedias, etc). However, the live studio audience only had approx. 20 seconds or so to make their selections. In such time constraints, either a person knows (and can remember) the answer or they will have to make a guess at the answer. It is important at this point to distinguish between *'informed'* guessing — where each individual interprets the information available in the question in some rational way, and random (or *'blind'*) guessing — where each individual simply accepts that they don't possess the knowledge necessary to ensure a correct answer and therefore makes an ad-hoc selection of one of the four possible answers without thought or supporting rationale.

If the majority of the members of the live studio audience had just made a random guess then the aggregated answer percentages would have been very unlikely to

have resulted in the percentage amounts shown in table 6.1. Given the highly specific knowledge required to know the answer, it is much more likely that the studio audience answered in an informed guessing manner and some information in the question collectively biased a large section of the studio audience members to answer in a very homogeneous intuitive manner. In fact in table 6.2 we can perform a two-tailed Chi-Squared statistical assessment — based upon the actual answering percentages of the studio audience that night, and what would have been (approximately) expected if the vast majority of the studio audience had simply made an ad-hoc (i.e. random) guess.

In such an assessment x^2 results in an obtained value of 30.9. Given the number of rows and columns the degrees of freedom is 3 (i.e. rows = (4-1) = 3 and columns = (2-1) = 1, so 3 x 1 = 3 degrees of freedom). Any x^2 value greater than 16.27 represents a statistical significant probability of $p < 0.001$. Meaning, in statistical terms, that we can be confident that there is only a 1 in 1,000 chance that the live studio audience, answering in an ad-hoc random fashion to this question, would have produced the answer percentages shown in table 6.1.

It is therefore reasonable to assume that the vast majority of the audience, not knowing the highly specific knowledge required and not having time to consult a suitable information source, attempted some intuitive interpretation of the information contained in the question : *"Which world ocean is 5.5 million square miles in size?"* While it is impossible to ascertain exactly what this collective informed intuitive reasoning actually was, a reasonable suggestion, given the statistically significant bias, is that the vast amount of the studio audience members interpreted 5.5 million square miles as a lot of water and therefore intuitively assumed or rationalised that this very large amount of water must represent one of the two largest oceans in the world (i.e. either the Pacific or the Atlantic). This would at least go a long way to explaining the significant bias (i.e. 85%) of the studio audience answering either the Pacific or Atlantic oceans.

The fact that the collective intuitive view was completely wrong (all of the other three oceans had much larger water volumes) indicates that not only can peo-

ple make collective assumptions when knowledge is lacking, but the particular knowledge context in which such assumptions are made can sometimes result in collective biased judgements and interpretations that are completely incorrect.

This section has attempted to show that the context plays a significant role in individuals making certain assumptions. Not only this, but within a context of understanding, collective assumptions are often prevalent. While, as indicated earlier with enthymatic reasoning, this can facilitate communication and understanding, such collective assumptions can be the source of problems and error. This is particularly true if: a) the context undergoes change — resulting in long–established contextual assumptions becoming invalidated (i.e. consumer goods organisations in the 1970s); b) when different contexts are overlapping (i.e. American Airlines Flight 965); or c) when contextual knowledge is lacking (i.e. game show question) and people base decisions upon knowledge bounded intuitions to *'fill-in'* those gaps.

6.6 Chapter Summary

In this chapter the issue of assumptions has been discussed. It can be seen that assumptions can occur from many causes. The manner in which we reason can often be subject to assumptions — in fact making assumptions are necessary in many circumstances if we are to avoid needless repetition and explanation which would stifle communication and activity. Furthermore, assumptions are useful mechanisms in progressing a line of thinking or reasoning in both a formal and informal reasoning manner. However, it is important to identify assumptions in certain situations as these are often unspoken, implicit, and unquestioned limitations placed upon our thinking and reasoning which can have potentially adverse affects. This is particularly true when such assumptions represent part of our belief systems. In such instances unidentified assumptions can act as serious constraints and artificial limits upon our thinking episodes. It has also been recognised that the context, while often providing additional information cues that help us identify assumptions, can also result in collective or shared assumptions being made that can become invalidated over time, cause confusion when contexts overlap,

or can result in a kind of flawed *'group–think'* type paradigm when knowledge is bounded or knowledge acquisition is difficult. It can be seen from the issues and computer–based system examples given in this chapter that, broadly, assumptions could be categorised into:-

- **Implicit Assumptions:** in the form of unconscious constraints and limits placed upon our representation and reasoning about something;
- **Explicit Assumptions:** in the form of conscious reasoning mechanisms that allow us to progress a particular line of reasoning — either formally or informally;
- **Shared Assumptions:** in the form of those made collectively in a unquestioned manner — usually from some common context–of–interest;
- **Invalidated Assumptions:** in the form of either unconscious constraints or conscious reasoning in an individual or shared manner that, although are originally viable when made, can become invalidated as circumstances, demands and/or influences change over time.

Undetected assumptions can result in either inconsistent or incomplete comprehension of the true influences that pertain in any particular situation. Detecting assumptions then, is critical if greater dependability coverage of computer–based systems is to be achieved as such shortfalls of reasoning, thinking, and problem–solving have potentially major implications when our thinking, reasoning, and problem–solving becomes a pre–requisite activity to creating dependable computer–based systems.

Chapter 7

Purpose and Function

7.1 Chapter Introduction

In this chapter the issues of goal-setting are raised. Developing computer–based systems is a definite purposeful act. Artifacts embody the goals of those who conceive, develop, and deploy them. However, the issue of goal–directedness in this activity is very important when we consider the interrelatedness of system structures. Such systems often need to promote many goals, many of which, may not be promoted — or adequately promoted, during development.

This chapter first considers the controversial area of teleology and related causation before focusing upon the benefits and concerns raised by goal-setting and multiple goals.

7.2 Teleology

Teleology is a deep philosophical area which requires a greater coverage than is possible to provide in this chapter. In this section, however, the underlying philosophical topic of teleology is discussed to provide a theoretical background to the later associated issues of purpose, function ascription, and the psychological effects of goal-setting upon human task performance. In this section the origins of teleology, the reasons for its scientific rejection, and the types of teleology recognised from more contemporary theories is covered.

7.2.1 Origins

The origins of teleology date back to ancient Greek philosophy. The term *'telos'* means an end or goal, and the complete term *'teleology'* means to be end directed. The two philosophers, from ancient Greece, who established its essential foundations were Plato and Aristotle.

Although the foundational principles of teleology, between the two founders, are essentially consistent they, do depart in significant ways with regards to determining how any given entity's end state is conceived and achieved. Platonic concep-

tions of teleology relates to *'external'* teleology [146]. With external teleology the ends to be satisfied (or desired to be attained) will be accomplished by some external agent essentially outside the system–of–interest that is to be created, designed, or modified. The value or utility envisaged in attaining the desired goal is also subjectively determined from that external agent's perspective. Although Plato's own usage for this definition was to characterise a divine artisan who created the universe, it can be utilised to mean any creative act that results in an artifact. External teleology captures the notion of a creative entity being motivated into action and producing pre–existing notions and ideas of what the results of the creative act will (or should) be.[1]

By contrast, Aristotle, while acknowledging the relevance of Plato's teleological conceptions, was primarily interested in the biology of living things [147]. As a direct consequence, he preferred to extend teleological interpretations to accommodate an *'internal'* conception also [146]. With Aristotle's conception, the ends or goals belong to the entity or system itself and not those of any external agent. Additionally, any notion of value, good, or utility of attaining the desired end or goal is to be determined from the perspective of the entity or system itself. With such a conception, the purposiveness is subsumed, in any thinking or reasoning manner, and there need not be any pre-existing notions or ideas of the end or goal that will (or should) result. Instead, such an end or goal state is provided by the internal processes or structures of the entity or system itself. Delbruck [148] claims that Aristotle's principle of an internal *'unmoved mover'* that provides for change to an end or goal state without ever changing itself is probably one of the greatest conceptual innovations as it perfectly describes DNA that acts, creates, and develops a living organism — and yet remains itself unchanged in the process. The very existence of DNA, however, was not to be discovered until well into the 20th century.

In comparing the two originating conceptions of teleology McLaughlin [[146]: p. 17] notes:

[1] Such original teleological ideas are fundamentally related to a formal causation cf. subsection 7.2.4.

> *"The reality of external teleology can scarcely be denied: the reality of internal teleology is what is really at issue."*

Mayr [147] explains that due to Aristotle's essential interest in biological systems he, in error, over extended teleological conceptions to include the non–living world, and, in part, it was this that subsequently resulted in teleological interpretations of scientific phenomena being vehemently rejected during the scientific revolution of the 17th, 18th and 19th centuries.

7.2.2 Rejection of Teleological Explanations

In addition to teleological explanations being rejected through Aristotelian attempts to explain non–living phenomena, teleology, by the 17th century, was intrinsically linked with vitalistic explanations that provided religious and metaphysical theories to explain nature [149]. This was due to much earlier attempts by the Catholic church to provide Christian interpretations of Aristotle's teleological philosophies (particularly idealistic causation of finalistic and formal causes see subsection 7.2.4). This resulted in reinforcing rejection by the scientific communities (in particular by prominent natural philosophers such as Francis Bacon and Renae Descartes).

By the late 17th century, mechanism had become the dominant scientific paradigm championed by (Sir) Isaac Newton that described the universe in purely physical causation terms. This would subsequently be further reinforced in biology by Charles Darwin in the 18th century that portrayed the origin of all life on earth in terms of purely materialistic causation terms of natural selection and adaptation. Both of these cases alluded to only explain *"how"* the universe and life on earth worked using materialistic and efficient causation explanations. This is in contrast to *"why"* explanations that would require reference to finalistic and formal causation, which, by then, had become scientifically discredited.

It was not until the early 20th century with the advent of relativistic and sub–atomic physics discoveries that the limits of the mechanistic world–view would

became exposed and questioned.

7.2.3 Types of Teleological Processes

Despite these shortfalls in teleological explanations biologists have long argued for the heuristic and empirical merits of posing teleological questions of the *"why"* persuasion. It has been therefore widely recognised in philosophy that in order to meaningfully apply teleological explanations, a careful categorisation of valid teleological definitions are necessary to avoid the confusion presented when teleological explanations unwittingly *'criss–cross'* issues of vitalism, holism, and reductionism.

To begin with Mayr [[147]: pp. 19–20] argues that such definitions must be immune to any of the following objections:

- Teleological statements and explanations must not imply the endorsement of unverifiable theological or metaphysical doctrines in science;
- Explanations for biological phenomena that are not equally applicable to inanimate nature must not constitute a rejection of a physiochemical explanation.
- The assumption that future goals were the cause of current events must not seem to be in complete conflict with any concept of causality.

To uphold these criteria Mayr [147] introduces two definitions of teleological processes.

The first is termed ***Teleomatic*** processes. Teleomatic processes are processes where causation is simply the consequence of natural laws. One such example is where gravity provides the natural end state for a rock that is dropped into a well. Another is where a heated bar reaches its natural end state (under thermodynamic laws) when its temperature with the prevailing environment reaches the

point of equalibrium. In teleomatic processes, systems and entities, the end directed causation is only in a passive manner — controlled deterministically by external forces and boundary conditions. In this sense it is an automatic process.

The second is termed ***Teleonomic*** processes. With Teleonomic processes the behaviour of the entity owes its causation towards a definite end or goal state to the execution of some internal program that can both anticipate the desired end or goal state and can regulate the organism's executive mechanisms and functions towards achieving that end or goal state. An obvious example would be the genetic DNA coding of a hen's egg that will transform into a chicken (and ultimately, in time, into a mature hen). Another example is where a computer's electronic circuitry and hardware devices acts in a determined manner when it is provided with the appropriate programmed instructions (i.e. software). An interesting extension to this definition is also to note that the internal program may be a *'closed'* or *'open'* program. For instance, the hen's egg represents a closed genetic program that determines the ultimate end or goal state (i.e. a chicken/hen) whilst the encoded computer program may be either a closed or an open program. It is closed if it's program does not need or cannot acquire other information. Alternatively, if it requires certain information input parameters, or is capable of intaking information, then it is an open program. Mayr notes that most behaviour of higher organisms is controlled by open programs which require or can incorporate additional input information in the form of learning, conditioning, experience, etc. Once such open programs have been *'filled-in'* with additional information it then becomes the equivalent to the closed program in its regulation of teleonomic behaviour.

An important point to note with these two definitions (and one that will be made more clear in the discussion to follow below) is that in both teleological definitions no reference can be made to an intentional purposeful act. The rock never intended to fall down to the bottom the well, the metal bar did not intend to be the same temperature as the room, the hen's egg never intended to become a chicken/hen, and (say the closed) computer program never intended to print on the screen *"Hello World"*. Such statements are silly and inappropriate teleologically for these type of processes. As Rosenblueth et al [[150]: p. 19] noted in

their cybernetic considerations of teleology:

> *"The basis of the concept of purpose is awareness of "voluntary activity." Now, the purpose of voluntary acts is not a matter of arbitrary interpretation but a physiological fact. When we perform a voluntary action what we select voluntarily is a specific purpose, not a specific movement or act. Thus, if we decide to take a glass containing water and carry it to our mouth we do not command certain muscles to contract to a certain degree and in a certain sequence; we merely trip the purpose and the reaction follows automatically."*

What we can see from the examples of teleomatic and teleonomic behaviour is that, whilst end or goal directed, in every case there was no voluntary behaviour present. The systems in the examples had no control of the process and therefore had no freedom to choose what acts will be performed. Therefore, these examples of teleomatic and teleonomic processes, discussed by Mayr [147], represent non-intentional forms of teleological processes.

McLauglin [146] includes, but also, extends Mayr's [147] teleological definitions by including into the definitions: a) classifications of intentional and non-intentional processes; b) making distinctions between teleological processes and teleological entities (in terms of artifacts, organisms, traits, and human institutions); and c) by providing subclassifications within these classifications and distinctions (i.e. a and b). He does this by providing a range of (considered) grammatically valid teleological statements, as follows:

1. The man ran in order to catch the train;
2. The cat opened the door in order to get the cream;
3. The wasp hunts bees in order to feed its larvae;
4. The function of the thermostat in the furnace is to keep the water from going above a certain temperature — and thus to help it provide steady heat;

	Processes	Entities
Intentional	(1) Human Actions (2) The Behaviour of Higher Animals	(7) Simple Artifacts (4) Parts of Complex Artifacts
Non-intentional	(3) Behaviour of Lower Animals (8) Organic Development (9) Historical Chiliasm	(10) Invertebrate Artifacts (6) Biological Traits (5) Social Institutions and Cultural Practices

Table 7.1: Teleological Classifications [source:[146]: p. 38]

5. The (latent) function of witchcraft persecutions among the Navaho's is to lower intra-group hostility;

6. The function of the heart is to circulate the blood;

7. The purpose (or function) of knives is to cut;

8. The cell became specialised in order to develop into a lung;

9. The Second Dutch War was necessary in order for England to become a world power;

10. Spiders spin webs in order to catch food.

The above teleological statements of McLauglin are entered into their appropriate teleological categories in table 7.1.

Examples (1) and (2) represent the classification of intentional teleological processes. This category deals directly with *final* causation in terms of some aspect of mental representation of a desired end or goal state is responsible (i.e. a cause) for the current human or animal behaviour (i.e. the effect) of running for the train or opening of the door. The two important considerations are: firstly, future anticipated goals are the cause of present events (i.e. running and opening) which runs in reverse to the traditionally accepted sequence of causality (i.e. the effect precedes the cause instead of the cause preceding the effect); secondly, both the man and the cat demonstrate voluntary behaviour (i.e. the man chose to run for the train, and the cat chose to open the door) which, as discussed earlier, is essential to ascribe consciously purposeful actions or activity.

Examples (4) and (7) represent the classification of intentional teleological entities. This category is particularly interesting because purposeful ascriptions attributed to artifacts blurs the distinctions between *final* and *formal* causation depending upon the complexity of the artifact and the context in which it occurs. For instance, if we consider first the simpler artifact of the knife in example (7) we can see that the functional identity of a knife can easily have many possible *"functional propensities"* [cf. McLaughlin [146]: p. 51]. It can be used (as it was designed for) as an implement for cutting. However, it could also be used as a screwdriver, a wall scraper, a lever, etc, etc. This multi–functional ascription is particularly common for simple artifacts where the function is completely externalised in a holistic way. The possible use functionalities that a simple artifact like a knife could be put to also introduces interesting issues surrounding *final* and *formal* causation. We can see that functional ascription with the knife relies on no more than a formal representation of conceiving that particular function of the knife to perform a different function without ever having to redesign or modify it at all. This is what McLaughlin terms a *"virtual"* functional ascription . How well the knife performs when ascribed other functional ascriptions — other than which it was designed brings into focus the tension (and possible mismatch) between its intended designed purpose and its (now new) ascribed usage function. For example using the knife as a lever may result in very poor performance (and be potentially dangerous). Here we can see that with such simple artifacts the final causation — in terms of the functional goal (or purpose) for which it was conceived, created, and designed (i.e. to cut with), in no conceptual way, restricts a re–conception of how it could be used. The fact that this, by some, may be judged to be a misuse of the artifact is a side–issue when considering teleological explanations. This led McLauglin [[146]: p 206] to concede:

> *"....functions of artifacts are, in the last analyses, based on mental events: beliefs and pro attitudes...Paradigmatically, we actually design and make the artifacts, the artifacts actually have the effects intended, and the effects are beneficial as expected. However, none of these need necessarily be the case for an item to have an artefactual function."*

The point McLaughlin wished to make with this comment is that with simple artifacts we can change the purpose and function relationship of the artifact as quickly as we can change our mental representation (i.e. formal cause) irrespective of its originally intended function (i.e. final cause) for which it was designed or created. With complex artifacts like example (4) however, the functional identity of the artifact is slightly more objective and identifiable. This is because, unlike simple artifacts, its functional contribution is both highly specialised and internalised within a broader system boundary. With simple artifacts the function can be changed as quickly as our intentionality towards it, without necessarily any structural change of the artifact needing to take place. Whilst this may also be the case with systemically intentionalised artifacts, it is so, to a lesser extent. However, it is still quite possible, even with complex artifacts, to ascribe a different purpose to an already existing function — in order to derive latent functionalities (in terms of intended design). A good example of this, from a computer–based system perspective, was given in chapter 5 (section 5.2.2) as issue 5 (Obscure security flaw conflict). Here we can appreciate that although the intended designed (i.e. manifest) function is to ensure an authorisation function for an ATM user via PIN code while allowing at all times a complete undo facility. The facility of allowing the ATM user to completely undo the transaction without recording any failed PIN code authentication attempts actually leaves open the possibility for a fraudster to completely enumerate a targetted ATM customer's PIN code. Therefore, while the design intention is to provide a flexible authentication PIN code function which promotes the well established usability principle of always allowing the user to undo a transaction, the unintended freedom within the designed function allows a fraudster with the purpose of gaining illegal access to an ATM customer's account, to ascribe an additional (i.e. latent) function to this PIN code authentication function as having the additional functionality of a complete PIN code enumerator. This ability to ascribe latent functionalities to manifest functions that they are not intended or designed to satisfy is at the heart of many security vulnerabilities in computer–based systems and further reinforces the need for a wider holistic and integrative consideration of both the technical and human systems as subsystems–of–interest within a computer–based system perspective.

Examples (3) (8) and (9) represent the classification of non-intentional teleological processes. Examples (3) and (8) are cases of teleonomic teleological explanations discussed earlier. This is where the end or goal states are determined by an internal program execution. Example (3) is an interesting distinction between examples (1) and (2) mentioned earlier. By contrast to examples (1) and (2), example (3) represents a much lower order of organism sophistication. Whereas the man and the cat represents voluntary purposive intelligent behaviour (i.e. intentional), example (3) of the wasp represents a limited level of internal program execution that constitutes involuntary instinctive behaviour (i.e. non-intentional). Example (8) is also teleonomic, but this time, represents an internal (and closed) program execution of a physiological nature. Example (9) is more of a teleomatic process. Teleological explanations can only be offered with the benefit of hindsight that then reveal the external material causation that the Second Dutch War had subsequently upon the end state of England's world position.

Examples (5) (6) and (10) represent the classification of non-intentional teleological entities. Example (5) can be very confusing as organisations are often thought of as having desired objectives and goals, etc. In this case it is, at first glance, very appealing to assert that organisational structures have intentionality. However, what is of interest here in this classification is the latent functional causation that cultural practices have upon the organisation itself. What must be understood, however, is that the cultural climate of the organisation or institution may not be in alignment with the formal or primary intentions of the organisation. Classic examples include a blame culture climate that can lead to emergent (and unintentional) behaviours of the organisation that are in direct conflict with its formal or primary goals (e.g. a blame culture in an organisation that requires high levels of safety). As will be seen in chapter 8 where the ATM case study issues of chapter 5 are analysed within a more suitable computer–based system dependability representation, assumptive reasoning involved in promoting one particular goal can often result in unintentional and undesirable interdependencies that compromise and undermine the promotion of other goals. When this occurs, during the creation process, the compromised goals in the eventual artifact can also be understood in terms on non–intentional teleological entities and reinforces the need

for a more holistic and integrative representation to help prevent this occurence during the creation process.

7.2.4 The Four Causes

As can be seen from the previous sections, causation is inherently related to issues of teleology. In the following subsections Aristotle's four types of causation are considered. Despite the scientific criticisms of Aristotle's views of teleology, his ideas of causation are more complex and sophisticated than at first glance. Aristotle identified that causation could take any one (or more) of four essential forms:

1. Final causation:
2. Formal causation:
3. Material causation:
4. Efficient causation:

The textbook example often used is that of building a house that illuminates the roles of each causation factor in reaching or achieving a required or desired end–state (or goal).[2] The *material cause* in house building relates to the stone, brick, timber and tiles that go to make up the physical structuring of the house. This includes the tools, methods, and techniques employed in putting the actual materials together in such a manner as to construct a house. The *efficient cause* represents the labourers, joiners, bricklayers, and plumbers involved in the actual construction. The *formal cause* is not as clearly categorised, however. As mentioned earlier, Aristotle's conception of the *'unmoving mover'*, recognised that there is a necessity for abstract intervention that precedes actual attainment or achievement of the intended goal. The *formal* cause represents some preceding conception or control of what is to be created. In this sense the *formal* cause relates to a blueprint or model of what is to be built, and in the example of building a house, the nearest role that fulfills this causation is that of the architect who first designs and refines

[2]Whether this is intentional or unintentional.

Causal Type	House Building	Military Battle
Material	Stone, bricks, mortar, etc	Guns, swords, tanks, etc
Efficient	Labourers, joiners, electricians	Soldiers
Formal	Plans, drawings, models, etc	Military strategy, etc
Final	Need / purpose for house	Political / economic reasons etc

Table 7.2: Aristotelian Causal Typology Example [source: [149]: p. 12]

a model, plan, scheme, or blue print of what the house will look like etc. The *final cause* represents the purpose for wanting to build the house in the first place. This could be for any number of utility reasons such as shelter, luxury, necessity, commercial gain, or social reasons.

Ulanowicz [149] provides another example which he believes overcomes some of the blurring between *formal* and *final* causation. Ulanowicz uses a military campaign as an example. In this case, the *material cause* represents the weapons and ordnance used by each side. The soldiers fighting on the battle field represent the *efficient cause.* The *final cause* relates to the broad reasons why each side became involved in the conflict to begin with (i.e. imperial ambitions, economic, etc). Ulanowicz argues that it is the generals who provide the *formal* causation, through strategic military planning and decision making that influences not only the shape of the battle as it unfolds but can affect the success or failure of the eventual outcome of the whole endeavour.

Table 7.2 captures the two examples of causation. Ulanowicz goes on to point out that Aristotelean notion of causalities is hierarchical — in that all the causal forms participate at different levels and influence the eventual outcomes at different scales. For example, the generals enact an immediate influence over the success of the entire campaign through the quality of their experience, intuition, and foresight (i.e. *formal* cause) in strategic military planning. The soldiers (i.e. *efficient* cause) also influence the campaign, but only on a smaller subfield scale. The same could be said for the quality and quantity of guns, ordnance, etc. Heads of state (i.e. *final* cause), on the other hand, exert an influence that goes way beyond the issues of the campaign itself.

It is pointed out, by Ulanowicz, that although *material* and *efficient* causes tend to exert their influence at a subset or subfield level, their effects, if severe enough, can propagate up the scales in some cases. Furthermore, it is noted that the formal cause acts at the *"focal"* level of observation and its alignment with that of the *final* cause is the most crucial in determining successful attainment of the end goal.

7.3 Goal–Direction

In the preceding sections teleological and causation issues have been considered. It can be seen that designing and developing dependable computer–based systems represents a straightforward external and intentional teleological activity. In this activity the goals and purposes of the computer–based system artifacts are what we, as their creators, envisage and embed into them. However, although true, this is not so clear as it may at first seem. A dependable computer–based system artifact must be reliable, secure, usable and maintainable. All of these well known attributes represent goals of the artifact, yet, firstly, these goals cannot be directly promoted in any structural or architectual sense, and, secondly, this means that a single artifact may need to contain, embody, and promote many goals. Furthermore, how can we assure that such goals are acceptably represented in the first place? These issues are raised and discussed in the following two subsections.

7.3.1 Single Goals

The influence of single goals on human performance has been extensively studied by industrial psychologists over nearly four decades. In a comprehensive survey of experimental findings on goal-setting effects upon human performance Locke & Latham [151] argue that the findings represent one of the most reliable and replicable areas of psychology.

Goal-setting research focuses on the relationships that consciously set task goals have upon human task performance. The core findings over the years concerns the relationships that exist between the difficulty of the goal set and the specificity. Difficult goals regularly produce more effort — however this levels off

once an individual reaches the limits of their ability. Specific goals set often lead to higher human performance than *"do your best goals"* and specific and difficult goals consistently produced higher human task performance and reduced variability between human performance.

Four positive mechanisms related to goal-setting have been consistently found during experimentation. Firstly, goal-setting provides a directive function through focusing activity and attention towards goal relevant activities and away from goal irrelevant activities. Locke and Latham noted that this mechanism of goal–setting influences at both the behavioural and cognitive level. Secondly, goal–setting provides an energizing function. Higher goals tend to motivate people to employ greater effort than lower performance goals. Thirdly, goal-setting influences persistence towards achieving the goal. It has been observed a number of times during experimentation that when participants control the time they can spend on a task, they spend longer on the task when explicit goals were set. Finally, goal-setting promotes an arousal and discovery function. It has been found on a wide range of tasks that setting goals stimulate the thinking process resulting in them searching out task relevant knowledge and information appropriate to achieving the goal. When individuals are confronted with a task they will automatically employ knowledge, information, and skills directly relevant to attaining the goal. In task situations which require knowledge, information or skills that people do not already posses then they will deliberately draw upon previous experience they deem relevant to that particular goal–setting context and apply it. In completely novel goal-setting situations where they do not possess even relevant previous experience then they will deliberately plan and develop new task strategies that will enable them to achieve the goal.

Despite these very positive and longstanding findings of goal–setting influences upon human performance, Locke and Latham did find a number of moderating effects upon goal–setting research. Firstly, is the issue of the particular individual(s) natural commitment towards achieving the goal. Factors that were considered critical include:

- **The importance of the goal itself.** Locke and Latham noted that public

announcement of commitment can often increase perceptions of importance of the goal to an individual;

- **Leadership can often increase commitment.** In particular Locke and Latham emphasised the importance of increasing confidence in a persons ability to be capable of achieving the goal;
- **Self–assignment and participative assignment.** These were also considered important in getting commitment from an individual;
- **Reasoned rationale.** The wider contextual importance of being able to achieve the goal has also been found to instill commitment.

Secondly, feedback upon performance, in terms of progress made is also an important consideration. Locke and Latham recognise that goal–setting, by nature, is a discrepancy creating process that requires careful feedback on task progress. Lastly, the intrinsic complexity of the task itself can act as a moderating influence. As complexity of the task increases higher levels of skills, experience, and knowledge is demanded whereby goal–attainment becomes dependent upon the individual's ability to plan and seek out new task strategies. Locke and Latham report on two important aspects of goal–attainment with intrinsically complex tasks. One aspect relates to assigning only *learning* type goals rather than definite or specific outcome goals when the nature of the task is inherently complex. When learning goals are set on complex tasks it has been found that performance is significantly higher. The second aspect relates to the careful structuring and alignment of proximal (sub) goals with distal (overall) goals. Studies on goal–setting on complex tasks have shown significant performance increases when overall distal and sub proximal goals are intelligently structured over just a distal or *"do your best"* generic goal

7.3.2 Multiple Goals

While goal–setting research has provided robust human performance increases it must be remembered that the vast majority of the research involved tasks with only one explicit goal to be achieved. The development and deployment of a

dependable computer–based system, on the other hand, involves an artifact and context which introduces and requires many goals to be simultaneously satisfied and attained. This introduces issues of multiple goal attainment.

To begin with, multiple goal attainment introduces issues of cognitive limits in terms of ability to focus and have sufficient knowledge, skills, and experience to consider many goal–related issues when performing a task. This was the issue Shallice [152] considered in developing and analysing a cognitive model of consciousness based upon previous psychological research. His resultant analysis argued that only one goal, plan. or mental scheme can be maximally activated at any one time during cognitive reasoning. This suggests that, in situations where multiple goals need to be promoted and attained, a single individual will be (at worst) oblivious to the non–activated goal attainment issues or (at best) be incapable of simultaneously providing sufficient consideration of important multiple goal–related issues. This theory reinforces Weinberg and Schullman's [153] findings of multiple goal–setting influences upon human performance in computer programming. In a number of studies to attempt to understand the huge performance variations in computer programming, Weinberg and Schullman set industry programmers primary and secondary quality and productivity goals on a number of programming tasks. What they found was that programmers tended to focus exclusively on their primary goal while treating secondary (or unstated) goals as *"free variables"* to be traded–off in pursuing their main goal.

Within a system development context multiple goals introduce a problematic interdependency problem where in promoting or focusing mainly/solely upon one goal, relationships between goals can be subtly created. The nature of these relationships can be of one of three types:

1. **Complementary.** This is where promoting one goal inadvertently also reinforces or promotes another related goal within the system;

2. **Compatible.** This is where promoting one goal inadvertently does neither promote nor compromise another related goal within the system;

3. **Conflicting.** This is where, in promoting one goal, another related goal within the system is inadvertently compromised.

It can be seen from these three inter–goal relationships that conflicting interdependencies are of a major concern in compromising the system. This is especially true if it occurs or results without detection.

7.4 Chapter Summary

The development of computer–based systems is an intentional teleological activity whereby the creators who conceive and construct the functionality do so purposefully. Artifacts therefore embody the goals of their creators — even though, in the case of simple artifacts, their purpose context can be altered merely by the conception of the user. In the case of more complex artifacts, such as computer–based systems, this is much less likely to occur, but can be at the core of many security vulnerabilities etc when it does. A particular problem with such complex artifacts as computer–based systems is that they require many goals to be promoted simultaneously — such as maintainability, reliability, security etc. Despite the fact the goal–setting research has shown that setting individuals explicit conscious goals can have a significant positive effect upon human performance (in both behaviour and cognition), it is also recognised that, with such complex artifacts, people cannot be expected to have the knowledge, experience, skills, or cognitive capacities to consider all possible inter–goal relationships that can occur in multi–goal promotional situations. If representation and promotion of multiple goals is to be achieved, therefore, some other way of resourcing and organising development teams will need to be considered. In the next chapter the ATM issues from chapter 5 will be revisited and reformatted to show how by setting different developers different goals can help unearth conflicting inter–goal relationships that can undermine computer–based system dependability. Furthermore, it will be shown that often these relationships occur because of some assumptions that are made about the goal being promoted and/or context being considered.

Chapter 8

Discussion of a Goal–Diversity Process Intervention

8.1 Chapter Introduction

In this chapter considerations for a process intervention that attempts to enhance computer–based system dependability via improved assumption detection by means of forcing diversity into a development team through diverse goal–setting is discussed. Section 8.2 provides a justification for considering the setting of diverse goals to attempt to achieve diversity. This is done by drawing upon the literature and arguments already presented in the preceding chapters of the thesis. Section 8.3 then introduces and justifies the usage of an already established notation that employes a non–functional analysis and synthesis of computerised systems. Lastly, in section 8.4.2 this notation is then utilised, in its slightly adapted form, to provide an insight into the expected benefits a goal–diversity process intervention may yield in obtaining a greater assumption detection coverage.

8.2 A Goal–Diversity Process Intervention

Chapter 3 provided an initial view of the important attributes one might expect in a dependable software creation process. From this viewpoint, it is possible to begin to consider: a) the inter–related dynamics of process technology inputs (e.g. tools, methods, and techniques), human resource inputs, the nature of the software creation task, the particular application domain, and overall process management; and b) how latent and active fault–phenomenologies can occur in both the creation process and its immediate process environment — as threats to achieving a dependable process. Viewing the creation process as a system–of–interest in its own right, it is possible to begin to consider how this creating system could utilize these elements to increase its fault–tolerance in avoiding the introduction of faults into the system it creates. As chapter 2 highlighted, the employment of redundancy and diversity into a software artifact has provided substantial increases in dependability in the presence of residual software faults, following this lead then, it is reasonable to consider how process redundancy can also aid dependability increases in tolerating imperfect creation process elements to increase avoidance of faults into the software artifact. Focusing upon human resources, it can be seen from chapter 3 that the employment of human redundancy to achieve human

diversity to increase fault–avoidance already exists in many guises. Examples provided in chapter 3 include: i) ad–hoc or natural human diversity approaches — such as pair–programming, egoless programming, and, more recently, open source development; ii) forced–diversity approaches — whereby diverse process technology is applied to aid fault–avoidance; and iii) to a lesser extent, composed diversity approaches — whereby diverse human resources are carefully composed into groups and teams based upon some uncontrollable psychological dimension such as personality or culture, etc.

Improving the dependability of the software creation process through some diverse human resource process intervention is further complicated, however, when considerations of a computer–based system are to be included. As discussed in chapter 4, a computer–based system perspective expands the system boundaries outwards to include the technical computer system and its interacting human system(s) as subsystems–of–interest. This wider systemic view is more complex as it becomes clear that sociological, organisational, and situational influences can combine to result in judgements of undependability for strictly non–technical reasons. Such examples, considered in chapter 4 include where notions of system purpose of the system vary in potentially conflicting ways in contextual areas of responsibility, motivations, and values etc. To help unearth such differing system purpose perspectives and judgements it is necessary to recognise that computer–based systems require a higher holistic and integrative understanding of dependability from important, but often conflicting, stakeholder contexts–of–interest. For this reason four generic contexts–of–interest of: a) the utility context; b) the engineering context; c) the deployment context; and d) the evolution context were devised as a guiding integrative understanding framework for computer–based system conceptions. When this computer–based system conception was applied to understanding a number of reported failures in the long–standing domain Automatic Teller Machine (ATM), it was indicated that many of the flaws, vulnerabilities, etc resulted from various assumptions being made.

In reviewing the literature on assumptions in chapter 6 and relating this literature to the ATM case study in chapter 5, four broad categories of assumptions can

be appreciated. These are: i) Implicit assumptions; ii) Explicit assumptions; iii) Shared assumptions; and iv) Invalidated assumptions. It was highlighted in the literature of assumptions in chapter 6 that often assumption identification can be extremely difficult and requires a significant level of conflict, challenging, and tension to detect them. Therefore a particular question of the thesis was: *"What form of human diversity process intervention could help improve assumption detection coverage during software creation?"* In answering this question diverse goal–setting was accepted for a number of reinforcing reasons. Firstly, from the literature on goal–setting in section 7.3.1 in chapter 7, industrial psychologists have found, over the last 30 years of studies, that goal–setting represents one of the most robust and replicable ways of increasing human task performance. Secondly, from both a computer–based system and psychological research perspective, goal–setting influences and focuses human cognition and behaviour — effecting peoples' values, reasoning, and priorities, which is considered crucial for unearthing different stakeholders' notions of purpose (and the underlying assumption set supporting them). Thirdly, because of these cognitive influences, setting different goals offers a more practical, feasible, and controllable way for an organisation to employ human redundancy/diversity than other less controllable forms of (say) composed diversity. Fourthly, because of these practicalities of goal–setting, and the fact that software development is a clear case of external teleology (cf. section 7.2.1), satisfactory levels of dependability representations, during the creation process, can be promoted. Finally, different goals inevitably result in higher levels of conflict, challenges, and tension. Diverse goal–setting therefore provides the necessary task climate for helping unearth assumptions.

A more detailed and specific example of the stages and assumption identification benefits of the proposed goal–diversity process intervention is discussed and exemplified in section 8.4.

8.3 Non–Functional Notation

In this section the nature of non–functional attributes will first be discussed in section 8.3.1. It will be noted that non–functional attributes have dependability

consequences for both the creation process and the created artifact. Next, a well established non–functional requirements framework will be introduced in section 8.3.2 before its merits and suitability of application to demonstrate the anticipated benefits of a goal–diversity process intervention (documented and illustrated in section 8.4.2) is provided in subsection 8.3.3. Lastly, some important differences between the established non–functional framework and the goal–diverse process intervention is discussed in subsection 8.3.4.

8.3.1 Non–Functional Attributes

In this subsection the nature of non–functional attributes is briefly discussed. However, before doing so, let's provide a suitable context via a quote from Malan and Bredemeyer [116] [p. 2] that is (perhaps) all too similar and frequent in many large–scale software development projects:-

> *"One development team, being close to its function–complete checkpoint, was frantically scrambling to meet benchmark targets that the marketing team was just then putting together for system test. An architecture assessment revealed that some of these quality requirements could not be met by the current architecture without significant rework. This problem of attempting to work quality in at the end of the development phase has been around as long as we have been doing software development...Another team started out with the goal of creating a system that would satisfy current user requirements and provide the basis for quickly developing other applications. After putting the engineers through object–oriented training and spending months on analysis and design, the project started to feel the pressure of the impending release date. As this pressure intensified, design reviews and code inspections were scuttled and key architects and engineers left the team disgruntled by the long workdays and corruption of the vision of creating a reliable, extensible and evolvable system that would solve the development pressure problem in future releases. More and more engineers were added to the team to make up for this attrition. Under this pathological cycle, the design degenerated and*

was going to be pretty much abandoned. Quality problems emerged and excalated out of control. Somehow, through sheer heriocs on the part of the engineers, the application was eventually released. It met the critical customer requirements, but came nowhere close to the organisation's goal of reducing the time–to–market of follow–on releases...Simply put, either the non–functional requirements were not specified (in time), or compromised without explicit attention to the trade–offs involved. Not paying attention to eliciting, documenting and tracking non–functional requirements makes it harder to prioritize and make trade–offs between quality of the product, the cost to develop and enhance it, and the time–to–market of current and future releases. Without quality targets to guide the architects and engineers, design choices are arbitrary, and it is hard to assess the system during architecture and design reviews and system test."

This quotation is all too reminiscent of chapter 3 in that it demonstrates quite clearly how the software creation process is made up of a complex interrelated set of dynamic influences that can cyclically undermine dependability.[1] More importantly, and in specific relation to non–functional attributes, while non–functional attributes are often thought of as desirable properties of the created software artifact, it can be appreciated that the explicit presence of non–functional attributes are not only important in providing a dependable software artifact, but their presence in driving the creation process is critical to promoting a dependable process. In the context of this thesis, it is their presence for driving the requirements and

[1]Note in particular :- 1) How loss of *process controllability*, as a desirable process attribute, can act as a latent process environment error that then further introduces faults into the artifact through fatigue and monotony of heavy overtime working to accelerate project schedules; 2) How subsequent reduction of fault detection of the process then occurs via violations of important design reviews and code inspection stages; 3) How the combination of these two influences then result in further loss of process control due to unnecessary rework; 4) Also note how desirable process attribute of *technology applicability* was undermined — in the form of not knowing the necessary methods (object–oriented) within the creation process also led to a loss of process control — through the project schedule then having to accomodate learning (which few project managers ever include within the work scope during planning); 5) Lastly, note the lack of desirable process attribute of *human performance predictability* — in that after losing project schedule control the work remaining could not be factored out for schedule acceleration by increasing the number of engineers involved in the software creation task.

design decision–making of the software creation process that is primarily focused upon. In this respect the thesis is focused upon process–oriented non–functional attributes in a qualitative reasoning manner, rather than product–oriented non–functional attributes in a quantitative verificational way.

Functional requirements relate to *'what'* the system does in terms of transformational behaviour of how inputs are processed into outputs and can be further broken down into states and structural elements that enable or constrain those states. By contrast non–functional requirements are holistic integrative properties of the entire system and are therefore often referred to as properties or attributes of the system as a whole. Locally satisfied non–functional properties of a subsystem may not mean globally optimal assurance of that non–functional property or attribute in the entire system–of–interest as there exists the potential for many subtle interdependencies (as discussed in chapter 7) between distinct parts and subsystems that can, when combined, result in conflicting or antagonistic relationships between them that result in an emergent compromisation of various involved non–functional attributes.

When viewed in a qualitative process–oriented perspective, it can be appreciated that non–functional attributes have three fundamental characteristics, as follows [cf. Chung et al [154]]:-

1. **Subjectivity.** Non–functional attributes can be percieved, viewed and valued in different ways by different people;

2. **Relativity.** Non–functional attributes can, depending upon the particular system type and domain characteristics, have a relative importance and priority attached to them;

3. **Interactivity.** Within a system, as a whole, attempting to promote one non–functional attribute can subtly compromise and undermine other important non–functional attributes.

A further issue that is worth considering in respect to the relationships that exist between functional requirements and non–functional properties or attributes, is

that, unlike functional requirements, non–functional attributes cannot be directly promoted within the system. Instead, when qualitatively reasoning about them, they exist only in the form of desired goals to be achieved (i.e. non–functional goals) and require (as chapter 7 emphasised in focusing upon the relationships between purpose and function) a functional realisation step — in the form of a functional ascription, in order to promote them. Such an example is discussed by Chung et al [154] in respect to promoting the non–functional attribute of performability. They note that the performance attribute in a accounting system may be the goal of achieving a fast response time for customer accounts. As they highlight, this non–functional goal, in itself, cannot be directly promoted, but requires some functionalisation to promote fast response time. One such functional ascription that could help promote this perfomance attribute for customer accounts could be the use of indexing of customer accounts. This would provide some direct functionalised means for achieving the non–functional goal of fast response time for customer account.

Therefore, we can add one more characteristic of non–functional attributes, that is implicit in Chung et al's [154] framework, but becomes more explicit in respect to chapter 7, and this is the characteristic of:

> 4. **Indirectivity.** Non–functional attributes cannot directly be promoted but require some functional ascription selection in order to be realised.

As we will see in section 8.4.2 with the analysed and synthesised ATM case study examples from chapter 5, it is this characteristic of indirectivity combined with the subjectivity, relativity and intertactivity a given functional ascription can create that makes the promotion of non–functional attributes difficult and problematic — in terms of promoting the wider super–ordinate goal of dependability.

8.3.2 The Non–Functional Framework

The non–functional requirements framework proposed by Chung et al [154] is the result of both theoretical and field research that has been progressed by the

authors over approximately fifteen years. In this section the essential components and the process stages are described. Their particular suitability for consideration of providing exampled envisaged benefits of a goal–diverse process intervention will be discussed in subsection 8.3.3, later.

The essential main component involved in the non–functional requirements framework involves the notion of a *soft–goal.* This concept implicitly captures the indirect nature of non–functional attributes — in that they are desirable properties of the software artifact, but ones that, during the creation process, cannot be directly implemented. Chung et al identify four main types of soft–goals in the non–functional requirements framework, as follows:-

- **Main soft–goal.** These are the familiar non–functional attributes that are commonly known about in connection with software artifacts and the dependability community and include such top–level desirable attributes as: safety; security; usability; reliability; maintainability, etc;

- **Sub soft–goal.** These soft–goals are a refinement decomposition of non–functional attributes and illustrate the more specific contribution that can be made to the main soft–goals by thier achievement. A familiar example from the dependability literature would be how the sub soft–goals of achieving integrity, confidentiality and availability directly contributes to the achievement of the main soft–goal of security [cf. [68]];

- **Operationalising soft–goals.** These soft–goals relate to the harder functionalisation design decisions that are made to promote non–functional attributes. In the framework, they are illustrated in a heavier emboldend form to visually indicate this. However, the authors still refer to them as soft–goals as they are used as a analysis representation for exploring interdependencies and trade-offs they create, and do not represent fixed design decisions. Furthermore, as Chung et al [154] highlight, operationalisations do not always involve functionalisation in the form of design and implementation components and can include, more broadly, such things as rules, constraints, data, and information. Therefore, operationalisation soft–goals,

while considering more definite non–functional promoting aspects, still are represented as soft–goals to demonstrate their fluid and changeable nature as development progresses;

- **Claim soft–goals.** These soft–goals capture the thinking and reasoning components of the framework as the developer considers non–functional attributes. They can be attached to any relationships that the other three types of soft–goals may create (i.e. upward direct contributions or implicit/explicit interdependencies across non–functional attributes detected). They are often in the form of justifications and priority interpretations that are provided by the developer in consideration and promotion of non–functional attributes.

Having described the essential components of the non–functional framework (NFR), it is now possible to complete this brief introduction of the framework by discussing the process stages employed in using this non–functional requirements approach. In total six stages are identified, as follows:-

1. **Domain and System Type Consideration.** At the top level, the process essentially begins with careful consideration of the particular domain characteristics and system type to be created. This is particularly important for the consideration of non–functional attributes because, as mentioned in subsection 8.3.1, non–functional attributes have a relative characteristic that helps unearth thier particular importance and priority in a particular development situation;

2. **Functional Requirements Definition.** These are also critically important as it is only through gaining a good understanding of the particular informational, data, and transformational elements required, can the developer later get to grips with the interactive nature of non–functional attributes and detect important interdependencies between possible function ascriptions in deciding, selecting and promoting particular non–functional attributes;

3. **Identifying Relevant Non–Functional Attributes.** Here, the developer must identify which non–functional attributes are most important. The com-

bination of how well stages 1 and 2 above are performed will directly impact on how well the developer succeeds in this stage;

4. **Decomposition of Non–Functional Attributes.** It is at this stage when the developer decomposes the main soft-goals down into more specific sub soft–goals by identifying them and determining and justifying the particular, more specific, direct contribution they make. It is noted by Chung et al [154] that the direct upward contributions may require all sub soft–goals to be contributing or just a subset of them to contribute. The two cases can be exemplified as: i) All contributing is an *AND* restriction whereby all sub soft–goals must be contributing so for example with security it may be deemed that the sub soft–goals of: intergity *AND* confidentiality *AND* availability must all contribute; ii) alternatively a subset contribution would be an *OR* restriction, such as, for example, (again with the attribute of security) where it is deemed that only confidentiality *AND* integrity *OR* availability must contribute in this case;

5. **Operationalising of Non–Functional Attributes.** This is where, having identified the relevant main non–functional attributes soft–goals in 3 above and decomposed them down into more specific sub soft–goal contributions, various functional ascriptions, in the form of design alternatives, are contemplated that fulfil the functional requirements in order to begin to promote and realise the non–functional attributes for the system under development;

6. **Dealing With Interdependencies.** A natural step that follows 5 above is once a number of design alternatives have been contemplated, the various interdependencies that can exist between them are analysed. As highlighted in chapter 7, these can be of a complementary, compatible, or conflicting nature, and all these types of relationships must be considered and recorded. This is where claim soft–goals will more frequently be used as it becomes inevitable in complex system development that certain soft–goals cannot be fully achieved and therefore acceptable trade–offs with supporting priorities and justifications will need to be documented in the form of claim soft–goals.

It is stressed by Chung et al [154] that although, when considered in this way, the non–functional process stages may look like a linear waterfall type development process model, this is not the case and in actual application through many studies conducted by the authors, the non–functional process requires many iterations with plenty of feedback between the various stages. This was also the experience of the author in using the framework for providing examples for section 8.4.2, below.

8.3.3 Suitability of the Non–functional Framework

In this subsection a justification for the selection of the NFR by Chung et al [154] is provided for its utilisation in section 8.4.2 as an initial demonstration into the assumption identifaction benefits of the proposed goal–diversity process intervention. In order to do this, the underlying philosophy and benefits, proposed by the NFR framework, are contrasted with the issues raised by such a goal–diversity process intervention.

Perhaps the most fundamental justification for utilizing the NFR framework proposed by Chung et al [154] is that, at its core, is the principle of putting non–functional attributes as the foremost important consideration in the mind of the developers. Typically, as the quote by Malan and Bredemeyer [116] earlier demonstrates, much *lip–service* may be paid to promoting quality or non–functional attributes, but the *lived–in* experience time and again is that they are either never considered, considered arbitrarily, or considered too late within the software creation process. The philosophy of the NFR framework is to ensure that non–functional attributes drive the process right from the start and ensure that quality is built into the product as it progresses through the software development cycle. This makes the NFR framework, in essence, a consistent method for demonstrating the expected benefits of a goal–diversity process intervention, as, at its root, this process intervention is also aimed at utilizing the psychological/performance benefits expected from goal–setting by predisposing developers to promote dependability attributes to ensure that dependability considerations also drive the software creation process.

Probably the next most fundamental justifaction for the suitability of utilising the NFR framework for exemplifying the anticipated benfits of a goal–diversity process intervention is the fact that it is, in principle, qualitatively process–oriented. Not only, in trivial terms, does this make it suitable for consideration of a process intervention, but, as we can appreciate from chapter 4 on computer–based systems, extending the boundaries outwards to consider both the technical and human systems as subsystems of interest introduces greater subjective sociological, psychological, and organisational considerations as purpose ascriptions and failure judgements can result in technically dependable systems being judged to be undependable for non–technical situational and contextual reasons. Computer–based system consideration, therefore, demands that both subjectivity and relativity characteristics of non–functional attributes be handled by any applied method, as different computer–based system contexts-of-interest will both provide different meanings and place different priorities and importance upon non–functional attributes. Since the goal–diversity process intervention benefits expected are those for computer–based systems, then whatever method chosen to exemplify those expected benefits would inherently not only need to place non–functional attributes as a process–oriented driver, but also would need to provide a qualitative representation and reasoning framework that encompasses considerations of subjective purpose ascriptions, relative importance and priority judgements. The inclusion of the soft–goal within the NFR framework is not only useful for ensuring non–functional attributes are represented and drive the development process, they are also fundamentally based, from a qualitative and subjective reasoning perspective, upon a dialectical form of reasoning from Artificial Intelligence (AI) research that inherently accommodates for subjectivity in reasoning whereby strict *AND, OR, NOT* reasoning is dispensed with and contributions and interdependencies are considered for how they fully/partially influence and positively/negatively effect non–functional attributes, respectively. The form of reasoning adopted by the NFR framework is usefully based upon the real–world situation, in considering such complex contributions and interdependencies, of achieving adequately satisfied or satisficing promotion of non–functional attributes which is important in unearthing and making explicit, within the creation process, the degree of dependability and undependability of a computer–based system. This subjective

qualitative reasoning is therefore highly suitable for consideration of computer–based systems as subjective sociological, organisational, strategic, and situational interpretations of non–functional attributes can be more appropriately represented and reasoned about when the process intervention of goal–diversity is applied for consideration of computer–based systems.

Another fundamental benefit of using the NFR framework, by Chung et al [154], is that it provides a holistic approach through accommodating both a top–down analysis and bottom–up synthesis anticipated as being vital in the goal–diversity process intervention. As was presented in subsection 8.3.2, relevant main soft–goals are further refined into more specific contributing sub soft–goals, which are then further promoted by consideration of design alternatives via oparationalised soft–goals. This stage is essentially decompositional, top–down and analytical — in terms of promoting non–functional attributes. Following this stage, multiple relevant non–functional attributes are then considered for how subtle interdependencies are created with different operationalised soft–goal alternatives — with regards to how they positively or negatively impact upon promotion of non–functional attributes. This is fundamentally a synthesis approach stage that begins the representation, selection, reasoning, and justification of how different design alternatives can be organised into a system that can be rationalised in non–functional attribute promotion terms. In the goal–diversity process intervention considered in this thesis, it is anticipated that the analysis stage will be performed individually, whereby individual developers can analyse one non–functional attribute and perform considerations of design alternatives that directly promote this attribute, but the major synthesis part of indentification of subtle interdependencies between multiple non–functional attributes — in terms of reasoning and justification of multiple non–functional attributes into an agreed rationalised eventual (computer–based) system will be performed in a collaborative meeting stage. In this respect then, the goal–diversity process intervention approach demands an two[2] staged holistic consideration of promoting dependability by both individual

[2]However, although, in analytic and synergistic considerations of the Goal–Diversity process intervention, it can be considered as a two staged process, when considering an additional inspection stage, in chapter 10, the analytic stage is divided into two stages of: a) Individual Analysis; and b) Individual Inspection

analysis and collaborative synthesis of non–functional attributes. Any such approach would, by definitions presented earlier in subsection 8.3.1, need to accommodate for the inherent indirectivity and interactivity characteristics presented by consideration of non–functional attributes. In this regard, as can be seen from subsection 8.3.2, the NFR framework incorporates both representation and reasoning about such characteristics and therefore provides another justification for its suitability for use in exemplifying the expected benefits from a goal–diversity process intervention.

Finally, as a last justification for the usage of the NFR framework, by Chung et al [154], for exemplifying the expected benefits of a goal–diversity process intervention, its reasoning representation is valuable in identifying various harmful assumptions in promoting non–functional attributes during development. As discussed in subsection 8.3.1, non–functional attributes, by nature of thier indirectivity and interactivity characteristics often introduce subtle interdependencies during system development. Chung et al [154] highlights that these can be of two fundamental types: i) Explicit intentional contributions, in the form of direct conscious reasoning during considerations of how direct upward specific sub soft–goals contribute to main soft–goals, and also operationalisation soft–goals (i.e. design alternatives) contribute to specific soft–goals; and ii) Implicit unintentional interdependencies that result in the form of unconsidered positive/negative interrelationships between multiple non–functional attributes (i.e. main and sub soft–goals). This is where the incorporation of claim soft–goals (i.e. in subsection 8.3.2) is invaluable in forcing the developer to record these during analysis, as claims are, by nature, defined as *"Statements made as being true without being able to give proof of them being true."*[3] Within the NFR framework, claims represent the subjective and relative justifications and priorities made by various developers which are inherently based upon their own beliefs, experiences, values, training, preconceptions, etc. Therefore, as chapter 6 discussed, act as a documentable source for capturing the underlying assumptions being made. From a goal–diversity process intervention perspective, aimed at increasing assumption detection to improve computer–based system dependability, the NFR framework

[3]Oxford Dictionary definition.

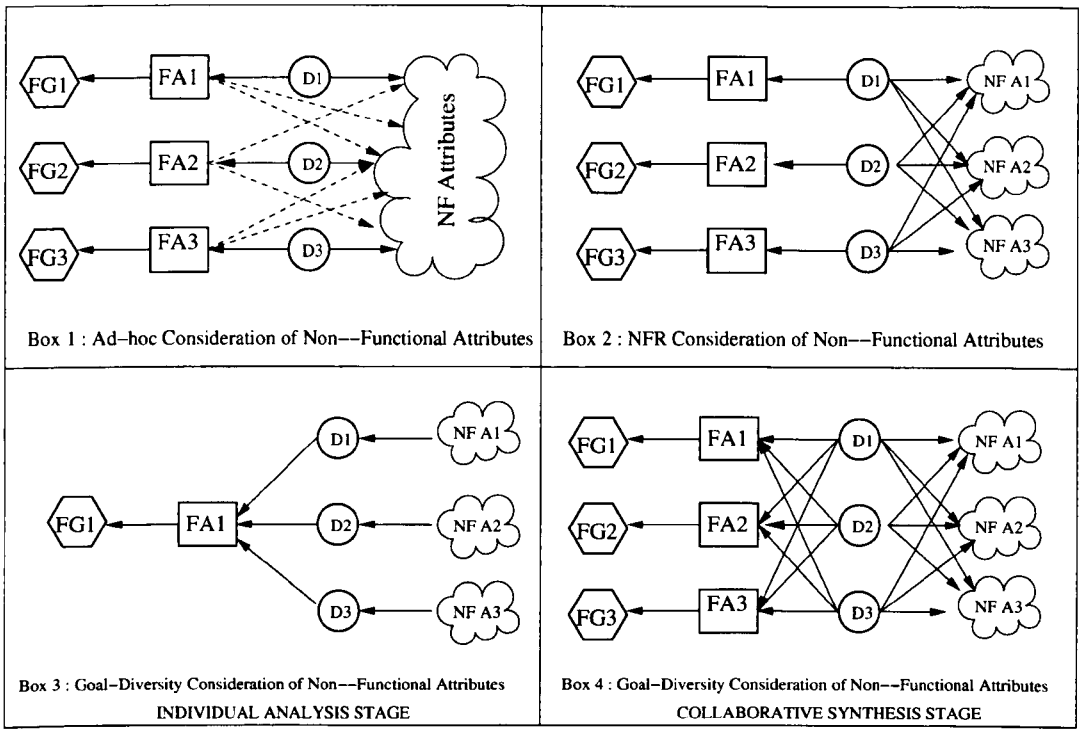


Figure 8.1: Differences in Approaches

provides an ideal representation as often such subtle and unintentional interdependencies are the results of various assumptions that occur through over/under representation of non-functional goals (i.e. relativity influence) or through ascribing different meaning/purposes to non-functional attributes (i.e. subjectivity influence). Since the NFR framework structure ensures that such claim soft-goals are attached to explicit contributions, interdependencies and operationalisation decisions, made by the developer(s), this then provides a means by which, at the later collaborative stages of goal-diversity, existing assumptions can be detected — along with implicit assumptions that result in interdependent consequences between multiple non-functional attributes.

8.3.4 Important Differences Between Approaches

Although, as subsection 8.3.3 shows, the NFR framework has important representational and reasoning justifications for helping to illustrate the expected benefits

of a goal–diversity process intervention, a direct comparison between the two, also reveals the existence of important differences.

In order to better understand these differences between the NFR framework, and the proposed goal–diversity process intervention, it will be useful to first describe, in some level of detail, the three possible ways in which non–functional attributes can be dealt with during development (i.e. process–oriented view). These three possible ways are illustrated in figure 8.1 and are described in the bullet points below:-

- **Ad–hoc consideration of non–functional attributes.** This is illustrated in the top–left box (i.e. box 1) in figure 8.1. The hexagons represent the functional requirement goals (i.e. FG) of a given development project. The squares represent some functionalisation or functional ascription (i.e. FA) that can help realise those functional requirement goals (i.e. FG). A total of three functional requirement goals (FG) are shown along with three functional ascriptions (FA) to achieve them in this particular example. The three circles illustrate the individual developers (i.e. D) whom are tasked with fulfilling the functional goals via design and implementation components represented by functional ascriptions (FA). The ad–hoc consideration of non–functional attributes results from no systematic aspects within the development process that ensures they are considered, promoted, and tracked as development progresses. They are therefore either not considered at all, considered arbitrarily or considered too late in the development life–cycle. Often, in many real–world software development projects promotion of non–functional attributes in the process is reliant upon an arbitrary and implicit promotion of non–functional attributes through the particular desires, expertise, experience and quality interests of the individual developers involved. This is why, in figure 8.1, the non–functional requirements are non–distinct and are represented by an amorphous cloud. As is often realised later, with such an approach, none or ad–hoc consideration like this, of non–functional attributes during the software creation process, frequently results in harmful negative interdependencies that are created during functional ascription and implementation of functions to achieve requirements

that undermine both dependability of the process and eventual delivered software artifact (if, indeed, it's ever delivered that is!). These harmful interdependencies are illustrated in the figure, in box 1, as dashed lines from the functional ascriptions back to the non–functional attributes that they compromise;

- **NFR consideration of non–functional attributes.** This is illustrated in the top–right box (i.e. box 2) in figure 8.1. The hexagons again represent the functional requirement goals (i.e. FG) of a given development project. The squares represent some functionalisation or functional ascription (i.e. FA) that can realise those functional requirement goals (i.e. FG). A total of three functional requirement goals (FG) are shown along with three functional ascriptions (FA) to achieve them. The three circles illustrate the individual developers (i.e. D) whom are tasked with fulfilling the functional goals via design and implementation components represented by functional ascriptions (FA). As already discussed in subsection 8.3.2, the NFR framework approach, by Chung et al [154], improve upon this situation by ensuring a systematic representation and reasoning framework that ensures that consideration of relevant non–functional attributes (in the form of soft–goals) drive the software development process. By doing so, developers have more opportunity to become explicitly aware of the harmful interedependency consequences of thier functional ascriptions to achieve functional requirements — thereby reducing the possibility of undermining the overall quality of the software artifact. In figure 8.1 this is shown by the solid lines that lead from the developers to distinct non–functional goals (NF As) that are considered relevant for this particular development.

- **Goal–Diversity consideration of non–functional attributes.** There is a two stage process envisaged in this process intervention. The first stage is illustrated in the bottom–left box (i.e. box 3) of figure 8.1 and involves individual analysis of a single functional requirement goal (i.e. FG) — in terms of considerations made about various functional ascriptions (i.e. FA) in the form of design alternatives and implementation components to realise them. As can be seen from figure 8.1, in comparison to the previous two

approaches, goal–diversity differs at this stage by ensuring that all three developers (i.e. D): i) consider the functional realisation of one functional requirement goal while; ii) individually promoting a single and distinct non–functional attribute only (i.e. NFG). The second stage is illustrated in the bottom–right box (i.e. box 4) of figure 8.1 and involves the collaborative synthesis stage whereby after all developers have analysed individually all three functional requirement goals (i.e. FG) individually (and considered the most appropriate possible functional ascriptions (FA)), a meeting takes place where all three developers, committed still to promoting only one particular and distinct non–functional attribute goal (i.e. NF A) compare, challenge, and argue the various merits and drawbacks in promoting the three non–functional attribute goals while satisfying the three functional requirements goals (i.e. FG). Obviously, this will result in conflicts as subtle negative interdependencies between the various design options require re–prioritisation and trade–offs, and as section 8.2 highlighted and suggested earlier, this is where assumptions, in the form of various claim soft–goals, will be potentially detected.

Contrasting the possible ways that non–functional attributes can be considered during the development life–cycle, it can be appreciated, as discussed in subsection 8.3.3, that the NFR framework, proposed by Chung et al [154], is useful in demonstrating the expected assumption detection benefits of a goal–diversity process intervention. Primarily this is because it provides a representation, reasoning and philosophy that allows the expression of associated complex information to be illustrated and recorded in a manner superior to what could be achieved as easily in any textual way. However, within the context of things considered within this thesis so far, the NFR framework can also be criticised on three aspects when compared with the proposed goal–diversity process intervention in helping to unearth various harmful assumptions that may often compromise computer–based system dependability.

The first criticism relates to its assumption detection capability — especially with regards to implicit assumptions. As it was highlighted in the assumption chapter

6, implicit assumptions often relate to the absence of conception not its presence, this is due to the fact that a single individual is unlikely to indentify an implicit assumption by him or herself because the assumption acts as a conceptual constraint on that particular thinking episode and therefore, acting as a constraint, precludes any possibility of detecting it by definition. This introduces problems within the existing NFR framework as it expects a single developer to consider many non–functional attributes at once and therefore inherently expects the developer to individually identify subtle unintentional interdependencies between them that are often underpinned by such implicit assumptions related to his or her values, beliefs, biases, etc. By contrast, the proposed goal–diversity process intervention makes no such expectation as the collaborative synthesis stage, where multiple developers (predisposed to promoting different non–functional attribute goals) incorporates a more challenging and conflicting phase that can help identify such implicit assumptions.

The second fundamental criticism relates to cognitive limits in performing very complex conceptual tasks. As recognised in the literature of chapter 7, achieving multiple goals *'hits'* cognitive limitations as usually only one such goal can be maximally activated at any one time during multiple goal problem solving. At best, this can result in other goals being under or inappropriately emphasised, and, at worst, can result in other important goals not even being considered at all. Again this introduces potential problems for the NFR framework as, implicitly, a developer using the methodology is tasked with having to promote multiple non–functional attribute goals simultaneously. In the context of this thesis, this is considered to be a dubious expectation of the NFR framework methodology in identifying explicit and implicit assumptions that can often result in unintentional negative interdependencies that can compromise the dependability of a computer–based system. With the proposed goal–diversity process intervention, this potential cognitive limit is accommodated for as throughout both the individual analytic and collaborative synthesis phases a single developer is only tasked with the responsibility of promoting a single non–functional attribute. Even at the synthesis stage, where multiple non–functional attributes need to be considered for harmful interdependecies, trade–offs, and priorities, this is done in a collaborative manner

between developers that are considering the various arguments and justifications of thier fellow development colleagues from only a single non–functional attribute perspective.

The final criticism relates to the expectation of the NFR framework that a single developer posseses the necessary experience and knowledge in promoting multiple non–functional attributes — deemed relevant to that system type and domain characteristics pertaining for particular software development project. This aspect is only handled generally within the NFR framework. By contrast, the proposed goal–diversity process intervention ideally expects a level of specialism of the individual developers in the particular non–functional attribute they are expected to promote. For example, a developer tasked with the promotion of security will have specialist knowledge and experience in security considerations, a developer tasked with the promotion of maintainability will have specialist knowledge and experience in maintainability considerations, etc. Within the wider perspective of a computer–based system, the goal–diversity approach also implicitly expects that each non–functional attribute specialist will intelligently interpret their expertise into differing computer–based system contexts–of–interest of the utility context, the deployment context, and the evolution context to enhance a more encompassing and synergistic coverage of assumption identification during the development process.

As a last point, and whilst not a direct criticism of the NFR framework, perhaps the most important distinction between the NFR framework and the proposed goal–diversity process intervention, is that the proposed goal–diversity process intervention utilises human redundancy and human diversity within the process to achieve an increased level of process dependability. Firstly, it uses human redundancy, in terms of duplicated effort for a given task. This can be seen from box 3 (bottom–left) by comparison with both the ad-hoc and NFR framework considerations of non–functional attributes in figure 8.1. Note that, by comparison, the analysis stage employs three developers for the realisation of a functional requirements goal, whereas the other two approaches in box 1 (top–left) and box 2 (top–right) employ the much more typical concurrent engineering principles of

factoring–out development effort in parrallel by ensuring a specific task is allocated to a specific developer. Secondly, it can be seen, however, from figure 8.1, that it generates human diversity by predisposing the three developers to promoting three distinct non–functional attributes. From chapter 7 it can be remembered that goal–setting introduces interesting cognitive influences that motivate individuals to search–out task specific information to fulfil those goals. Furthermore, by definition, setting an individual goal will effect thier mental model (i.e. formal cause) thereby inherently sensitizing them to value, prioritise, and judge the same given thing (i.e. in this case a functional requirement goal) in different ways. It is a major expectation that when such a process intervention is employed it will result in greater exploration and coverage that will improve the identification of flawed assumptions that can ultimately undermine and compromise the eventual dependability of computer–based systems.

8.4 Goal–Diversity – Analysis and Synthesis

In this section an initial set of examples, in the form of two scenarios, are used to indicate how the envisaged and proposed process intervention of setting diverse non–functional goals can be employed during the software creation process. In subsection 8.4.1, the two stages of individual analysis and separate inspection stages are exemplified, using a simple scenario, to show how the subjective, relative and indirective aspects of non–functional attributes both cause and help detect harmful assumptions. In subsection 8.4.2 the final third collaborative meeting stage is considered. While this subsection does not consider the particular group team dynamics and inevitable trade–off conflict and negotations involved, it does utilise the Chung et al [154] framework and provide more extensive examples of the ATM case–study in chapter **5** to provide sufficiently rich examples of how the interactive nature of non–functional attributes can further help detect harmful assumptions during this collaborative synthesis stage.

8.4.1 Analysis Examples

The envisaged goal–diversity process intervention has three fundamental stages. The first two of which, belong to the analysis phase. The first analysis stage involves predisposing a number of developers to promote a single non–functional attribute deemed critical to creating a dependable software artifact. The result is a number of functional ascribed analysis of a proposed solution from each of the non–functional predispositions. The second analysis stage involves the separate cross inspection of each of the predisposed analysis solutions so that each developer who promoted a non–functional attribute solution in the first stage can then compare and contrast the other predisposed analysis solutions from their own single non–functional attribute predisposition. Each of these first two phases are considered using a simple development scenario in subsections 8.4.1.1 and 8.4.2 below to make this clearer.

8.4.1.1 First Stage — Individual Goal Promotion

The proposition of the proposed goal–diversity process intervention, at this first stage, is that by predisposing individual developers to promoting a single non–functional attribute, during analysis, will sensitise them from being less likely to make harmful assumptions that will directly mitigate their non–functional goal, and make them more likely to make harmful assumptions that will indirectly compromise other non–functional goals being promoted by other developers.

To provide a simple software development scenario, let's consider that a new catalogue customer accounting system has to be produced that keeps track of telephone customers' orders and billing. Customers phone in orders to a telephonist who then enters the orders and updates the customers outstanding bill. Customers can also pay over the phone by a credit/debit card which the system then relays to the credit/debit card financial institution for payment.

After the initial functional requirements, it is deemed that the non–functional attributes of: a) maintainability; b) performability; c) reliability; and d) security are critical to the overall dependability of the software system. Four developers are

therefore separately employed to produce an individual design analysis solution. Each one is predisposed to ensure that while performing the design analysis that the decisions they make must solely focus on prioritising only one non–functional attribute. So for instance, developer 1 solely focuses upon promoting maintainability, developer 2 solely focuses upon promoting performability, developer 3 focuses upon promoting reliability, and developer 4 focuses upon promoting security.

After each of the four developers have performed their individual analysis solutions, they make three copies of them and distribute them to the other three developers ready for the second stage of separate inspection (see subsection below).

8.4.1.2 Second Stage — Separate Inspection

This stage is performed individually in separation, firstly as an important assumption detection phase in itself, and secondly as a preparation stage for the collaborative meeting so that each developer can gain a richer understanding of the software development problem — in terms of the important dependability attributes.

It is at this stage that the relative characteristic of non–functional attributes becomes important, as each developer continues to inspect each of the other developers' design analysis solutions — while still being predisposed to promoting their own single non–functional attribute. It is a proposition of this second stage that as each individual developer separately compares and contrasts each of the other developers' design analysis solutions, the relative undesirable consequences for their own prioritised and promoted non–functional attribute will help unearth potentially harmful assumptions that could compromise that non–functional attribute.

To return to the simple catalogue accounting system scenario, during this second separate inspection stage, when developer 1 (promoting maintainability) inspects the other three design analysis solutions from the other developers, he/she notices that to speed–up performability developer 2 has advocated that some of the more

commonly used functions should eventually be implemented inline as he/she (developer 2) has calculated that within the main execution loop these functions are the main time–consumer of processing resources and the time spent in executing these functions is expected to be favourable to the time spent jumping around calling them if they were not inline. However, developer 1, comparing and contrasting his/her analysis solution while still promoting maintainability, is concerned that such a solution will compromise their goal by undermining the overall cohesion of the classes which these particular inline functions represent. When developer 3 (promoting reliability) compares and contrasts the other three developers' design analysis solutions, during this separate inspection stage, he/she is also not happy with developer 2s (promoting performability) design analysis solution, as he/she has advocated for a recursive set of functions for printing accounts, copying account lists, and searching file storage, which, while producing performance benefits, appear to be particularly complex to developer 3 (promoting reliability) and provides a lack of exception handling defenses against potential run–time faults. Lastly, when developer 4 (promoting security) compares and contrasts the other three developers' design analysis solutions, he/she notices that developer 1 (promoting maintainability) and developer 2 (promoting performability) have both employed very strict normalisation (up to 5th normal form) and enhanced indexing of fields to promote future reporting flexibility and speed of information access, respectively. However, developer 4 (promoting security) has deliberately violated strict normalisation principles to ensure enhanced confidentiality and privacy of customers credit/debit card details into a separate table, with 1–to–1 correspondence, so that such a subset of sensitive customer information can by encrypted and contain more restrictive access rights.

From this brief example, with the catalogue accounting system, it can be appreciated, that in ensuring individual developers first perform a design analysis while promoting a single non–functional attribute, and secondly, later compare and contrast each others design analysis solutions allows both the relative and integrative characteristics of non–functional attributes to help detect potentially harmful assumptions that underpin such design analysis solution decisions.

8.4.2 Synthesis Examples – Using ATM Case Study

In the third stage of a collaborative meeting it is anticipated, with the goal–diversity process intervention, that the issues raised in the second stage (in sub-section above) will be further discussed and debated. It is in this stage that either breakthrough solutions or trade–offs will need to be sought or agreed, respectively, to determine the most appropriate or feasible solutions regarding the extent to which the non–functional attributes can be promoted. If breakthrough solutions are found then conflicting relationships (and their underlying assumptions supporting them) can both be satisfactorily accommodated. If not, then at least the degree of both dependability and undependability, concerning the particular software artifact to be developed, can be made explicit — along the levels of assumption validity allowable.

Rather than continue with a simple example, in this section, we draw–upon such issues raised in section 8.3 by exemplifying the expected benefits of a goal–diversity process intervention with the nine ATM computer–based system case study issues raised in chapter 5 to illustrate how combining computer–based system contexts with a multiple diverse goal–orientated approach, that includes the subjective, relative, and interactive characteristics of non–functional attributes, can result in greater overall assumption identification coverage. Within a computer–based system perspective, claims and reasoning approaches often appear reasonable when viewed from a particular computer–based system context–of–interest or non–functional attribute goal to be promoted. However, as will be appreciated, they can create harmful or conflicting relationships with other important non–functional attribute goals necessary to create a dependable computer–based system.

The reader should note that the illustrations attempt to capture the assumption identification events expected during the second synergistic phase of a goal–diversity process intervention when all the developers meet and discuss, argue, and judge — and therefore produce more integrative system perspective.[4] As will be seen,

[4]This phase is more consistent with perceiving dependability as a super ordinate system goal at a higher and more holistic level.

the NFR framework provides a sufficiently rich representational and reasoning structure to capture the complex collaborative development stage situation in a manner that would not be so easily ellaborated textually. Finally, the terminology and icons used is adapted from Chung et al [154] methodology on the Non–Functional Requirements Engineering approach. To aid the reader in understanding the NFR framework representations, a subset of the visual modelling icons used is explained in Appendix section A on page 380 in section A.1, along with additional terminology and diagrams specifically used for exemplifying the benefits of a goal–diversity process intervention in appendix section A.2 on page 382.

8.4.2.1 Encryption Policy — Issue 1

The first issue relates to the many security flaws that can be introduced through employing proprietary software encryption. This is illustrated in figure 8.2. It can be seen that there are two top–level non–functional goals involved. One concerns the organisation's strategic budget limitations. The other involves the non–functional attribute of security. The organisation's overall spending budget is further divided into sub–goals. One of which, that is relevant to this issue, is the ATM budget for commissioning, developing and deploying a network of ATMs. Another important sub–goal would be the associated costs of the encryption policy to be adopted to protect the confidentiality of bank and ATM customer's account details during transactions. The top level ATM security goal is then further divided into important sub–goals to be achieved. One of these, relevant to this issue, involves the sub-goal of the confidentiality of the customer's accounts. Correspondingly, an important sub-goal to achieving confidentiality is the particular nature of the encryption policy to be enforced or promoted. In chapter 7, the issue of functional ascription was discussed in the context of goal–directedness and teleological explanations. It should be pointed out that these functional explanations are important in analysing the ATM problems illustrated in the diagrams. The emboldened clouds in figure 8.2 on page 208 represent potential functional instantiations of how to promote the security sub–goal of confidentiality. This confidentiality goal represents a non-functional goal (what Chung et al [154] call *"soft–goals"*) as they cannot be directly implemented, but require some functional

implementation (Chung et al use the term *"operationalize"*) in order to achieve them. In this example two possible functional options exist: either the encryption can be implemented in hardware or software.

At this point it is necessary to consider the generic computer–based system contexts–of–interest. Two are illustrated. The utility context represents, in the ATM application domain, the higher strategic business assessment surrounding the commercial value expected from investing in an ATM network by the financial organisation. It is reasonable to suggest that the budgetary goal of the ATM network will have a major influence upon which encryption functionalisation will be employed. Since this decision will take place within the utility domain a justification for choosing a proprietary software functionalisation over a professional (industry standard) hardware functionalisation may employ a priority to financial criteria over technical criteria. This may result from some uninformed (or knowledge bounded) reasoning, or, worse still, may be politically motivated to place a priority of financial criteria over technical criteria. In either situation, it is likely to become manifest in the assumptive claim that software encryption is as secure as hardware encryption. In this regard, although both non–functional attributes (i.e. goals) of ATM budget and security are considered together, the assumption that software encyption is as secure as professionally standardised hardware encryption is used to emphasise and prioritise financial budget criteria in the form of an explicit justifying proposition.

It is obvious from chapter 5, that such a decision undermines the dependability of the ATM as it leaves open the potential for unintentional faults, malicious code (i.e. intentional), or systemic design flaws that can compromise the encryption integrity of the software. Such vulnerabilities will most likely become evident after deployment (i.e. shown in the oval in figure 8.2 on the following page).

8.4.2.2 Authorisation Policy — Issue 2

This issue relates to the system effects of the financial institution's disbelieving attitude that all unauthorised withdrawals are either due to customer carelessness

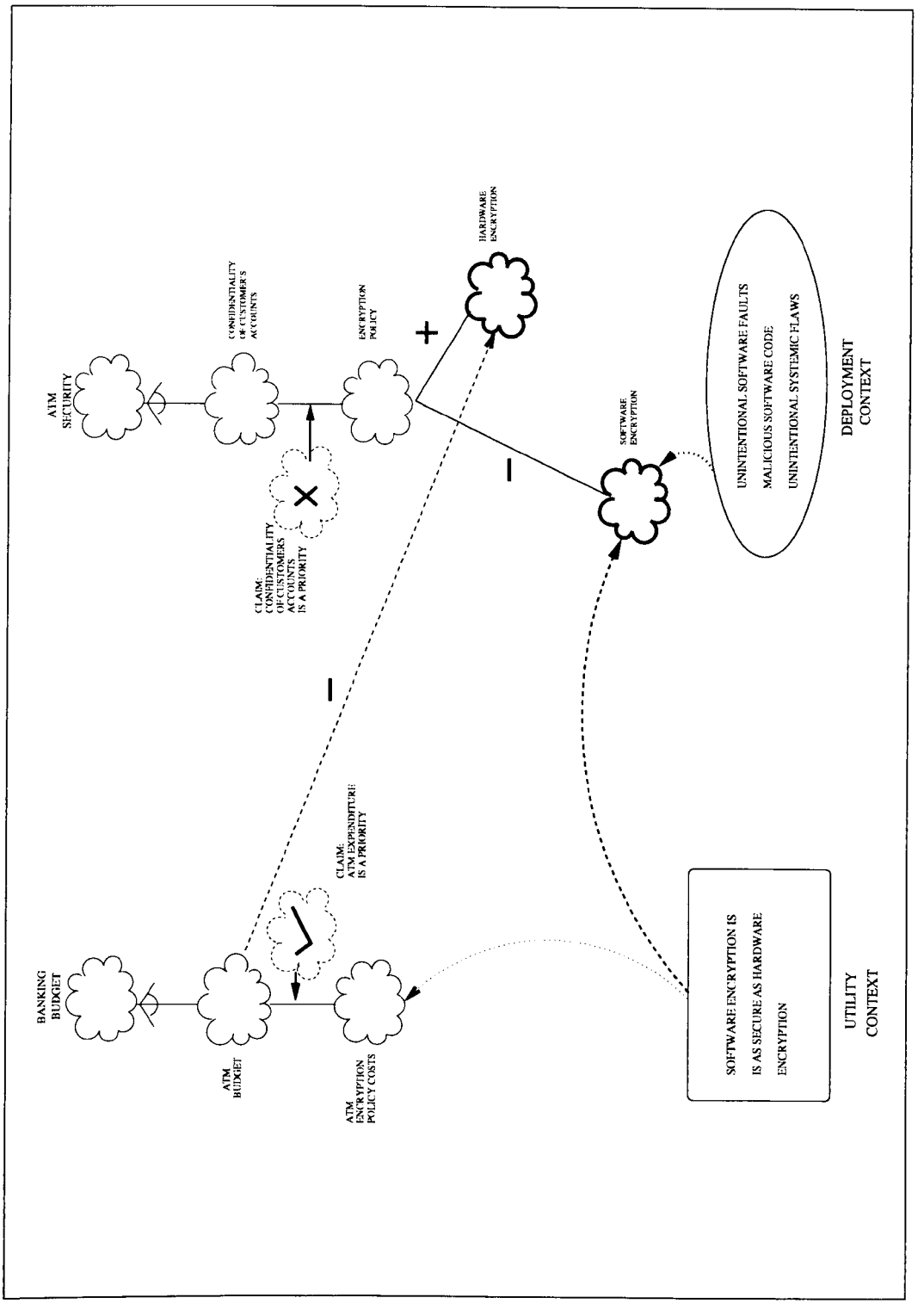


Figure 8.2: Encryption Policy Issue 1

or collusion. The issue is illustrated in figure 8.3 on the next page. In particular it can be seen that the two non–functional attributes of safety and security are involved. With safety, the sub–goals of the financial institution's safety and the customer's safety are further illustrated. These are then decomposed into further specific non–functional goals of: a) protection of the institution from financial loss; b) protection of the institution from loss of reputation or public confidence; c) protection of the ATM customer from wrongful conviction; and d) protection of the customer from financial loss.

It is shown from the persistent maintenance of this attitude that the financial institution acts as a kind of functionalisation (i.e. to realise the purpose). However, while this positively protects the institution from damage to its reputation and financial loss, it also undermines the safety goals of the ATM customer who is more likely now to be accused of attempting to obtain money by deception or not be reimbursed for any unauthorised withdrawals — due to them not being believed.

This is further undermined by the vulnerabilities of the functional implementation of achieving ATM authorisation via an ATM card and PIN — which has been widely proved to be vulnerable to unscrupulous insiders during engineering or maintenance of ATMs (i.e. engineering and evolution contexts) or from various methods by external fraudsters within the deployment context such as:-

- Shoulder–Surfing
- Social Engineering Techniques
- Bogus ATMs
- Skimming Devices

In this case, as discussed in chapter 5, the assumptive claim that all unauthorised withdrawals must be the result of carelessness or collusion is in the form of a deliberate public supposition made to protect the financial institution's own safety from loss of reputation or financial costs. However, this assumption is underpinned by a lack of coverage representation of the non–functional attribute of

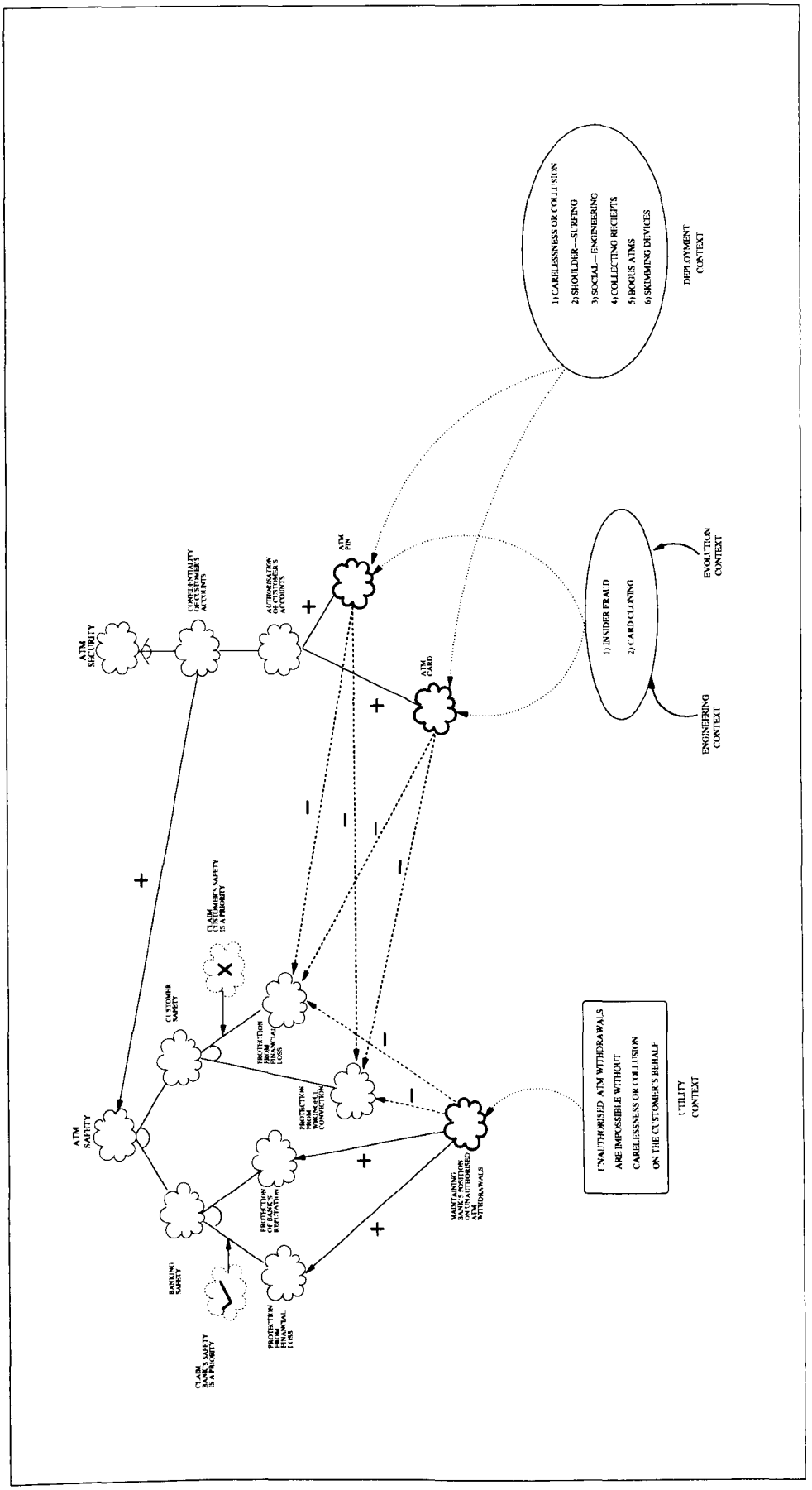


Figure 8.3: Authorisation Policy Issue 2

safety — in terms of the potential undesirable consequences that maintaining this position can have upon ATM customers.

8.4.2.3 Human Error Analysis — Issue 3

This issue is the simplest of the nine issues. It is simple because it involves only one goal i.e. safety. It is illustrated in figure 8.4. It can be seen that the sub goal of safety is the non–functional attribute of protection from financial loss of the customer. In this case the functionalisation involves the human–reliability development option of either allowing for human error during an ATM transaction or not by anticipating the potential for the ATM user to commit a post–completion error by forgetting to take the cash from the mouth of the cash dispenser. To promote the non–functional goal involves incorporating a timed retraction of the cash back into the ATM to prevent passers–by or subsequent ATM users stealing the cash. Incorporating this functionalisation promotes accommodation for human error and protection of financial loss (i.e. hence the plus sign) and omitting it undermines the non-functional sub goal.

An interesting facet involves how this functionalisation may be omitted from the engineering domain. This omission may be due to some implicitly shared assumption caused through a lack of knowledge about the deployment context or some constraint on that particular thinking episode concerning this human–reliability issue. In the diagram in figure 8.4 it has been shown as an assumption that ATM customers are hardly likely to forget to take their cash at the end of an ATM cash withdrawal. This, however, under represents the non–functional attribute of safety consequences (with respect to loss of cash) that the ATM customer can experience with this failure.

8.4.2.4 Opportunistic Theft — Issue 4

This issue is a kind of extension of issue 3 — concerning the retraction of cash back into the ATM. It is illustrated in figure 8.5. In this case it was reported in chapter 5, that unless the cash retracted back into the ATM is audited, it is possible for ATM users to remove part of the dispensed cash before retraction. Two

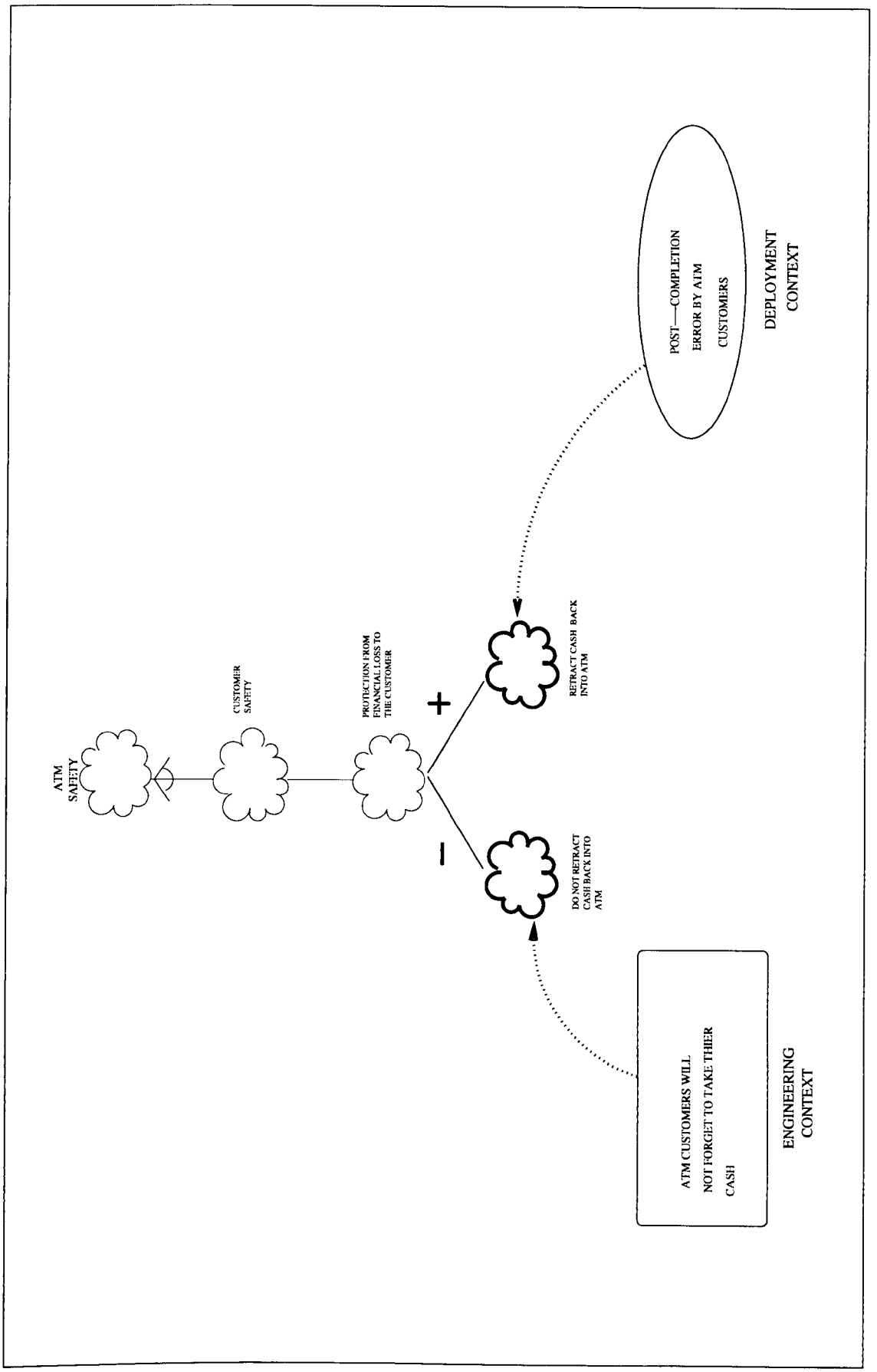


Figure 8.4: Human Error — Issue 3

non–functional goals are of interest in this issue. Firstly security, and secondly safety. With regards security, a particular sub–goal relevant to this issue concerns the integrity of the accounts. An important sub–goal of this is to ensure effective auditing and recording of all ATM transactions so that any physical money deviations can be subsequently traced. This therefore involves two possible functionalisations to fulfill these goals. Either money retracted is to be carefully audited or not. If it is, then this directly promotes traceability and the integrity of accounts (i.e. hence the plus sign). If not then it directly undermines the traceability and integrity of accounts (i.e. hence the minus sign in the figure).

It can be seen that if this functionalisation is not employed then it can potentially undermine the safety concerns of the financial institution — in terms of the non–functional sub–goal of protection from financial loss of the institution from opportunistic theft from the deployment context. Employing this functionalisation, however promotes the safety sub goal.

Finally, it is shown in figure 8.5 on the following page that if this functionalisation of auditing and recording retracted cash is not anticipated by the engineering context it may have resulted from some implicit assumptive reasoning that when cash is retracted it is solely because the ATM user has unintentionally forgot to take their cash i.e. a genuine mistake. In such cases it is reasonable to assume that all of the requested cash will be retracted. The choice of omitting extra traceability functionality is demonstrative that this implicit assumption possibly results from an under representation of safety concerns of protecting the bank institution from potential financial loss from such opportunistic theft by legitimate ATM customers.

8.4.2.5 Obscure Security Flaw Conflicts — Issue 5

Recall from chapter 5 that this was identified as one of the more complex examples and this is certainly illustrated in figure 8.6. Three CBS contexts (i.e. deployment, engineering & utility) are involved, along with four dependability attributes of

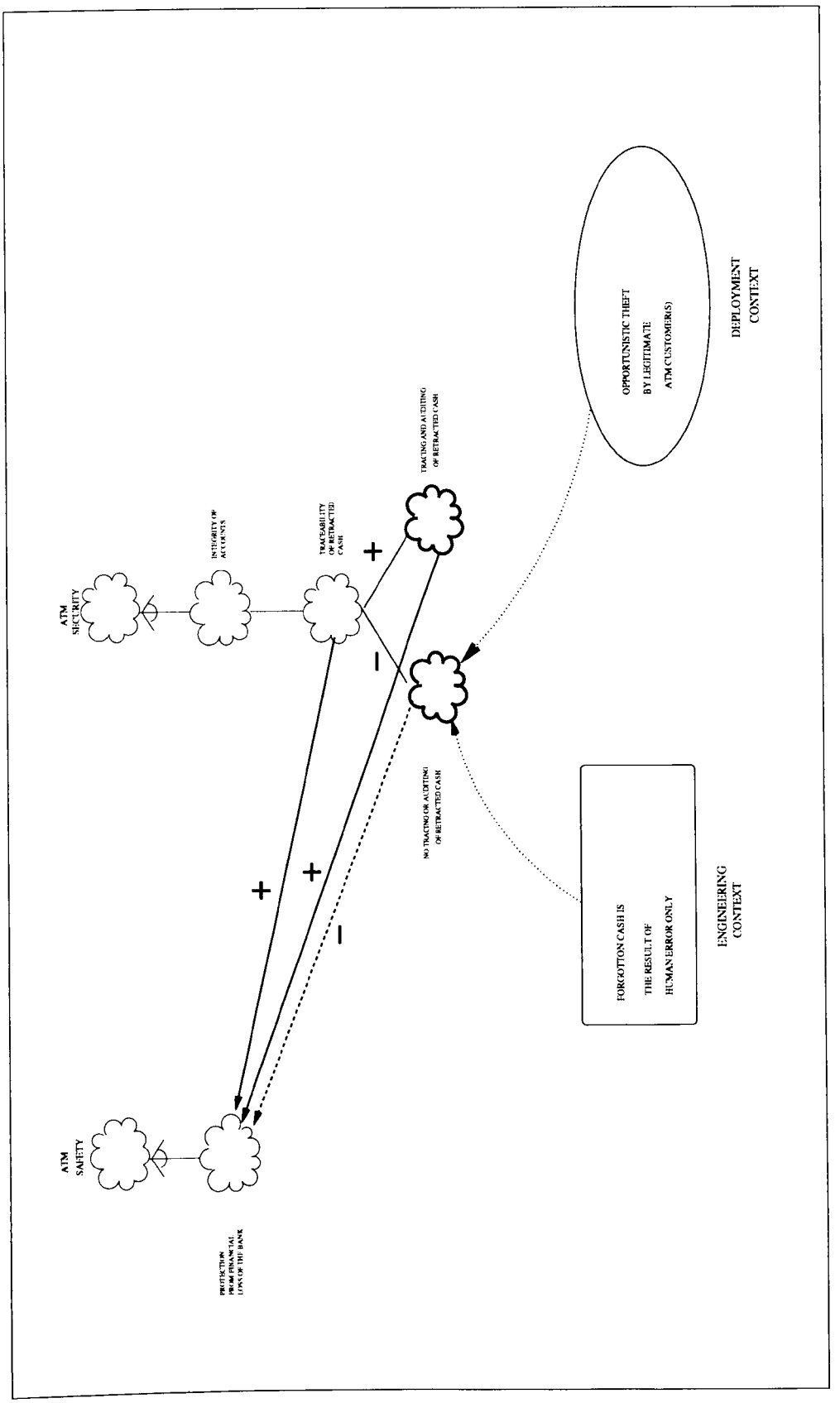


Figure 8.5: Opportunistic Theft — Issue 4

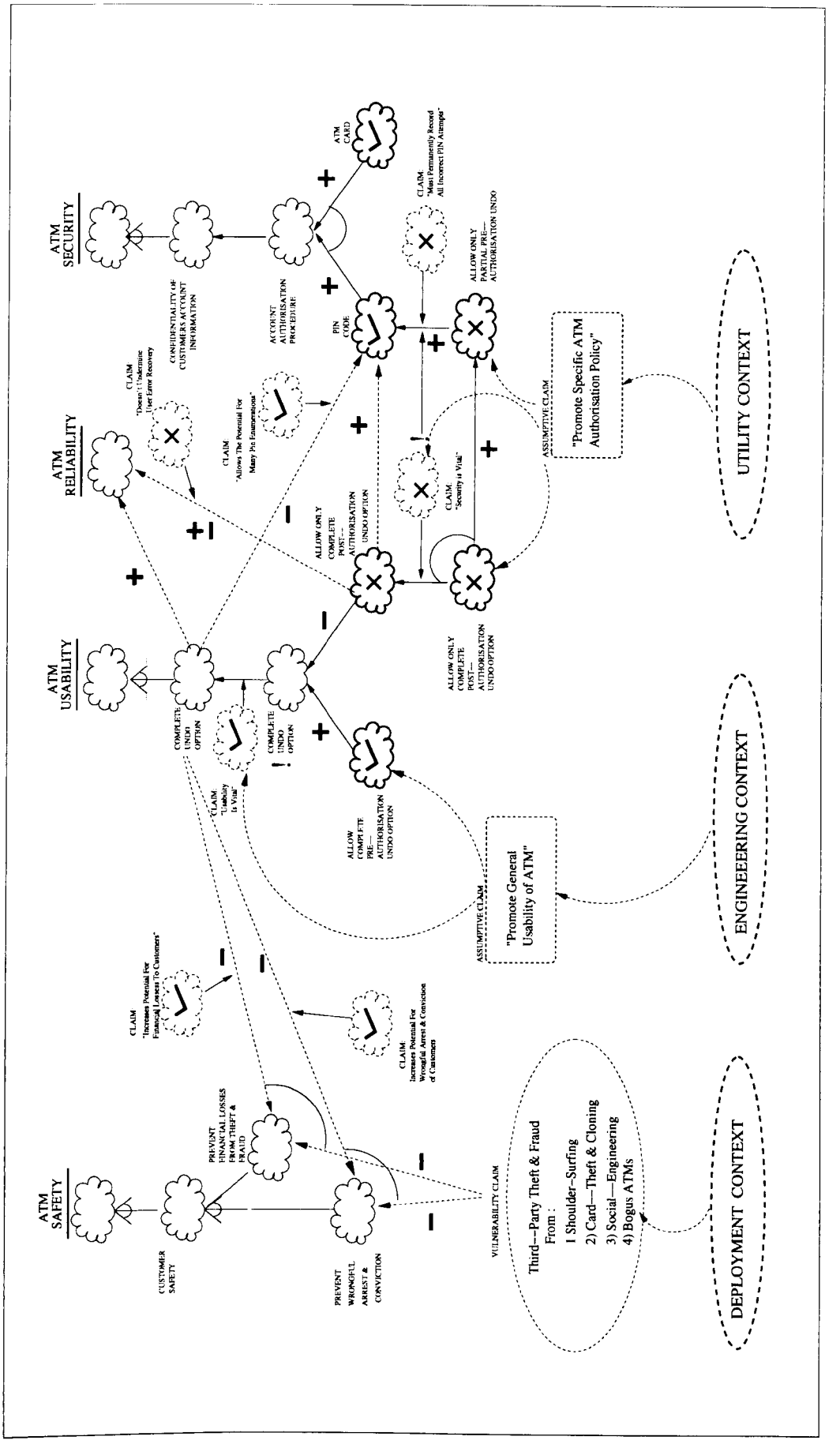


Figure 8.6: Obscure Security Conflict Issue 5

safety, usability, reliability,[5] and security.

A particular aspect relating to assumptions, with this issue, is how *inductive reasoning,* within a particular context, can result in shared contextual or knowledge based assumptions. In this case, the generic HCI engineering knowledge of promoting usability by always allowing the user, interacting with the computer system, to completely undo the transaction so that unanticipated human errors and conditions (i.e. discussed in chapter 5) can be recovered from (and hence promote reliability — in terms of transaction error recovery). This inductive premise results in the engineering domain prioritising the goal of usability — to always allow complete transaction undo without ever really being aware of, how, in doing so, antagonistic (or conflicting) dependencies arise with customer safety and secure authentication procedures within the specific context of an ATM application. In this respect, as discussed in previous sections, the assumption does not really exist, as an argumentation supposition (or presupposition), in the mind(s) of the engineer(s), rather it becomes a shared constraint upon their particular thinking episode about promoting usability (and indirectly reliability as contributing criteria justifacation). This is an important point to state, as the qualitative interdependency reasoning and operalisations (or implementation) claims enforced in figure 8.6, would imply such an argumentation (either internally within a particular engineer or collectively between engineers) has actually taken place. Here then, it is possible to argue that this implicit assumption results from an over–emphasis placed upon usability criteria and user error–recovery that failed to detect that when user authentication is required a user should never be able to perform a complete transaction undo operation until after they have successfully authorised themselves. Other than this, during the authentication operation, any failed authorisation attempts (i.e. in this case via PIN code) should be permanently recorded if the integrity of the security policy is to be maintained.

[5]In this case reliability is promoted through a subgoal (not shown for space reasons on graph) of transaction error–recovery.

8.4.2.6 Interaction Consistency and Completeness — Issue 6

This issue relates to ensuring that the flexibility of the embedded software in the ATM does not extend beyond the firmware or physical limitations of the ATM. It is illustrated in figure 8.7. In this case, the Daily Withdrawal Limit (DWL) — controlled by the software, is made to be flexible to accommodate the many different types of accounts by different customers (i.e. this will be stimulated by the utility context to offer a more flexible customised service to customers). However, over time, if this exceeds the limitations of the relevant firmware devices — such as the cash dispensing mechanism and width of the cash slot then erroneous behaviour is likely to result (i.e. as reported in chapter 5).

Three non–functional goals are involved in this situation. These are reliability, availability, and safety. With respect to reliable operation of the deployed ATM, it is important to promote the sub–goal of interaction consistency — in terms of ensuring that all the controlled composite parts (i.e. software, hardware, and firmware) are controlled in a consistent manner. The sub goal of interest in this issue to help this is to ensure that firmware and software interaction consistency is prioritised. This means that the lower functionalisation of the firmware cash handling and dispensing mechanisms have constraints, limitations, and flexibility consistent with the limitations, constraints, and flexibility of the embedded software control.

If this firmware/software interaction is not consistently controlled then it will impact unfavourably (in dependability terms) upon a) availability — in terms of allowing customer access to required cash amounts; and b) safety — with regards to customer(s) potentially losing cash through the banking institution not believing that the inputted cash requirement (or part of) was not dispensed.

With regards to contextual and assumption issues, it can be argued that this may arise in the engineering and/or evolution contexts where the engineers involved assume that their responsibilities only extend to embedded software control flexibility of the Daily Withdrawal Limit (a separation of concerns issue). It can be

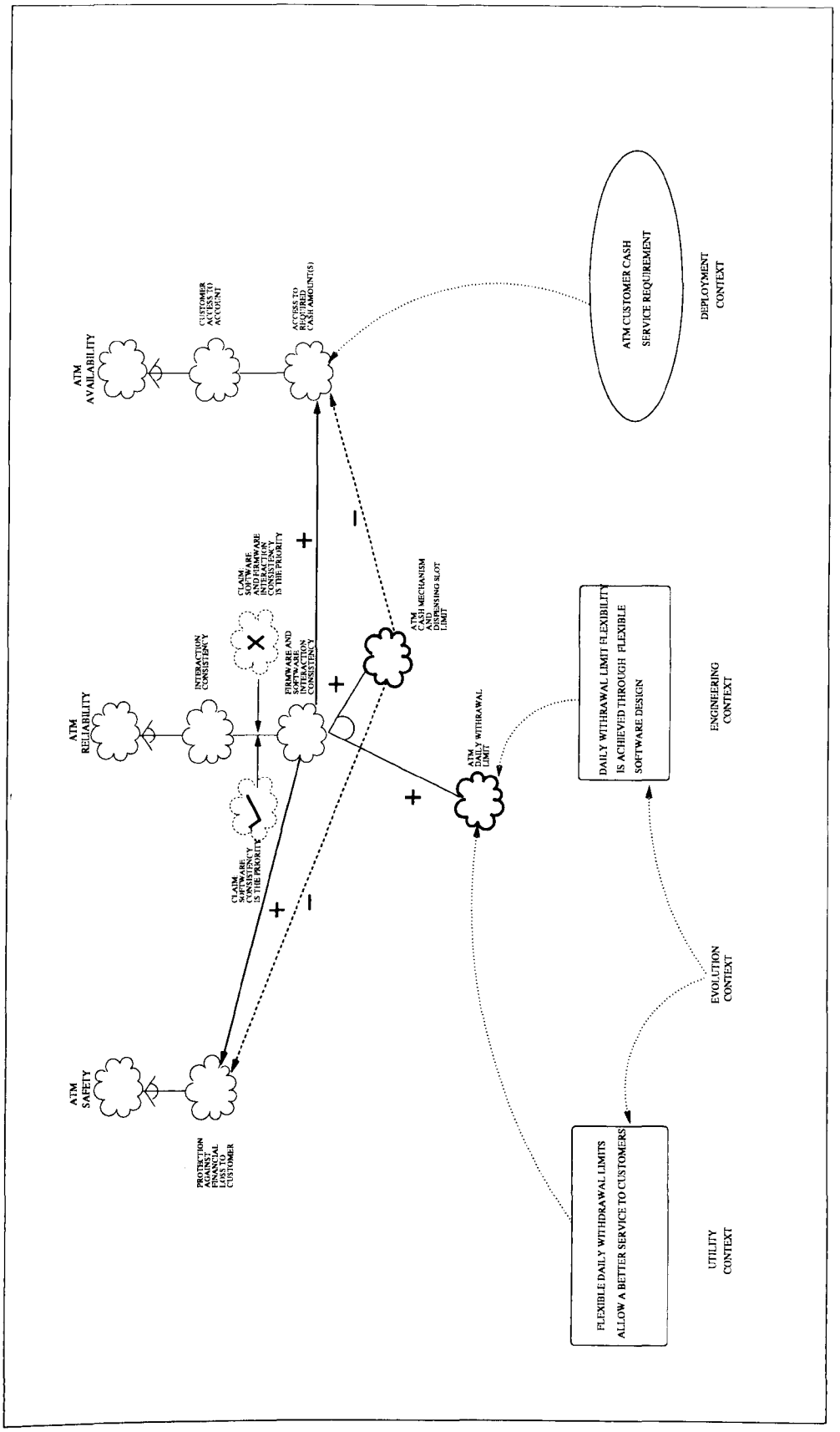


Figure 8.7: Interaction Consistency — Issue 6

appreciated that such assumptions, depending upon the nature of the engineering organisation, could be either made in: a) an implicit unquestioned manner — perhaps due to the culture, structure, or working practices of the particular organisation in a collective way that results in the software engineers collectively assuming that consistency between the firmware and software is not thier responsibility; or b) may even be explicitly considered and reasoned about but eventually it is supposed, as a simplifying reasoning mechanism, that the utility context will not extend DWL to beyond the maximum withdrawal limits imposed by the physical firmware of the cash dispensing mechanisms. In either case it can also be seen from this issue that, from a computer–based system standpoint, there is under representation of other important non–functional attributes that have interdependent relationships of how this assumption (either implicit or explicit) will negatively impact upon safety and availability. Lastly, it should be noticed, from an evolutionary context perspective, that, as the embedded software DWL are extended beyond the physical firmware limitations, the assumption will inevitably become invalidated in certain circumstances.

8.4.2.7 State Representation Completeness — Issue 7

This example is similar in some respects to issue 6. It concerns the need for adequate state representation completeness of the embedded software control to ensure reliable behaviour of the ATM. It is illustrated in figure 8.8. In this case the need to represent the physical state of the amount of cash in the cash magazines is necessary if the ATM is to avoid dispensing *'fresh air'* while debiting the ATM customer for the full amount.

Three non-functional goals are involved. These are reliability, availability, and safety. With regards to reliability, it is important that the need to represent the state of the cash magazines is prioritised if the non-functional sub goal of reliability (i.e. state representation completeness) is to be represented or promoted. This requires the functionalisation goal of ensuring a software state representation of the cash magazines. If this is functionalised then it directly promotes both: a) availability — in terms of promoting the sub goals of ATM customer access to required cash

amounts; and b) safety — in terms of protection against the ATM customer losing money — if the financial institution later does not believe that the ATM short changed the customer. Equally, if this functionalisation is not employed then it impacts on both availability and safety sub goals in a negative manner.

With regards to contextual and assumption issues, firstly, it can be seen that such an assumption will tend to emerge from the engineering context and possibly considerations from an evolutionary context perspective. Secondly, the assumption may well result from them restricting their responsibility to only software concerns believing that any state representation of the cash magazines is outside their scope of interest and resides with the system or firmware engineers (i.e. a separation of concerns issue). Either, that, or they may (perhaps) reasonably assert that the day to day servicing of the ATM will ensure that sufficient cash is always available. Again, it can be appreciated that such an assumption could be made in either an unquestioned implicit manner, or, even if explicitly considered, be used as a simplifying assumption — depending upon the engineering context's particular structure or culture, etc. In either case, it can be suggested that such an assumption can be viewed as a lack of a sufficiently holistic and integrative computer–based system perspective which fails to adequatly represent the negative influences and interdependencies of other important non–functional attributes of customer safety and cash availability. Finally, it can also be seen that such an assumption has an evolutionary dimension (in terms of servicing and maintenance), as while the assumption may be viable most of the time, it will periodically become invalidated when the cash magazine goes empty or so low that they cannot fulfil an ATM customer's cash withdrawal requirement.

8.4.2.8 Environmental Adaptation Issues 8 and 9

This issue relates to the changing temporal and geographical environments of ATMs. It has been an ongoing strategy of financial institutions to offer more ubiquitous and remote access to ATMs over the past 20 years or more. However, as the goal–analysis in figure 8.9 reveals this introduces important computer–based system dependability issues — in terms of a wider computer–based system con-

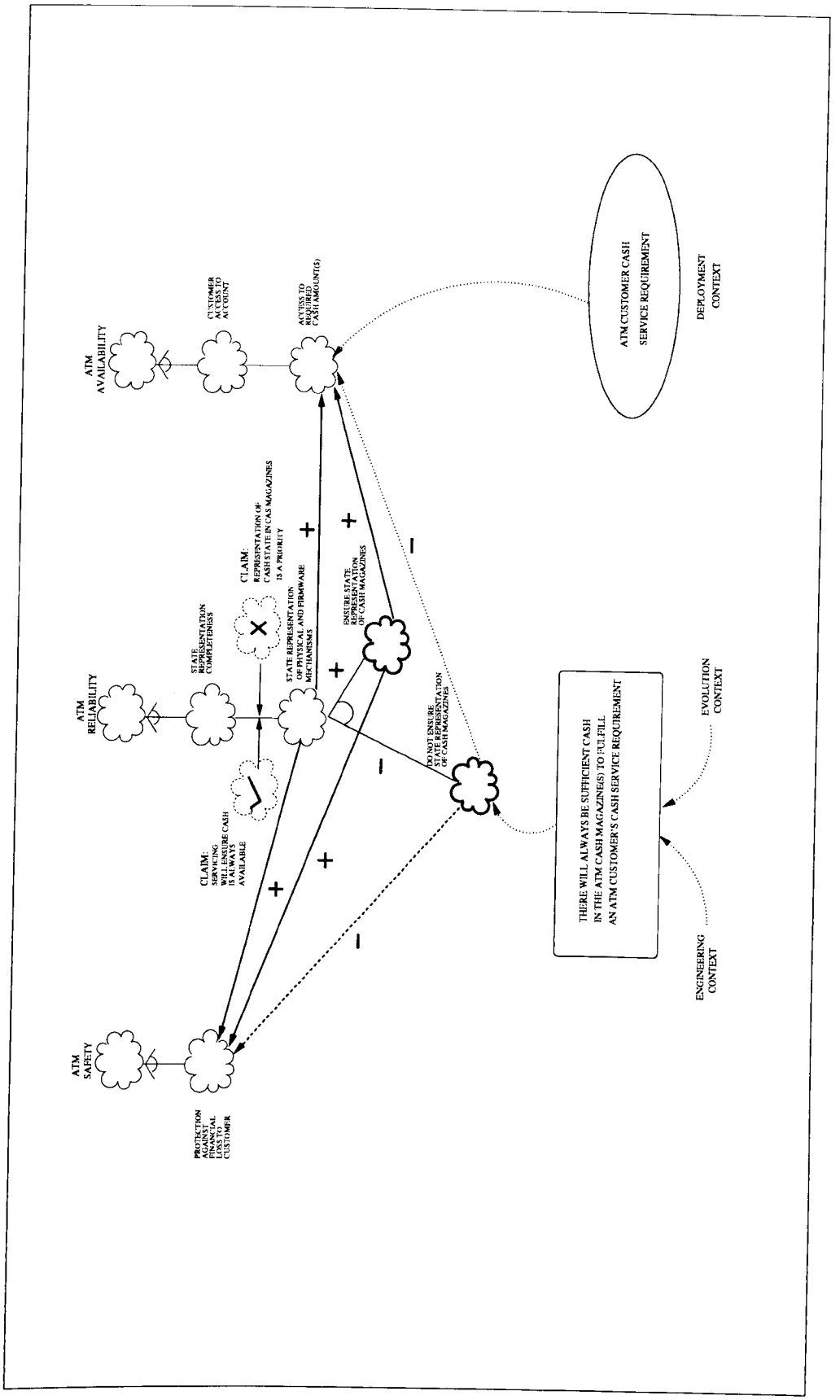


Figure 8.8: State Representation — Issue 7

ception that includes considerations of not only how different purpose ascriptions can be attributed to non–functional attributes, but also the changing deployment context issues within the specific ATM application domain.

The non–functional goals of interest involve availability, safety and security. In terms of availability, offering both ubiquitous 24 hour access and/or access at remote locations can undermine other important non–functional goals of: a) safety — regarding the increased potential for physical attack upon ATM customers to get access to their accounts; and b) security — as off–peak and/or remote ATMs are likely to become targets for gaining fraudulent access without being seen or looking suspicious by wearing disguises to circumvent any ATM cameras or CTTVs.

In order to help prevent this it would be necessary to introduce some functionalisation that reflects the changing temporal or geographical ATM environment by reducing or restricting access to ATMs in such off–peak or remote locations. This would reduce the potential for legitimate ATM users to use such ATMs and it would not necessarily impact unfavourably on ATM availability as most legitimate ATM users would rarely choose to use such ATMs at such times or locations anyway (i.e. hence both the plus and negative sign on the reduce/restrict functionalisation relationship). It would, however, impact favourably upon the non–functional sub goals of confidentiality of accounts (i.e. security against fraudulent account access) and protection of customer from physical attack (i.e. safety).

In terms of contextual and assumptive considerations it is obvious that the utility context would be motivated to explitly assert that ubiquitous ATM access is a positive customer service attribute as it gives an overt impression of proficiency for the financial organisation. Most the time, this is a complementary view of service availability when considering the more specific dependability definition of readiness of service. However, this explicit assertion underpins a different set of valuing criteria and priorities that make a different purpose ascription to the non–functional attribute of availability, by the utility context, when specifically considering the deployment nature of the ATM domain. Here, the interpretation, of

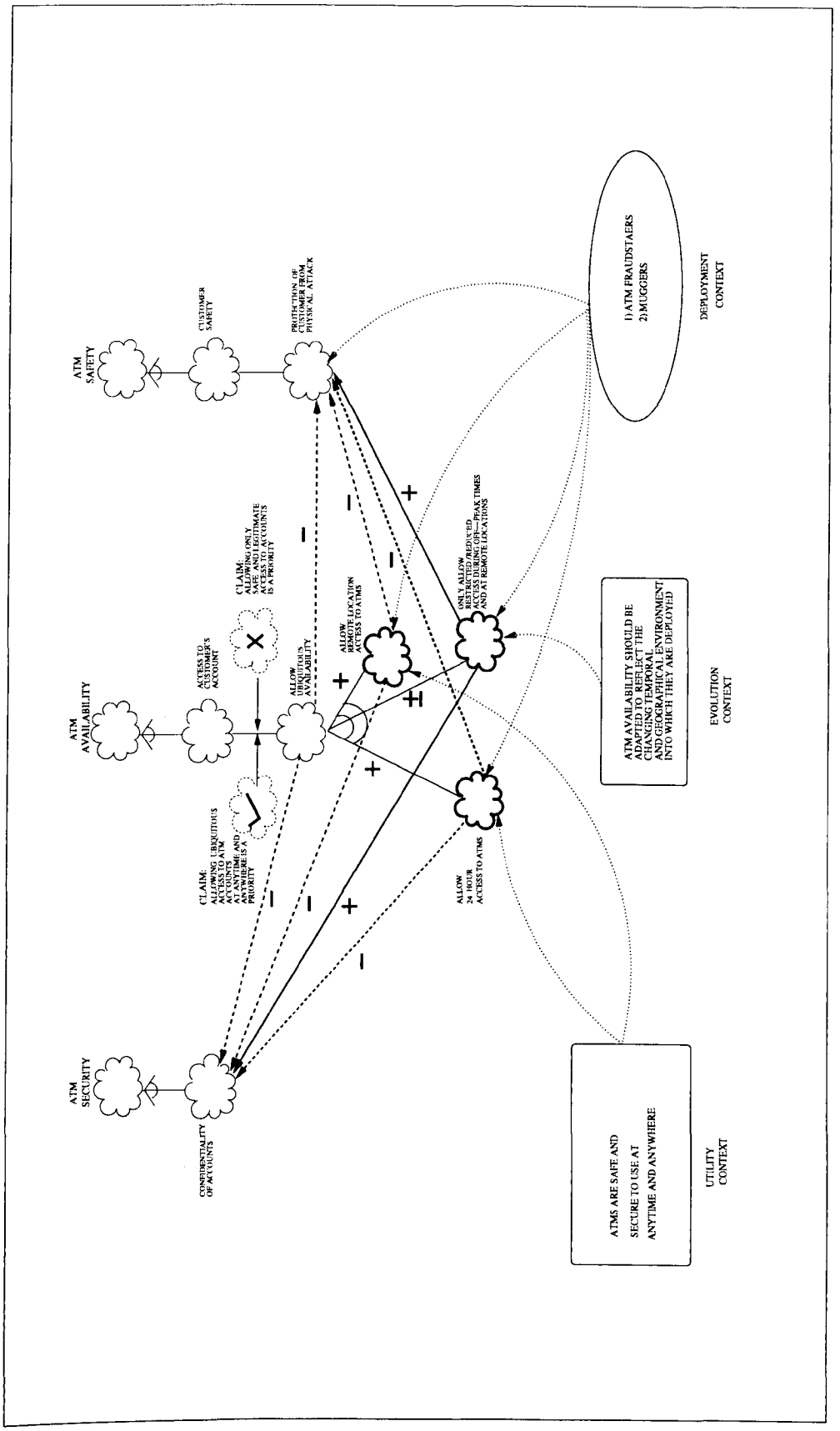


Figure 8.9: Attack and Fraudulent Access Concerns — Issues 8 and 9

ASSUMPTION NATURE / EVALUATED CAUSE	IMPLICIT ASSUMPTIONS	EXPLICIT ASSUMPTIONS	SHARED ASSUMPTIONS	INVALIDATED ASSUMPTIONS
UNDER REPRESENTATION OF NF GOALS	ISSUE 3 ISSUE 4 ISSUE 6 ISSUE 7	ISSUE 2 ISSUE 6 ISSUE 7	ISSUE 3 ISSUE 6	ISSUE 6 ISSUE 7
OVER REPRESENTATION OF NF GOALS	ISSUE 5	ISSUE 1	ISSUE 5	
DIFFERENT PURPOSE ASCRIPTIONS OF NF GOALS		ISSUE 8 ISSUE 9		ISSUE 8 ISSUE 9

Figure 8.10: Assumption Types and Evaluated Causes

promoting a positive commercial customer service imperative, can result in negative deployment consequences. The occurence of which, demonstrably reflects a failure to provide a more integrative and holistic computer–based system conception and evaluation of how offerring a ubiquitious service anytime and anywhere can negatively and undesirably impact upon important non–functional attributes of customer safety and account security when considering the ATM application domain. Furthermore, this is another example where assertive assumptions that *"ATMs are safe and secure to use anytime and anywhere"* is too encompassing and can subsequently become invalidated.

8.5 Chapter Summary

In this chapter a discussion of the expected beneficial effects of a goal–diversity process intervention has been presented to provide greater assumption detection coverage when considering the extended boundaries of a computer–based system.

Research on goal-setting has provided robust examples of how both behavioural and cognitive human performance can be increased. It therefore offers definite potential for improving the conceptual and motivational performance of human developers, as necessary input resources, within the software creation process. In order to improve the dependability of computer–based systems, however, requires a greater and more encompassing creation process–oriented approach that accommodates for such influences as different purpose and meaning ascriptions as well as different priorities and emphasises of issues during the creation of such software systems. Adopting a qualitative process–oriented view of non–functional attributes introduces characteristics that are not only important in representing such computer–based system influences, but are also intrinsically important to improving the dependability status of both the resultant artifact and creation process when they are used as a systematic driver of the many complex development decisions, choices, and trade–offs that are involved in any real–world software development situation. An existing non–functional requirements methodology was introduced and applied in this chapter to demonstrate the expected envisioned assumption detection benefits that could be expected from a goal–diversity process intervention. It was selected because of its suitability for representing, reasoning, and recording complex information visually in an elaborated form that would not be easily captured using a textual format. It is already well known from many years of research in applying this existing methodology that ensuring that non–functional attributes drive the software creation process in a systematic way significantly improves the eventual quality of the eventual software artifact. However the expected beneficial effects of a goal–diversity process intervention go further than this by overcoming some implicit shortfalls in this existing methodology that have been highlighted in previous chapters of this thesis. These include the inclusion of human redundancy and human diversity; improved likelihood of implicit assumption detection; reduced cognitive overload in having to consider non–functional attributes; and greater specialism, experience, and expertise in promoting non–functional attributes. The nine ATM issues from chapter 5 were then exemplified using a slightly adapted form of this existing methodology to demonstrate that, at the envisioned synthesis stage of the goal–diversity process intervention, greater overall computer–based system assumption coverage may be possible.

Figure 8.10 illustrates a summary table of possible flawed assumptions that could have resulted in reduced computer–based system dependability of ATMs when conceived as computer–based systems. The table highlights the broad categories of assumption types identified in chapter 6, along with the possible non–functional attribute failures of the creation process in not detecting them. Contained within appropriate sections of the table matrix is the specific ATM issues evaluated in this chapter and it reveals that a number of assumption types can be interpreted as the result of under/over–emphasising non–functional attributes or through different computer–based contexts–of–interest ascribing different meanings and purposes to them. Such examples, evaluated in this chapter, begin to suggest that a goal–diversity process intervention can help gain greater assumption coverage detection via adopting a qualitative process–oriented approach that simultaneously leverages human redundancy/diversity while representing and reasoning about non–functional attributes in a more systematic manner.

Chapter 9

Software Inspections and Physical Searching

9.1 Chapter Introduction

In chapter 8 a goal–diversity process intervention was proposed and discussed that utilises the characteristics of non–functional attributes to help detect harmful assumptions that underpin many development decisions that can compromise the overall dependability of the creation process, and ultimately, the eventual dependability of the created software artifact. As the ATM examples in chapter 8 reveal, this is particularly true when considering the wider system view of computer–based systems. As inspection plays a key central part in the proposed process intervention, it is important, as a pre–consideration for designing and developing a simulation model for chapter 10, to discuss the many issues and aspects involved in software inspections. Furthermore, since chapter 6 highlighted that assumptions are intrinsic to conceptual reasoning, and it can be argued that detecting them can be loosely related to a searching type task, a brief literature of the longstanding research area of search theory is also covered as it provides categorisations and terminology that can be considered useful in designing and developing a simulation model in chapter 10.

9.2 Software Inspections

As one of the issues covered in the previous discussion of chapter 8 concerned assumption detection, it is appropriate to briefly cover a well established and analogous detection process to detect software faults — known as *"Software inspections."* Software inspections of artifacts and documents has emerged as one of the most effective ways to help assure the quality of software artifacts.

Program code reviews , in the form of informal peer review, have long been practiced [64]. Software inspections, on the other hand, represent a much more structured and formal approach. Fagan [155, 156] is credited as providing the original underlying theories, structure, and research in this area. His original research found that a combination of design and code inspections resulted in a detection effectiveness of 82% of all known defects in the artifact [155]. Later research conducted reported a even higher 93% detection effectiveness of all known de-

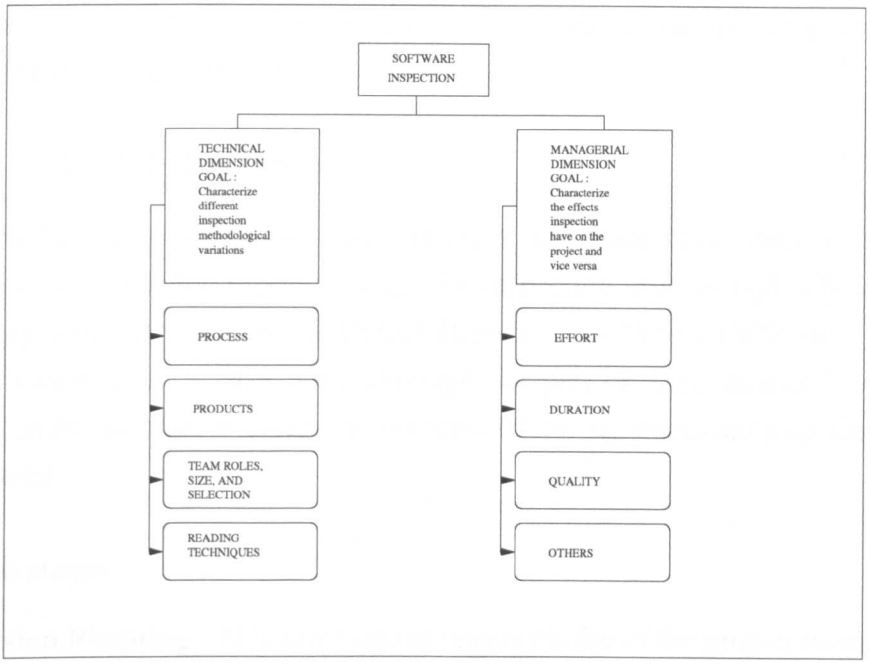


Figure 9.1: Software Inspection Taxonomy [source [157]]

fects [156].

Although initial research primarily focused upon inspections of design and the program artifact, the inspection process has expanded to include inspections of other *'upstream'* artifacts — such as problem definitions, customer requirements, and developer requirements connected with the software process phases of requirements engineering and specification.

9.2.1 The Inspection Process

In an attempt to structure and organise the software inspection phases into the broader software process to aid further inspection research and facilitate inspection planning in software development projects, Laitenberger and DeBaud [157] conducted an exhaustive survey of the software inspection literature — encompassing nearly twenty years of research and considering more than four hundred articles. In discussing the software process and related dimensions, this chapter adopts a subset of the structured life–cycle, concepts, and relationships taxonomy

they devised. A graphical subset adaptation of their taxonomy is reproduced in figure 9.1 on the page before.

9.2.1.1 The Technical Dimension

The first dimension to consider in the technical dimension of software inspections involves the inspection process itself. The inspection process typically involves four stages of: a) Planning; b) Defect Detection; c) Defect Collection; and d) Defect Correction. Furthermore, although not part of these generic inspection stages, an introductory overview and process follow-up stages are sometimes also introduced.

Process stages

Inspection Planning . This involves the responsibility of the project manager (or delegated technical expert) to organise, prepare, coordinate and control the subsequent stages. This may be in strict planning terms of specifying who and how many inspectors will be involved and how long and how many times the artifacts will be inspected, along with administrative features, such as, how the defects will be recorded, evaluated, and collated. Additionally, the planning phase may involve a technical overview phase where the artifact(s) to be inspected are prepared and explained to the inspection team in order to emphasise particular application domain issues and expedite learning and understanding of the artifacts. Gilb and Graham [158] referred to this planning phase as the *"kicking–off meeting"*.

Defect Detection . This is the central stage and represents the core purpose of the whole process. The purpose being to allocate both inspectors and time to carefully scrutinize the artifacts in order to identify potential defects. This stage may be performed by non–interacting inspectors working alone or performed collectively in the form of an inspection meeting. Additionally, there may be both an individual inspection stage (sometimes referred to as the preparation phase) and a subsequent meeting stage. Whether to have just one individual phase, or one collective meeting phase, or both, has proved to be a controversial issue (see subsection 9.2.2 on page 239) as it is difficult to determine which is the best configuration and to jus-

tify both must result in greater detection effectiveness as it introduces time/cost implications that can (in some cases) increase overall project costs and schedule extensions.

Defect Collection. Because, in most software inspection processes, more than one person participates it is necessary to collect and document the individual defect detections made by the inspectors once the defect detection stage is completed. This stage inherently involves the evaluation of individual defect detections made by the inspectors as: a) disagreements may be involved in describing a defect and this will need to be investigated; and b) it is common for individual inspectors to identify defects that actually don't exist (often termed false positives). One benefit of performing the defect detection stage as a collective meeting is that such problems can be quickly discussed and reconciled. If, on the other hand, the defect detection stage is performed individually, then correction can involve a variety of formats. One variety is to hold a highly structured collection meeting which is managed and focused upon the issues of defect evaluation and correction. Another variety is where only a few (more experienced) inspectors take part and is chaired by a moderator. Finally, the issue of defect evaluation and correction can be held in isolation through ongoing discussions and correspondence by remote communication methods (i.e. email etc). As with the defect detection stage, the issue of holding meetings can, in some instances, increase costs and extend delivery schedules.

Defect Correction. This is the last phase in the inspection process and involves the reworking and removal of defects detected, evaluated, and agreed as undesirable attributes in the artifact. An additional phase may involve a follow–up stage where the authors of defect correction activities are checked to see that defects detected and agreed have been resolved in the artifacts. This is sometimes performed by another inspection member or someone (i.e. inspection leader or moderator) appointed with the responsibility.

The products

As mentioned earlier, software inspection was originally focused on design and code artifacts. However research into the economics of defect detection and correction involved in the software development life–cycle [cf. Boehm [100]] identified that leaving the activity of defect detection and correction until late in the software development life–cycle can exponentially increase the correction/removal costs involved. As a consequence, software inspection has been introduced, in recent years, into *'up–stream'* software artifacts of requirements documents and requirements specifications also. Listed, defect inspections involve the following type of software life–cycle documents :-

- **Requirements Documents**;
- **Specification**;
- **Architecture Models Documents**;
- **Design Models and Documents**;
- **Implementation Code**;
- **Test Cases**;

It can be seen from this list that software defect inspections now encompass the entire span of possible software artifacts produced during the software engineering life–cycle.

Inspection team

One of the fundamental aspects that define an inspection approach from other informal code review methods (i.e. walkthroughs , etc) is the formalisation of team roles — in terms of what functions and responsibilities each bring to the software inspection process. Laitenberger and DeBaud [157] identify seven possible contributing team roles within the inspection team , as follows:-

1. ***Organizer:*** The organizer is charged with the responsibility of inspection planning;

2. ***Moderator:*** The moderator takes the role of technical leadership, within the inspection process, and is responsible for ensuring the procedures from each stage are followed;

3. ***Inspector(s):*** Inspectors, as the name implies, are responsible for scrutinizing, detecting, evaluating, and correcting software artifact defects;

4. ***Reader/Presenter:*** The reader or presenter is responsible for describing the artifact(s) to be inspected to expedite the learning and understanding of the artifact to the other inspectors involved;

5. ***Author:*** The author is responsible for the preparation and (any) corrections that are necessary for the artifact to be inspected. Additionally, he/she is responsible for supporting the reader/presenter if any questions are asked or posed by the inspectors;

6. ***The Recorder:*** The recorder is responsible for the logging of all defect detections made by the inspectors;

7. ***The Collector:*** The collector is responsible for consolidating all the defects detected by the inspectors.

It should be noted from this list that a member of the inspection team may be responsible for more than one role in the inspection process. This is obviously the case with inspectors as most the members of the inspection team will be involved in this role.

The size of the inspection team is a subjective issue and largely depends upon such factors as the size of the development product, the experience of the inspectors involved, and the commitment of the wider organisation to software quality assurance etc. A further, but related issue, is who to assign to the inspection team and what members of the team are best suited and sufficiently experienced to undertake what roles? Inspectors should be sufficiently knowledgeable of the software system being developed, and experienced, in the broader sense, in order to

help ensure the success of the inspection process. However, wider project and organisational issues of staff availability and training issues also constrain/facilitate such decisions. For instance, although inexperienced project members may extend inspection durations and reduce initial defect detection effectiveness, it has also been recognised that the software inspection process is a good opportunity to develop and train inexperienced staff.

Reading Techniques

Since artifact defect inspection depends upon the syntactic, semantic, visual, and logical understanding of textual/graphic/symbolic based documents (be there textual documents, design models, or code, etc) it has been recognised, within the software inspection research area, that reading/reviewing techniques play an important influence in both individual inspector defect detection and improving the performance, in terms of detection effectiveness and efficiency, of the inspection team — as a whole. The selection of a particular reading technique will largely be one of the responsibilities of the inspection organiser.

The following reading techniques make–up the many reading variations that have been developed in the field of software inspection over the last twenty years or so.

Ad–Hoc Reading: This is one of the most used techniques. As the term suggests, no guidance, rules, or support is provided with the software artifact — in terms of how the inspectors are to conduct the inspection of the artifact. Additionally, no inspection process intervention takes place — in terms of allocating additional or different artifact representations or information to certain inspectors. With this reading technique all the inspectors receive the same representations and information.

Checklist Reading: These are the next most popular. As the term suggests, a list of questions relating to important general issues are provided — in addition to the software artifact to be inspected. This extra information is the responsibility of the role of the author of the inspection team to produce. Whilst checklists provide extra information, their use has often been criticised as only providing very general

coverage and make demands of personal inspector experience and/or knowledge of the domain for effective illumination and detection of faults. Another criticism leveled at the use of checklists is that they can constrain exploration of the defect space down to only those on the checklist.

Stepwise Refinement Reading: This is a reading technique that is much more structured and requires greater specific representations and information. This approach requires the inspector(s) to abstract the function from the coded artifacts. This is repeated until all function points of the artifact have been covered. These are then compared with the specification to help detect any defects.

Active Design Review Reading: This reading technique also offers greater specific support to the inspector. With this approach, inspectors are given clear roles and responsibilities to ensure a more active, rather than passive involvement. Because of the active nature, this technique is inherently suited more towards collective inspection meetings than individual inspection effort. It consists of three steps: 1) an overview step; 2) a defect detection step — where the author ensures active participation through asking questions to guide the inspectors. These questions are carefully chosen to ensure that an in-depth elaboration and understanding of the artifact has been achieved by the inspectors in their answering. In some cases this will even require the creation of assertions or extracts of lines of code to demonstrate full understanding; and 3) defect collection is performed during the meeting. It should be pointed–out, however, that this technique breaks up the inspection meeting into smaller sub meetings also — where specialised consideration is also given to individual quality properties;

Scenario–Based Reading: More recently, research in inspection reading techniques have been focused into providing custom guidance in the form of detailed scenarios to limit the individual inspectors attention to the detection of particular defects — as laid–out in the custom–based scenario. Such an approach involves a significant process intervention, as inspectors are given deliberately different representations and information about the software artifact under scrutiny. However, such deliberate process interventions place a high reliance upon the qual-

ity, design, and content of the scenario representations and information provided. Scenario–based reading techniques have a number of subtle variances, as listed below:-

- ***Defect–Based Reading:*** This scenario-based variation is aimed at providing different inspectors with scenarios that focus their attention upon different defect classes relevant to the application domain concerned. This may also include highly customised sets of questions a given inspector must answer.

- ***Perspective–Based Reading:*** This scenario–based reading technique starts with the assumption that the software artifact under review should be scrutinized from the perspective or viewpoint of different stakeholder(s) involved with the software system. The underpinning reasoning for such an assumption is that there is no definite or single interpretation of what software quality represents. As a consequence, the viewpoint of what constitutes, for example, maintainable or secure etc, may be interpreted and prioritised differently. For each individual inspector a number of perspective–based scenarios are created and the inspector must focus upon this view in order to answer them. So far research on perspective–based scenarios have been mainly limited to defined roles and stakeholders within the software development process (i.e. maintainer, tester, etc).

- ***Functional–Point–Based Reading:*** This scenario–based reading technique generates scenarios in the format of functional point analyses — in terms of inputs, files, inquiries, and outputs. The functional point scenarios are combined with sets of questions that directs the attention of an individual inspector to that particular function.

N-Fold Inspection Reading: This reading approach makes the explicit assumption that when a large group of potential inspectors are split into subsets of smaller inspection teams, then inspection teams will detect different defects. N-fold inspections, in this sense, place a large emphasis upon the role of human redundancy in the inspection process to increase defect detection and overall dependability [159]. The approach is particularly advised, by their advocates, to be used upon

safety–critical applications where the cost of detection may well be outweighed by the consequent impact of such defects resulting in loss of life or catastrophic damage to the wider environment. However, its application employs an ad-hoc reading approach.

9.2.1.2 The Managerial Dimension

Inspection effort

The first consideration in this dimension concerns the amount of effort a software inspection process is likely to take. The amount of estimated effort must be considered against a number of other factors. These include: a) the inspection process effort against the negative effects it will have on project costs and schedule; b) the positive effects it will have on increasing overall software quality of the system — through (hopefully) lower residual fault levels (or densities).

By definition, software artifact inspection is a human activity. Therefore, the first main consideration for the project manager — concerning effort, is the number and experience of the inspectors involved in the inspection process. However, there is more to consider than just the issue that the more experienced the inspectors involved then more defects may be found more quickly, as although this may be the case, it is also true that the more experienced inspectors will (broadly) cost more money per unit of effort.

Other considerations that can impact upon inspection process effort are less controllable by the manager. These include the size and complexity (difficulty/novelty, etc) of the software system under development. The larger and more complex this is — then the larger and more difficult the software artifacts will be to inspect. This will not only impact upon defect detection effectiveness, but also directly upon the levels of effort involved.

Inspection Duration

A major management consideration in the inspection process is the interval duration that is created by the software inspection process activities. This can be

a critical consideration when the project is mission–critical to an organisations success or there is a strategic time–to–market criticality for delivery of the software system. This delay can be created by the inspection process because of: a) human resources (inspectors and other inspection process roles) employed in the inspection process cannot be performing other development work; and b) other human resources and software artifacts not involved in the inspection process may be dependent upon work to be done by these resources and/or completion of the inspected artifact before other development work can proceed.

In addition to inspection process duration being influenced by the number and experience of the inspectors involved, the organisation of the software inspection process itself can have a direct impact upon inspection duration. An obvious example is whether the inspection process involves collective inspection meetings to detect software artifact defects.

One further consideration, not explicitly covered in Laitenberger and DeBaud's [157] survey of software inspection, is the fatigue effect upon software inspectors. Biffl and Halling's [160] study of nominal software inspection teams, using diverse reading techniques, found, that as individual inspection duration progressed, inspection defect detection effectiveness tended to decrease.

Software Quality

In software inspection terms, the quality of the software system is directly related to the number of defects contained in the artifact (i.e. the defect density). Over the past twenty years or so, Laitenberger and DeBaud's [157] survey reports on the quality increases — through defect detection during the software inspection process over a wide range of sources. In terms of defect detection effectiveness , many of the studies report defect removal rates of between 50–90% of known defects being detected (one study reported only 38%).

Detection Efficiency

A final managerial consideration relates to the detection efficiency of the software inspection process. Defect detection efficiency relates to the number of defects found in proportion to the inspection effort duration. It is usually expressed as a ratio of defects per effort duration. In Laitenberger and DeBaud's [157] survey there appears to be a wide variance of defect detection efficiency ratios, with ratios reported from 0.67 hours per defect detected — up to 11.6 hours per defect detected.[1]

9.2.2 Software Inspection Process Loss Issues

Much speculation surrounds whether the defect detection is better suited to an individual or group activity. In Fagan's [155] original study of software inspection he argued that a group detection activity allows for synergy to take place during inspection through the interaction between group members. Fagan explains this effect as the *"phantom inspector"*. In conflict with Fagan's view, however, is the repeated finding that study after study of aggregated and/or group performance on software inspection tasks continually show that, quite apart from any synergizing effect (i.e. whole is greater than the sum of its parts), inefficiencies are introduced that makes group performance, in the defect detection task, diminishing in nature (i.e. whole is less than the sum of its parts).

To better explain these issues consider the diagram of three performance functions in figure 9.2 on the next page. In 1) the Additive Function, performance grows predictably and linearly as extra human resources are added. An example in software defect detection would be where one inspector detects ten defects, two inspectors detect twenty defects, and so on. 2) the Synergistic Function is the views originally held by Fagan [155]. An example in software defect detection is where one inspector detects ten defects, but two inspectors detect thirty defects, three inspectors detect fifty, and so on. The last example 3) the Diminishing Function is where there exists some inefficiency within the process that results

[1] However, this variance involves different reading techniques, different artifact products, different inspectors, and different phases in the wider software development life–cycle.

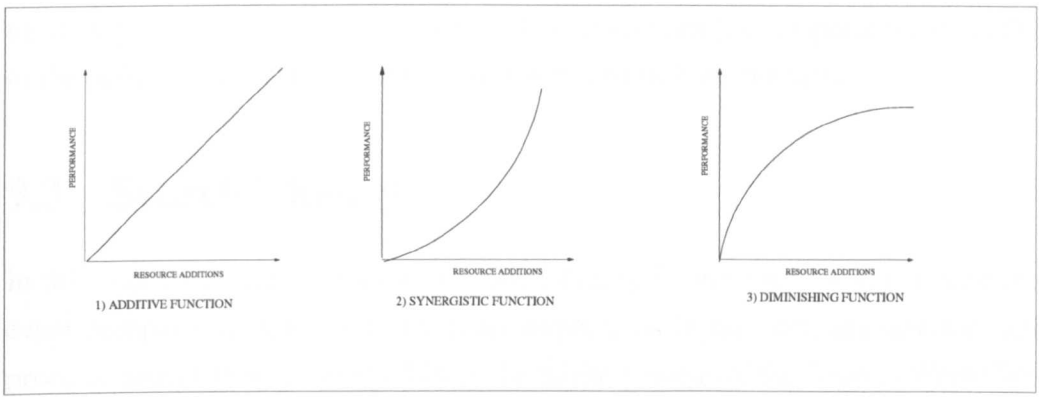


Figure 9.2: Three Group Performance Functions

in what is often termed *"process—loss"*. An example of this type of function is where one inspector detects ten defects, but two inspectors detect only eighteen, three inspectors detect between them only twenty four, and so on. With this type of function there reaches a point where adding extra human resources becomes virtually pointless.

In software inspection, research indicates time and again that the defect detection task results in a diminishing function [cf. [159], [160], [161]]. In order to better understand why such process losses occur it is necessary to understand the nature of the task and what is deemed a desirable feature or output that should emerge from the task. With software defect inspection, process losses occur due to individual inspectors detecting the same defects. So, for example three inspectors each detect ten defects each, but at the later defect collection phase, it is realised that inspectors 1 and 2 found two defects in duplicate, inspectors 1 and 3 found two defects in duplicate, inspectors 2 and 3 also found two defects in duplicate, and all three inspectors found one defect in triplicate. When we take these duplicate and triplicated detections into account, we can see that, although they detected ten defects each, they did not actually find thirty defects in total. When we calculate this out, in terms of unique defect detection, collectively, they only detected 15 defects uniquelly and duplicated/triplicated detections on 7 other defects. So only a total of 22 different defects were actually found between the

three inspectors — not 30.[2] As more human resources (i.e. inspectors) are added to the task, this overlapping of detection tends to further increase.

9.3 Search Theory

In this section a brief overview of search theory is provided — along with relevant comparisons with software fault inspections in the software development process. Search theory emerged from the military needs of the Second World War to improve U-Boat detection in the north Atlantic. It provides a long–established underlying theory and terminology which is useful in considering human redundancy and human diversity aspects within a fault detection context. The detailed survey by Benkoski et al [162] is used to provide the essential ideas, notions, and theories. However, other references are also used to provide more specific ideas and examples.

Subsection 9.3.1 provides a brief history of the emergence of search theory and the main pioneers in the field. Subsection 9.3.2 describes the ideas and terminology that relate to a specific branch of search theory known as one–sided searches. Subsection 9.3.3 provides additional search theory ideas and terminology relating to two–sided searches. Although the latter is not so directly relevant to the sections that follow, it is still a large and significant area in the search theory field and is therefore included for reasons of completeness. Subsection 9.3.4 then introduces the theories and terminology that will be introduced and used throughout the chapter 10 — along with limitations that must be acknowledged when considering conceptual, rather than physical searches.

9.3.1 Brief History

Search theory is recognised essentially as one of the generic mathematical models that have emerged from the area of operational research (OR). The underlying

[2]This can be calculated as follows: i) 3 (triplicated detections) - 2 (inspectors) = 1; ii) 12 (duplicated detections) / (2 inspectors detections) = 6; iii) 1 + 6 = 7 (triplicated/duplicated detections); iv) 15 (unique detections) + 7 (triplicated/duplicated detections) = 22 total different detections.

theories and terminology were initially developed from the necessities imposed by the Second World War in order to counteract the threat of German U–boats sinking merchant and military ships of the USA and UK in the North Atlantic.

This work was essentially undertaken by the US Navy's Antisubmarine Warfare Operations Research Group (ASWORG) in 1942. Bernard Osgood Koopman (cf. Morse [163]), along with a number of colleagues at ASWORG, are now largely credited with providing the pioneering theoretical work that launched search theory as a practical technique for conducting physical searches and undertaking search planning.[3] This early theoretical work was later collected together into a single publication authored by Koopman entitled *"Search and Screening"* (cf. Koopman [164]). Search theory recognises a number of important underpinning concepts in its application. These are:-

- a prior distribution on target object locations;
- a function relating search effort and detection probability;
- a constrained amount of search effort;
- and the optimization criterion of maximizing probability of detection — subject to a constraint on effort employed.

As well as military and search and rescue fields, search theory has been utilised to provide insights into many other application contexts. These include missile detection; oil detection for the petro–chemical industries, and mining, etc.

Historically, since its theoretical inception, practitioners have recognised that search theory has broadly gone through four generations. These are:-

1. **Classical Search Theory (1942-1965):** where search theory primarily focused upon one–sided physical searchers;

[3]Search Theory today is not only used by the military, but is also practised by a number of civil organisations — such as mountain rescue, coastguard services, etc.

2. **Mathematical Search Theory (1965–1975):** where mathematical analysis was improved and extended. This included the introduction of two–sided search–theoretic issues also;

3. **Algorithmic Search Theory (1975–85):** In which search theory was improved and defined into identifiable proceduralisations that could be performed on computers;

4. **Dynamic Search Theory (1985–present day):** with the advent of more sophisticated computer hardware/software, search theory was extended into the fields of sophisticated simulation and decision–support modelling to improve the utility of its application and increase and extend the boundaries of knowledge of the field generally.

9.3.2 One–Sided Searches

One–sided search theory is essentially concerned with searching for a target object that is *'passive'* in nature. This means that the target object of interest in the search does not react to the searcher in any intelligent way in order to avoid or prevent detection. In this search situation the goal of the search is normally to maximize detection of the target object of the search and minimize cost/time involved — in terms of effort expended in detecting the target object, by the searchers.

However, although the target object is passive in one–sided searches, it need not be a stationary object and target objects may be either mobile or static. With a stationary target object, the problem is to allocate an optimal amount of effort to detect the target object. Secondly, with stationary target objects, an exponential detection function is assumed. Frost [165] highlights that actually there are three detection functions possible in one–sided searches. These are shown in figure 9.3 on the following page.

The linear function called the *"definite range"* function represents a situation where the search process can be highly controlled and can be factored out with resources employed to achieve a 100% detection at 100% coverage of the search

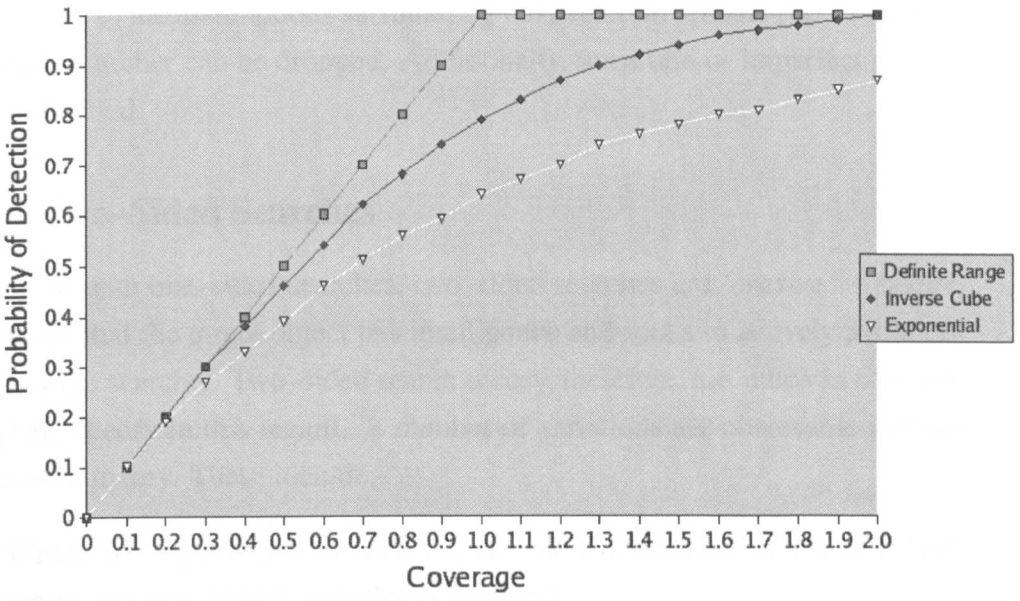


Figure 9.3: Probability of Detection Functions

area. The exponential search function reflects a search situation which, by contrast to the definite range search, has uncontrolled negative influences. This variation results in the search resulting in diminishing returns over larger and larger coverages of the search area. The inverse cube search function reflects a search process which is (in Koopman's own words) *"a middle case"*, that has been empirically observed in many physical search situations.[4] The inverse cube search function approaches 100% detection at search space coverages of about 200%. It can be clearly seen from the graph in figure 9.3 that while the inverse–cube search is as effective at detecting objects over larger coverages, it is not as efficient upon finite search resources employed as the definite range search function.

One–sided searches have a number of extensions when later generation search theory research is considered. Three later directions have been concerned with: a) changing the goal of the search; b) target objects can be in multiples and the assumption of persistence of the object can be disregarded (i.e temporal target

[4]One example is the detection of U-boats and warships from the air, but many other examples also exist

objects); and c) the assumptions surrounding the detection sensors (i.e. searchers) of a single searcher can be dropped. Additionally, uncertain or imperfect sensors can be modelled.

9.3.3 Two–Sided Searches

By contrast with one–sided searches, two–sided searches are *'evasive'* in nature. This means that the target object has intelligence and seeks to actively avoid detection by the searcher. Two–sided search theory, therefore, has much in common with game–theory in this regard. A number of variations are observable in two–sided search theory. These include:-

- Where the target object has either a) complete information of the searcher's search strategy or b) is completely ignorant;
- The actual pay–off functions can vary from the probability of detection, to probability of capture, to time to capture or detection, etc;
- The target object may be mobile or immobile. In the later case, the target object can only evade the searcher by choice of initial location or position in the search space;

A number of search games are well known and exemplify two–sided search theory. One in particular is the *"Princess and the Monster"*. As the title suggests, the princess is the pursued object and the monster is the pursuant searcher. The princess must try and avoid capture by the monster and, as listed above. A number of variations on the search theme exist.

9.3.4 The Search Process

Whether one-sided or two sided searches are involved, the search process usually involves three factors: a) the search environment ; b) the sensors (i.e. searchers) ; and c) the target objects (cf. Champagne [166]). In addition, the goal of the search process is to attempt to optimize the pay–off function . While these can vary between different search situations there is usually a focus upon maximizing search

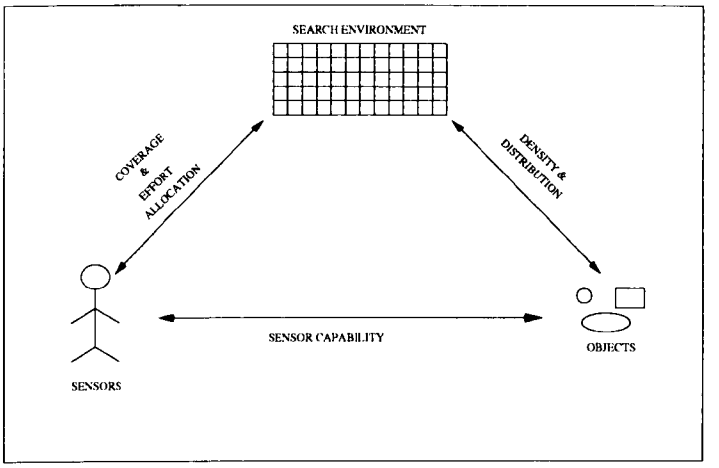


Figure 9.4: Essential Factors in Search Process

effectiveness (i.e. number of target objects detected) and minimizing search effort involved (i.e. search effort translates directly in *'real–world'* searches into time and cost but can also have safety critical considerations — such as how quickly can a person lost at sea be recovered (cf. Frost [165]). A search strategy that is as effective as another search strategy, but uses less effort will normally be considered the better strategy as it is more search efficient.

In figure 9.4 the main factors of the search process are illustrated along with direct relationships that exist which are of interest in this chapter. These, along with the terminology, will be briefly discussed.

The search environment is a fundamental factor in any search situation. It is within this search space that the target objects will be contained. The distribution and density of the target objects directly affect search planning since locations or regions within the search space that contain larger densities than others reflects an uneven object distribution and searching within these particular regions or locations will, in preference to others with lesser densities, have a favourable detection effectiveness and efficiency outcome.

The issue of the sensors is also an important factor. A key issue, in relation to the target objects, is how sensitive or capable are they in detecting the target ob-

jects? In search theory, sensors (or searchers) can range from perfect through to oblivious at detecting target objects. Another issue directly associated with the sensors is how much effort they allocate. Such constraints on effort will directly affect how much coverage of the search environment can be undertaken. Coverage is a key issue in search theory which, at one end, is directly influenced by particular search strategies, search paths, and search constraints, and at the other end directly influences the eventual search process functions of detection maximization (i.e. detection effectiveness) and search effort resource minimization (i.e. detection efficiency).

9.4 Chapter Summary

In this chapter the issues and aspects of software inspections and search theory have been covered and discussed. Software Inspection has a reasonably long history within the software engineering literature, and has proven, in many cases, to be instrumental in improving the overall quality of software artifacts, through the dependability means of fault–avoidance (e.g. detection and removal). Software inspection fundamentally incorporates a technical and managerial dimension. The technical dimension involves consideration of: a) ***process*** — the many process responsibilities for planning, detecting, collecting, and correcting software faults; b) ***products*** — although, initially, introduced to detect code faults, the value and success of software inspection for detecting faults, has resulted in its application in many other life–cycle phases such as, requirements engineering, specification, design, and even test cases, also; c) ***team roles*** — since software inspection is essentially a human activity, various team roles and responsibilities have been devised over the years; and d) ***reading techniques*** — there exists a wide range of approaches to characterising, representing and emphasising how the syntactical, semantic, visual and logical information can be interpreted and assimilated during inspection. It has been noted that such predispositioning intervention can play an important role in improving detection effectiveness and efficiency during inspection. The managerial dimension involves consideration of: a) ***effort*** — how much effort, and by whom, is always an ongoing and important resourcing consideration for inspection management, as it can impact positively or negatively upon overall

project schedule and/or budgets; b) ***duration*** — the length of time to allow for inspections is always an important managerial issue as the needs of one project must be balanced–off against other organisational demands and includes issues of whether a particular planned inspection process should be individual and/or collective; and c) ***quality*** — the overall impact, in terms of value, of including an inspection phase must always, in the end, be judged both on its immediate quality impact on the software artifact, and also its longer term positive *'downstream'* creation process impacts on improving the managability of the overall creation process. Search theory has went through a number of advancements — since it was first founded during the Second World War. The area primarily considers physical searches, although, due to its longstanding history, search theory has produced a useful structured set of categorisations, terminology, and notations that make it highly useful in preliminarily considering what factors and relationships are useful in designing and developing a future simulation model for chapter 10.

Chapter 10

Search Simulation Model

10.1 Chapter Introduction

In this chapter a search simulation model is used to suggest and indicate the potential assumption detection benefits that may be possible via the employment of diverse development goals. Section 10.2 draws upon issues raised in the previous eight chapters in order to provide a design rationale for a simulation model that can capture a feasible subset of the influences of diverse goals and their consequent effects upon assumption detection. Section 10.3 considers the relevant verification and validation issues along with the more definite concrete simulation process issues. Section 10.4 introduces and justifies envisaged modelling configuration issues along with a sensitivity analysis. Section 10.5 then performs a number of simulation experiments to provide an initial indication of the assumption detection benefits that may result from such a goal–diversity process intervention.

10.2 Design Rationale

In this section a design rationale is provided for the simulation model. In providing such a rationale it is necessary to make as explicit as possible the consequences and impacts for the simulation model's eventual design by unearthing the following three aspects: i) what are the alternatives? ii) which alternative was chosen? and iii) why the alternative was chosen from the set of alternatives. However, as Chwif and Paul [[167]: p. 450] stress, with regards to removing simulation model complexity:-

> *"Since modeling is an abstraction of reality, model results (not the model itself) should be close to reality, not exactly the same. Who needs the complexity of reality in the model when that complexity was what started the project (to begin with)?"*

Therefore, implicitly, when providing a rationale, sometimes the design alternative chosen will be justified upon the basis of removing infeasible and unnecessary model complexity that would detract from, and undermine, the overall confidence in the simulation model's eventual output results. Next, it is not always possible to illuminate a clear set of design alternatives or a clear set of criteria for favouring

one design alternative over another when more than one alternative exists. In such cases, it is inevitable that the design selection decision is, somewhat, arbitrary in nature. In these design situations the arbitrary nature of the design decision will also be made explicit. Finally, the role of the overall objective or goal of the simulation model is also an important factor in favouring one design alternative over another.

Robinson [in: Chwif and Paul [167]] notes that, in considering the design possibilities of a simulation model, there are two fundamental dimensions that need to be considered. The first is simulation model scope, in terms of the breadth of phenomenon to be modelled.[1] The second fundamental dimension involves the level of model detail, in terms of the depth of phenomenon to be modelled.[2] It can be appreciated that two (or more) simulation models can have the same model scope but have very different levels of detail. Moreover, it can also be appreciated that when considering model scoping design alternatives, the overall simulation model's objective(s) or goal(s) play a significant role in providing selection criteria for justifying eventual design alternatives. The role of the simulation model's objective(s) and its impact and consequence upon modelling scope is provided in subsection 10.2.2. Finally, in terms of level of modelling detail, the major selection criteria for justifying the eventual design alternatives relate to the need to remove infeasible and unnecessary complexity that can undermine eventual confidence in the simulation model's outputs. The role of removing infeasible and unnecessary complexity and its impact and consequences upon the eventual level of model detail is provided in subsection 10.2.3.

[1]Chwif and Paul provide the example of a manufacturing system. With such a system, modelling scope would concern whether to model the entire facility or just one work–center. If the entire facility were to be simulated, this would be taking a very wide modelling scope. If, by contrast, only one work–center process were to be simulated, this would, comparably, be taking a much narrower modelling scope.

[2]To continue the Chwif and Paul example with a manufacturing system, if only a single work–center facility were to be modelled, level of model detail would be concerned with whether to model: 1) processing times; 2) breakdowns; 3) shift patterns; and/or 4) material handling equipment, etc.

10.2.1 Model Goals

As mentioned earlier, in section 10.2, the simulation model's overall goals provide a significant set of selection criteria for helping determine model scope. In this section the various approaches to promoting non–functional attributes (including the proposed goal–diversity process intervention) discussed and illustrated in detail in chapter 8, are rationalised in terms of what is both desirable and feasible to simulate.

It can be seen from chapter 8, that, including the proposed goal–diversity process intervention, there are three approaches to considering non–functional attributes during the software creation process. These are:-

- **Ad–hoc Consideration of Non–Functional Attributes:** where there are no systematic aspects or process intervention involved. Non–functional attributes, and their consideration, rely solely upon the arbitrary expertise, professionalism, and knowledge of the developers concerned. At best, the nature of the application domain and/or the particular system being developed may raise certain non–functional attributes as being important,[3] in the minds of the developers involved, but this does not constitute a systematic or interventionist treatment of non–functional attributes;

- **Systematic Consideration of Non–Functional Attributes:** where non–functional attributes are explicitly considered. This may be a process orientated or product orientated approach that provides for early and continued representation throughout the development process;

- **Goal–Diversity Process Intervention:** where non–functional attributes are not only promoted in a systematic and explicit process–orientated manner, but unlike existing systematic approaches, the process intervention is geared towards utilising the nature of non–functional attributes to justify human redundancy through generating human diversity to increase assumption coverage during the earlier phases of the software development process.

[3]For example, in such domains and systems as nuclear process control, safety will be naturally raised as a priority. In real–time domains and systems, performability will be naturally raised as a priority, etc.

From these distinctions it can be seen that the first goal of the simulation model is to:-

> COMPARE THE HUMAN DIVERSITY BENEFITS OF THE EXISTING APPROACHES (OF CONSIDERING NON–FUNCTIONAL ATTRIBUTES) UPON ASSUMPTION DETECTION COVERAGE UNDER AN CONSTANT HUMAN RESOURCING CONSTRAINT.

With this first goal of the simulation model, it is important to note that assumption detection effort will be factored out under a constant human resourcing constraint to determine the diversity effects upon assumption detection process loss issues highlighted in consideration of software inspections in section 9.2.2 in chapter 9. As the literature from search theory indicates from chapter 9, section 9.3.2, it is a useful baseline to ensure that the constant human resource constraint represents (a theoretical) 100% effort coverage of the search space to be searched, as this allows a logical comparison between the different approaches being considered. Therefore, this human resource constant will represent a 100% human effort coverage, capable (in theory) of searching the entire search space.

The second simulation model goal is a follow on from the first goal, in that, should the simulations reveal that a significant increase in assumption detection is possible with the goal–diversity process intervention, then to what extent is it useful to factor–out an overall human resource constraint? It is hypothesised that an overall resource constraint can be factored–out, up to, but not exceeding the number of useful goal predispositions possible. Once this limit has been reached, there will be a gradual loss in assumption detection effectiveness as developers begin to over–represent a given non–functional attribute and therefore begin to detect the same assumptions. Additionally, consideration of this issue works in reverse also, in that, in terms of considering dependability as a super–ordinate goal (as discussed in chapter 4), if important non–functional attribute goals are not represented then this will result also in reduced assumption detection through under representation. The second goal of the simulation model is therefore stated as follows:-

COMPARE THE ASSUMPTION DETECTION EFFECTS OF THE PROPOSED GOAL–DIVERSITY PROCESS INTERVENTION WHEN THE ACHIEVABLE NON–FUNCTIONAL ATTRIBUTE GOALS ARE BOTH OVER AND UNDER REPRESENTED UNDER A 100% RESOURCE ALLOWANCE COVERAGE.

These two goals of the simulation model provide a focal reference point for the later scoping and level of detail design decisions in subsections 10.2.2 and 10.2.3 below.

10.2.2 Model Scoping Decisions

In this subsection the high–level breadth or scoping design decisions are considered and determined for the simulation model. These will be in the format of: a) consideration of what alternatives are available; b) which alternative was selected; and c) why that alternative was selected from the available options. It should be noted, however, as stated earlier, the overall goal(s) of the simulation model are also an important criterion for favouring a particular option. Furthermore, in decision situations where there is no clear criterion for favouring one alternative over another, then an arbitrary selection will be made and this will be made explicit.

What stage(s) of the proposed goal–diversity process intervention will be simulated? It can be seen from chapter 8, that there are three envisaged stages to be considered for possible simulation.

Briefly, the simulation modelling *alternatives* are:-

- **Stage 1. Individual Analysis Stage:** Where each of the developers perform a separate analysis of the proposed system while being individually predisposed to promote a single non–functional attribute goal;
- **Stage 2. Individual Inspection Stage:** Where each of the developers, still predisposed to promoting the same non–functional attribute goal, separately inspect the other developers' analysis from stage 1;

- **Stage 3. Collective Meeting Stage:** Where all of the involved developers, still individually promoting there own non–functional attribute goal, meet and collectively contribute to any breakthrough decisions[4] or negotiate inevitable trade–offs.

Of these *alternatives*, the simulation modelling *selection* made is to only simulate stage 2 of the Individual Inspection Stage. The *rationale* for such a selection is that: a) it is not necessary to simulate stage 1 of the Individual Analysis Stage as this only produces the various assumptions and the goal(s) of the simulation model are to determine the assumption detection benefits of human diversity. Stage 1 can therefore be provided via the configuration of the simulation model to allow the creation of the assumptions for detection; and b) it is not really feasible to attempt to simulate stage 3 of the Collective Meeting as, although this stage is important in providing assumption detection through the synergistic negotiating and interaction of the developers, attempting to simulate such group and team dynamics would introduce too much complexity and subjectivity which would compromise the confidence in the eventual simulation results. Consequently, only the Individual Inspection Stage 2 will be simulated as this relates directly to the simulation model's goal(s) of assumption detection benefits from human diversity, but does not need to consider any group or team dynamics.

What phase and what products of the software engineering life–cycle does the simulation model relate to? From considerations of the software inspection issues in chapter 9, subsection 9.2.1.1, it can be seen that there are a number of *alternatives* possible. First, in terms of the software engineering life–cycle, it is possible for inspections to take place at many phases, including, the requirements engineering phase, the specification phase, the high–level architectural design phase,

[4]Breakthrough decisions reflect situations where seemingly intractable design conflicts are simultaneously resolved. A good example from World–War 2 is the Russian T34 tank. Previously, tank designers where unable to resolve the two important conflicting dimensions of mobility and protection. To optimise protection the tank must have extra armour, however, this increases weight significantly and has a restrictive effect upon the tank's overall mobility in battle — which is also critical. The Russian T34 tank overcame this, to a great extent, by being the first tank design to utilise sloping armour, which simultaneously offered greater protection with less actual overall weight. Today, all modern tanks employ sloping armour for this reason.

the lower–level module design phase, the code implementation phase and testing phase (e.g. generation of test cases etc). Secondly, regarding the various software engineering products, there are domain requirements documents, specifications, architectural representations, module designs, implementation code, and test cases. Obviously, from this, it can be seen that, broadly, there is a one–to–one mapping between the phase of the software engineering life–cycle and the product it produces. The *selection* of the process and products the simulation model is to representively simulate is the *'upstream'* software engineering life–cycle process phases and products of the requirements phase, the specification phase, and high–level architectural design phase. The *rationale* for such a decision relates to the overall goal(s) of the simulation model as the goal(s) wish to consider the assumption detection benefits of human diversity. While harmful assumptions may be made in any of the software engineering life–cycle phases, it is expected that it is in the phases of requirements engineering, specification, and architectural design that, because of their intrinsically conceptual nature, the most harmful assumptions can be made and the greatest dependability benefit achieved from any such process intervention that can improve assumption detection. Furthermore, the simulation modelling goals, quite specifically, relate to non–functional attributes, and it is widely accepted (cf. chapter 8) that their consideration early–on in the software engineering life–cycle brings the biggest reward, in terms of dependability and quality.

What stages of the generic software inspection process does the simulation model incorporate? With regards to the technical dimensions in subsection 9.2.1.1 in chapter 9, of the generic software inspection process, there exists a number of *alternatives* to consider. These include: a) the planning stages; b) the detection stage; c) the defect collection stage; and d) the defect correction stage. The *selection* made is to only simulate the detection and collection stage, but, in order to configure the simulation model to simulate these stages, it is also necessary to allow the simulation model certain features of the planning stage. The *rationale* for such a design decision is twofold. Firstly, in terms of allowing flexible configuration to simulate different approaches to promoting non–functional attributes, which is an important simulation modelling goal, it is necessary to allow certain

inspection planning activities such as number of inspectors, the duration of effort, etc. However, these features are only a subset of the normal software inspection stage and such administrative and specific software inspection features — such as the *'kick–off'* meeting, how defects will be recorded, and facilitation of domain understanding, etc are all precluded in the simulation model as such planning activities are unnecessary and can be abstracted from. Secondly, since the simulation model's goal is to focus upon the assumption detection issues, only the detection and collection stages will be actually simulated in the simulation model, it is an arbitrary decision not to include the defect correction stage as it could be argued that certain human–error influences of misinterpretation and misjudgement of what consistutes a defect are important, particularly with certain non–functional attributes, such as security, and consideration of false positive and false negative defect judgements. However, it is decided that to include such human–error issues could overly complicate the simulation model and that this simulation model will only consider defect sensitivity and not defect judgement phenomenon during assumption detection. Furthermore, as will be discussed in considering the more specific level of detail design decisions in subsection 10.2.3 below, such human–error aspects, during detection, are highly subjective and result in indeterminate modelling relationships with other modelling concerns.

What equivalent team–roles, of the generic inspection process, does the simulation model incorporate? In considering what team–roles should be represented in the simulation model, it should be first pointed out, as already stated earlier in this section, that the simulation model does not consider the interactive nature of the collective meeting stage (i.e. stage 3) in the simulation, and will only consider the aggregated individual inspection stage (i.e. stage 2). Therefore when considering the team roles of the generic software inspection process in subsection 9.2.1.1 in chapter 9, these are interpreted more as individual task roles that should be considered in terms of model scope. It can be seen that there are various *alternative* task roles of: a) the organiser; b) the moderator; c) the inspector; d) the reader/presenter; e) the author; f) the recorder; and g) the collector. It was decided only to include the task roles of the inspector, the recorder, and the collector would be *selected*. The *rationale* for doing so is, again, twofold. First the

role of the recorder is to log assumption detections and the role of the collector to determine unique and duplicated detections is functionality that the simulation model will need to incorporate into its internal logic. Secondly, since the goal of the simulation model is to determine the assumption detection benefits in combination with human diversity, it is obviously necessary that the role of the inspector would need to be incorporated into the simulation model. The other roles reflect the more *real–world* inspection environment (e.g. such as the reader/presenter for facilitating domain understanding, etc) which can be considered out of scope for the simulation model.

In summary, therefore, in terms of scoping decisions, we can summarise the scope of the simulation model in the following explicit rationalised terms:-

- Only the second stage of Individual Inspection of the proposed goal–diversity process intervention will be simulated;
- Analogically, the process phases and products inspected relate to the more highly conceptual phases of the requirements engineering phase, the specification phase and the architectural phase, as it is in these phases that harmful assumptions are more likely to occur;
- The simulation model's scope will accomodate, for configuration purposes, a subset of the planning stage of the inspection process along with other inspection process stages of inspection and collection;
- The simulation model's functionality will incorporate inspection task roles of the recorder, the collector, and the inspector;

These explicit simulation model scoping decisions now determine a justified and narrower breadth of modelling concerns which can be elaborated, in greater levels of detail in section 10.2.3, below.

10.2.3 Level of Model Detail Decisions

In this section the simulation model's design will be elaborated in depth within the overall model scope determined in section 10.2.2 above. Again, the overall

design decisions made will be of the format: a) consideration of what alternatives are available; b) which alternative was selected; and c) why that alternative was selected from the available options. It should be noted, however, as stated earlier, that the overall goal(s) of the simulation model are also an important criterion, at this detailed level, for favouring a particular option. Furthermore, in decision situations where there is no clear criterion for favouring one alternative over another, then an arbitrary selection will be made and this will be made explicit.

As discussed, by Chwif and Paul [167], a model is a simplified abstraction of reality and should be: *"...close to reality...not exactly the same"*. Therefore, at this level–of–detail modelling, it is always necessary to simplify the model in order to remove complexity that would render the model unnecessarily subjective and reduce confidence in its results. In order to aid in demonstrating the actual complexity involved in such a simulation model, so that such unnecessary complexity can be omitted, in performing the simulation model's initial analysis, the author used a conceptual modelling format known as an *Influence Diagram* — as used and demonstrated by Robson [168]. Robson justifies its usage in such detailed modelling situations, as follows [cf. [168]: p. 14]:-

> *"...the influence diagram is particularly useful because it can display all the decisions which need to be made, in addition to distinguishing between the input, process and output variables which have been identified."*

Input variables are those variables that can be measured or controlled by the simulation model user. They reflect the simulation model's input parameters which can be changed and reconfigured — depending upon the simulation model's purpose for simulating. The intermediate variables are those that can be indirectly calculated or influenced from the input or other intermediate process variables. They essentially capture the simulation model's internal behaviour. The output variables are those variables which will be used to make judgements and/or decisions about the phenomenon being simulated.

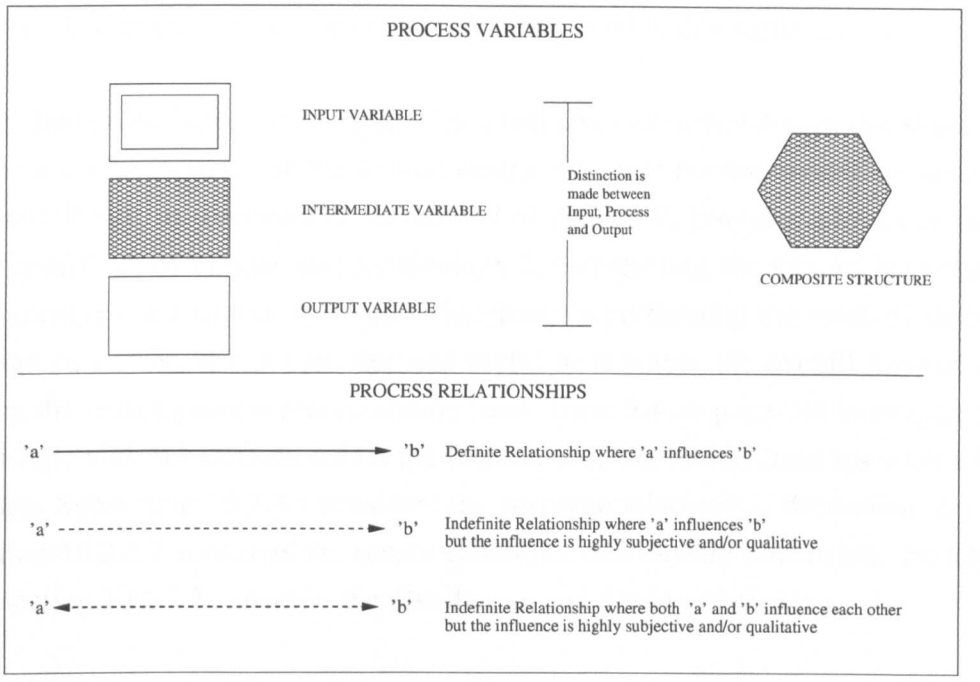


Figure 10.1: Influence Diagram Notation Used [source: Robson [168]: p. 14]

In addition to displaying design decisions and distinguishing between input, process, and output variables, Robson notes that the influence diagram also captures the relationships that exist between the various types of variables, and emphasises, that in developing a model, the nature of the relationships is a fundamental simplification criterion for deciding what can be feasibly modelled in order to remove unnecessary subjectivity and complexity. Robson stresses that there are, in principle, two types of relationships. First, there are definite relationships. These are relationships between variables that have a more obvious and objective relationship that can be more easily quantified. By contrast, secondly, there are indefinite relationships. These are relationships that contain much subjectivity and variability and are, consequently, much more qualitative in nature.

In considering the simulation model's level–of–detail design, the use of influence diagramming will be used and, as a selection criterion, the simulation model's design decisions may also be determined by simplification considerations — concerning the indefinite relations between variables. Figure 10.1 is a diagramatic

key to the particular influence diagram notation used in this section.

As a final point, before undertaking the actual level–of–detail design decisions, it will be useful to structure the overall design rationale for this modelling scope. Search theory, as discussed in section 9.3 of chapter 9, provides a longstanding and useful set of notions and terminology for structuring the various important relationships and factors involved. Therefore, in performing the level–of–detail design rationales, it has been deemed useful to structure the overall discussion using the search process representation from figure 9.4 on page 246 in chapter 9. The following subsections reflect the fundamental search relationships from this figure. Subsection 10.2.3.1 considers the coverage relationship dimension. Subsection 10.2.3.2 considers the sensor capability relationship dimension. Finally, subsection 10.2.3.3 considers the distribution and density dimension.

10.2.3.1 Coverage Dimension

In this section the many level–of–detail design decisions relating the searcher to the search space will be considered. As noted in subsection 3.3.1.1, in chapter 3, coverage, in dependability terms, is a critical aspect for ensuring that the situational representativeness of the computer–based system is sufficiently close to the actual situations experienced during operation. As shown in chapter 8, when these do not closely match, it can often be the result of some flawed assumption being made about the system or domain in question. Therefore, since the simulation model's goal(s) concern the assumption detection, consideration of assumption coverage is an important simulation modelling dimension to consider. This section documents and discusses a previous analysis performed, by the author, and illustrated using the influence diagram notation in figure 10.2.

The diagram shows four input variables of: Inspection Duration; Searchers; Locations; and Process Intervention. There are also ten intermediate variables of: Inspection Expertise; Location Difficulty; Searcher Learning; Searcher Selection Bias; Searcher Fatigue; Location Comprehension; Searcher Location Memory; Searcher Location Selection; Search Space Size; and Location Selection Rate;

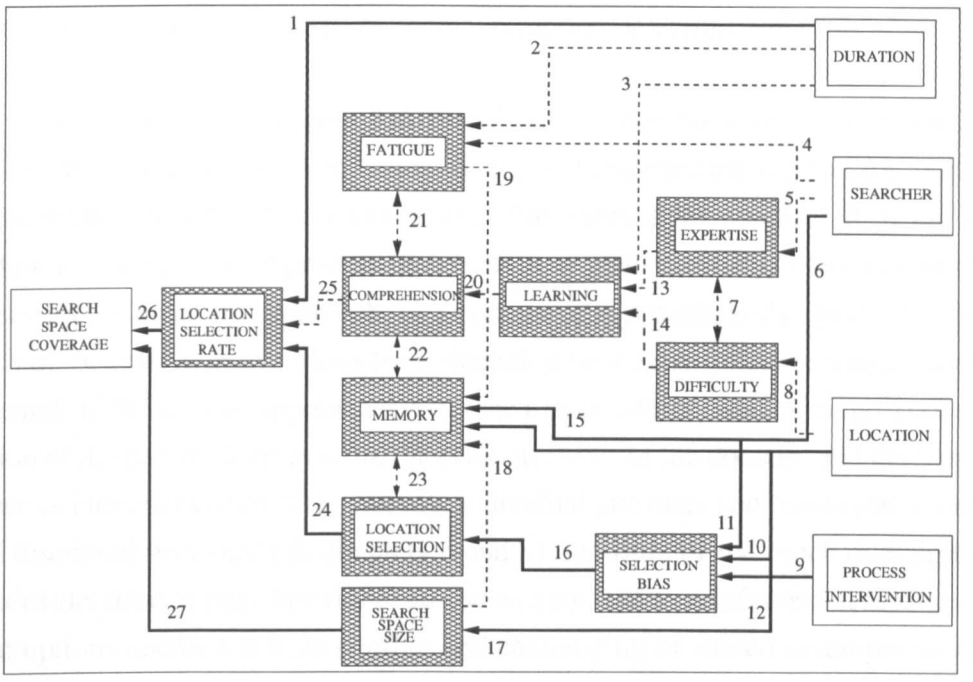


Figure 10.2: Influence Diagram Analysis of Coverage Dimension

There is one output variable of Search Space Coverage. The various definite and indefinite relationships that connect these various variables are numbered 1...27, and these, and the variables, will be discussed in more detail in the level–of–model detail rationale below.

In the design rationale that follows the simulation design decisions will be framed into categories of: a) decisions justified by the goal(s) of the simulation model; b) decisions justified by the scope of the simulation model; c) decisions justified on removing unnecessary complexity and subjectivity of the simulation model; and d) arbitrary design decisions deemed necessary to progress the simulation model from a more practical standpoint.

Goal Related Decisions.

The following simulation modelling design decisions are justified by the purpose of the simulation model, in terms of, comparing the various approaches of considering non–functional attributes and issues of what effect under/over representation

of non–functional attributes have upon assumption detection.

The input variable of **Process Intervention**. As can be seen from subsection 10.2.1, the first goal of the simulation model is to compare the various approaches to promoting non–functional attributes. The alternatives considered in the literature and the proposed process intervention of goal–diversity, advocated in this thesis, are: a) an ad-hoc approach; b) a systematic approach; and c) goal–diversity. Each of these have implications for influencing how assumption coverage may be affected. In the ad–hoc approach there is the real possibility that, through a combination of the nature of the system, the characteristics of the domain, and the homogeneous influences of culture, training, individual priorities and biases (mentioned and discussed previously in chapters 6 and 8), too little or too much overlapping of consideration of non–functional attributes may lead to insufficient sensitivity of assumptions and/or too little conflict and challenging of shared assumptions and purpose ascriptions. In the systematic approach there is the possibility that the individual will be asked to promote too many non–functional attributes at once, which, as chapter 7 discussed, results in too much cognitive processing, and consequently, a shallower level of assumption coverage may result. With the proposed goal–diversity process intervention, there is the need to predispose a developer to the promotion of a single non–functional attribute throughout the higher conceptual phases of the development life–cycle — in order to sensitise them and justify human redundancy in the task to gain a wider and more diverse assumption coverage. Therefore, from these alternative approaches, and the simulation modelling goal of being able to compare the assumption coverage efficacy of the different approaches, it is necessary to be able, in some way, to emulate a process intervention so that the simulation model user can influence an individual searcher's search pattern as they search. Here, we can get an insight into how assumption coverage can be modelled in the simulation model from the software inspection process from subsection 9.2.1 in chapter 9 with respect to reading techniques.

A major human redundancy and human diversity consideration relates to the relationships of coverage and sensor capability between the factors of the searcher, the search environment, and the target objects and the different reading techniques

Reading Technique	Coverage Sensitive	Defect Sensitive
Ad–Hoc	No	No
Checklists	No	Yes
Stepwise Abstraction	Yes	Yes
Active Design Review	Yes	Yes
SB/ Defect–Based	Yes	Yes
SB/ Perspective–Based	Yes	No
SB/ Functional Point–Based	Yes	Yes
N–Fold	No	No

Table 10.1: Reading Technique Classification

that have been evolved and used in the software inspection process to improve artifact coverage and defect detection effectiveness.

In table 10.1 a classification is produced characterizing the reading techniques from subsection 9.2.1 in chapter 9 into which search theory factors and relationships they attempt to affect. These can be further composed into a 2 x 2 matrix encompassing four different different characterizations based upon two dimensions of Defect Space Coverage and Defect Sensitivity. This is illustrated in figure 10.3.

For example, ad-hoc and n-fold reading techniques reflect no reading predispositioning support — in terms of defect space coverage or defect type sensitivity. In this regard, from a search theoretic standpoint, it reflects a uniform search function where each searcher or inspector, probabilistically, is equally likely to search certain regions of the defect space and detect certain defects. In this sense they are, as a group of individual inspectors, more uniform on the dimension of defect space coverage and uniform on the dimension of defect types. Checklist reading techniques, however, while offering no support, in terms of *where* in the artifact to look, attempt to predispose the inspector to check for certain types of defects. In this regard, from a search theoretic view, while, like ad-hoc and n-fold reading techniques, the group of individual inspectors are equally likely to search different regions of the defect space, they are more likely to be differently sensitised to look for different defect types (assuming that different checklists are handed

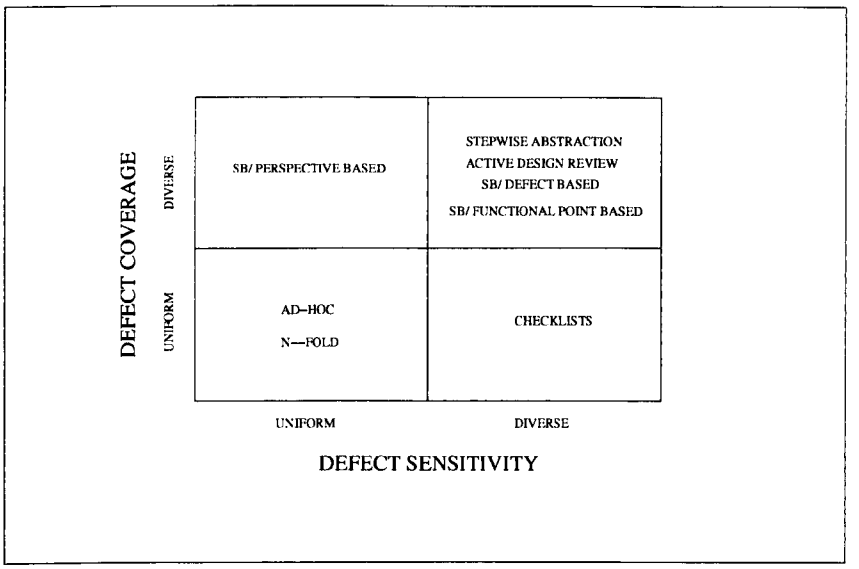


Figure 10.3: Diversity/Uniformity Focus of Reading Techniques

to different inspectors, otherwise, this reading technique would be the same as an ad-hoc or n-fold approach). Perspective—based reading techniques, on the other hand, attempt to predispose different inspectors to view the artifact under inspection from different stakeholder standpoints. This means that, although they may look at the same part of the artifact, they are likely, or more probable, to bring a different meaning, emphasis, or value interpretation because of this reading technique's predispositioning. However, there is no attempt to sensitize the inspectors to look for certain types of faults with this reading approach. Therefore, it can be seen that, in this case, the group of individual inspectors are different (or diverse) in terms of searching the defect space whilst having the same probability of detecting certain defect types. Reading techniques like active design review and functional–point, and step wise refinement appear to attempt predispositioning on both defect coverage and defect sensitivity as they tend to force the inspectors to cover (at least) the inspection artifact (i.e. scenarios) and present open–type questions that predispose inspectors to focus on certain defect type classifications (this is particularly true of the defect–based approach under the scenario–based reading technique). Assuming that these reading approaches use these to attempt inspection process diversity by giving individual inspectors different reviews/scenarios

and focused questioning based upon them, then this has the potential to make each inspector different or diverse on both defect space and defect types.

Therefore, it is a design decision, in the interest of promoting the simulation model's goal(s), that the input variable of Process Intervention emulates the diversity/uniformity dimensions of the software inspection process and allows a searcher to be configured in a probabilistic manner to be more/less likely to search some locations than others, and be more/less likely to detect some assumptions than others. In the influence diagram on this coverage dimension in figure 10.2, this coverage process intervention dynamic is modelled by the variables of: Searcher, Location and Process Intervention (input variables), and Selection Bias and Location Selection (intermediate variables), and includes the definite relationships numbered 9, 10, 11, 16, and 24 which will be modelled as a probability distribution for each searcher on a location type of each location within the search space.

The input variable of **Duration**. In subsection 10.2.1, it is noted that the simulation modelling goals require a comparison of the various approaches to promoting non–functional attributes under an equivalent human resourcing constraint. This is important as, in a real–world situation, the effectiveness of the proposed goal–diversity process intervention needs to be compared to the other approaches in terms of equivalent human effort duration deployed. For example, if the second stage of individual inspection is allocated (say) 3 hours each per developer promoting a single non–functional attribute, and 5 non–functional attributes are deemed important to represent, then the total human resourcing constraint for the second stage of individual inspection is 15 hours in total. If the efficacy of the proposed goal–diversity approach is to be compared with the others then, in the case of the systematic approach, each of the 5 developers involved would spend 3 hours promoting all 5 non–functional attributes — and told to spend 36 minutes on each throughout the inspection. With the ad–hoc approach, the 5 developers would simply be given 3 hours each and told to think about quality issues they believe are important. The point to note is that in comparing the three approaches the developers, under each approach, would need to be given comparable duration in order to compare the effectiveness of each approach. Furthermore, as noted

in chapter 9, search theory also recognises that in order to compare the detection effectiveness and efficiency of competing search strategies, it is necessary to ensure that all strategies operate under a constant and equivalent resource constraint that reflects a theoretical 100% sufficient search effort coverage as this not only allows comparison baselines between competing search strategies, but also unearths and makes explicit the inefficiencies and ineffectiveness of each particular search strategy when compared generically to the definite range search. In the simulation model it would have been possible to design the behaviour so that different searchers could search for a different number of resource units, but this would have contravened the search theory principle and the overall modelling goal(s) of the simulation model. Therefore, the design decision is to ensure that all searchers search under a constant resource constraint, and that no matter how many searchers are involved, the theoretical 100% sufficient search effort coverage constraint will be factored–out by the number of individual searchers involved in the search. This ensures, that not only are competing searchers comparable — in terms of competing efficacy on coverage to a common baseline, but that also, each individual search can be compared to a generic search strategy to achieve a more general and comparable assessment of how effective a particular search approach is.

In the influence diagram in figure 10.2, these dynamics are captured by the input variable of Duration, the intermediate variable of Location Selection Rate, and the output variable of Search Space Coverage. Furthermore, the relationships 1 & 26, between these variables, is a definite one whereby the user of the model can input a number that reflects the number of resource units a searcher will search for. This figure can be between 1 and the overall resource constraint divided by the number of searchers employed. So, for example, in a 1000 location search space, the user of the model can set the resource constraint anywhere from 1 resource unit to 1000 resource units. If there is 1 searcher employed and the user sets the resource constraint to a (theoretical) 100% coverage constraint (i.e. 1000 resource units), then the single searcher will search for 1000 resource units. On the other hand, if there are 5 searchers employed in the search, then each searcher will search for 200 resource units.

Scope Related Decisions.

The following simulation design decisions are justified upon the scope of the simulation model, in terms of: a) only simulating stage 2 of Individual Inspection; b) the *'upstream'* phases and products of the software development process; c) inclusion of the planning, inspection, and collection phases of the inspection process; and d) inclusion of human roles of the recorder, inspector and collector.

The input variable **Searcher**. As discussed in subsection 10.2.2, on modelling scope, the scope must accomodate for the inspection process phases of inspection, and team role of the inspector. In an obvious way, then, the input variable of Searcher fulfils the modelling scope of these two requirements. Less obvious, however, the modelling scope section noted that the stage simulated is stage 2 of individual inspection. Therefore, while the possible modelling alternatives could have been to model searchers creating the objects in the search space, and/or, modelling searchers interacting with each other and detecting objects in the search space, both of these are considered out of modelling scope of the model. In terms of the coverage dimension, the searcher acts as an independent and non–interacting agent that decides to direct their search governed by a process intervention probability distribution that predisposes the searcher to be more/less likely to determine *where* they search in the search space. As, already mentioned, this is dependent upon them being more/less predisposed to select a location belonging to a certain type. Of course, the flexibility, in the search simulation model, means that the predisposition probability distribution can be configured in such a manner that makes the searcher(s) equally likely to select a location of any type, which is important in fulfilling the configuration predispostion necessary for simulating a systematic approach to promoting non–functional attributes.

It can be seen from the influence diagram in figure 10.2 that the Searcher input variable has many relationships of both a definite and indefinite type. As previously mentioned, one of the definite relationships relates to the process intervention that biases location selection in order to emulate assumption coverage diversity. The only other definite relationship from the Searcher input variable

relates to Searcher Memory (i.e. number 15) which will be discussed further, in terms of design decisions, in the arbitrary section below. All the other relationships from the Searcher input variable are indefinite, in nature, and these will be further discussed, in terms of design decisions, in the simplification decisions section below.

The input variable **Location.** In section 10.2.2 of modelling scope, it was noted that the scope of the model should account and accommodate for: a) the inspection process phases of, not only the inspector, but also for a certain amount of inspection planning and collection phases; and b) the team roles of, not only the inspector, but also the recorder and collector. In terms of modelling scope, the input variable of Location, in the simulation model satisfies both of these scope requirements in an abstract way, as, although it would have been possible to simulate the behaviour without the concept of a location (i.e. such as merely selecting certain numbers within a number range), the object–oriented design approach of creating a location object that could: i) record the number of times a searcher has selected a location; ii) record the number of times different searchers have detected an object in duplicate; and iii) provide a container for holding multiple objects to be detected, clearly affords an effective and efficient way to fulfil the modelling scope requirements of recording and collection of objects detected, not only, uniquely and in duplicate, but also whether they were undetected etc.

Furthermore, the concept of a location, as an input variable to the simulation model, also helps fulfil the modelling scope of inspection planning, as the number of search locations, directly determine not only the size of the search space to be searched (and hence the resource constraint limits — as mentioned above), but in configuring the number of search space locations to use is also fundamental in determining the density of the objects hidden within the search space. Therefore, the notion of Location as an input variable is fundamental to satisfying the modelling scope of planning during configuration of the simulation model.

Finally, it can be seen from the influence diagram in figure 10.2 that the input variable of Location has a number of relationships with intermediate variables of

both a definite and indefinite type. Its definite relationship (i.e. number 10) as a member of location type to help fulfil the modelling goal of a process intervention to influence search space coverage has already been discussed. The other definite relationship with Searcher Memory will be discussed in the arbitrary decision section below. The only indefinite relationship of the input Location variable is with Location Difficulty (i.e. number 8) and this will be further discussed in the simplification related decisions below.

Simplification Related Decisions.

The following simulation model design decisions are justified primarily upon indeterminate relationships identified in figure 10.2 that result in overly complicating the simulation model and hence reducing the simulation model's confidence through introducing too much subjectivity.

It can be seen from the influence diagram in figure 10.2 that there are a number of indeterminate relationships between input, process, and output variables. The most intuitive way to discuss these is to start at the topmost input variable, and work across to the left and downward in the diagram.

The first indeterminate relationship is between the input variable of **Duration** (i.e. number 2) and the intermediate process variable of **Searcher Fatigue** and the indeterminate relationship between the input variable of **Searcher** and Searcher Fatigue (i.e. number 4). As mentioned in chapter 9 in section 9.2.1, the longer an inspector performs an inspection, the more fatigue effects can undermine detection. This can manifest in a number of other influences — such as impeding comprehension of the artifact being inspected, as indicated by the mutually influencing indeterminate relationship between Searcher Fatigue and **Comprehension** (i.e. number 21), or begin to undermine the searcher's memory of where they have searched, in terms of coverage, resulting in unnecessary and wasteful duplication of individual effort — as indicated by the indeterminate relationship between Searcher Fatigue and **Searcher Memory** (i.e. number 19). Although this is intuitively plausible it is also highly subjective, variable, and qualitative, in nature, and in the interests of maintaining and promoting confidence in the eventual

search simulation model's outputs, the influential dynamics of searcher memory is omitted from the simulation model.

The next indeterminate relationship to consider is the one that exists between the input variable of **Searcher** and the intermediate process variable of **Searcher Expertise** (i.e. number 5). As chapter 3 section 3.2.2 indicated, there is an enormous amount of human performance variability in software design and programming. If we are to accept that similar levels of performance variability exists in detecting software defects, then it is clear, in an absolute sense, that individual inspectors, will differ widely in their ability to gain greater coverage of the software artifact, during the inspection process. This can be broadly explained in terms of personal superiority of intelligence and experience etc, which we can generalise as expertise. However, as already indicated, this is an absolute interpretation of expertise, there is also the potential for a more relative interpretation of expertise due to particular knowledge of a particular domain or type of system under development. This is illustrated in the influence diagram of figure 10.2 by the dual influencing indeterminate relationship between Searcher Expertise, Location and **Location Difficulty** (i.e. number 7 and 8). Obviously, novelty and unfamiliarity of the particular system to be developed and/or application domain to be understood can seriously increase/decrease the task difficulty level, depending upon the individual inspector's prior knowledge and experience of that particular system and/or application domain. However, trying to objectify, within such a simulation model, this influential dynamic of Searcher Expertise, in both the absolute and relative sense, is highly subjective and qualitative and any attempt to do so would not only over–complicate the simulation model, but could, in doing so, seriously undermine the confidence of the eventual output information. For these reasons, the dynamics of searcher expertise is omitted from the simulation model.

The next indeterminate relationship in the influence diagram of figure 10.2, is between the input variable of Duration, the intermediate process variables of Searcher Expertise, Location Difficulty and **Searcher Learning** (i.e. numbers 3, 13, and 14). In the real–world sense of the software inspection process, there is little doubt that learning what the purpose and function of the software artifact

is aiming to achieve is critical to detecting defects. Additionally, as chapters 6, 7 and 9 have argued, this is also critical in detecting harmful assumptions as many such assumptions manifest in system flaws because of conflicting purpose ascriptions. In terms of Duration, the longer an inspector spends studying a particular software artifact, the deeper and more completely they will come to understand it. However, this can also be influenced, by both the particular inspectors expertise (in both the absolute and relative sense) and also the intrinsically complex nature of the artifact to be learned. Nevertheless, as before, although these influential dynamics will no doubt have an impact upon an individual searcher/inspectors coverage of the artifact, the dynamics at play here have far too much variability, subjectivity, and interdependability in thier influences upon Searcher Learning, therefore, in the interests of maintaining objectivity and confidence in the simulation model's outputs, these influential dynamics are omitted from the model's design.

The next indeterminate relationship to consider is that between the intermediate process variables, of Searcher Fatigue, Searcher Learning, Searcher Memory and **Searcher Comprehension** (i.e. numbers 20, 21 and 22). As already hinted in the previous paragraph, comprehension of the purpose, function and behaviour of the software artifact being inspected is critical in detecting defects and harmful assumptions. The process that leads to greater comprehension coverage is that of learning. However, there are a number of potentially moderating dynamic influences upon Searcher Comprehension. The first, which has already been discussed in this section, is that Searcher Fatigue can inhibit comprehension over time. Another potentially moderating factor upon Searcher Comprehension is Searcher Memory. The inability to retain knowledge and information over time and hence forget what has already been learned can obviously undermine comprehension. This is particularly relevant to the software inspection process where inspections are interrupted or are staged with sizable breaks in between. Once again, while it is fairly obvious that, in the real–world inspection process, these dynamics will have an impact upon eventual coverage of the software artifact to be inspected, it is nonetheless, again another example of highly subjective influences that can compromise the eventual confidence of the simulation model if included.

For these reasons, the dynamics of searcher comprehension are omitted from the design of the eventual simulation model.

Finally, the last indeterminate relationship to be considered is that between the intermediate process variable of Searcher Comprehension and Location Selection Rate (i.e. number 25). Since the search space is made up of lots of locations (of various types), then the rate at which a searcher selects them over time will have an obvious influential dynamic upon coverage. By selection rate, it is meant that per period of Duration, how many locations does the searcher select? It is intuitive that, in the real–world inspection process, the inspectors ability to comprehend the purpose, function and behaviour of the software artifact will have a direct bearing upon the number of (say) methods, functions, modules, or objects they can be covered during a given duration constraint. As already discussed, the notion of comprehension and its preceding dependencies of Searcher Expertise, Location Difficulty, Searcher Learning, and Searcher Fatigue etc, introduce much subjectivity and unpredictable variability into the design of the simulation model, and therefore, the effect of Searcher Comprehension upon Location Selection Rate must also be omitted for this reason. Notwithstanding this, however, there is still a need to ensure that a flat Location Selection Rate is included into the design of the simulation model to fulfil the simulation model's goal(s) for comparing different searching strategies under a constant resource constraint. Therefore, to keep the Location Selection Rate simple and comparable across all diversity/uniformity configurations, 1 resource unit (RU) will be required by all searchers to select and search a Location. This means that the Location Selection Rate is 1RU per Location Selection, which keeps the mathematics fairly simple and straightforward for determining resourcing constraints in connection with the search space size.

Arbitrary Related Decisions.

The following simulation model design decisions are justified, in a pragmatic manner, arbitrarily as there are no clear decision–making criterion alternatives to compare.

There is only one arbitrary simulation modelling design decision concerning the coverage dimension and this is the intermediate process variable of Searcher Memory. Although, as stated in the section(s) above that this intermediate process variables dynamic influences upon Searcher Comprehension is subjective, it does allow, in terms of individual inspection effort, a real–world aspect to be accommodated within the search simulation model that can be more objectively modelled with regards to its dynamic influences upon Searcher Location Selection (i.e. number 23). This is why the relationships between the input variables of Searcher and Location have been diagrammed as definite relationships in the influence diagram of figure 10.2. The searcher memory can be modelled as a probability factor for each searcher employed between the ranges of perfect memory (i.e. p(1.00)) of remembering when an individual searcher has already selected and searched a particular location, and completely imperfect memory (i.e. p(0.00)) of remembering when an individual searcher has selected and searched a particular location. Furthermore, not only does this arbitrary design decision allow an aspect of more real–world influences to be accommodated within the search simulation model, but additionally, it also affords a more real–world influencing aspect into the model that can subsequently be analysed for its sensitivity effects upon various diverse/uniform searching configurations. One aspect of the intermediate process variable that will not be accommodated into Searcher Memory, however, is the effect of the **Search Space Size.** Intuitively, the larger the search space or software artifact being inspected, the greater the potential of a searcher or inspector forgetting and unwittingly re–inspecting a part of a software artifact or re–searching a location by mistake. This aspect is omitted from the search simulation model's design (as indicated by the indeterminate relationship number 18) because the relationship between artifact size or search space size and its progressive potential for undermining Searcher Memory is highly subjective and qualitative and therefore removed in the interest of promoting modelling confidence in the eventual outputs of the simulation.

In Summary

From the influence diagram in figure 10.2, the eventual design decisions of the search space coverage dimension can be stated, as follows :-

- Duration will be modelled as Resource Units (RU) allocated to Searchers employed in the search. 1 RU is required by a searcher to select and search a search space Location. Therefore the Location Selection Rate will be 1 RU per 1 Location selection and search. Total RU can be within the range of 1 to the size of the search space (i.e. number of locations) and this total will be factored–out between the number of searchers employed to ensure a theoretical 100% search resource coverage constraint to ensure a comparable baseline between competing search configurations;

- Searcher has a memory probability range from imperfect (i.e. p(0.00)) to perfect (i.e. p(1.00)) for remembering whether they have already selected and searched a particular location. A Searcher represents an inspector in the inspection process and is modelled as a non–interacting agent with other searchers employed;

- Location models, in an abstract manner, the necessary inspection process aspects of recording and collecting, as well as being instrumental in search planning. Locations can also be assigned to be a member of a type of location, which may or may not be selected and searched more by a particular searcher;

- Process Intervention captures the ability to predispose certain searchers employed in the search to be more/less likely to select and search certain locations more/less than others. This will be modelled by each searcher employed in the search having a probability distribution that predisposes or biases them to search certain location types more/less than others.

Lastly, in connection with the search space coverage dimension, the output variable reflects the overall collective coverage achieved by the searchers involved.

10.2.3.2 Sensitivity Dimension

In this section the many level–of–detail design decisions that relate the searcher to actual detection of the objects will be considered. It should be noted, that detection can be difficult or imperfect for two reasons. Firstly, as subsection 10.2.3.1 highlighted, in order to find an object the searcher needs to be in the location where the object(s) reside. This first aspect relates to coverage. To provide a more specific example, someone may find it difficult to find their car keys. They know *'what'* they are looking for, and will be able to detect them once they see them, but finding them may become difficult because they don't know *'where'* they are. Detection can become particularly difficult for the coverage dimension if the location *'where'* something resides is unusual in some way. For instance, in the car key example, the person may have left their car keys in a location that is, to them, very unusual. Let's say when they last had them they came into the house and went straight to the toilet — inadvertedly placing them on the bathroom window ledge. Normally, they would place them on the kitchen table, the living–room sideboard, or the coffee table. Now when they come to need them again, they search all their usual or normal locations but cannot find them, they spend a long time searching through pockets, the back of the sofa, the bedroom, etc until they, by chance, go to the bathroom and glance at them on the window ledge. The important point to remember is that detection, overall, became difficult, not because of the difficulty in sensing the particular object to be found, as the searcher in question knows exactly *'what'* they are looking for, but instead, because the object sought was contained or located in a location, that, in the mind of the searcher seeking them, was unusual in some way (i.e. bathroom window ledge). A final point worth mentioning at this stage, in consideration of the coverage dimension, is that a containing location is not *'unusual'* in any absolute sense, but instead, only in a relative sense with respect to a given searcher. So the window ledge, as a containing location for car keys, is not, in any absolute way, an intrinsically unusual place for a set of car keys, as another person may consider such a location within their house as quite a usual place to find their car keys — since they nearly always go to the toilet when first coming in and frequently place their car keys on the window ledge. Therefore, the *'usualness'* and *'unusualness'* of the particular

containing location of any object is essentially a relative phenomenon between a containing location, an object, and a given searcher. Secondly, however, detection of an object may become difficult due to its essential nature being difficult to detect — even when the searcher is in that particular location, and therefore, has the potential to detect it. This relates to the sensitivity dimension. A good specific, and physical, example of something being difficult to detect, even when we are in the location in which it is contained, is things like a needle, a contact lense, etc. Such things are physically, small and this, in a physical way, reduces our capability to detect them. At least, in a physical sense, on this sensitivity dimension, we can appreciate that, to a large extent, physical objects may be difficult to detect in an absolute sense. Nevertheless, it is also possible that, even when an object is contained in the location under search, an object can be difficult to detect due to: a) the nature of the containing location. For instance, Search Theory (cf. chapter 9 section 9.3) recognises that conducting searches on open plains is much easier and more successful than conducting searches in heavy forrestation. A conceptual searching analogy to this may well be a software inspector inspecting a very complex, unfamiliar, and novel concurrent processing module. Therefore, detection, on the senstivity dimension, can also be relative to the difficulty of the containing location; and b) detection, on the sensitivity dimension, may also become difficult and be undermined due to the indistinguishability of the sought object from other very similar objects contained in the same location. We have all seen game shows where the contestant has to find a particular thing (i.e. say a ball) from an area, container, or location holding a number of similar things. The well known saying *"Where's the best place to hide a pebble?...on a Beach!"* is also another clear case of this physical searching and detection phenomenon. Therefore, at least in a physical sense, on the the sensitivity dimension, something may become difficult to detect or find, due to its similarity of other things contained there.

This section, is concerned with illuminating a number of detailed–level design decisions focusing upon the sensitivity dimension. From what has been discussed above, it is possible to see that, at least drawing upon physical examples, something can be difficult to detect in both an absolute and relative way, and this section is aimed at enumerating a number of design decisions focusing on the searcher's

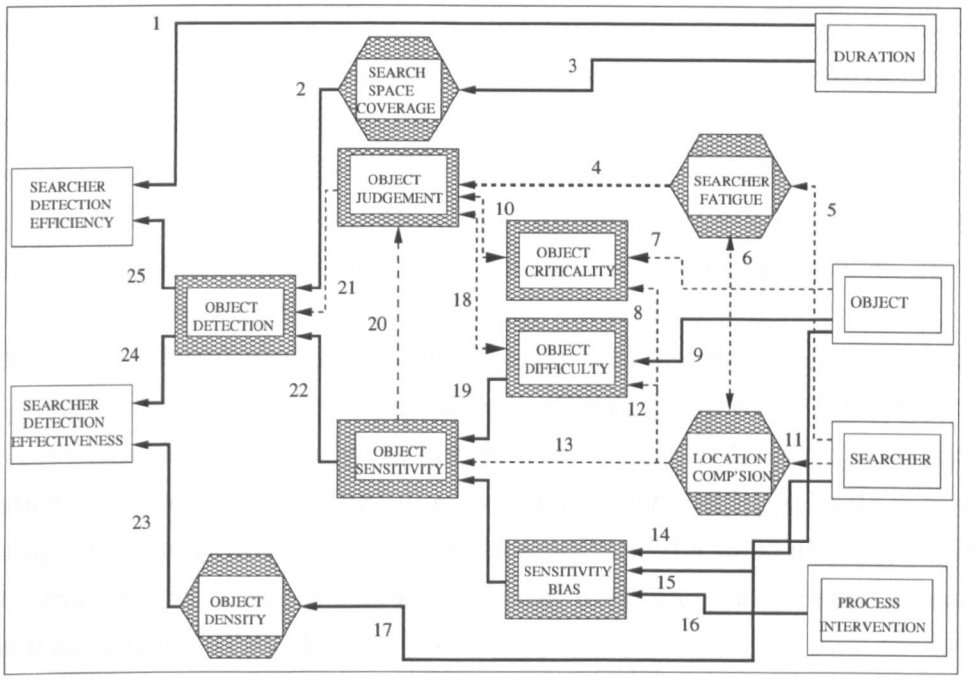


Figure 10.4: Influence Diagram Analysis of Sensitivity Dimension

capability to detect or sense an object once they have selected or covered the location and hence have the potential to detect or sense the objects being sought. As in subsection 10.2.3.1, a previous analysis has already been performed and diagrammed as an influence diagram in figure 10.4 to aid explanation of these design decisions.

The influence diagram shows four input variables of: Duration, Object, Searcher and Process Intervention. There are also six intermediate process variables of: Object Criticality, Object Difficulty, Sensitivity Bias, Object Sensitivity, Object Judgement, and Object Detection. In contrast to the influence diagram illustrating the coverage dimension in figure 10.2 on page 262, the influence diagram on the sensitivity dimension in figure 10.4 illustrates a number of composite variable structures imported from the coverage dimension diagram and the distribution dimension diagram in figure 10.5. As mentioned earlier in this section coverage is a prerequisite to detection and affords the potential to detect. Therefore, it is necessary to include a number of influences from the two other influence dia-

grams to capture these conditions. The composite structure variable notation is used to ensure inclusion while minimizing the illustrative complexity of the diagram. Finally, there are two output variables of Searcher Detection Efficiency and Searcher Detection Effectiveness. The various definite and indefinite relationships that connect these variables are numbered 1...24, and these, and the variables, will be discussed in more detail in the level–of–model detail rationale sections below.

In the design rationale that follows, in these four sections, the simulation model's design will be framed into the same four categories as in section 10.2.3.1 of: a) design decisions justified by the goal(s) of the simulation model; b) design decisions justified by the scope of the simulation model; c) design decisions justified on removing unnecessary complexity and subjectivity from the simulation model; and d) arbitrary design decisions deemed necessary to progress the simulation model from a more practical standpoint.

Goal Related Decisions.

The following simulation modelling design decisions are justified for the purpose of the simuation model, in terms of, comparing the various approaches of considering non–functional attributes and issues of what effect under/over representation of non–functional attributes have upon assumption detection.

The input variable of **Process Intervention.** The justification for the inclusion of the input variable, has been largely discussed in dealing with the coverage dimension in subsection 10.2.3.1. From the modelling goals in section 10.2.1, it can be seen that there are potentially three approaches that need to be compared within the simulation model. These are: i) an Ad–Hoc consideration; ii) a Systematic consideration; and iii) the proposed Goal–Diversity process intervention consideration. As mentioned earlier in this section, there are two fundamental dimensions to detection. These are a coverage dimension, in terms of *'where'* to search, and a sensitivity dimension, in terms of *'how'* sensitive the searcher/inspector is, when the containing location is actually searched (i.e. covered). These two dimensions are both in evidence within physical searches, from search theory in chapter 9 section 9.3, and also within the software inspection literature in chapter 9 section 9.2,

as exemplified by the predispositioning potential of the reading techniques used by the software inspection process. As was discussed in subsection 10.2.3.1, concerning the coverage dimension, these reading techniques can be classified into predisposing inspectors over a 2 x 2 matrix of search space coverage predispositioning (coverage dimension) and/or object sensitivity predispositioning (sensitivity dimension). Therefore, from these alternative dimensions, and the required simulation modelling goals for simulating the human redundancy and human diversity efficacy of these possible approaches, it is necessary, in some way, to be able to emulate a process intervention so that the simulation model user can configure the model in such a way to influence an individually modelled searcher's object sensitivity pattern to various objects as they acquire the potential to detect objects after selecting a location that contains objects.

It is a modelling goal related design decision, then, that the input variable of Process Intervention, on the searcher sensitivity dimension, allows any searchers employed in the search to be configured in a probabilistic manner to be more/equal/less likely to detect some objects than others in the search. In the influence diagram of the sensitivity dimension in figure 10.4, this sensitivity process intervention dynamic is modelled by the variables of Searcher, Object and Process Intervention (input variables) and their interactive detection behaviour upon the intermediate process variables of Sensitivity Bias, Object Sensitivity and ultimately Object Detection. The interconnecting relationships that exist between these variables (i.e. numbered 14, 15, 16, and 22) are all definite, in nature, as they can be reasonably modelled as a probability profile in terms of capturing each individual searcher's sensitivity capability of detection as a probability between zero and one. As in the case for coverage and locations in subsection 10.2.3.1, objects hidden within the search space, will be members of differing sets of object types that searchers can be made more/equal/less sensitive to — over the range of those object types.

The input variable of **Duration**. As noted in section 10.2.1 on simulation modelling goals, all the comparisons of the differing approaches to considering non–functional attributes must be under a comparable human resourcing constraint so that there is a baseline for comparing the efficacy of the competing search ap-

proaches. In this regard, on the sensitivity dimension, the input variable of Duration plays the same role as that on the coverage dimension in subsection 10.2.3.1 to ensure that overall detection performance, by the searchers employed, is related to the same resourcing constraints. This is captured diagrammatically in the influence diagram of figure 10.4 as a composite variable structure of Search Space Coverage, the input variable of Duration, and Object Detection — connected by two definite relationships (numbered 2 & 3). The composite structure captures a simplification that has already been discussed earlier in this section, that coverage is a prerequisite to actual detection that involves, itself, the investment of human search or inspection effort in terms of searcher comprehension, searcher learning, etc. However, on the sensitivity dimension, the input variable of Duration also plays an additional role in promoting the simulation model's goals with regard to directly providing one of the output variables of Searcher Detection Efficiency. Detection Efficiency, as chapter 9 highlighted, is a well used measure in both the areas of Search Theory and Software Inspections as it characterises how productive the human search and detection resources have been during the search/inspection process. This output variable of Searcher Detection Efficiency will be further discussed, in terms of promoting modelling goals, below, and in doing so directly shows how the input variable of Duration, on the sensitivity dimension, contributes to the simulation model's goal, in an additional way, through allowing the comparison of different search simulations to be modelled and configured. In the influence diagram in figure 10.4 this relationship is definite (i.e. number 1) and shows how the input variable of Duration is directly related to producing the output variable of Searcher Detection Efficiency.

The output variable of **Searcher Detection Efficiency.** As suggested above, this output variable directly contributes to the simulation model's goals as its use is to allow the detection productivity of search resources employed over different competing search strategies to be compared. Calculating the detection efficiency involves dividing the total number of objects (or assumptions, defects, faults etc) by the Duration of time exploited in the search or inspection. This is shown in the influence diagram of figure 10.4, by illustrating how the input variable of Duration and intermediate process variable of Object Detection is connected by

two definite relationships numbered 1 and 25, respectively. The resulting number captures how many (or what fraction of) objects were found or detected per period of time exploited. As discussed in subsection 10.2.3.1, Duration is abstracted to resource unit (RU) allowance for the searchers up to a theoretical 100% coverage constraint. Therefore, the detection efficiency of each search captures the number of objects (or fraction of objects) detected per 1 Resource Unit (RU) expended during the search.

The output variable of **Detection Effectiveness.** This, like Detection Efficiency, is also a variable that promotes the comparison goals of the simulation model, as it also affords a measure of how to compare different competing search strategies that will be configured in the simulation model. However, unlike Detection Efficiency, that is aimed at capturing the productivity of searching and inspection resources employed in the format of periods of duration, instead, the output variable of Detection Effectiveness captures the capability of the search or inspection resources involved with regard to the number of objects (or assumptions, defects, faults, etc) that are possible to find or detect within the search space. Therefore, calculating Searcher Detection Effectiveness involves dividing the total number of objects hidden, within the search space, by the total number found or detected by the search resources employed. The dynamics are illustrated visually in the influence diagram in figure 10.4, between the input variable of Object, the composite variable structure of Object Density (i.e. the actual number possible to detect), the intermediate process variable of Object Detection, and the output variable of Searcher Detection Effectiveness. These four variables are connected by three definite relationships of 17, 23 and 24.

Scope Related Decisions.

The following simulation design decisions are justified upon the scope of the simulation model, in terms of: a) only simulating stage 2 of Individual Inspection; b) the *'upstream'* phases and products of the software development process; c) inclusion of planning, inspection, and collection phases of the inspection process; and d) inclusion of human roles of recorder, inspector and collector.

The input variable **Searcher**. As discussed in subsection 10.2.2 on modelling scope, the scope must accomodate for the inspection process phases of inspection, and team role of the inspector. In this respect, then, the input variable of Searcher fulfils the modelling scope of these two requirements. However, the modelling scope section noted that the stage simulated is stage 2 of individual inspection. Therefore, while the possible modelling alternatives could have been to model searchers creating the objects in the search space, and/or, modelling searchers interacting with each other and detecting objects in the search space, both of these are considered out of modelling scope of the model. In terms of the sensitivity dimension, the searcher, again, acts as an independent and non–interacting agent that decides to direct their search governed by a process intervention probability profile that predisposes the searcher's sensitivity to be more/less likely to detect some objects contained within a given selected location. The flexibility, in the search simulation model, however, also means that the predisposition probability profile can be configured in such a manner that makes the searcher(s) equally likely to detect an object of any type, which is important in fulfilling the configuration predispostion necessary for simulating a systematic approach to promoting non–functional attributes.

It can be seen from the influence diagram in figure 10.4 that the Searcher input variable has relationships of both a definite and indefinite type. As previously mentioned, one of the definite relationships relates to the process intervention that biases object detection sensitivity in order to emulate assumption sensitivity diversity of the real–world situation. These definite relationships exist between the input variable of Searcher, Object, Process Intervention and the intermediate process variables of Sensitivity Bias, Object Sensitivity, and Object Detection and are numbered 14, 15, 16, & 22). All the other relationships from the Searcher input variable are indefinite, in nature, and these design decisions will be further discussed in the simplification decisions section below.

The input variable **Object.** In section 10.2.2 of modelling scope, it was noted that the scope of the model should allow for: a) the inspection process phases of, not only the inspector, but also for a certain amount of inspection planning and

collection phases; and b) the team roles of, not only the inspector, but also the recorder and collector. In terms of modelling scope, the input variable of Object, satisfies both of these scope requirements — as an abstraction, as, although it would have been possible to simulate the behaviour without the concept of an Object, the object–oriented design approach of creating an Object class facilitates: a) the recording of the number of times a searcher has detected an Object; b) the recording of the number of times different searchers have detected an object in duplicate. Both of these facets clearly affords an effective and efficient way to fulfil the modelling scope requirements of recording and collection of objects detected, not only, uniquely and in duplicate, but also whether they were undetected etc.

In addition to this, the concept of an Object, as an input variable to the simulation model, also fulfils the modelling scope of inspection planning, as the number of Objects hidden within the search space, directly helps determine such fundamental planning and user configuration issues — such as Object Density etc. Therefore, the inclusion of the concept of Object, as an input variable, is fundamental to satisfying the modelling scope of planning during configuration of the simulation model.

Lastly, it can be seen from the influence diagram in figure 10.4 that the input variable of Object also posseses a number of relationships with other intermediate process variables of a definite and indefinite type. Its definite relationship (i.e. number 15) as a member of object types helps fulfil the modelling goal of process intervention to influence searcher detection sensitivity. Its only other definite relationship concerns the intermediate process variable of Object Detection Difficulty (i.e. number 9) which will be discussed, in terms of design decisions, in the arbitrary related decisions section below. The only other relationship (i.e. number 7) is indefinite, in nature, and this design related decision will be discussed in the simplification related decision section next.

Simplification Related Decisions.

The following simulation model design decisions are justified primarily upon the indeterminate relationships identified in the influence diagram analysis in figure

10.4 that result in overly complicating the simulation model and, by doing so, undermine the simulation model's objectivity.

It can be seen from the influence diagram in figure 10.4 that there are a number of indeterminate relationships between the input, process, and output variables. Again, the best way, perhaps, to discuss these is to start at the topmost input variable, and work across to the left and downward in the influence diagram.

The first indeterminate relationship is between the input variable of **Searcher** and the composite variable structure of **Searcher Fatigue** (i.e. numbered 5). As was discussed in subsection 10.2.3.1, on the coverage dimension, Searcher Fatigue has many interrelated influences that can be highly variable and difficult to model — and which can, on an individual searcher, impact upon the overall coverage that can be achieved. Intuitively, on the sensitivity dimension, Searcher Fatigue can also impact upon a searcher's capability to detect an object as fatigue effects can undermine both the necessary concentration and comprehension of what constitutes a harmful assumption (or defect, fault, etc), and thereby also impact upon the searcher or inspectors judgement. For these, reasons, the influence diagram in figure 10.4 includes these indeterminate relationships from a Searcher and Searcher Fatigue perspective, via:- a) the dual influencing indeterminate relationship (i.e. numbered 6) between Searcher Fatigue and (the compound variable structure) **Location Comprehension**, since again, intuitively, fatigue can influence comprehension and understanding, while, equally, as was discussed in subsection 10.2.3.1, on the coverage dimension, comprehension involves learning effort which ultimately, over–time, leads to fatigue effects; and b) the indeterminate relationship from Searcher Fatigue and **Object Judgement** (i.e. numbered 4), as, fatigue will ultimately impact upon a searcher or inspector's judgement of what constitutes a flawed assumption or defect. These influences, while being relevant in both a physical and conceptual sense, is perhaps even more influential within the domain of conceptual searching since, intuitively, a lack of concentration and appropriate representation, can quickly result in reducing a searcher's senstivity capability to correctly judgeing the impact of some implicit, explicit, shared, or purpose ascription assumption that can subsequently undermine the dependability of the

eventual artifact during later operation. It can be seen, therefore, that the compound variables of Searcher Fatigue and Location Comprehension — along with the intermediate process variable of Object Judgement, have potentially many subtle interdependent influencing relationships with one another. While, as already stated, they, no doubt, can positively/negatively impact upon a searcher's detection capability, on the sensitivity dimension, they are of a nature that are highly subjective, variable, and qualitative, and, in the interests of promoting the simulation model's simplicity and objectivity, these influential dynamics are omitted in the eventual simulation model's design.

The next indeterminate relationship to consider is between Object and **Object Criticality** (i.e. numbered 7). It is, in a real–world fault detection situation, intuitively obvious that different assumptions and faults will have different consequential effects upon the overall dependability of the eventual software artifact during subsequent operation. Therefore, in a broad and brief distinction, the dependability criticality of a flawed assumption or fault can be categorised on two dimensions:- a) in terms of the consequential impact that it may have upon the eventual judged dependability of the software artifact during operational usage; and b) the frequency upon which the flawed assumption or fault will manifest in the fault=>error=>failure chain (cf. chapter 2). Consideration of faults on these two dimensions reveals that there is a 2 x 2 matrix for categorising the criticality nature of any such defect, fault, or flawed assumption. The most critical category is those faults, defect or assumptions that have both a high consequential impact and frequency of occurence during operational usage — as these will undermine judgements of dependability of the software artifact the most. The least critical are those faults, assumptions or defects that have both a low consequential impact and frequency of occurrence — as these type of faults will have the least influence upon judgements of dependability and may often be considered, by the users, to be little more than an anoyance during operation. The other two categories relate to either a high/low impact and high/low frequency, and such judgements of the eventual dependability criticality will, to large extent, be dependent upon a number of other specific circumstances — such as the application domain, the type of system in question, as well as the specific circumstances in which these

faults and defects manifest, etc. Such issues, of fault criticality, were previously raised and discussed in chapter 3 subsection 3.4.2 in providing some justifications of process redundancy with the distinctions made between a productivity effect and a quality effect of employing human redundancy and diversity. It can be seen from these issues and those definitions discussed in chapter 3 that considerations of Object Criticality relate directly to a quality effect measure of search or inspection detection. Additionally, from the influence diagram in figure 10.4 we can see that Object Criticality has also a number of other indeterminate relationships with both Object Judgement and (the composite variable structure) Location Comprehension (i.e. numbers 8 and 10). Firstly, it is intuitively clear from what has already been discussed in the previous paragraph that determining the negative criticality influence (in terms of both consequences and frequency of occurence) places a large amount of judgement upon the particular searcher or inspector — particularly in conceptual searching/detection. Furthermore, as chapters 6, 7 and 8 on assumptions, teleology and functionalism, and ATM analysis indicates, this judgement of what is, and what is not, a harmful assumption or fault really reflects a relative judgement based upon the searcher's particular bias, values, knowledge, CBS context–of–interest, and non–functional goals being promoted etc. It is for these reasons that the indefinite relationship between Object Criticality and Object Judgement (i.e. numbered 10) is dual influencing — in order to capture this relative issue in determining an object's criticality. Secondly, from what has already been discussed in the previous paragraph, it is intuitively obvious that comprehension and understanding will have a major influence in determining a searcher or inspector's detection capability for establishing a given assumption or fault's criticality. For this reason, the influence diagram includes another indeterminate relationship (i.e. numbered 8) between Location Comprehension and Object Criticality. Lastly, whilst, in a real–world inspection situation, there is no doubt, that, of all the defects that may be detected, some will be more/less critical, in terms of dependability than others, what is at issue here is what is possible to reasonably objectify for the simulation model. The simulation model, like all models, necessitates a generalisation and abstraction of reality, and as already discussed, not only do issues of object criticality involve many other intederminate relationships with other variables, but assessing the criticality of a defect or flawed assump-

tion relies, to a great extent, upon the relative judgement and understanding of the searcher or inspector, but also the specific nature of a particular applicational domain, system type and operational circumstances. Therefore, it was decided that such issues, influences and dynamics — in the real—world situation, would unnecessarily compromise the simplicity and objectivity of the simulation model, and for these reasons, the simulation model only infers a productivity effect measure of human search diversity in terms of the number of objects detected and not the criticality of any particular detected/undetected object upon overall dependability.

The last set of indeterminate relationships to consider in the influence diagram of figure 10.4 relate to the input variable of Searcher to the composite variable structure of Location Comprehension through to the intermediate process variables of **Object Sensitivity** and Object Judgement (i.e. numbers 11, 13 & 20 & 21). As stated earlier in this subsection — in consideration of the simulation model's goals, the searcher's object capability, on the sensitivity dimension, must be capable, within the simulation model, to be predisposed to different sensitivity biases throughout the search simulations in order to fulfil the simulation model's goals of comparing different non–functional attribute considerations upon assumption detection. However, from what has already been discussed in the preceding paragraphs concerning Location Comprehension, it is intuitively clear that the searcher or inspector's ability to apply their expertise (in both an absolute and relative sense) and adequately understand the nature of the thing being searched and inspected will also have an impact upon their detection sensitivity capability. For this reason the influence diagram illustrates this highly qualitative and varying indeterminate relationship from Location Comprehension to Object Sensitivity as a dynamic that will ultimately influence a searcher or inspector's detection sensitivity (i.e. numbered 13). Additionally, as was discussed in the previous paragraphs, the Object Judgement involved will have a critical impact upon *'what'* the individual searcher or inspector decides what is, and what is not, a harmful assumption or defect, etc. As was pointed out also, with regards to Object Judgement, this is a highly relative consideration and is also fundamentally influenced by such variables as Searcher Fatigue. From the influence diagram it can also

be recognised that if the searcher or inspector's object detection sensitivity is impaired in any way, then this will also impact upon their ultimate judgement of what constitutes a harmful assumption or defect etc. For this reason the indeterminate relationship between Object Sensitivity and Object Judgement is included in figure 10.4. Finally, it can be argued, in an intuitive way, that if, for some reason (e.g. via Searcher Fatigue, Location Comprehension, or Object Sensitivity), a searcher or inspector's judgement becomes incomplete or impaired then this will ultimately influence not only overall **Object Detection** measures (i.e. Detection Efficiency and Detection Effectiveness), but, in a real–world situation, can also result in increasing errors of judgement. It has already been discussed in subsection 10.2.2 on simulation model scope that human error in detection can result in false positive and false negative judgements about what objects are detected, and this aspect is included into the influence diagram of figure 10.4 as another indeterminate relationship (i.e. number 21), since, as it was highlighted there, such dynamics of misjudgement reflect a very subjective, specific, varying, and qualitative modelling concern. Finally, then, although such influences of Object Sensitivity, Object Judgement and Object Detection can result in erroneous detections, these are all highly subjective and have many antecendent and interrelated dependencies which would both reduce the simulation model's objectivity and increase its complexity — if introduced. For this reason the interrelated influences of Location Comprehension, Object Sensitivity and Object Judgement upon particular Object Detections is omitted from the design of the simulation model.

Arbitrary Related Decisions.

The following simulation model design decisions are essentially justified in a practical way — since there is no clear decision–making criterion alternatives to compare.

There is only one arbitrary simulation modelling decision, on the object detection sensitivity capability dimension, and this essentially concerns the intermediate process variable of **Object Detection Difficulty**. From the influence diagram in figure 10.4, it can be seen that this intermediate process variable is connected

by a number of relationships of both a definite and indefinite type (i.e. numbers 9, 12 18 & 19). This is because, during the analysis of the many influences, on the object detection sensitivity dimension, Object Difficulty contains, from a real–world searching and inspection situation, a number of highly subjective and interrelated influences and connections, respectively. Firstly, in terms of (the compound variable structure) Location Comprehension, it is intuitively clear from the previous subsection that searcher expertise, learning, and comprehension can have an obvious positive or negative impact upon the ease or difficulty in detecting any given particular assumption or defect etc. For this reason the relationship between Location Comprehension and Object Detection Difficulty (i.e. number 12) is modelled as an indeterminate relationship. Secondly, from the influence diagram in figure 10.4, it can also be seen that there is a dual influencing indeterminate relationship between Object Judgement and Object Detection Difficulty (i.e. numbered 18). This relationship is both indeterminate and dual–influencing because, as has already been discussed in the previous subsection on modelling simplification related decisions, judgement is a highly subjective and relative dynamic. For instance, what's judged to be difficult for one searcher or inspector to detect — may, at the extremes, be very easy for another searcher or inspector to detect. In fact, this relative detection phenomenon is actually at the *'heart'* of such fault detection processes — such as Pair–Programming, Egoless Programming, and Open–Source Software Development (cf. subsection 3.4.1.2 in chapter 3). Alternatively, however, there are also two definite relationships connecting Object Detection Difficulty. In the influence diagram, the first one is between the input variable of Object and Object Detection Difficulty (i.e. numbered 9). It has been arbitrarily decided to model each object's difficulty as a probability factor that can be in the range of p(0.00) and p(1.00), since, although, within the real—world searching and detection situation, object detection difficulty is essentially a relative dynamic, as discussed above, it has already been highlighted earlier also in this subsection, that at least within the physical searching domain, an object or thing can be difficult to detect in the absolute sense (i.e. needle, contact lens, etc), and, therefore, its inclusion does facilitate a richer and more real–world influence (at least in physical searches) that can be easily modelled quantitatively. Furthermore, like the arbitrary inclusion of Searcher Memory, on the coverage di-

mension in subsection 10.2.3.1, including Object Detection Difficulty within the simulation model also permits, at least to some extent, the potential sensitivity influence this variable may have upon various competing diverse/uniform search configurations. It is for these arbitrary and practical issues in progressing a useful simulation model that this intermediate process variable's influence is included in the simulation model's design. This is why, in the influence diagram in figure 10.4, the relationships between Object, Object Detection Difficulty and Object Sensitivity is modelled as a definite relationship (i.e. numbered 9 and 19) as it will allow the model user to control searcher object detection sensitivity.

In Summary

From the influence diagram in figure 10.4, the eventual design decisions of the sensitivity dimension can be summarised as follows: -

- Duration, on the sensitivity dimension, contributes to the simulation modelling goals directly as it is intrumental in providing a fundamental measure in determining the detection efficiency of a particular search strategy configuration;

- Searcher, on the sensitivity dimension, will not include the possibility of object judgement. Therefore, objects reflect definite entities that are to be detected by the searchers employed in the search and detection is devoid of any potential for human–error on the part of the searchers (i.e. no potential for false positive or false negative judgements). Additionally, a searcher represents, analogically, an inspector in the inspection process and is modelled as a non–interacting agent with other searchers employed in the search;

- Object, in an abstract manner, satisfies a number of the simulation model's scope and goals. Firstly, in terms of modelling scope, its inclusion facilitates important aspects of the inspection process to be modelled — in the form of inspection planning to record the number of times an object has been detected etc. Secondly, in terms of modelling goals, the inclusion of an object directly facilitates the simulation model's goals through its potential to be assigned to an object type that allows diverse/uniform process

interventions to be incorporated into a search strategy configuration upon the searcher detection sensitivity dimension;

- Process Intervention captures the ability to predispose searchers employed in the search to be more/less likely to detect certain objects more/less than other objects. This will be modelled by each searcher employed in the search having a probability capability profile that predisposes or biases their object detection capability to detect certain objects of a certain type more/less than other objects of another type. Therefore, the process intervention dynamic along the searcher detection sensitivity dimension directly promotes the goals of the simulation model.

Lastly, in connection with the sensitivity dimension, the output variables reflect an important comparison measure for comparing the various diversity/uniformity search strategy configurations envisaged. In this regard these output variables directly promote the simulation model's goals of comparing the human diversity benefits of existing approaches to promote dependability during the creation process.

10.2.3.3 Distribution Dimension

As in subsections of 10.2.3.1 and 10.2.3.2, a previous analysis has already been performed and diagrammed as an influence diagram in figure 10.5 to aid explanation of these design decisions.

The influence diagram shows three input variables of: Object, Types and Location. There are four intermediate process variables of: Object Types, Number of Objects, Number of Locations, and Location Types. Finally, there are four output variables of: Object and Object Types Distribution, Object Density, Location and Location Type Distribution, and Object Types to Location Types Distribution.

It can also be seen, from the influence diagram in figure 10.5 that there are 14 relationships numbered 1..14. Unlike the influence diagrams of 10.2 and 10.4 on

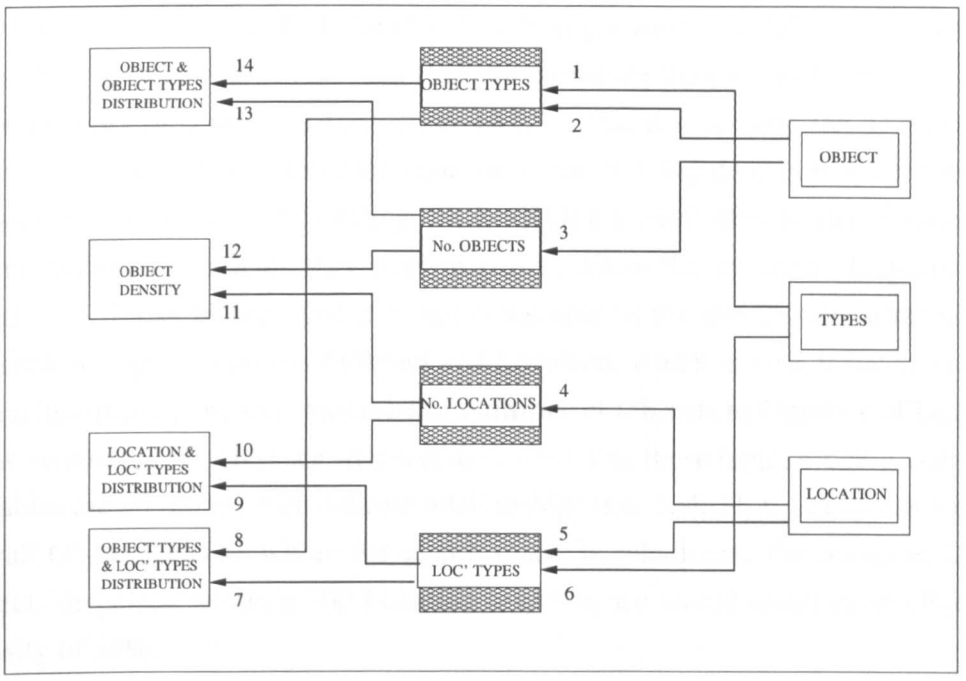


Figure 10.5: Influence Diagram of Distribution Dimension

the coverage and sensitivity dimensions, all of the relationships are of a definite nature — in terms of being quantifiable.

In the design rationale that follows, the simulation model's design will be framed into the same categories as in subsections 10.2.3.1 and 10.2.3.2 of: a) design decisions justified upon fulfilling the goal(s) of the simulation model; b) design decisions justified by the scope of simulation model; c) design decisions justified on removing unnecessary complexity and subjectivity from the simulation; and d) arbitrary design decisions deemed necessary to progress the simulation model from a more practical standpoint.

Goal Related Decisions

The following simulation modelling design decisions are justified for the purpose of the simulation model, in terms of, comparing the various approaches of considering non–functional attributes and issues of what effect under/over representation of non–functional attributes have upon assumption detection.

The output variable of **Defect Density** directly supports the modelling goal of the simulation model, as it can be seen from the influence diagram of figure 10.4 on the sensitivity dimension in subsection 10.2.3.2 that it was imported as a composite structure and is an intrinsic variable in determining detection effectivenes — which is important for fulfilling the modelling goal(s) of comparing various diverse/uniform search strategy configurations. From the influence diagram, in figure 10.5, it can be seen that this output variable of the distribution dimension requires the input variables of **Object** and **Location**, which in turn, is determined by the intermediate process variables of **Number of Objects** to **Number of Locations**, within the search space. It can also be seen, that these input, process, output variables are all related with definite relationships (i.e. 3, 4, 11 & 12) — since the overall Object Density, within the search space, is calculable. For instance, 250 Objects dispersed within a 500 Location search space would result in an Object Density of 50%.

Scope Related Decisions

The following simulation design decisions are justified upon the scope of the simulation model, in terms of: a) only simulating stage 2 of Individual Inspection; b) the *'upstream'* phases and products of the software development process; c) inclusion of planning, inspection, and collection phases of the inspection process; and d) inclusion of human roles of recorder, inspector, and collector.

There are two scope related design decisions to consider on the distribution dimension.

The first scope related design decision, as can be seen from figure 10.5, relates to the possible distribution potential of Objects to **Object Types**. In the influence diagram this output variable requires the input variables of Object and **Types** and is determined by the process variable of Object Types. It can also be seen that there are four definite relationships relating the input, process, and output variables (i.e. numbers 1, 2, 13 & 14). All of these relationships are definite — since there are of a quantifiable nature. For instance, Objects could be distributed over the

number of Object Types unevenly such as: Object Type 1 has 25 Objects, Object Type 2 has 50 Objects, Object Type 3 has 75 Objects, Object Type 4 has 100 Objects.[5] Here we can see that there are 250 Objects of 4 Object Types, but they are in a positive skewed distribution over the 4 Objects. The point worth mentioning here, analogically, with the simulation model, is what, in essence is this capturing, in connection with assumption detection and the proposed Goal–Diversity process intervention? It can be argued that, intuitively, what is being indicated is that during the first stage of Individual Analysis, developers employed in providing independent and diverse non–functional analyses representations are much more likely to make certain undesirable assumptions that compromise some non–functional attribute than others. While, correspondingly, being much less likely to make certain other undesirable assumptions that compromise some other non–functional attribute. Not only does this need to be empirically validated, but, in terms of the simulation modelling scope, such issues relate to the first stage of the Individual Analysis — where assumptions are generated and manifested within the resultant analysis solutions. It is reasonable, for now, due to the inherent arbitrary nature of assumptions, to consider that all assumption types are equally likely to occur during the first stage of Individual Analysis, and, therefore, the distribution of Objects over Object Types will be kept uniform.

The second scope related design decision, as can be seen from figure 10.5, relates to the distribution of Object Types over Location Types. In the influence diagram this output variable requires the input variables of Object, Types and Location and is determined by the process variable of Object Types and **Location Types**. It can also be seen that there are six definite relationships relating the input, process, and output variables (i.e. numbers 1, 2, 5, 6, 7 & 8). All of these relationships are definite — since there are of a quantifiable nature. For instance, Object Types could be distributed over Location Types quite unevenly so that a given Location Type could contain more Objects of a certain type than Other Location Types — such as Location Type 1 could contain 10% of Object Type 1, 20% of Object Type 2, 30% of Object Type 3, and 40% of Object Type 4, while Location Type 2 could

[5]Note here, that, in comparison to Object Density in a 500 location search space, the density would still be 50%. Distribution of objects, no matter how uneven, do not alter the density.

contain 20% of Object Type 1, 30% of Object Type 2, 40% of Object Type 3, and 10% of Object Type 4, etc. Again, the point worth mentioning, analogically, with the simulation model, is what, in essence, is this capturing, in connection with assumption detection and the proposed Goal–Diversity process intervention? It can be intuitively argued that what this type of uneven distribution is argueing for is that during stage 1 of Individual Analysis, predisposing developers to promote certain non–functional attributes results in them being more likely to make certain harmful assumption types that compromise certain other non–functional attributes while being much less likely to make other harmful assumption types that would compromise another certain non–functional attribute.[6] Once again, it can be argued that this is a phenomenon that needs to be empirically validated. Furthermore, in terms of the simulation modelling scope, such issues relate to the first stage of the Individual Analysis — where assumptions are generated and become manifest within the resultant analysis solutions. It is therefore reasonable, at this point, due to the inherent arbitrary nature of assumptions, to consider that predisposing a developer to promote a particular non–functional attribute, during the first stage of Individual Analysis, will result in them being equally likely to make assumptions that could compromise any of the other important non–functional attributes. Therefore, the distribution of Objects Types over Location Types will also be kept uniform.

Simplification Related Decisions

The following simulation model design decisions are justified primarily upon the relationships identified in the influence diagram analysis in figure 10.5 that result in overly complicating the simulation model and, by doing so, undermine the simulation model's objectivity.

There is only one simplification design decision for the distribution dimension. This can be seen from the influence diagram in figure 10.5 to be the distribution of Locations over Location Types. In the influence diagram this output variable re-

[6]For example, what is being suggested by such a distribution is that (say) predisposing a developer, during stage 1, to promote Safety means that he/she is much more likely to make assumptions that compromise Security, than (say) Availability.

quires the input variables of Location — and is determined by the process variable of Number of Locations and Location Types. It can also be seen that there are five definite relationships relating the input, process, and output variables (i.e. numbers 4, 5, 6, 9 & 10). All the relationships relating the input, process and output variables are of a definite nature as they can be quantified quite objectively in principle. For instance, Number of Locations can be distributed quite unevenly over Location types such as Location Type 1 could have 50 Locations, Location Type 2 could have 100 Locations, Location Type 3 could have 150 Locations and Location Type 4 could have 200 Locations.[7] It can be seen that in such a distribution of Locations, within the search space, Locations are positively skewed over Location Types. Once again, the point worth mentioning, analogically, with the simulation model, is what, in essence, is this capturing, in connection with assumption detection and the proposed Goal–Diversity process intervention? It can be intuitively argued that what this type of uneven distribution is argueing for is that some non–functional attributes, during the 2nd Stage of the Individual Inspection, requires a greater search for harmful assumptions than some other non–functional attributes. While this also, to a large extent, would require empirical validation, it does raise more complex and unusual issues also for the inspection process. Firstly, if, during the Individual Inspection stage 2, some non–functional attributes had larger conceptual search spaces that required larger coverage than other non–functional attributes, then this would mandate that developers inspecting analysis representations would need to be allocated different durations of time as required effort allocation to cover these uneven Location distributions would naturally require different inspection/search durations. Therefore, if coverage is unevenly distributed between the non–functional attributes to be inspected, then this would also mean that effort allocation between the inspecting developers would also need to be distributed unevenly. Since, in the example of Number of Locations to Location Types given above, if all the four searchers were allocated 125 RUs each (theoretically sufficient effort allocation to gain 100% coverage of the search space i.e. 500

[7]Note here also, that with such a distribution of Locations over Location Types, the size of the search space is 500 Locations in size. If 250 Objects were hidden in this search space then the Object density would remain at 50%. This highlights again that distribution does not effect the actual density output variable, although it may significantly alter the efficiency and effectiveness of the search.

resource units) then the searcher predisposed to searching Location Type 1 would have enough RUs to gain 250% coverage of that Location Type while the searcher predisposed to search Location Type 4 would only have enough RUs to gain 62.5% coverage of that Location Type. Secondly, and connected directly with the first issue, this is not what would normally happen in the actual software inspection process — since the team members of the inspection team usually are allocated an equal period of time to conduct an inspection. Furthermore, as was discussed in chapter 9 and both the previous subsections of the coverage and sensitivity dimensions, extending inspection durations may well introduce fatigue effects that could undermine overall detection. Therefore, although the relationships are of a definite nature and can be quantified, such distributions are not proven and require additional search resource complexities, so, in the interests of not wanting to overly complicate the initial simulation model and possibly compromise confidence in the outputs it produces, the distribution of Number of Locations over Location Types will be kept uniform.

In Summary

From the influence diagram in figure 10.5, the eventual design decisions for the distribution dimension can be stated, as follows:-

- Distribution of Objects over Objects Types will be kept to a uniform distribution;
- Distribution of Object Types over Location Types will be kept to a uniform distribution;
- Distribution of Locations over Location Types will be kept to a uniform distribution;

Lastly, in connection with the distribution dimension, the output variable of Object Density is an important inclusion as a dynamic in the search simulation model as it directly contributes and promotes the modelling goal(s).

10.3 Search Simulation Model

In this subsection verification and validation of the search simulation model will be discussed along with a brief overview of the actual search simulation model that has been implemented — including the simulation modelling approach and building process. The verification and validation issues are discussed in subsection 10.3.1 and the simulation modelling approach and process is presented in subsection 10.3.2.

10.3.1 Model Verification and Validation

This subsection discuses the relevant simulation modelling verification and validation issues in order to: a) ensure adequate satisfaction that the actual simulation model's behaviour operates as expected; b) to discusss relevant *real–world* representational issues — in terms of the searching and detection phenomenon that the simulation model aims to capture.

10.3.1.1 Verification

It is important to ensure that the simulation model behaves as the design rationale in section 10.2 intended. It can be seen from the previous section that there were three dimensions of interest, namely: a) the coverage dimension, that relates a searcher and searcher effort issues to the search space; b) the detection sensitivity dimension, that relates a searcher to target objects; and c) the distribution dimension that relates target objects and their density to the search space.

Within these dimensions, confidence in the search simulation model's dynamic behaviour can be increased by performing a number of relating test configurations in order to help ensure confidence that the eventual implemented model performs properly.

First, on the coverage dimension, it is important to ensure that the searcher memory facility, discussed in the previous section, performs as expected. This can be checked by configuring the simulation model to have only one searcher employed

with 100% resource allocation units so that the searcher with perfect memory (i.e. p(1.00) of remembering which locations they have searched in) will cover the entire search space, and will produce the equivalent of a exponential random search[8] when the memory parameter is set to completely imperfect (i.e. p(0.00)). This test was performed, and the simulation model performed as expected: a) when the searcher had perfect memory they performed a complete coverage of the search space (i.e. a definite range search): and b) when set to imperfect memory (i.e. p(0.00)) the searcher only achieved (approx) 63% coverage – as expected. Another test on the coverage dimension is to ensure that the coverage predispositioning is behaving as expected. To do this, the simulation model is reconfigured so that there is a 1000 location search space made up of 500 locations of type 1 and 2. The two searchers are perfectly predisposed to ensure a partitioned search (i.e. both are diversely set to probability of p(1.00) to select different location types) and probability of zero in selecting the other searcher's location type. If searcher memory is also perfect (i.e. p(1.00)) for both searchers, then a 100% partitioned search can be expected. This test was done and both searchers performed a complete partitioned coverage of the search space — as expected.

Next, is the sensitivity dimension, the only check, on this dimension, is to ensure that the sensitivity predispositioning is working in the simulation model as expected. This can be tested by reconfiguring the search simulation model so that two searchers completely duplicate the search coverage of the search space, but are diversely predisposed to two different object types with 100% (i.e. p(1.00)) sensitivity on one object type and zero (i.e. p(0.00) detection sensitivity on the other searcher's object type. If object density is set high to 200% and perfect memory is also set, then when the searchers duplicate on every search space location location (on average) they should detect only the 1 object type they have

[8]A random exponential search, can be defined as ***Exponential Search = 1-(q^r).*** Where ***q*** is the probability of not searching a location within the search space for every RU allowance ***r.*** So, for example, if a searcher searches within a 1000 location search space and is allocated 1000 RU allowance, ***q*** is the probability of not selecting a search space location = 1-1/1000 = 0.999 of any location during the search not being selected. Therefore, the exponential search, in this configuration case, would be 0.999^1000=0.3677 and 1-0.3677 = 0.6323 indicating that, in terms of coverage of the search space, there was only (approx) 63% coverage — despite there actually being sufficient search resource effort allocation for 100% coverage.

perfect sensitivity to, and never detect the other object which they are entirely oblivious to — resulting in all of the target objects of both types being detected — but only one type being detected by one searcher. When this test was performed, each searcher detected all of their own object types only and collectively detected all of the 200 hidden objects diversely — as expected.

Finally, the distribution dimension is verified. However, this dimension merely handles the uniform distribution of target objects within the search space and in performing the previous tests and configurations it has been highlighted that the random/uniform allocation of target objects is behaving in the simulation model as expected.

10.3.1.2 Validation

Like all models, the search simulation model represents an abstraction from reality. The issue of model validation relates to the eventual simulation model's level of acceptable representativeness of sufficiently important real–world influences. In this case, the influence of diverse non–functional attributes upon assumption identification. In section 10.2 a design rationale was justified for the model to determine a reasonable and satifactory set of decisions in order to arrive at a search simulation model's design that fundamentally and practically captures the important real–world influences of such a searching and detection phenomenon — with respect to employing human redundancy/diversity. In the paragraphs below these are discussed in terms of model validation issues they raise.

Inclusions of real–world influences: In the interests of retaining additional real–world human detection and searching influences, two additional real–world influences included in the search simulation model's design were: a) on the coverage dimension, Searcher Memory. As discussed in section 10.2, this aspect could be acceptably modelled in an objective quantifiable manner that aids additional associated real–world detection and searching aspects to be analysed and included in the simulation experiments; b) on the detection sensitivity dimension, Object Detectability. This detection and searching influence allows the simulation model

to include an obvious influence that *'things'* detected or searched for may intrinsically be more/less difficult to detect than other *'things'*. Although, as mentioned in section 10.2, that only an absolute, and not relative, interpretation can be modelled, inclusion of this searching/detection influence facilitates analysis and promotes a more real–world valid search simulation modelling aspect.

Defense of uniform distribution: As discussed in section 10.2, on the distribution dimension only a uniform distribution of target objects within the search space is modelled. Therefore, the only dimensions of uneven distributions modelled, by the search simulation model, is: a) on the coverage dimension — in terms of *'where'* a searcher searches; and b) on the detection sensitivity dimension — in terms of *'how'* good a searcher is at actually detecting objects. Although, in using a software defect inspection analogy in the model, the validity of this simulation modelling decision may be criticised from the point of view that, in software systems, faults are not likely to always be distributed evenly throughout the artifact, it should be pointed out that the subject of the thesis, and explicit goal of the simulation model, is not to focus exclusively upon the manifestations of faults, but instead, upon the flawed assumptions that potentially cause them to become manifest. In this regard, as chapters 5, 6, and 8 indicate, assumptions have an intrinsically arbitrary, implicit and subjective nature that, as far as the literature and examples indicate, in the thesis, do not necessarily have uneven distributions. In the absence of any such empirical or experimental evidence, it seemed reasonable, as a first search simulation modelling exercise, to design the model to simply provide a uniform distribution of analogous object representations. Moreover, in the wider context of human redundancy and diversity benefits, in promoting dependability, if the search simulation model's results can indicate that, even with a uniform object distribution, that human diversity can be usefully leveraged to achieve greater defect detection, then this is, by itself a meaningful finding of the thesis.

Future Validation Issues: real validation of the model would require extensive research and investigation of the major influences upon assumption detection — that goes beyond what can be feasibly achieved within the limited resources of

a single PhD study. Such issues include additional influences like: a) Criticality and Judgement issues relating to specific domains — in terms of how does the criticality of assumptions (i.e. impact and frequency) undermine dependability and influence detection rates and what broader categories, within those identified in chapters 6 and 8, can be usefully devised and intergrated into the simulation model; b) Learning and Fatigue effects, upon detection related issues. There is, within the literature of Operations and Project Management literature on learning curves, but little, to my knowledge, directly relates its effects upon fault detection etc. In order to integrate such influences, greater empirical and experimental knowledge is required; c) Fault/Assumption Distribution, if there were sufficient time and research resources available then it may be possible to validate the simulation model from the perspective of being able to sufficiently persuade others that the search simulation was providing reasonably accurate detection results by being able to get more realistic diversity configurations and checking the resultant outputs from real inspection related data. Such data could, for instance, provide information on actual assumption/fault distributions to check whether these are random, or, if not, to what extent they are unevenly distributed.

In summarising these search simulation model validation issues, it must be admitted, due to a combination of limited suitable data and finite research resources allowed, the simulation model's design and configurations that follow this section represent, at best, explorative indications of what human redundancy/diversity benefits may be possible from incorporating diversity on the coverage and sensitivity dimensions with a uniform target object distribution. However, I do believe, that through careful analysis provided in the thesis and design of the eventual simulation model the design and subsequent configurations are not widely unrepresentative in presenting some evidence in support of the stated modelling goals in section 10.2.

10.3.2 About the Simulation

The search simulation model was developed in Java 2 using the Borland J-Builder Integrated Development Environment (IDE) — version 9. Due to both time re-

strictions and inexperience on the author's part with the language, it is presently only in the form of a text–based menu design.[9] The following subsections introduce and explain the design, interaction menus, and simulation modelling process adopted. It should be noted, however, that this section explains only the essential features of the search simulation model used — as the actual model is much richer with many other configurations possible.[10]

10.3.2.1 The Simulation approach

Simulation modelling is used in preference to other modelling approaches for a range of reasons [cf. [169]]. These include modelling situations where mathematical analytical techniques are believed inadequate or would be overly complex to apply. Another reason for adopting a simulation modelling approach is that its use can provide an insight into the problem without affecting or disrupting the actual area of interest. Finally, simulation modelling approaches are very useful when large numbers of variables need to be considered in a timely fashion. In such situations simulation modelling can compress time–scales and offer predictive solutions ahead of decision–making schedules.

In this thesis, a simulation modelling approach was specifically adopted to: a) gain greater understanding of diverse assumption detection and search–related issues which appear to have received little coverage in the literature, and which cannot be easily observed in the software development process; and b) to provide research insights into future possible empirical studies on fault–detection tasks within the wider DIRC Research Programme.[11]

[9]However, it was designed so a Graphical User Interface (GUI) could be added later.

[10]In all, the search simulation model's code size was (approx.) 23,500 lines of code (LOC) and includes simulation features that specify: a) search planning and search resource coordination of oblivious, duplicated, and unique search space regions; b) more complex effort allocation configurations — such as sectors make further demands upon searchers allocating sufficient resource units; and c) individual detection capabilities of the searcher — such as searcher memory etc. These features (with the exception of searcher memory) were not configured for the search simulations used in this chapter.

[11]DIRC is an Interdisciplinary Research Programme for improving the dependability of Computer–Based Systems...see website: www.dirc.org.uk

10.3.2.2 The Simulation process

There are five important steps to a simulation model's construction in the modelling process. These are [cf. Lucey [169]: pp.225–226]:-

1. Identify the input variables, distinguishing between controlled and non–controlled variables;
2. Where appropriate, determine the probability distribution for non–controlled variables;
3. Identify any parameters and/or status variables;
4. Identify the output variables;
5. Determine the simulation model's essential logic.

In the paragraphs that follow these five steps will be covered.

Input variables

Controlled variables represent those that, in a real search situation, would be under the direct control of management. In the simulation model these correspond to the search effort allocation dimension in figure 9.4. More specifically, they involve the number of searchers employed and the amount of effort each searcher can expend in searching for target objects (i.e. labelled resource units in the simulation model).

Uncontrolled variables represent those that, in real world search conditions, cannot be directly controlled by management. These correspond to the complexity (or difficulty) embodied in the searching task. Such variables include such factors as the size of the search environment (i.e. search space) and the dimension of the number of target objects hidden in the search environment (i.e. target object density) in figure 9.4.

Probability distributions

A simplifying assumption made in constructing the simulation model was to assume that target object density is random (or evenly distributed).[12] Therefore, in terms of target object density distribution, the probability distribution is fixed in the simulation model.

Parameter variables

Parameters also represent input variables to the simulation model, but they consist of input variables, that, for all simulation runs — during a given simulation configuration, keep a constant value. In the search simulation model, parameters reflect such factors that influence the effort allocation dimension — in terms of the constant number of resource units that are exhausted in searching any particular selected search environment location for target objects (in the search model this remains at 1 resource unit per location searched). Another dimension parameter that retains a constant value for all simulation runs concerns the target object(s) detectability (set between 0.01 (p) (i.e. very hard object to detect by any searcher) to 0.99 (p) (i.e. easy object to detect by any given searcher). Finally, the searcher capability dimension from figure 9.4 also reflects an example of parameter variables concerning the introduction of: a) search space location types and/or target object types to facilitate diversity predispositioning; and b) individual searcher location type and object type diversity profiling. [13]

Status variables

In some models, variables behave differently during simulation runs because of additional variation status introduced into the model to promote accurate modelling behaviour of the real domain under investigation — due to temporal, sea-

[12]This is an assumption often made in other analytical models — such as statistical modelling approaches, although it must be identified as a modelling assumption as claims exist that, in fault detection, faults are not equally likely to occur throughout the system but can often result in uneven densities biased towards more complex parts, modules, objects, or subsystems of a software system (i.e. a kind of Pareto dynamic). However, this would be a more sophisticated future simulation modelling inclusion.

[13]It should be noted, however, that the searcher diversity profiling/predispositioning to location types and object types can also be modelled as status variables also ...see status variables section.

sonal, or inherent inconsistencies the real domain invokes. In the search simulation model the searcher capability dimension from figure 9.4 on page 246 by allowing the searcher's diversity or uniformity predispositioning towards search space locations and/or target objects to vary between individual simulation runs.[14]

Output variables

The output variables reflect the end–state results from the simulation after calculations and randomised interactions, internally modelled in the form of (controlled/uncontrolled) input variables, parameters, status variables (if any), and probability distributions, have been performed. The output variables must provide data/information of the simulation that satisfy the original simulation model's intended objectives. In the search simulation model, the two output variables are: a) detection effectiveness — in terms of number of target objects (of whatever type modelled) are detected as a proportion of the total target object density modelled. The calculation is pretty generic and simple and reflects a common method adopted in much of the software inspection literature. The detection effectiveness calculation is: $detection\,effectiveness = \frac{number\,of\,objects\,detected}{total\,number\,of\,objects\,hidden}$; and b) detection efficiency — in terms of the number of target objects detected (of whatever type modelled) as a proportion of total effort allocation (in abstract resource units). Again, this is quite a well accepted calculation method used in software inspection to reflect the amount of effort required to detect a single fault. The detection efficiency calculation is: $detection\,efficiency = \frac{number\,of\,objects\,detected}{total\,number\,of\,resource\,units}$. It should be finally pointed out that these calculations are performed at the end of all required simulation runs and reflect the mean average over the total number of individual simulation runs performed.

[14]Such a situation is when location and/or target object predispositioning is capturing an uncontrolled team compositional type diversity (i.e. personality or cultural heterogeneity) in the simulation. In order to accurately attempt to capture its detection effectiveness or detection efficiency results over many such compositional teams, the simulation model must alter the diversity predispositioning — in–keeping with the uncontrolled nature of actually composing many such teams in reality.

Logic of the simulation model

A final modelling construction stage in the simulation modelling process is to define the main essential logic that underpins the calculations and randomised interactions of the input variables, parameters, and status variables etc. This is typically done in the form of a flow–chart that unearths the dynamic flow and decision points involved.

In figure 10.6 the essential simulation model's logic is illustrated in the form of a flow–chart. As mentioned earlier in subsection 10.3.2 the actual model is much richer and contains a number of additional input variables, parameters, and status variables etc. Figure 10.6 reflects the simplified logic view of the simulation model's calculations and interactions relevant to the simulations adopted in this chapter.

The logic reflects the flow of each searcher searching under diverse or uniform location type and object type conditions. It can be seen that the simulation starts with the random selection of a search space location number. Providing the searcher has sufficient resource units left to search the selected location (representative of effort allocation of time and/or cost in the *'real–world'* domain), the searcher has the potential to search the location. If not, this searcher stops searching. If they have sufficient resource units, then the first consideration in the search simulation model is to determine if parameters of location type diversity have been included (this models the diversity dimension of coverage see subsection 10.3.2). If this parameter is not included then the flow of the simulation model proceeds to allow the searcher to search the search space location and decrements one resource unit allocated. If this parameter is included, then the searcher will have a location diversity profile probability of selecting this type of location to search in. Providing the randomised probability of the searcher is less than or equal to their probability profile of searching the location then they will select this location as a location to search and their resource unit allocation will be decremented by one.

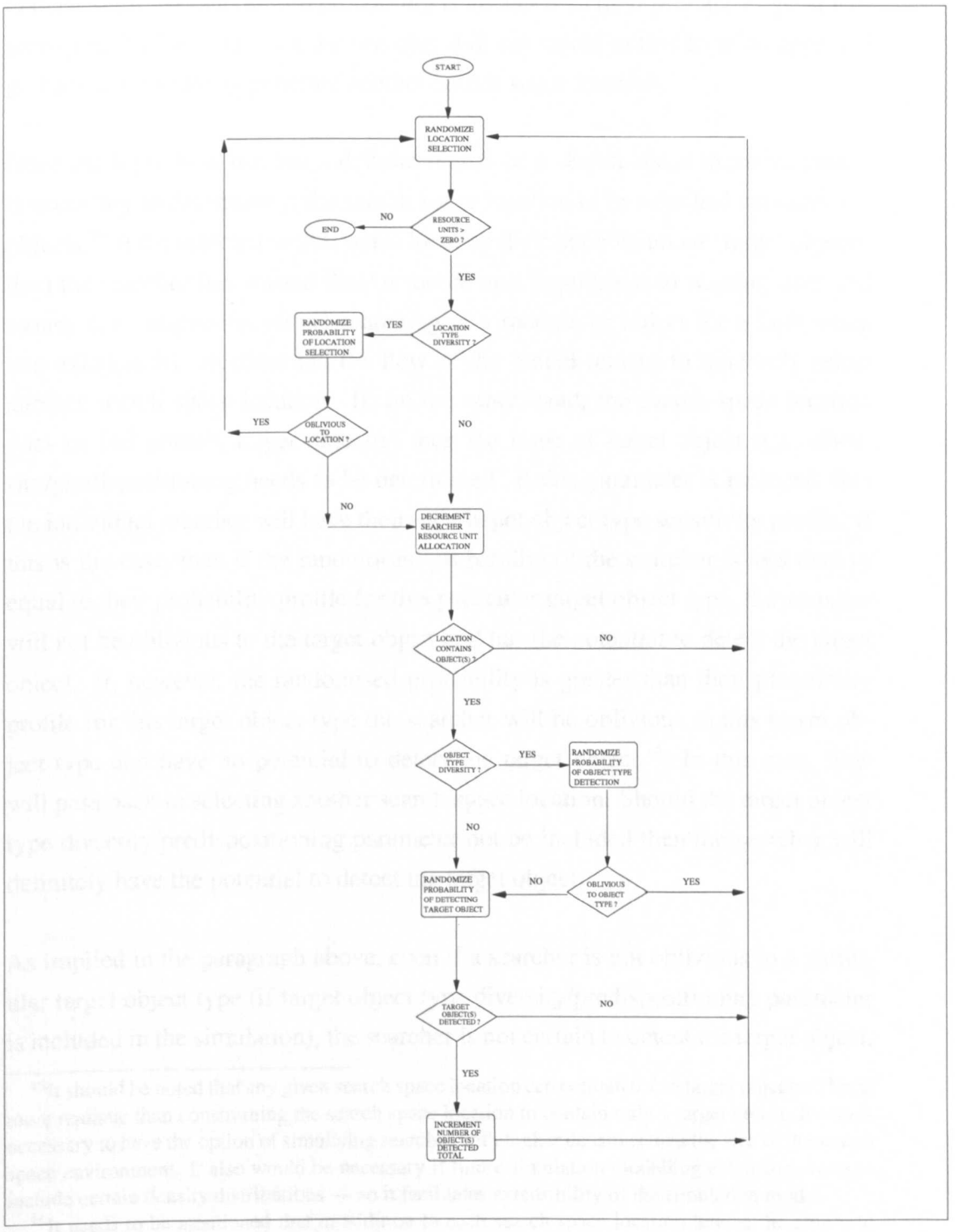


Figure 10.6: Main Simulation Logic

Otherwise, if the randomised probability is greater than their probability profile of searching the location, then the searcher will not select to this location type and go back and randomly generate another search space location.

Once the logic flow reaches a definite search of a search space location, then it is necessary to determine if the search space location to be searched contains any objects.[15] If the selected search space location does not contain any target objects, then the searcher has wasted their resource unit (equivalent to wasting time and money (i.e. inspection effort) searching in a module or object for a fault when non exist) in its selection and the flow of the model returns to randomly select another search space location. If, on the other hand, the search space location does in fact contain target object(s) then the issue of target object type diversity/predispositioning needs to be determined. If this parameter is included then the individual searcher will have their own target object type sensitivity profile. If this is the case, then if the randomised probability of the searcher is less than or equal to their probability profile for this particular target object type, the searcher will not be oblivious to the target object and has the potential to detect the target object. If, however, the randomised probability is greater than their probability profile for this target object type the searcher will be oblivious to this target object type and have no potential to detect the target object.[16] In this case, flow will pass back to selecting another search space location. Should the target object type diversity/predispositioning parameter not be included then the searcher will definitely have the potential to detect the target object.

As implied in the paragraph above, even if a searcher is not oblivious to a particular target object type (if target object type diversity/predispositioning parameter is included in the simulation), the searcher is not certain to detect the target object.

[15]It should be noted that any given search space location can contain *0,1..n* target objects. This is more realistic than constraining the search space location to contain only 1 target object. It also is necessary to have the option of simulating searches with higher densities than the size of the search space environment. It also would be necessary if future simulation modelling extensions were to include certain density distributions — so it facilitates extensibility of the simulation model.

[16]It needs to be mentioned that in addition to each search space location having the ability to contain *0,1...n* target objects, any single search space location can contain many target objects of different types.

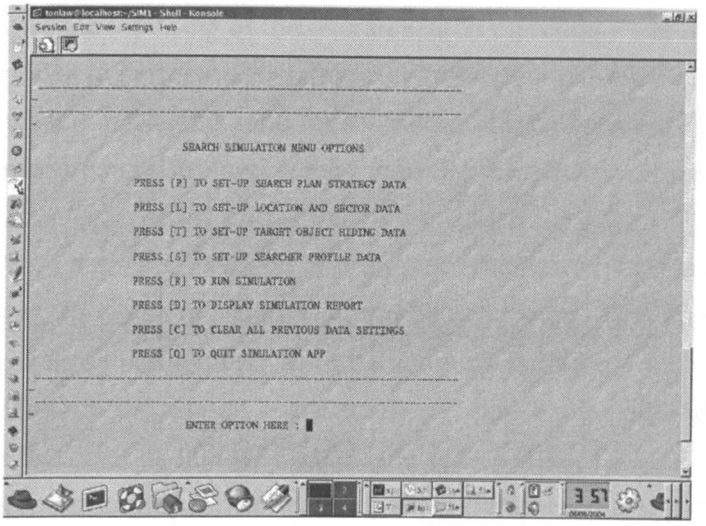


Figure 10.7: Main Menu Screen Options

This is because, independent from any target object type diversity/predispositioning, each target object has a general detectability probability. This is why, in the flow–chart logic, irrespective of the inclusion of target object type diversity/predispositioning, the logic flow flows next to randomising the probability of the target object (of whatever type) being detected by the searcher. If this is less than or equal to the searcher's detection proficiency then the searcher will detect the target object. The total target object detection count will then by incremented by one before flow control passes back to randomly selecting another potential search space location to search. Conversely, if the randomised probability is greater than the searcher's proficiency then they will not detect the target object and flow will pass back to randomly selecting another potential search space location to search.

This then concludes the discussion of the search simulation model's actual design and construction within the simulation modelling process.

10.3.2.3 Brief Overview of the Simulation Model

In this subsection an overview of the actual search simulation model is provided. As mentioned earlier in subsection 10.3.2 the simulation model is a text based menu design. In figure 10.7 the main menu options available are shown. Menu

option "P" allows the inputting of the overall search planning — in terms of allowing the user options to perform a resource constrained or unconstrained search and whether they wish to perform a randomised or coordinated search. If the user opts to perform a resource constrained search then the maximum number of resource units per searcher (i.e. effort allocation) is presented. Additionally, If the user selects to perform a constrained and coordinated search then they are required to set–up the degree to which the search should be conducted. This involves the setting of a further three parameters: a) the amount of individual searcher resource units that will search the search space uniquely; and b) the number of individual searcher resource units that will search the search space locations in duplicate with other searchers employed. Collectively these parameters and controlled input variables allow for the beginnings of an overall search plan.

The next main menu option "L" allows the user to define the search space environment (see figure 9.4). Firstly the user is asked to enter the size (in search space locations) of the entire search environment. If a coordinated overall search strategy has been selected above then the user is asked to decide a) whether they wish all the searchers to be oblivious (irrespective of diversity/predispositions) to some subset of the search space; b) whether they wish all searchers to search part of the search space uniquely; and c) whether they wish all the searchers to search some part of the search space in duplicate with other searchers. If *yes* is selected to any of these options then the user is prompted to enter the total number of search space locations that are to be searched or avoided in the way desired.

Following these selections, the user is asked to input the parameter(s) of search space location types. If some diverse predispositioning is required then multiple search space location types can be entered. If not, then the user inputs only 1 type to be simulated. One of the richer features, not employed in simulations in this chapter, involves the ability to determine the number of search space location sectors within each individual search space location. This allows for a much more complex effort allocation simulation where the individual searchers must allocate a number of resource units on selection of a search space location to search. In this chapter this option is precluded. This is done by allocating only one sector

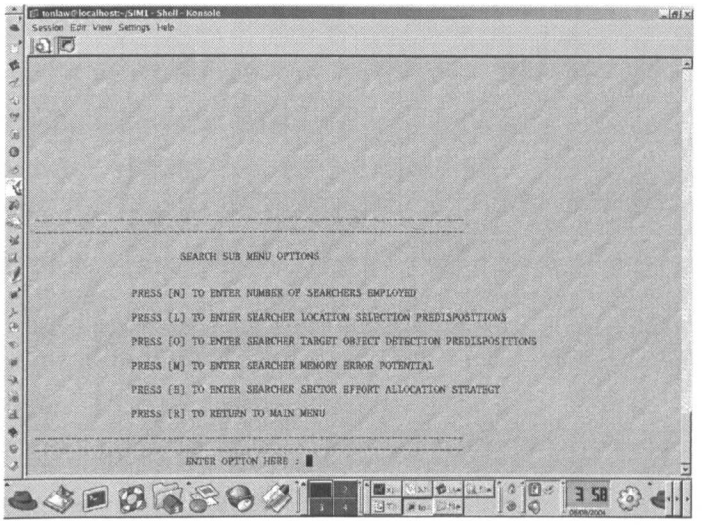


Figure 10.8: Searcher Sub Menu Screen Options

per search space location.[17]

The next main menu option "T" allows the user to define the uncontrolled variable inputs of target object densities. It should be noted that the simulation model is designed to accommodate any density level.[18] These are entered as the total number of target objects to be randomly hidden within the search environment. Next, the user is asked to enter the parameter of target object types. If diversity predispositioning is required then multiple target object types may be entered. If not, then only one target object type should be entered. If multiple sectors per search space location has been included (not used for simulations in this chapter), then the next input option allows the user to distribute the number of target objects among the sectors in a search space location. This can be randomly distributed

[17]In this more complicated model, target objects (of whatever type) can be allocated to sectors contained by search space locations. The more sectors per location means that search situations arise where individual searchers must allocate sufficient effort if they are to stand any potential of detecting a target object i.e. say a location contains ten sectors, and sector nine contains a target object, the searcher must allocate at least nine resource units to provide sufficient effort coverage to have the chance of detecting the target object.

[18]From the software inspection literature, this allows an analogous model of artifact quality i.e. the larger the density the poorer quality of the artifact and vice versa.

or the user can specify which sector should contain target objects.[19] Next, the detectability of the target objects can be entered. This can be any probability, as a decimal fraction, between 0.01 (i.e. very hard to detect) and 1.00 (i.e. will definitely be detected, subject to search space and effort coverage). The simulation model allows the user to either randomise these on every simulation run (i.e. as a status variable) or as a constant throughout all simulations (i.e. as a parameter).

The next main menu option "S" takes the user to the searcher sub menu screen illustrated in figure 10.8. From this sub menu the user can enter the controlled input variable of number of searchers to employ in the search. This is chosen by selecting "N". By entering "L" the user can now specify the searchers' diversity/predispositioning profile to search space locations types. This profile can be: a) set to a random diversity/predispositioning (i.e. as a status variable) on each simulation; b) set to be a constant uniform predispositioning across all searchers employed (i.e. as a parameter) across all simulation runs; or c) set to a specific and constant profile pattern, by the user (i.e. as a parameter) across all simulations. The next sub menu option "O" allows the user to specify the searchers' diversity/predispositioning profile to target object types. This profile, again, can be: a) set to a random diversity/predispositioning (i.e. as a status variable) on each simulation; b) set to be a constant uniform predispositioning across all searchers employed (i.e. as a parameter) across all simulation runs; or c) set to a specific and constant profile pattern, by the user (i.e. as a parameter) across all simulations. Sub menu option "M" allows the user to specify the level of all the searchers' memory — in terms of remembering which search space locations they have already searched. This parameter can be set as a probability figure in decimal fraction format between 0.00 (i.e. no memory of previous search space locations searched) and 1.00 (i.e. perfect memory of previous search space locations searched). The last sub menu option "E" allows the user to determine how much effort coverage (in terms of resource units) will be allocated to searching a search space location. This parameter is only of any real influence when the simulation model is configured with multiple sectors per search space location. In

[19]Note: Multiples of target objects can be allocated to any single sector in a search space location

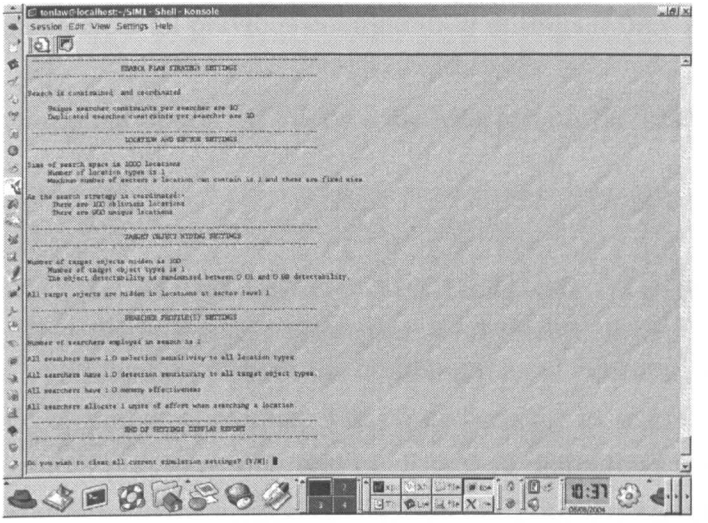


Figure 10.9: Configuration Settings Screen

such a configuration, the user can: a) ensure sufficient effort coverage is allocated to search all sectors within a search space location; b) allocate effort coverage to only those search space locations that contain (at least) one target object. This is a kind of partial knowledge search; or c) only allocate a specified amount of search effort coverage per search space location as a constant parameter. In the simulations, in this chapter, no search space location sectors are configured. Therefore, effort coverage allocation is one resource unit requirement to completely search one selected search space location.

The sub menu option "R" returns the user back to the main menu options screen (see figure 10.7).

The next main menu option "R" allows the user to actually run the simulation. This is entered as any positive integer (i.e. greater than or equal to one). Following the entry of this figure, the simulation automatically starts. Main menu option "D" allows the user to view the output variables of detection effectiveness and detection efficiency — once the simulation has terminated. Main menu option "C" allows the user to view all the configurations previously set–up by them — in case the model user forgets which input variables, parameters, status variables etc have been entered. Figure 10.9 illustrates this screen. Additionally, from this

screen the user can clear all previous configuration settings if they wish.

Finally, the main menu option "Q" allows the user to quit the search simulation model.

10.4 Configuration of the Simulation Model

In this section a clarification of both the combinations of potential search strategies and degree of predispositioning will be discussed. In subsection 10.4.1 a reasonable degree of predispositing will be discussed that allows both an insight into the potential benefits of acheiving diversity on both the search space coverage and object sensitivity capability dimensions, while attempting to retain a feasible and realistic degree of diversity/uniformity predispositioning. Furthermore, as hinted at in providing a rationale in section 10.2, it will also be necessary to clarify that the two dimensions capture different dynamics in their respective predispositioning dimensions of search space coverage and object sensitivity capability. In subsection 10.4.2, these configurations from subsection 10.4.1 will be analysed for sensitivity to important simulation parameters maintained in the simulation model to help retain some more realistic and real–world influences — as discussed in the design rationale section 10.2.

Firstly, however, from what has been discussed so far in this thesis, and particularly what has been more explicitly highlighted in this chapter, we can see that in order to fulfill the simulation model's goal(s) from section 10.2.1, it is necessary to simulate for comparison the three potential approaches to promoting non–functional attributes during the early phases and products of the software development process. To begin with there is the Ad-hoc approach to considering non–functional attributes. As has already been discussed, this approach can result, through common homogeneous and uniform influences of the the system type, application domain, and the broader collective cultural and experiential backgrounds of the specific developers, in their predispositions being more uniform and overlapping in their promotion and consideration of non–functional attributes. Next, there is the Systematic approach to considering non–functional attributes. Again,

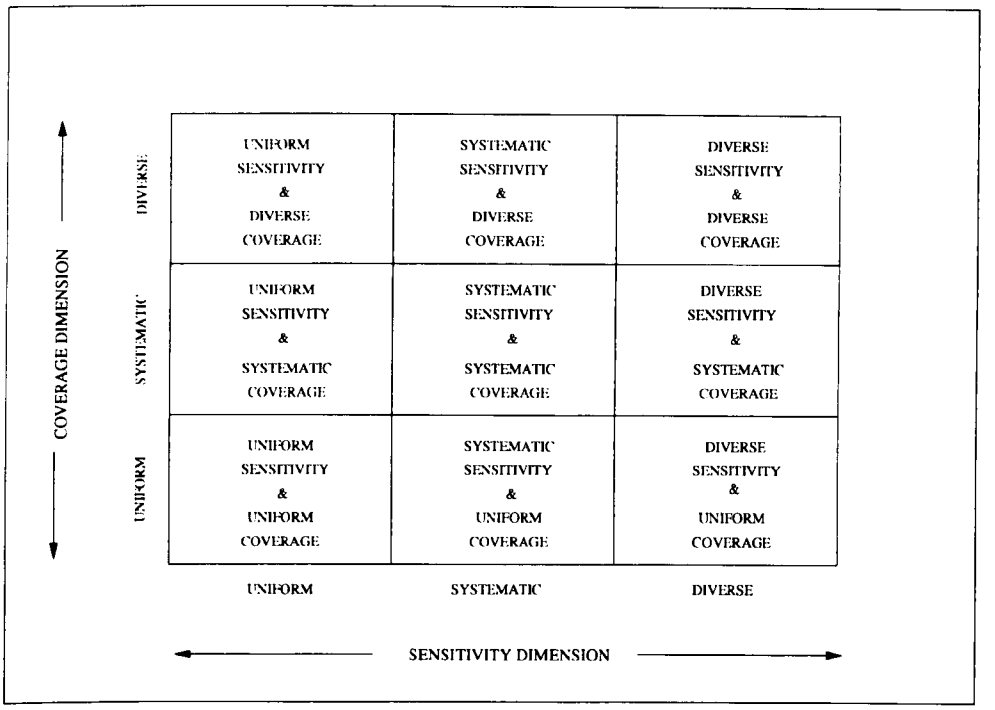


Figure 10.10: The Nine Possible Search Strategis

as has already been discussed in this thesis, whilst such consideration is an improvement, within the development process, in an interventionist manner, it also fails to recognise the cognitive limitations of developers who are expected to simultaneously promote multiple interrelated goals. As both research and anecdotal evidence has shown, when multiple goals are to be simultaneously considered, there is the inevitability that what in fact happens is that developers constantly switch between representations and overall achieve a more shallow consideration. Finally, there is the proposed goal–diversity process intervention of this thesis. In this situation, not only is this interventionist to help ensure that important non–functional attributes of dependability are considered, but this process intervention, unlike the Systematic approach, also recognises the cognitive limitations involved in the promotion of multiple goals at once. With this proposed process intervention, as has already been discussed, a single goal is promoted throughout the three envisaged stages allowing a more acute and deeper consideration of important non–functional attributes that contribute to overall computer–based system dependability — as a super—ordinate system goal.

When we interpret these three approaches into the two critical dimensions of search space coverage and object sensitivity capability, it can be seen that there are in fact nine possibilities to consider. These are shown in figure 10.10. Looking from left to right and bottom to top, it is first possible to see that it is possible to characterise the Ad-Hoc approach as one that, at worst, results in too much overlapping, as the Uniform Coverage and Uniform Sensitivity influence (UC & US). The next two along the bottom offer, at least a variation of this theme that is worth considering, in terms of less pessimistic possibilities whereby, although coverage is essentially overlapping and uniform, there may be less influence of this upon object sensitivity capability. The middle case is therefore the Uniform Coverage and Systematic Sensitivity variation (UC & SS). At the most optimistic, within this uniform coverage, there is the variation of Uniform Coverage and Diverse Sensitivity (UC & DS) which considers the possibility that whilst developers may be heavily uniform and overlapping, in terms of the non–functional attributes they consider, they do tend to be, at times, more diverse in flawed assumptions they detect. Moving up one row, it is possible to characterise the Systematic consideration of non–functional attributes — in terms of a shallow and more switching consideration. At the far left of the middle row, it is possible to consider the more pessimistic case that although the searchers are constantly switching their coverage predispositioning, there is a uniform or homogeneous bias influencing them — in terms of their likelihood to detect the same types of objects. This is characterised by the Systematic Coverage and Uniform Sensitivity dimensions combination (SC & US). The centre box of the middle row attempts to capture the more middle case, that if developers or searchers keep switching their promotions and considerations, then they are likely to also have a more systematic, if perhaps, shallow sensitivity to detecting objects also. This middle case is the Systematic Coverage and Systematic Object Sensitivity combination (SC & SS). The far right box of the middle row captures, again, a more optimistic possibility for the Systematic consideration of non–functional attributes. Here, whilst searchers achieve a shallow and ever–switching consideration of the search space, they are more diverse in terms of which object types they are more likely to detect. This more optimistic case is provided by the Systematic Coverage and Diverse Object Sensitivity combination (SC & DS). Finally, the top row attempts to characterise

the proposed goal–diversity process intervention, by offering more diversity with respect to coverage — due to a deeper and more focused consideration of different non–functional attributes throughout the early stages of the software development process. At the far left, however, is the pessimistic case where, although searchers are more likely to be diverse with regards to search space coverage, they are, for some reason, more likely to be uniform or homogeneous in object detection sensitivity to the same types of objects. This is characterised by the Diverse Coverage and Uniform Sensitivity combination (DC & US). Next is the middle case situation, where, although the searchers are more diverse in covering the search space, for some reason, they have only a shallow and equal object detection sensitivity capability. This strategy combination is the Diverse Coverage and Systematic Object Sensitivity (DC & SS). Finally, there is the optimistic diversity case for the proposed goal–diversity process intervention, whereby, due to a more diverse coverage of the search space, searchers become more diverse in their object detection sensitivity capability. This search strategy is the Diverse Coverage and Diverse Object Detection Sensitivity (DC & DS).

Apart from these configuration considerations, it has been decided to model five developers to test the first simulation modelling goal from section 10.2.1 of comparing the three potential approaches to considering non–functional attributes. The reasons for this are that, in consideration of dependability, within the wider literature, five non–functional attributes are often considered fundamentally important. Therefore, five seems a reasonable number to configure the simulation model with. However, in order to consider the second modelling goal from section 10.2.1 of the effects of under/over representation of important non–functional attributes, two to ten location types (i.e. goals) will be configured to offer sufficient insight into the over/under representational dynamics and influences concerning the proposed goal–diversity process intervention.

Furthermore, each search strategy configuration is simulated 1000 times utilizing a 500 location size search space with human resources factored–out to a theoretical 100% search coverage effort allocation — as discussed in detail in various parts of the design rationale in section 10.2.

Sensitivity Profile	Object Type 1	Object Type 2	Detection Effectiveness
Searcher_One	0.75	0.25	0.50
Searcher_Two	0.25	0.75	0.50

Table 10.2: Equivalent Object Detection Effectiveness

10.4.1 Predispositioning

In this subsection there are two issues to be dealt with. Firstly, there remains the issue of how both the uniformity/diversity dimensions of search space coverage and object sensitivity capability capture different dynamics. This is discussed in subsection 10.4.1.1. Secondly, there is a justification necessary for arriving at a reasonably realistic degree of uniformity/diversity predispositioning. This is dealt with in subsection 10.4.1.2.

10.4.1.1 Differing Dimension Dynamics

While both dimensions characterise uniformity and diversity predispositioning, it is important to note that the two essentially capture different dynamics. On the coverage dimension, location selection is capturing an option dynamic whereby if one option is not taken another must i.e. 2 location types presents the searcher with a forced either/or choice situation. Conversely, with object detection sensitivity, the simulation model is capturing a capability dynamic i.e. given that the searcher has covered the area where two objects are, and therefore has the potential to detect the objects, how capable are they at detecting them is the issue.

To illuminate this further, let's provide an analogy. Imagine a man is faced with being forced to select one of four rooms to enter by opening the door and entering. Each room has a probability of being entered, either room 1 will be entered, or room 2, or room 3 or room 4. The probability of any one of the 4 rooms being entered will have to sum to p(1.00). Because of whatever influences, any room may have a more or less likelihood of being selected for entry by the man but the sum of the probabilities of the 4 rooms being entered must always sum to 1.00 as it is a forced choice dynamic that is actually being captured. Now let's say that the man has to wear a special pair of spectacles before selecting and entering a room,

once he has entered a room a number of shapes are on the wallpaper of all the four rooms (no matter which room is picked the wallpaper is the same and has the same shapes in all the rooms) there is a square shape, a triangle, a circle, and a star on different walls of each room. The spectacles are specially made to impair the man from seeing the shapes, indeed, let's say, that the design of the spectacles are the result of 1,000 such trials and experiments of asking people to enter a room and report what shapes were detected. From these experiments the glasses are known to impair detection of the triangle 10% of the time, the square 25% of the time, the circle 50% of the time, and the star 75% of the time. Note that, in this case, the probabilities do not add up to one, as, each shape's probability captures the single detection capability or if subtracted from 1.00 (i.e. triangle 1.00 - 0.90 = 0.10) its detection impairement influence over the detection of the four different shapes. For instance, the man in my example after entering the room may walk back out and report that he has not detected anything, or he may say *"I seen a triangle and a square, or a circle and a square"* or any such combination — including all four shapes. The important point to note is it is not like when he selected a room, in this case he does not have to detect anything at all. It's not the case that if he doesn't detect the triangle he must detect the square, or the circle, or star — as it is not a forced choice situation and therefore the probability over all four shapes in the room being detected does not have to add up to p(1.00).

However, in terms of detection effectiveness, the spectacle's overall detection sensitivity to all the shapes is important if we wish to compare two sets of different glasses with equivalent overall detection sensitivity over the four shapes. The important issue here is that the overall detection effectiveness capability profile over the number of object types must be the same — even though a searcher may be more or less capable and effective at detecting certain object types than others. So for instance, on object type 1 *searcher_one* may be 0.75 effective at detecting object type 1, and 0.25 effective at detecting object type 2. Conversely, on object type 1 *searcher_two* may be 0.25 effective and on object type 2 be 0.75 effective. We can see from table 10.2 of this example that although both searchers are significantly diverse in their detection capability over the two object types, they are, however, equal in terms of their overall detection effectiveness because $\frac{(0.75 + 0.25)}{2\ Object\ Types} =$

Configuration	Type 1	Type 2	Type 3	Type 4	Type 5	Check
Coverage	0.30	0.25	0.20	0.15	0.10	Prob' Dist = p(1.00)
Sensitivity	0.70	0.60	0.50	0.40	0.30	Det' Effect = 0.50

Table 10.3: Diverse Mid–Case for Analysis

0.50 detection effectiveness for both *searcher_one* and *searcher_two*. If we don't ensure that overall detection effectiveness is equal in comparing different predisposition configurations, then we will not be comparing the detection benefits of diversity, but instead, getting confused outputs influenced by the detection effects of more/less capable individual searchers.

Therefore, the coverage dimension captures a probability selection distribution that must always sum over the number of location types to p(1.00). The object detection sensitivity dimension captures a capability dynamic of a success/failure of detection profile over the number of object types configured in the simulation model, the sum of which, must always be equivalent between compared searchers in terms of their detection effectiveness if we wish to compare the diversity/uniformity influences of two or more competing search strategies.

10.4.1.2 Predispositioning

This subsection considers what amount or degree of predispositioning is suitable for conducting a sensitivity analysis in subsection 10.4.2 — in order to get an insight into some of the realistic detection dynamic behaviour concerning the influence of: a) Searcher Memory; b) Object Detection Difficulty; and c) Object Density. It seems reasonable, in this case, to take a mid–point in terms of both the dimensions of coverage and sensitivity — since, so long as these different dimensional configurations capture the nine possible uniformity/diversity categories, then a mid-case of search space coverage and object detection capability should allow a reasonable insight into the potential influences of the three chosen sensitivity parameters on the nine possible search strategies.

Therefore, in terms of coverage dimension, we can see that they should vary around the 0.20 area and sum, over 5 location types to p(1.00) as discussed above

Uniform Coverage	L1	L2	L3	L4	L5
Searcher 1	0.30	0.25	0.20	0.15	0.10
Searcher 2	0.30	0.25	0.20	0.15	0.10
Searcher 3	0.30	0.25	0.20	0.15	0.10
Searcher 4	0.30	0.25	0.20	0.15	0.10
Searcher 5	0.30	0.25	0.20	0.15	0.10

Table 10.4: Uniform Coverage Characterisation

in subsection 10.4.1.1. With regards the sensitivity dimension, we can see from subsection 10.4.1.1 that the overall detection effectiveness over 5 object types should be 2.5 and should vary around 0.50.

It can be seen that in order to both achieve a middle case, for sensitivity analysis in subsection 10.4.2, and still allow uniform/diverse overall detection performance comparisons of the nine possible search strategies discussed in section 10.4 earlier, then the coverage distribution and sensitivity profile will need to be configured as in table 10.3.

In order to accommodate each of the nine search strategies, these mid–case figures, on both dimensions, will need to be capable of characterising: a) uniformity; b) systematic; and c) diversity so that all nine combinations are possible for configuration.

First, then, we can see, amongst 5 searchers on the coverage dimension that uniform coverage can be chracterised as in table 10.4. It can be seen that the probability distribution sums to p(1.00) and also attempts to capture coverage unformity — since all of the searchers involved are biased to Location types 1 & 2 and away from Location types 4 & 5.

Secondly, we can see that amongst 5 searchers on the sensitivity dimension that uniform sensitivity can be characterised and configured as in table 10.5. Again, on this dimension it can be seen that the sensitivity results in 0.50 detection effectiveness overall and the detection between the searchers is biased towards Objects types 1 & 2 and away from Object types of 4 & 5.

Uniform Sensitivity	O1	O2	O3	O4	O5
Searcher 1	0.70	0.60	0.50	0.40	0.30
Searcher 2	0.70	0.60	0.50	0.40	0.30
Searcher 3	0.70	0.60	0.50	0.40	0.30
Searcher 4	0.70	0.60	0.50	0.40	0.30
Searcher 5	0.70	0.60	0.50	0.40	0.30

Table 10.5: Uniform Sensitivity Characterisation

Systematic Coverage	L1	L2	L3	L4	L5
Searcher 1	0.20	0.20	0.20	0.20	0.20
Searcher 2	0.20	0.20	0.20	0.20	0.20
Searcher 3	0.20	0.20	0.20	0.20	0.20
Searcher 4	0.20	0.20	0.20	0.20	0.20
Searcher 5	0.20	0.20	0.20	0.20	0.20

Table 10.6: Systematic Coverage Characterisation

Next, with regards to 5 searchers on the coverege dimension it's possible to see that systematic coverage can be characterised as in table 10.6. Here, the table shows that because multiple goals are simultaneously being considered the probability distribution is equal across all Location types to capture the dynamic that the searcher keeps switching their focus.

Next, with regards to 5 searchers on the sensitivity dimension that Systematic sensitivity can be characterised as in table 10.7. In this case, the individual success/failure probabilities on each Object type across the profile is also equal in order to capture the dynamic that, because the searcher keeps switching Location type focus, the detection also becomes equally likely and unlikely.

Systematic Sensitivity	O1	O2	O3	O4	O5
Searcher 1	0.50	0.50	0.50	0.50	0.50
Searcher 2	0.50	0.50	0.50	0.50	0.50
Searcher 3	0.50	0.50	0.50	0.50	0.50
Searcher 4	0.50	0.50	0.50	0.50	0.50
Searcher 5	0.50	0.50	0.50	0.50	0.50

Table 10.7: Systematic Sensitivity Characterisation

Diverse Coverage	L1	L2	L3	L4	L5
Searcher 1	0.30	0.25	0.20	0.15	0.10
Searcher 2	0.10	0.30	0.25	0.20	0.15
Searcher 3	0.15	0.10	0.30	0.25	0.20
Searcher 4	0.20	0.15	0.10	0.30	0.25
Searcher 5	0.25	0.20	0.15	0.10	0.30

Table 10.8: Diverse Coverage Characterisation

Diverse Sensitivity	O1	O2	O3	O4	O5
Searcher 1	0.70	0.60	0.50	0.40	0.30
Searcher 2	0.30	0.70	0.60	0.50	0.40
Searcher 3	0.40	0.30	0.70	0.60	0.50
Searcher 4	0.50	0.40	0.30	0.70	0.60
Searcher 5	0.60	0.50	0.40	0.30	0.70

Table 10.9: Diverse Sensitivity Characterisation

Next, with regards to 5 searchers on the coverege dimension. Diverse coverage can be characterised as in table 10.8. In this example, because the individual searchers maintain more focus and have been deliberately predisposed to be different, then each searcher is more/less likely to search certain Location types in a complimentary diverse manner to every other searcher (e.g. Searcher 1 is 3 x more likely to search Location type 1 than Searcher 2, and Searcher 5 is 3 x more likely to search Location type 5 than Searcher 1, etc).

Next, with regards to 5 searchers on the sensitivity dimension, the diverse sensitivity can be characterised as in table 10.9. Here, the searchers are diversely complimentary about searching Locations of a certain type, they retain focus for longer on that Location type and hence are more likely to be more diversely complimentary on the Object sensitivity dimension — in terms of being more likely to find Objects of different types.

These six characterisations of the three potential approaches to considering non–functional attributes allow us to fulfil the simulation modelling Goal One from section 10.2.1, as they can be combined in different combinations to allow the configuration of all the 9 possible search strategies in subsection 10.4.

Furthermore, as mentioned in subsection 10.4.1.1, the figures are not only arrived at to offer a middle–case for further sensitivity analysis on certain simulation model parameters, they are specifically chosen to uphold the characteristic natures of the two critical dimensions of coverage and sensitivity. The coverage dimension captures a choice dynamic and therefore, over the sum of the Location types, the added decimal fractions must always sum to 1.00, since it is a probability distribution that is being modelled. On the other dimension of object sensitivity, however, the dimension captures a capability dynamic, whereby individual decimal fractions capture the detection success/failure probability of detecting a certain object type. In this case, the sum over the total number of object types does not have to sum to 1.00. Instead, what is important, is that like is being compared with like, regarding the detection effectiveness of a given searcher under a given uniformity/diversity predisposition. Therefore, the sum averaged over the number of object types must be an equal decimal fraction to ensure that the searchers under differing diversity/uniformity configurations represent the same overall detection effectiveness. Only when these two criteria are met is the sole influence of coverage and sensitivity uniformity/diversity being observed between competing search strategies.

10.4.2 Sensitivity Analysis

In this subsection, the nine potential search strategies for comparison that utilize the middle–case configurations and predispositions from subsections 10.4.1.1 and 10.4.1.2 will be analysed for their absolute and relative sensitivity. The absolute variance is the amount each strategy varies as the particular simulation model parameter is altered over a range. The relative variance provides a comparison of whether or not any one or more of the nine search strategies is more/less sensitive by comparison to the others. The relative variance is the standard deviation divided by the mean average (cf. [169]) of the detection distribution recorded for each step alteration of the particular simulation model parameter being varied.

A total of three parameters are studied for their sensitivity impacts upon the nine search strategies in this section. In subsection 10.4.2.1, the parameter of Object

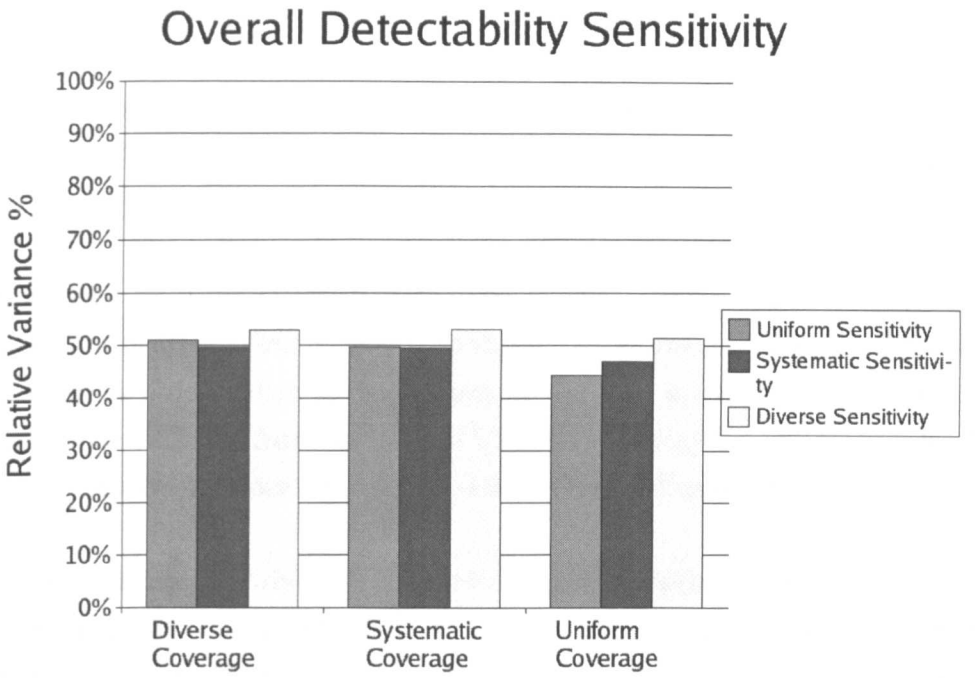


Figure 10.11: Object Detectability Sensitivity

Detectability is studied. In subsection 10.4.2.2 Object Density is studied. In subsection 10.4.2.3 Searcher Memory is studied. It should be pointed out that all of these parameters can be varied from 1.00 to 0.00 and only one at a time is varied over the range with the others held constant.

10.4.2.1 Object Detectability

The first simulation model parameter studied for its sensitivity on the nine potential search strategies is Object Detectability. As noted in section 10.2, Object Detectability was an important searching/detection dynamic that was kept in the simulation model's design so that, at least, some real–world complex dynamics could be approximated for sensitivity effects upon searching predispositions. As was noted there, however, the view taken concerning Object Detectability, is a simplified absolute perspective that assumes that an object can be difficult to detect by its own nature — rather than a relative influence that can exist between an object and a particular searcher.

In this section the nine predisposition searching strategies employing uniformity/diversity from the previous sections are studied for there absolute and relative variability under the influence of this simulation modelling parameter. A total of ten steps where taken with this parameter, varying the Object Detectability from 1.00 (perfectly detectable) to 0.10 (very difficult to detect) in steps of 0.10.

It can be seen from figure 10.11 that whilst there is a large degree of absolute variance — with all nine search strategies varying by as much (or approx. around) 50%. There is little relative variance between the nine strategies with the least relative variance coming from the UC and US search strategy (i.e. 44.36%), whilst the highest relative variance emerges from the SC and DS strategy (i.e. 53.02%).

This would indicate that whilst all of the nine search strategies are highly sensitive to the influence of Object Detectability, in an absolute way, all of the strategies are relatively affected in the same way — meaning that while they may be some detection performance loss by one searching strategy from one to the other, it is anticipated that this will be marginal.

10.4.2.2 Object Density

The second simulation model parameter studied for sensitivity analysis is Object Density. As discussed in section 10.2, the object density and the size of the search space are important user configuration considerations that allow the user to characterise the quality or dependability of the artifact in a simplified manner.

In this section the nine predisposition searching strategies employing uniformity/diversity from the previous sections are studied for there absolute and relative variability under the influence of this simulation modelling parameter. A total of ten steps where taken with this parameter, varying the Object Density from 100% density to 10% density in steps of 10%. By analogy, the fewer the objects that exist in the search space, the higher the quality of the artifact (given an absolute simplification), but the more likely searchers will expend resource unit (RU) effort for no return (i.e. finding/detecting) therefore the less efficient becomes the search.

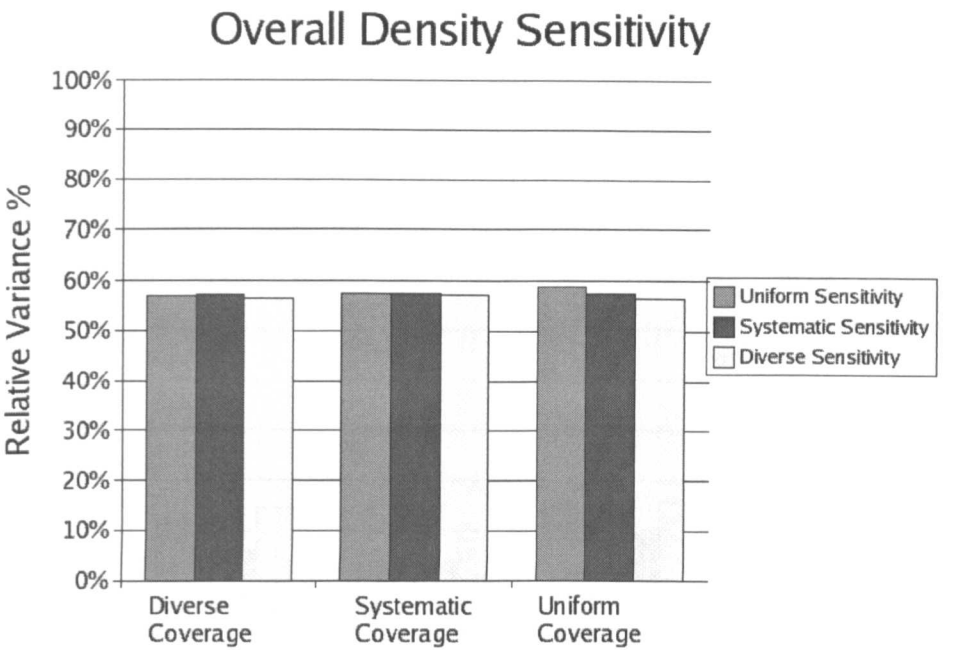


Figure 10.12: Object Density Sensitivity

It can be seen from figure 10.12 that there is a large absolute variance indicating that all of the nine search strategies are highly sensitive to this simulation modelling parameter of Object Density. However, in relation to each other, none of the nine search strategies are more sensitive to this particular parameter than the others, with differences between them being very marginal. This can be seen from the fact that the least affected, in relative terms, is the UC and DS search strategy (i.e. 56.34%) whilst the most affected is the UC and US search strategy (i.e. 58.80%).

As a result, it can again be reasonably concluded that, while the detection performance of all the search strategies will be highly affected by variances in Object Density, they will, to a large extent, all be affected relatively equally — meaning that no one strategy is unduly affected, in relation to the others, by this modelling parameter of Object Density.

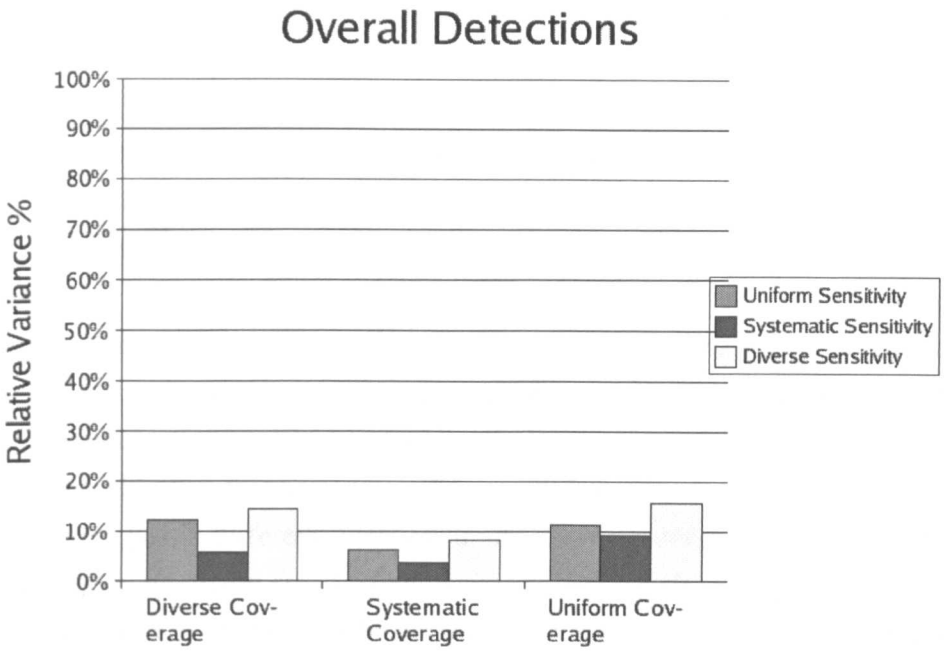


Figure 10.13: Searcher Memory Sensitivity

10.4.2.3 Searcher Memory

The last sensitivity parameter concerns the individual Searcher's memory to individually remember where they have searched. As section 10.2 highlighted, this is another simulation modelling parameter that was retained in the interests of designing a slightly more realistic simulation model that can be studied for its sensitivity influences and it is anticipated that the individual Searcher's memory will have an impact on both coverage and sensitivity dimensions, since the less they can remember where they have searched the less effective and efficient the search will become as they increasingly overlap upon themselves.

In this section the nine predisposition searching strategies employing uniformity/diversity from the previous sections are studied for their absolute and relative variability under the influence of this simulation modelling parameter. A total of ten steps where taken with this parameter, varying the Searcher's Memory from a probability of 1.00 (i.e. perfect memory) down to 0.10 in decreasing steps of 0.10 to see both the absolute and relative sensitivity effects it has upon searching performance

under the nine possible search strategies and configurations discussed in sections 10.4.1.1 and 10.4.1.

It can be seen from figure 10.13 that whilst the absolute sensitivity is relatively small, by comparison with the other two simulation modelling parameters in subsections 10.4.2.1 and 10.4.2.2 (i.e. the highest is 15.54%), the relative variance does seem more pronounced indicating that the individual potential search strategies are more likely to be individually affected than others. The least sensitive is the SC and SS strategy (i.e. only 3.55%) whereas the most affected strategy is the UC and DS strategy (i.e. on 15.54%) and in relative terms approx 4 times as sensitive. However, although this parameter of Searcher Memory does appear to affect individual strategies in different ways, it should also be remembered that the absolute variance is relatively small so whilst individual search strategies may well be differently affected by this simulation modelling parameter its overall affect is likely, on this evidence, to be also nominal.

10.5 Simulation Experiments

In this section the simulation model's configurations and predispositions from section 10.4 are employed to fulfil the goals of the model. It can be seen from that section in subsections 10.4.1 and 10.4.2 that the configurations allow a reasonable approximation of the three approaches to considering non–functional attributes, namely: a) Ad-Hoc Approach; b) Systematic Approach; and c) The proposed Goal–Diversity process intervention advocated in this thesis. The sensitivity analysis performed in utilizing and comparing these possible nine combinations of these approaches revealed that, in the main, all the search strategies are either equally sensitive, in the absolute sense, or any relative variance between them is very marginal (i.e. Searcher Memory). Therefore, the configuration settings involving a middle–case, discussed in subsection 10.4.1 will be utilzed for the diversity/uniformity detection performance comparisons. However, it will be more realistic to also ensure that the three simulation modelling parameters analysed for sensitivity in subsection 10.4.2 are also more realistically set to give a closer (or as close as a simulation model can expected to) approximation by integrating

a more realistic configuration of these three modelling parameters.

Therefore, the three modelling parameters of: a) Object Detectability; b) Object Density; and c) Searcher Memory will be configured for these comparison simulations as follows:-

- Object Difficulty: will be set at 0.50 for all objects hidden in the search space of all types. This is reasonable as it represents a middle case;
- Object Density: will be set at 50% so that half of the search space locations may not contain any objects within them. Again, this appears to represent a reasonable middle case;
- Searcher Memory: will be set to 0.75 effective, meaning that 0.25 of the time they may mistakenly, on an individual basis, overlap with themselves. The Searcher Memory has been set at the mid–point of the upper–level (i.e. between 0.50 and 1.00) since even in a conceptual searching situation, the searcher should often be capable of remembering what aspects and areas or concepts they've already searched, covered, detected and considered. It seems reasonable, therefore, to set the Searcher Memory someway at the middle of the upper–level.

Additionally, it is worth reiterating from subsection 10.4.1 that the search space size will be set to a 500 location size space of 100 location types between 1..5 and 250 objects will be hidden within this space of 5 object types — so 50 of each. Furthermore, as mentioned in the modelling goals, each searcher will be allocated 100 resource units, which collectively is capable, in theory, of sufficient human resource effort allocation to achieve 100% coverage of the search space. Finally each predisposition and configuration is simulated 1000 times and the performance averaged.

10.5.1 Modelling Goal One

As mentioned in the modelling goals of section 10.2.1, the first goal of the simulation model is to perform a comparison between the available search strategies

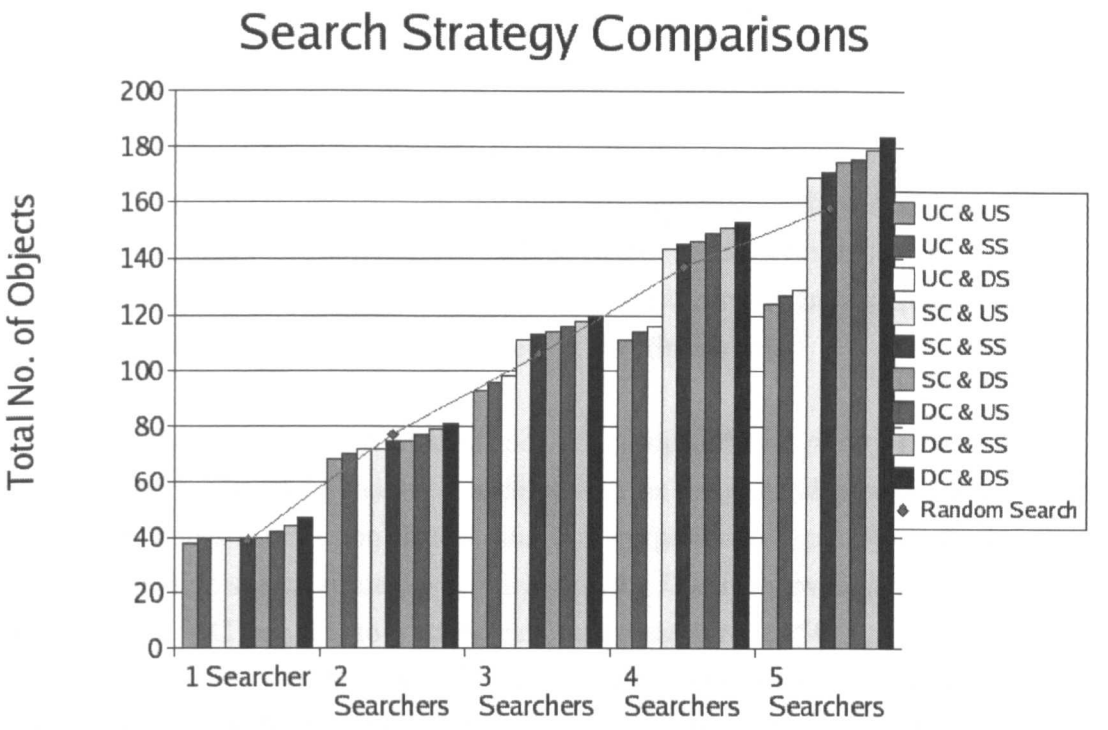


Figure 10.14: Comparisons of Search Strategies

possible. As was discussed in section 10.4, there are a number of possible diversity/uniformity predispositing variants on each of the three approaches to considering and promoting non–functional attributes. In subsection 10.5.1.1 below these various nine search strategies are compared. The collective searching performance of the searchers is continually aggregated so that the process loss involved in each strategy can be illuminated as each extra searcher, from 1 up to 5 adds an extra 100 resource units of effort. In subsection 10.5.1.2 a more specific subset of the nine strategies which capture the extremes available is illustrated and discussed to show what the specific performance upon detecting specific object types was.

10.5.1.1 Search Strategy Comparisons.

In this subsection the nine search strategies are compared. This is done by continually adding searcher effort under each particular uniformity/diversity coverage and sensitivity dimensions from 1 to 5 searchers.

Figure 10.14 visually illustrates the results of performing these simulations on each diversity/uniformity predisposition. While it clearly shows how all of the nine search strategies employed show diminishing detection effectiveness as more searchers are added, it also clearly indicates that the Diverse Coverage and Diverse Object Sensitivity search strategy progressively outperforms the other search strategies as more and more searchers are added to the search. This is because as more searchers are employed the diversity benefits start to result in less duplication and overlap on both the coverage and sensitivity dimensions.

Furthermore, when all these search strategies are compared to a purely random search (cf. Search Theory in chapter 9 section 9.3) the only search strategy that statistically produces a detection distribution that outperforms a purely random search from the nine search strategies simulated is the Diverse Coverage and Diverse Sensitivity (DC and DS) to a statistical confidence level of 5% achieving a Chi–Squared value of x at 9.84. The SC strategies (i.e. pessimistic, middle case, and optimistic) also performed well, although they did not produce a statistically significant superior detection performance than a random search. All the UC detection search strategies performed poorly by comparison, further indicating, that, uniform or homogeneous influences increase detection process loss, this is especially more pronounced as larger numbers of searchers are added to the search.

Finally, in terms of the modelling goal one comparisons, the simulation modelling analogies and corresponding simulation results begins to provide evidence that a detection strategy that diversely predisposes searchers/inspectors on both dimensions of coverage (i.e. *'where'* to search) and detection sensitivity (i.e *'how'* sensitive they are to relevant objects/faults/assumptions etc) provides increased utility and benefit from human redundancy and diversity — even with a random object distribution. By analogy, the finding, strengthens the position that the proposed goal diversity process intervention may well result in improved detection performance when applied to a detection type task.

10.5.1.2 Object Type Detection Performance

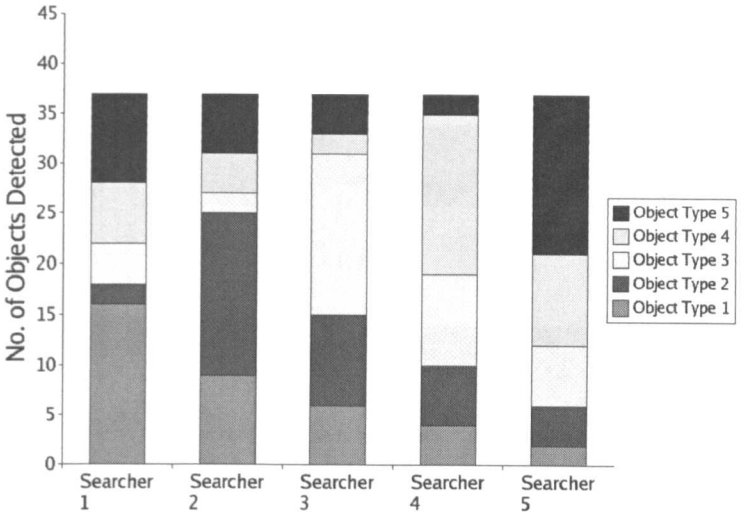


Figure 10.15: DC and DS Object Type Performance

Although, as subsection 10.5.1.1 indicated, that the DC & DS search strategy was the only search strategy that provided a statistically significant superior detection performance than a random search, it can be seen from figure 10.14, in the last subsection, that the other systematic predispositioning on both coverage and diversity also performed reasonably well — although it did not yield a statistically significant superior search when compared to a random search. In this subsection, the comparisons between the three different approaches to promoting non–functional attributes, analogically, will be performed by contrasting the object sensitivity performance of: a) the middle–case of the SC strategy, namely the SC and SS search strategy with the optimistic case of the DC and DS search strategy, to get a further impressionistic view of what aspects it may present in the wider context of the issues already raised in discussion chapter 8; and b) the pessimistic case of the UC strategy, namely UC and US with the optimistic case of the DC and DS search strategy to also discuss wider assumption detection issues raised in chapter 8.

In figure 10.15 the DC and DS strategy is illustrated. It can be seen from this figure that, in addition to producing the only statistically significant superior detection performance over a random search strategy, there is a wide variation in

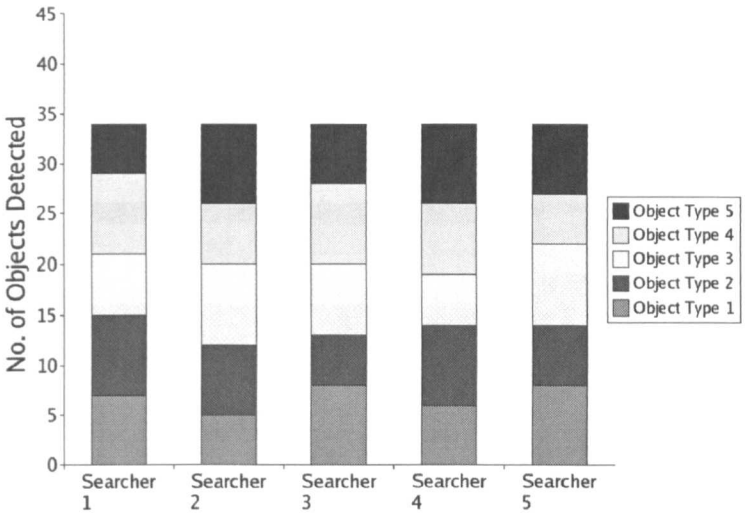


Figure 10.16: SC and SS Object Type Performance

terms of the detection sensitivity of certain object types. For example, Searcher 1 found a significantly larger number of Objects of type 1 than Objects of type 2. Correspondingly, Searcher 2 detected a significantly larger number of Objects of type 2 than Objects of type 3. This pattern pervades through the entire detection performance of all the searchers employed, whereby, each searcher was particularly prejudicial for detecting certain object types than others and between them they tended to be prejudicial to detecting more of different object types than the other searchers employed.

If we contrast this object type detection performance with that of the SC and SS detection search strategy in figure 10.16, it is possible to notice that, as mentioned earlier in subsection 10.5.1.1, although the overall detection performance of objects is quite good, by comparison to the DC and DS detection strategy in figure 10.15, the detection of object types, by the various searchers employed in this search strategy, is much less varying. In chapter 8 the overall different approaches to promoting non–functional attributes were considered. It was highlighted there that a particular concern of the systematic approach was that it places a potentially unrealistic cognitive burden upon a developer when considering multiple goal type promotion tasks. Furthermore, intuitively, within the stage 3 of the pro-

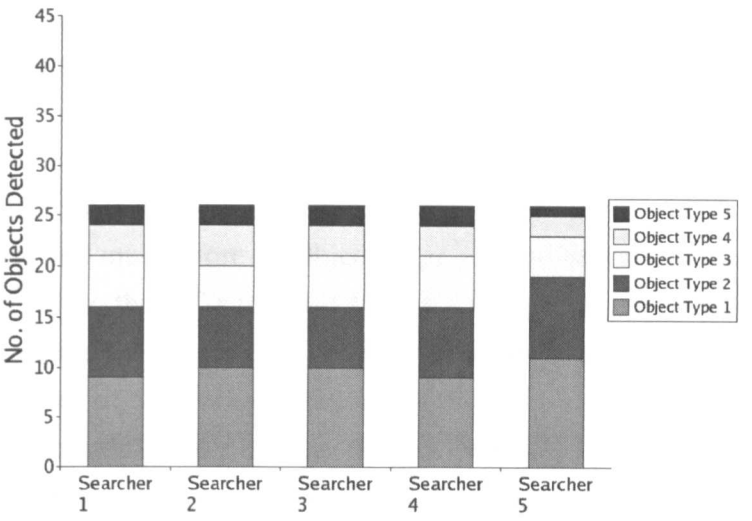


Figure 10.17: UC and US Object Type Performance

posed goal–diversity process intervention, it was envisaged that a potential benefit of performing goal promotions in this way, was that at the collaborative meeting stage, the human diversity generated in the previous two stages (i.e. individual analysis and individual inspection) is that it would predispose different developers to identify and detect different assumptions — which facilitates the necessary task climate of conflict at the collaborative meeting stage to further identify flawed assumptions, necessary trade–offs and possibly breakthrough decisions. It can be seen, in an intuitively impressionistic manner, that in contrasting the object type detection performance between the two strategies of DC and DS and SC and SS, although they both perform overall quite well, when we consider the level of variation of object type detections within and between the searchers employed, it is the DC and DS detection search strategy that promises to provide the necessary climate of conflict and challenge positively envisaged at the stage 3 of the goal–diversity process intervention — since the SC and SS detection strategy increases the likelihood that developers will derive less contrasting predispositions based upon a more homogeneous set of assumptions from the previous stage 2 (i.e. individual inspection).

The next comparison to make is between the DC and DS detection search strategy and the pessimistic case of the UC and US search strategy. The UC and US detection strategy is illustrated in figure 10.17. It can be seen, by contrast to the DC and DS detection strategy, that there is a strong homogenizing influence upon object type performance between the searchers employed. For example, searchers 1..5 all have detected much more of Object Type 1 than Object Type 5. Here we can see, by contrast to the DC and DS detection strategy, that, in terms of object type detection, while the UC and US detection strategy is also prejudicial in influencing what types of objects are more/less likely to be detected, they do so in a homogeneous manner. It can be remembered from chapter 6 and the discussion chapter 8, that assumptions can be shared. Such influences that can result in flawed shared assumptions are common culture, past experience, training, common context–of–interest, etc. In these situations implicit/explicit assumptions can often go unchallenged and can often act as reinforcing the installation and adoption of certain assumptions in an unquestioning manner. When considering stage 3 of the proposed goal–diversity process intervention (i.e. collaborative meeting) it can be appreciated that the situation presented by this UC and US search strategy is even more concerning than in the previous case of the SC and SS detection strategy, as such a heavily homogeneous predisposition of the same assumption types will not only be counter–productive in establishing the right task climate of conflict, but could easily act as reinforcing already indentified assumptions — since they will be largely shared and hence potentially reinforcing, instead of contrasting and conflicting.

In summarising this particular subsection, it can be appreciated, as already mentioned in chapter 6, that assumptions are inevitable reasoning and decision–making mechanisms in producing software artifacts. The point this section has attempted to illustrate, is not to argue that assumptions should be prevented from occurring, rather, it is the nature of the assumptions identified and how different and contrasting they are that can potentially aid their detection. In applying the proposed goal–diversity process intervention, it is not so much hoped that different developers don't make assumptions, as this is inevitable, what is hoped is that the diversity generated from predisposing different developers to promote different

non–functional goals results in sufficiently diverse and contrasting assumptions that this not only allows greater exploration and evaluation of those assumptions — in terms of their relevance and validity, but that it provides an appropriate task climate of conflict and challenge to identify further assumptions during stage 3 and help resolve trade–offs and stimulate breakthrough decision situations. From the analogous simulation strategy configurations compared, the superiority of the DC and DS detection strategy begins to indicate an impression that this level of contrast and conflict can be possible through diversely predisposing developers on dimensions of coverage and object detection sensitivity.

10.5.2 Modelling Goal Two

While it is clear from the simulation analysis in subsection 10.5.1 that: i) the most superior detection performance is the DC and DS search strategy; and ii) that this search strategy is the only one that is statistically significantly superior to a random search strategy, there is still a further analysis that can be carried out on this particular search strategy — in order to get a clearer indication of the diversity detection benefits of predisposing different individuals to different goals to achieve greater assumption–coverage. Such an analysis on the DC and DS search strategy needs to consider the direct relationships that exist between the number of searchers to the number of location types represented in this particular search strategy.

Table 10.10 on the following page provides the results of such a simulation analysis, where the number of searchers and number of search space location types are varied, in order to ascertain the diversity merits of ensuring that each individual searcher is predisposed to cover a particular search space location type that is different from all other searchers involved. This, therefore, implicitly assumes that there is an equal number of searchers to location types. However, in order to reinforce the diversity detection benefits of such a search strategy, it is insightful to consider the object detection performances when: i) more searchers (analogous to software inspectors) are involved in the search than location types (analogous to fault inspection goals); and ii) when there are more search space

Searchers/Locations	2 Loc.	3 Loc.	4 Loc.	5 Loc.	6 Loc.	7 Loc.	8 Loc.	9 Loc.	10 Loc.	Same No
2 Searchers	***39***	40	40	37	36	34	34	34	37	***39***
3 Searchers	49	***59***	59	58	56	54	55	55	55	***59***
4 Searchers	70	68	***80***	75	77	76	76	76	72	***80***
5 Searchers	88	87	91	***100***	97	96	96	95	86	***100***
6 Searchers	106	103	111	109	***117***	113	113	113	107	***117***
7 Searchers	117	116	119	123	127	***137***	133	131	119	***137***
8 Searchers	132	139	140	135	136	145	***153***	148	151	***153***
9 Searchers	148	148	151	159	155	159	167	***171***	167	***171***
10 Searchers	160	162	167	168	169	171	180	182	***184***	***184***

Table 10.10: Searcher and Location Comparison Results

Searcher/Loc. Type	Loc. 1	Loc. 2	Loc. 3	Loc 4.	Loc 5.
Searcher 1	0.75	0.0625	0.0625	0.0625	0.0625
Searcher 2	0.0625	0.75	0.0625	0.0625	0.0625
Searcher 3	0.0625	0.0625	0.75	0.0625	0.0625

Table 10.11: Under Representation Example

location types than searchers to predispose to them. As, was mentioned in section 10.2.1 concerning the goals of the simulation model, the second goal of the simulation model refers to two cases of: i) the *over representation of search/inspection goals in a search/detection effort*, where human resource effort exceeds the number of possible diversity predispositions possible; and ii) the *under representation of search/inspection goals in the search/detection effort*, where there are more predispositions possible than human resource effort available.

Each of the 81 individual simulations of the DC and DS search strategy were configured and simulated like those in subsection 10.5.1 — with only the number of searchers and search space location types differing.[20] In table 10.10 it can be seen that there are emboldened figures running diagonally across the table from top left to bottom right. These emboldened figures represent simulation configurations where there were an equal number of searchers to search space location type predispositions. Looking diagonally across the table, all of the other figures below the

[20]In subsection 10.5.1 simulations, an equal number of searchers and search space location types were assumed.

emboldened diagonal line represent simulation outputs from configurations that resulted from more searchers employed than goals (over representation), whilst all of the figures above the emboldened diagonal line of figures represent simulation outputs from configurations that resulted from more goals than searchers (i.e. under representation). The emboldened diagonal figures that represent equal representation between searchers and goals are also reproduced in the last (far right) column so that they allow an intuitive visual inspection, by the reader, for comparison to *under/over* goal representation simulation configurations when making comparisons horizontally back across the table.

It should be clear from the example shown in table 10.11 that the location types (analogous to goals) of 4 and 5 are under represented. By contrast, if there were only 2 search space location types — with 3 searchers employed, then such a predispositioning configuration would represent an over representation scenario, where one location type would have two searchers predispositioned to searching it at 0.75 predisposition. Both the under/over representation case figures, from table 10.10, are statistically analysed in subsections 10.5.2.1 and 10.5.2.2. This is done by comparing the *over/under* representation of search space location types to searchers to simulation configurations when there was an equal number of both involved in the search.

10.5.2.1 Under Representation of Goals

In this subsection the simulation configuration — when the goals were under represented is statistically analysed. This is a search situation when there were more location types than searchers to, individually, predispose them to. In order to ensure that we do not rely upon direct single comparisons of figures (i.e. comparing a single *equal* representation with *over/under* representation figures in table 10.10), groups of comparable *over/under* search simulation outputs to a group of *equal* simulation outputs are statistically analysed. If we were to look at table 10.10 we can see, for instance, that we can compare the under representation situation when there are (say) 10 search space location types involved and there are only 2 to 9 searchers to predispose them to — compared with the simulation outputs when

Goals to Inspectors	X^2 (Chi)	Hypothesis	Conf. Level	Conclusion
10 Goals & 2 - 9 Insp.	6.47 < 14.067	Accept H0	N/A	No Difference
9 Goals & 2 - 8 Insp.	1.93 < 12.592	Accept H0	N/A	No Difference
8 Goals & 2 - 7 Insp.	1.53 < 11.070	Accept H0	N/A	No Difference
7 Goals & 2 - 6 Insp.	1.56 < 9.488	Accept H0	N/A	No Difference
6 Goals & 2 - 5 Insp.	0.59 < 7.815	Accept H0	N/A	No Difference
5 Goals & 2 - 4 Insp.	0.43 < 5.991	Accept H0	N/A	No Difference
4 Goals & 2 - 3 Insp.	0.03 < 3.841	Accept H0	N/A	No Difference

Table 10.12: Under Representation of Goals Results

Inspectors to Goals	X^2 (Chi)	Hypothesis	Conf. Level	Conclusion
3 - 10 Insp. & 2 Goals	***17.45 > 14.067***	***Accept H3***	***@ 5%***	***Different***
4 - 10 Insp. & 3 Goals	***15.39 > 12.592***	***Accept H3***	***@ 5%***	***Different***
5 - 10 Insp. & 4 Goals	8.50 < 11.070	Accept H2	N/A	No Difference
6 - 10 Insp. & 5 Goals	6.33 < 9.488	Accept H2	N/A	No Difference
7 - 10 Insp & 6 Goals.	5.34 < 7.815	Accept H2	N/A	No Difference
8 - 10 Insp. & 7 Goals	2.18 < 5.991	Accept H2	N/A	No Difference
9 - 10 Insp. & 8 Goals	0.18 < 3.841	Accept H2	N/A	No Difference

Table 10.13: Over Representation of Goals Results

there are always an equal number of search space location types to searchers in the grouped simulation output figures between 2 to 9 searchers. The actual statistical analysis tables of how the statistical analysis was carried out, using Chi–Squared tests, are shown in full in appendix B, in section B.2 on page 385.

The statistical summary in table 10.12, of the Chi–Squared analysis, indicates that there is no statistical difference in the comparisons between equal and under representation of goals in a search effort under the configurations of the simulation model.

10.5.2.2 Over Representation of Goals

In this subsection, the search situation when locations/goals are over represented is analysed. This is a situation where there are more searchers than location types to predispose them uniquely to. Again, multiple groups of simulation outputs of the over representation search case are compared to the groups of search configu-

ration outputs when there are an equal number of searchers and location types (i.e. equal representation case). Again, the Chi–Squared statistical test is used and a complete set of statistical analysis tables is provided in appendix B, section B.3 on page 387 to show how this was carried out.

The statistical summary in table 10.13 on the page before, of the Chi–Squared analysis, reveals that when there are only a small number of search space location types and an increasing number of searchers employed, then the detection performance is undermined. In this case, it is possible to rationalise such a finding, by commenting that over representation of an inspection goal results in a homogeneous effect upon the inspectors involved — resulting in more and more process loss, in fault–detection terms, as more inspectors are added or employed in the inspection effort. It appears, from this, that predisposing inspectors to consider and pursue individual goals becomes more important as more inspectors are added, if detection effectiveness and efficiency is to be promoted and maintained.

10.6 Chapter Summary

This chapter has introduced a search simulation model as an analogous indicator of the potential human diversity benefits anticipated by the proposed Goal–Diversity process intervention progressed throughout this thesis and explicitly discussed earlier in chapter 8. The chapter began by establishing a design rationale for the search simulation model — drawing generally upon diversity and dependability aspects already discussed in this thesis and specifically using, analogously, the well established literature from both Search Theory and Software Inspections from chapter 9 to guide and inform the design. The design rationale was structured into areas of Modelling Goals, Modelling Scope, and Modelling Detail to provide a sound and coherent bases for justifying the search simulation model's eventual design. The Java implemented search simulation model was then introduced. This included considerations of: a) the verification and validation of the model in order to both check its dynamic behaviour operated as expected and, within the constraints of a PhD research, establish that the simulation model's eventual design and implementation captures the satisfactory real–world representative-

ness required to provide valuable insights into the benefits that may be expected from introducing human diversity on both the coverage and detection sensitivity dimensions in detecting uniformly distributed target objects. The simulation modelling process was then presented to illuminate the more concrete aspects of the implemented search model in terms of inputs, parameters, outputs and controlling logic. Next a justification for the configuration of the envisaged simulation experiments was presented. In this regard, categorising diversity upon both the coverage and detection sensitivity dimensions allows nine potential search strategies to be configured. These allow, analogously, a pessimistic, middle–case, and optimistic categorisation of the three identified ways of promoting non–functional attributes from chapter 8. Each of these were discussed along with diversity distribution and profile tables to illuminate a reasonable configuration for conducting the eventual simulations. Additionally, integrated real–world parameters were also studied for their impact upon the nine search strategies using a sensitivity analysis reported in the form of both their absolute and relative variance. It was found that whilst two of the parameters resulted in wide ranges of variance — in the absolute sense, their relative variance meant that the individual search strategies would be comparably affected. The other parameter, although showing differences of relative variance, varied little between the search strategies in any absolute sense. Finally, the two simulation experiments were performed to fulfil the search simulation modelling goals established earlier in the chapter. Simulation experiment one was to satisfy the human diversity assumption detection comparisons between competing ways in which non–functional attributes can be considered. Of the nine search strategies, the one most analogous to the envisaged process intervention of Goal–Diversity proved to be the only search strategy that produced a statistically significant superior detection performance than a random search. When a pertinent subset of these nine search strategies were further analysed upon the detection sensitivity dimension, it was found that this search strategy also results in significantly varying detection of particular object types which bodes well, in an intuitive way, to providing the right challenging task climate of conflict for helping to identify and detect additional assumptions in the last stage of the envisaged Goal–Diversity process intervention. Simulation experiment two was to establish the influence of over/under goal representation issues in adopting such a process

intervention. It was found that, although not statistically significant, under representation did not unduly influence search detection performance when important goals are under represented. However, at higher levels of over representation, a statistically significant inferior detection performance is found, indicating that human diversity is useful for detection type tasks up to the number of beneficial discriminating predispositions possible, but beyond that, adding extra human resources produces an undesirable homogenous influence.

Chapter 11

Conclusions

11.1 Summary of Work

This thesis has focused upon how computer–based system dependability can be promoted. The existing focus of dependability has primarily aimed its attention towards the inclusion of computational and structural redundancy to increase toleration of residual software faults during operational execution to prevent software service failures or ensure such service failures occur in a graceful and controlled degradation of delivered service. Due to the conceptual nature of software, achieving such fault tolerance often mandates the introduction of diversity interventions during the creation process to reduce the possibility of co–incidental or common–mode failures. To aid this purpose the dependability community has progressed a conceptual framework to guide practitioners in the field. The framework provides guidance on the important attributes upon which operators and users will base their judgements of trust and confidence in the system. The threats to achieving such dependability attributes are also included in the framework — along with the means by which attributes can be attained.

Although the primary focus of the dependability community on improving fault–toleration of the created software artifact has resulted in significant increases in software dependability, the ongoing ubiquity and pervasion of information processing technology into everyday life places an increased demand for further software dependability improvements. It is suggested, in this thesis, that two areas where further dependability improvements may occur concern: i) improving the dependability of the software creation process through the means of increased fault avoidance; and ii) providing a more holistic computer–based system perspective of computer systems and recognising that technically dependable computer systems can be judged to be undependable for non–technical reasons.

With regards to improving the dependability of the software creation process, through the means of increased fault avoidance, it has been suggested, in this thesis, that a similar conceptual framework perspective to the existing dependability framework should be applied that emphasises the desired attributes of a mature dependable process — along with the threats and means to achieving it. Unlike

some existing maturity models, that possess a hierarchical perspective, this thesis advocates the need for an integrated understanding of the process dynamics if the process is to become more mature and predictable. The need for a more dependable process also raises the issue of what role redundancy and diversity would play in a dependable creation process to achieve the required process attributes. In this respect, adopting a wider computer–based system viewpoint, it is possible to see the creation process as a valid system–of–interest. When viewed in this way, like introducing redundant and diverse structure and function into the artifact, the role of process redundancy (i.e. redundant/diverse process technology and human resources) also has the purpose of increasing the fault–tolerant nature of the creation process via strengthening the dependability means of fault prevention, fault detection, fault removal and fault forecasting. With human resources, there is already a number of examples of how human redundancy and human diversity has been employed to increase fault–avoidance (e.g. open–source development, egoless programming, pair programming). However, most of these make the assumption that additional human resources are additive (i.e. natural diversity), while some less well known approaches (in the software engineering and dependability communities) to achieving human diversity through human redundancy do not make such an assumption and directly intervene in the process to improve human diversity (i.e. cognitive diversity and heterogeneous groups/teams). With process technology, there are also examples of how diversity can be achieved through the application and redundant employment of existing tools, methods, and techniques to improve fault–avoidance in the creation process. Again, a major issue of concern involves the additivity of these process technologies to achieve effective diversity application. It has been suggested in this thesis, that two measures of establishing the utility of process redundancy and diversity involves both a productivity effect (i.e. quantitative assessment of number of faults prevented, detected, or removed) and a quality effect (i.e. the qualitative assessment of the impact, importance, or consequence of faults prevented, detected, or removed).

With regards to improving the dependability through a more holistic computer–based system perspective of computer systems, this thesis has introduced a number of generic contexts–of–interest that illuminate and provide a more holistic

perspective during the creation process. Such an approach views computer–based system dependability as a super ordinate system goal dependent upon the interpretations of purpose attributed to the computer system. It has been discussed that there are a number of potentially conflicting stakeholder interests during the creation, operation, and evolution of a computer system. Each stakeholder interest group can be compared to a context of interest that brings its own interpretations of such things as the purpose, motivations, and responsibilities involved in the creation and operation of a computer system. Failure to recognise these contexts–of–interest has, in the worst cases, resulted in technically dependable computer systems being judged to be undependable for non–technical reasons.

Widening out the boundaries of such a system view allows a greater holistic perspective of the primary and latent purpose ascriptions often made about real–world complex computer system applications which are essential to unearthing views of dependable and undependable judgements. A case study of the longstanding Automatic Teller Machine domain is subsequently presented to reinforce the value of such a computer–based system perspective. This case study in broken down into generic computer–based system contexts of: a) The utility context; b) The engineering context; c) The deployment context; and d) The evolution context. The value of such an holistic perspective is demonstrated in the case–study to show how vulnerabilities and faults often result through a separation of concerns or an over emphasis of one context of interest which is oblivious to the needs and requirements of another context in which the failure of the vulnerability or fault often propagates.

It was stressed that a particular requirement or responsibility of a dependable process view was to provide a greater understanding of the fault–phenomenology. This thesis has recognised that a major cause of many faults and vulnerabilities in computer systems results from assumptions made during the creation process. Assumptions are always prevalent in any non–routine decision–making situation, and are often necessary if decision–making, communication, and understanding is to be achieved in a decisive manner. However, the extreme levels of novelty and the highly conceptual nature of software development makes flawed assumptions,

that can result in faults due to incompleteness, inconsistencies, and incorrectness, particularly critical in the software creation process. Assumptions can be consciously or unconsciously made. Flawed conscious assumptions are often the result of some ambiguity during communication or due to some lack of knowledge about something. Flawed unconscious assumptions often result from some hidden bias, belief, or value system of an individual which can result from past experience or conditioning that allow a person to make unquestioned associations about some aspect of interest. From informal pragmatics and problem–solving literature, unconscious assumptions are often referred to as *'artificial limitations'* or constraints placed upon a particular thinking episode. It has been argued that only through divergent or lateral thinking that creates conflict and dissonance that challenges such biases, beliefs, and values, can such unconscious limitations be detected. Another area of assumptions concerns situations where people are likely to make the same flawed assumptions unconsciously. This can occur where people are collectively conditioned in some way by past experience, training, education, or culture. However, it can also occur when severe knowledge acquisition constraints are placed upon groups of people (i.e. a kind of group–think dynamic). Finally, even when assumptions are valid their validity may only be of a temporal nature as valid assumptions can be invalidated over time as the system or its surrounding environment evolves and changes.

It was emphasised earlier that an holistic computer–based system perspective that integrates and unearths the different purpose ascriptions is important to establishing the dependability and undependability judgements various stakeholder contexts make during computer system development. This is a purpose representation issue. Therefore, illuminating and making explicit the many goals and objectives is important to promoting dependability in two fundamental ways. Firstly, in terms of the created artifact, it is necessary to ensure that all the dependability attributes are adequately represented (i.e. security, availability, reliability, etc), as only when these are suitably represented will it be possible to establish the particular priorities and conflicts that the criticality of the system and the application domain requires. Secondly, in terms of the creation process, it is also necessary to ensure that different computer–based system contexts–of–interest are also ade-

quately represented to ensure that different purpose ascriptions, in the form of differing motivations, values, responsibilities, are unearthed to ensure a sufficiently holistic perspective. It can be seen, therefore, that the teleological aspect, in terms of desired artifact goals, and the wider creation process perspectives plays an important representation role in computer system dependability. From a philosophical perspective, computer system development is a purpose driven teleological activity. However, adequate teleological representation relies firstly upon an adequate formal causation, as a *blueprint*, in the mind(s) of the creators of an artifact if it is to be realised as a final causation factor or goal. This has been empirically established by industrial psychologists and software engineering researchers who found that explicit goal–setting results in both behavioural and cognitive effects that emphasise what is important and what is not important to achieving the goal. However, equally, psychologists have also established that complex multiple goal–setting tasks introduces cognitive limitations whereby only one goal can be maximally activated at any one time. It can be seen therefore, that acquiring adequate representation of desired dependability attributes (i.e. goals) therefore mandates some amount of human redundancy or effort. This will either require a single individual to repetitively consider development pursuing different goal perspectives, or require multiple people to simultaneously pursue development from different goals. It can be argued that the latter option is more advantageous as, has already been discussed earlier, a single individual is likely to make the same underlying unconscious assumptions — resulting in common assumptions pervading through the different goal perspective attempts. Whereas, using different individuals pursuing different goal perspectives allows the introduction of some initial natural diversity — where different people bring a naturally different set of underlying unconscious assumptions. As we have discussed earlier, detecting assumptions relies upon conflict and dissonant reasoning to challenge any preconceptions. At a later collective meeting phase, it is therefore, very likely that such underlying assumptions each person has will be unearthed and identified. However, such an approach also offers a more flexible way to achieve diversity as goal-setting naturally influences human behaviour and cognition — resulting in different individuals valuing, prioritising, and generating different decision–making criteria. This is a much more controllable way to achieve a form of focused diversity than trying

to compose a development team on some uncontrollable diversity dimension, of say, personality types, culture, etc, as in the human resourcing, demographic, and economic restrictions of a normal development organisation such development team/group compositions may not be practically feasible. Nevertheless, because human goal–setting directly affects behavioural and cognitive performance, it will directly result in influencing different developers — predisposed to promote a certain dependability goal to generate (knowingly and unknowingly) different sets of assumptions. Therefore, at a later meeting phase, a rich foundation for assumption conflict and detection should result. One drawback, is that, as this thesis has discussed, assumptions can be made in a collective shared manner — due to some homogeneous preconditioning influence of shared past experience, education, training, or culture, etc. Furthermore, as discussed in this thesis, context of meaning also results in people from the same context also sharing deeply held assumptions. It is for this reason that people from different contexts of the utility domain, the engineering domain, the deployment domain, and the evolution domain, also need to be predisposed to consider the development from differing goal–aspects — in order to help detect shared assumptions.

The discussion chapter uses a well established goal–oriented approach to requirements engineering to bring together the relationships between: i) a computer–based system view; ii) assumption detection; and iii) diverse goal–setting, to suggest how setting diverse goals and diverse computer–based system contexts–of–interest can help provide a rich holistic view of dependability and detect hidden flawed assumptions. It should be noted, that the flawedness of the assumptions may only be appreciated from a holistic computer–based system view of dependability, and may seem quite reasonable and justified from any particular individual or computer–based system context. Therefore, the discussion chapter identifies two types of assumptions that can be detected through the illustration of diverse goals and different computer–based system contexts–of–interest: Firstly, intra–context assumptions — where the application of diverse goals, within a single context, result in identifying a fault or vulnerability within the same context–of–interest; Secondly, inter–context assumptions — where the application of diverse goals and diverse computer–based system contexts–of–interest, in different con-

texts, result in identifying a hidden assumption which appears reasonable (and potentially shared) within a single context, but the vulnerability or fault, it raises, results in conflict and subsequent detection between two (or more) contexts–of–interest.

To achieve a greater understanding of the potential dynamics of diverse goal–setting, for unearthing assumptions that could result in vulnerabilities and faults, a simulation model of the essential dynamics was constructed. A simulation model was developed for two reasons. First, it was realised that to carry out a suitably large–scale empirical study was beyond even the feasibility of a large research programme — let alone the limited resources available for a Ph.D thesis. Secondly, a simulation model is ideal for exploring and manipulating different parameters and variables to allow greater understanding of the dynamics at work.

Two complementary and established areas guided the initial design of the simulation model. Firstly, since the focus of diverse goal–setting is essentially to improve the dependability of the software creation process through increased fault–avoidance, the literature on software inspections was selected as an appropriate influential area of fault detection. Furthermore, the literature on software inspections is vast and longstanding. Secondly, since fault–avoidance can be compared directly to a searching type task, the operational research area of search theory was also used as an influence upon the simulation model's design. Search theory is a very well established area with terminology, concepts, and structure that improved understanding of what essential factors the simulation model's design should be included.

From the simulation modelling chapter 10 it can be appreciated that detection and searching tasks can be difficult upon two dimensions: i) detection and searching, whether in a physical or conceptual sense, can be hampered or facilitated by effective and efficient coverage, in terms of *where* they search; and ii) even when effective and efficient coverage is achieved, effective and efficient detection of both physical and/or conceptual entities also depends upon *how* sensitive the searchers/inspectors are to those physical or conceptual entities. When search-

ing/detection is considered on these two dimensions it is possible to see that human redundancy and diversity can be characterised to provide nine interesting categories of how uniform, systematic, and diverse searching or detection can be resourced and deployed on a searching or detection type task. Using these categories, the various existing approaches to promoting non–functional attributes can be usefully approximated and configured, within the simulation model, by utilising coverage distributions and detection sensitivity capability profiles that can capture the potentially uniform predispositioning of the Ad-Hoc approach; the systematic multiple consideration of non–functional attributes of the Systematic approach; and the diverse human predispositioning of the proposed Goal–Diversity approach advocated in this thesis.

Usefully approximating these various approaches, within the simulation model, therefore allows an important modelling efficacy test of how effective and efficient the proposed Goal–Diversity process intervention is, by comparison to the two other approaches, in terms of demonstrating the potential process dependability benefits through improved assumption–detection proposed by this thesis. Therefore the first simulation modelling goal was to provide a detection comparison of the human diversity/uniformity benefits of the proposed Goal–Diversity approach with existing approaches (i.e. Ad–Hoc and Systematic) under an equivalent human resourcing constraint. The simulation model was configured and the outputs statistically evaluated for their detection performance. The simulation findings revealed that the analogous diversity configuration of the proposed Goal–Diversity approach was the only search configuration that produced a statistically significant detection distribution when utilising multiple search resources under a constant overall search effort constraint. The simulation findings, under suitable real–world modelling approximations, suggest that there are potential process dependability benefits to be gained via increased assumption/fault detection means that utilises human diversity via important non–functional goal predispositioning. The next simulation modelling goal was therefore to help establish, that, given there is a limitation to the degree of human diversity goal predispositioning that is possible, what would be the detection effects of under/over representation of such non–functional attributes. The simulation model was therefore configured to

compare detections when there was: a) more diverse predispositions that human searchers (i.e. an under representation situation); and b) fewer diverse predispositions than human searchers (i.e. over representation situation). Both these search simulation configurations were compared to the (ideal) case where there are an equal number of diversity predispositions as searchers diversely predisposed. The search simulations were performed and the outputs from each of the three configurations statistically evaluated. The findings suggest that if human redundancy and diversity performance, on a detection type task, is to be maximised, then every human resource added to the task should be appropriately predisposed to a different goal predispositioning — since over representation of a single goal results in a homogeneous detection effect that reduces the additivity of the human resources employed and hence increases process loss.

11.2 Limitations of Work

One of the main limitations of the thesis is that the value of setting diverse development goals could not be empirically tested. Whilst, as already stated above, this was judged to be infeasible — to conduct a sufficiently large–scale experiment, it must be explicitly stated as a limitation of the thesis, as it results in only a theoretical justification for the dependability benefits of improving process dependability through fault–avoidance utilising the promotion of diverse development goals.

Another limitation of the thesis is that, although a search simulation model provides some justification for the theory, the simulation is incapable of capturing the assumption detection capabilities of a resolution meeting stage where conflicting goal relations, both within and across, computer–based context–of–interest, are exploited to detect potentially harmful assumptions.

Finally, as chapter 10, subsection 10.3.1 on verification and validation of the search simulation model indicated, the simulation model is designed to study coverage and sensitivity diversity upon a uniform distribution of objects. As stated there, there is no empirical evidence or research that indicates, unlike fault distributions, that assumptions occur or are caused in an unevenly distributed fashion.

This highlights a particularly difficult aspect of the thesis, in that there is little definitive research and literature on the nature of assumptions and their pathology. The literature that does exist is largely anecdotal and sparse and results in an inability to establish a sound set of theoretical arguments for justifying how, in particular with this thesis, diverse goal–setting, and the inevitable goal–conflict relations these create, can help create the required conflict and dissonance required to detect flawed assumptions. Nevertheless, the finding that diverse coverage and detection sensitivity yields potentially superior detection performances with multiple searchers under a constant search effort constraint is still a useful and encouraging finding in connection with the potential process dependability increases that may be afforded via the introduction of such a human diversity process intervention.

11.3 Future Work

As indicated, in the Limitations of Work in section 11.2 above, a more robust finding in favour of a Goal–Diversity process intervention to increase process dependability via improved assumption/fault detection, would be if it could be replicably produced in a real–world situation. In this section of considering future work, then, some broad suggestions on how this process intervention could be experimentally researched is provided — along with some envisaged problems that may be faced.

In order to structure this appropriately, it is necessary to briefly recapture both the envisaged stages of the Goal–Diversity process intervention advocated, along with the hypothesised benefits that are expected. These can be discussed as follows:-

- **Stage 1:** ***Individual Analyses*:** because non-functional goals introduce a subjective and relative teleological reasoning dimension (see chapters 7 and 8) it is hypothesised that at the Individual Analysis Stage 1, predisposing individual developers to promote distinct non–functional attribute goals will both: decrease their probability of making harmful assumptions that compromise their promoted non–functional goal; and increase their probability

in making harmful assumptions that compromise other non–functional attribute goals promoted by their development colleagues;

- **Stage 2: *Individual Inspection***: Again, due to the inherently relative and subjective purpose influence of non–functional attributes, it is hypothesised that, during Stage 2 of Individual Inspection of other colleagues non–functional goal promotions, a given developer promoting a given non–functional attribute will be more probable in detecting other developers harmful assumptions that compromise their promoted non–functional attribute goal;

- **Stage 3: *Collaborative Meeting***: due to the interactive nature of non–functional attributes, it is hypothesised that the variety of different assumptions made, along with their consequent interactive positive/negative impacts upon competing important dependability attributes will create the appropriate task climate of conflict to: a) detect further potentially harmful assumptions; b) facilitate appropriate dependability trade–off negotiations; and c) stimulate creative breakthrough design decisions that reconcile such interactive non–functional conflicts.

From this breakdown of the proposed Goal–Diversity process intervention — along with their respective hypothesis at each stage, a number of potential measures of performance can be determined. These are: a) at *Stage 1*, a measure of the variety of assumptions made between the development team;[1] b) at Stage 2, a measure of detection sensitivity of each particular developer's capability in detecting other developers' assumptions made; and c) the number of new assumptions detected and the number of breakthrough decisions adopted.

Having considered the stages involved and the potential measures of efficacy at each stage experimental research could be initially suggested that compares the assumption detection efficacy of a Goal–Diversity process intervention with the other two approaches of considering non–functional attributes. As discussed in chapter 8 and chapter 10 there are three potential approaches. These approaches,

[1]The reader should note that this was also a finding of the search simulation model in chapter 10 subsection 10.5.1.2.

along with their benefits and drawbacks in promoting process dependability can be briefly reviewed as follows:-

- **Ad-Hoc Approach:** involves no process intervention to directly promoting dependability attributes and thus relies, largely, upon the professionalism, knowledge, and experience of the developers concerned;
- **Systematic Approach**: involves the explicit consideration of multiple dependability attributes by a single developer. As discussed in chapters 7 and 8 this ignores the cognitive overload complications of simultaneously promoting multiple attributes in multiple goal type tasks;
- **Goal–Diversity:** involves an explicit process intervention to achieve human diversity for important coverage of dependability attributes in the activity of analysis and design of computer–based systems. It recognises and overcomes the potential cognitive overload drawbacks associated with the Systematic approach by ensuring that each developer considers a single non–functional attribute during analysis, design and inspection stages.

From the literature sources, arguments formed and simulation findings contained in this thesis, it can be suggested that the following hypothesis can be initially formed for each approach as follows: a) with the Ad-Hoc approach, there is a greater chance that due to homogeneous influences of the system type, the application domain, and other individual and group uniformity influences (e.g. common culture, education, training, etc) there will be a much greater potential for harmful unconscious and shared assumptions to be undetected during development – resulting not only in a lower number of assumptions being detected, but also a lower variety; b) with the Systematic approach, due to the cognitive limitations imposed with this type of task, a much shallower consideration of assumptions is expected. Therefore, while there may be more variety of assumptions considered — in terms of their harmful consequences on certain non–functional attributes, it is expected that a reduced number will be detected; and c) with the Goal–Diversity approach, as mentioned earlier, it is expected, that, due to the increased efficacy of human diversity introduced, there will be both a greater variety of assumptions made and detected.

Having considered these approaches, along with expected outcomes, a possible experimental approach that could be considered would be a well established domain (say something like the ATM domain in chapter 5) that could be professionally analysed to illuminate a significant number of assumption related faults that have occurred over time and these could then be suitably categorised into which dependability attributes they tend to compromise. Using three groups of four participants this development task could be set so that each four member group could be advised to perform an analyses under the follow conditions: a) Group 1 could be the Ad-Hoc approach and told to perform the initial analysis/design and consider what they individually believe are important quality attributes to be promoted in the course of the task; b) Group 2 could be the Systematic approach and told to perform the initial analysis/design paying explicit attention to the non–functional attributes of (say) Security, Reliability, Maintainability, and Safety during the course of the analysis/design; and c) Group 3 could be the Goal–Diversity approach where each individual of the four group team is told to promote a different non–functional attribute in the course of of the analysis/design. For instance Participant 1 promotes Security, Participant 2 promotes Reliability, Participant 3 promotes Maintainability, and Participant 4 promotes Safety. It is important that each group performs this stage for the same length of time in total. For instance, (say) that four man hours is allowed for each group. Following this stage of the experiment, each analysis/design contribution could be professionally assessed to see what assumptions have been made and these could be carefully categorised into what dependability attributes they potentially compromise. From this, the performance of this first stage could be evaluated in terms of: a) number of potentially harmful assumptions made; b) the dispersion of assumption types over the four non–functional attributes involved to ascertain the variety of different assumptions made.

Next, the same groups individually inspect each others analysis/design contributions to evaluate them. Group 1 (Ad-Hoc) could be informed to highlight any analyis/design decisions made that could compromise the quality of the system. Group 2 (Systematic) could be informed to highlight any analysis/design decisions that compromise the non–functional attributes of Security, Reliability, Maintain-

ability, or Safety. Group 3 (Goal–Diversity) could be individually informed each to highlight any analysis/design decision they believe compromises their particular goal they are promoting (i.e. Participant 1 would highlight any analysis/design decision made by the other 3 participant within his/her group that he/she believed compromised Security). Again, it is important that all groups perform the inspection stage for the same period of time for all three groups. At the end of this stage the number and type of assumptions detected, by any one participant, along with the total and types for each groups will provide some indication of the assumption detection efficacy of the three approaches on both an individual and group performance level.

Finally, each group performs a collaborative meeting review where each participant in each group reviews and evaluates potential conflicts found from the inspection stage. So Group 1 (Ad-Hoc) are told to collaboratively evaluate any quality conflicts, agree trade–offs and breakthrough decisions to provide an overall analysis/design solution for the system being developed. Group 2 (Systematic) are told to collaboratively evaluate any Security, Reliability, Maintainability and Safety conflicts, agree trade–offs and breakthrough decisions to provide an overall analysis/design solution for the system being developed. Group 3 (Goal–Diversity) are also told to collaboratively evaluate any Security, Reliability, Maintainability and Safety conflicts, agree trade–offs and breakthrough decisions to provide an overall analysis/design solution for the system being developed. Again, a constant period of time is allocated for all three groups of participants. At the end of this third stage the number of new assumptions and breakthrough decisions offer an insight into the efficacy of the three approaches for the collaborative stage.

There are also a number of problems involved in performing such an experiment. The application domain and system type chosen for the development task may be, naturally, biased to be more relevant to the promotion of some non–functional attributes than others. In this case, it is important to choose a domain and system type that places a reasonably equal importance on a number of dependability attributes and frame the original requirements documents/specification for the task in such a way that does not imply that certain dependability attributes are more

important than others. Human performance variance is always a variable that can corrupt or produce misleading findings in such an experiment. This, as chapter 3 highlighted, is a particular concern in a software development type task. It may therefore be necessary to be very careful in selecting the participants and a pretest of skill, capability, and experience may be necessary so that participants can be assigned to groups to minimise human performance variability. Categorising assumptions and judging what constitutes as an assumption will be an inherently subjective influence. The best that can be offered in this regard, is to ensure that a thorough understanding of the domain is achieved so that assumption classifications can be as coherently relevant as possible. Finally, in studies of group behaviour, especially when diversity has been deliberately composed into the group conformation, it has been noticed that conflict can occur in two ways. Task conflict is largely seen as being desirable in increasing group/team performance — since it increases exploration and coverage of relevant task criterion. However, conflict can also be of a social inter–personal nature whereby group/team members begin to take offence of others conflicting remarks. Since, with Goal–Diversity, it is hoped that at Stage 3 of a collaborative meeting, the variety of assumptions identified stimulates a task climate of conflict, it may be necessary to have a group facilitator to ensure that conflict remains related to task issues and does not degenerate down into inter—personal conflict between group/team members. However, having stated this, from a purely research perspective, the finding that, for instance, the Goal–Diversity approach results in undesirably high levels of social conflict that undermines group performance is also an interesting finding.

Bibliography

[1] Laprie, J.C. Dependability: Basic concepts and terminology - in English, French, German, Italian, and Japanese. *Dependable Computing and Fault Tolerance. Springer-Verlag, Vienna, Austria.*, 1992.

[2] Laprie, J. C. "Dependable Computing: Concepts, Limits, Challanges,". *IEEE International Symposium on Fault-Tolerant Computing - Special Issue. Pasadena, California. USA*, pages 42–54, 1995.

[3] Avizienis, A., Laprie, J.C & Randell, B. Fundamental Concepts of Dependability. *Newcastle University Technical Report, Computer Science Dept, CS-TR-739*, 2002.

[4] Randell, B. Facing Up to Faults. *Turing Memorial Lecture. Dept of Computer Science. University of Newcastle upon Tyne. January 31st*, 2000.

[5] Heylighen, F & Joslyn, C. Cybernetics and Second–Order Cybernetics. *: in : Meyers, R. A. (ed), Encyclopedia of Physical Science and Technology (3rd edition). Academic Press. New York.*, 2001.

[6] Prata, S. C++ Primer Plus 2nd Edition. *The Waite Group Press. Corte Madera. CA.*, 1995.

[7] Moulding, M. R. Designing for high integrity: the software fault-tolerant approach. *: in: Sennett, C. High Integrity Software. Pitman Publishing. London. UK.*, pages 39–68, 1989.

[8] Cristian, F. Exception Handling and Tolerance of Software Faults. *: in : Lyu, M.R. (ed). Software Fault Tolerance. Chichester. Wiley.*, pages 88–107, 1995. See Chapter 4.

[9] Winder, R. & Roberts, G. Developing JAVA Software. *2nd Edition. Wiley. Chichester West Susses. UK.*, 2001.

[10] Curtis, B., Krasner, H & Iscoe, N. A Field Study of The Software Design Process for Large Systems. *Communications of the ACM*, 31(11):1268–1287, 1988.

[11] Senge, P. M. The Fifth Discipline: The Art and Practice of The Learning Organisation. *Bantam Doubleday Dell Publishing. Century Business. London. Great Britain.*, 1993.

[12] Bell, D. Software Engineering: A Programming Approach. *3rd Edition. Addison-Wesley. Harlow, London. UK*, 2000.

[13] Randell, B. System Structure for Software Fault Tolerance. *IEEE Transactions on Software Engineering.*, SE-1(2):220–232, June 1975.

[14] Torres–Pomales, W. Software Fault Tolerance: A Tutorial. *NASA Technical Report NASA/TM-2000-210616. Langley Research Center, Hampton, Virginia. USA.*, October 2000.

[15] Lewin, D., & Noaks, D. Theory and Design of Digital Computer Systems. *2nd Edition. Chapman and Hall. London. UK*, 1992.

[16] Littlewood, B., Popov, P.T., Strigini, L & Shryane, N. Modelling the Effects of Combining Diverse Software Fault Detection Techniques. *IEEE Transactions on Software Engineering*, 26(12):1157–1167, December 2000.

[17] Littlewood, B., Popov, P & Strigini, L. Design Diversity: An Update from Research on Reliability Modelling. *Proceedings of the 9th Safety-Critical Systems Symposium, Bristol.*, 2001.

[18] Hatton, L. N–Version Design Versus One Good Version. *IEEE Software*, pages 71–76, November/December 1997.

[19] Knight, J. C. A Large Scale Experiment in N-Version Programming. *The 15th Abbual International Conference on Fault Tolerant Computing (FTCS)*, pages 135–139, June 1985.

[20] Knight, J.C., & Leveson, N.G. An Experimental Evaluation of the Assumption of Independence in Multiversion Programming. *IEEE Transactions on Software Engineering*, 12(1):96–109, January 1986.

[21] Eckhardt, D. R., Caglayan, A. K., Knight, J. C., Lee, L. D., McAllister, D. F., Vouk, M. A & Kelly, J. P. J. An Experiemental Evaluation of Software Redundancy as a Strategy For Improving Reliability. *IEEE Transactions on Software Engineering.*, 17(7):692–702, July 1991.

[22] Kanoun, K. Real–World Design Diversity: A Case Study on Cost. *IEEE Software*, pages 29–33, July/August 2001.

[23] Anderson, T & Lee, P. A. Fault Tolerance: Principles and Practice. *Prentice Hall*, 1981.

[24] French, C. S. Data Processing and Information Technology. *DP Publications. Guernsey, Channel Islands. Great Britain.*, 1991. See Sections 10.17 and Appendix 3.1.

[25] Kernighan, B. W & Pike, R. The Practice of Programming: Simplicity, Clarity, Generality. *Addison Wesley Longman Inc, Reading, Massachusetts. USA*, 1999. see chapter 1: Style pp. 1–27.

[26] Parnas, D. L. On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM.*, 15(12):1053=1058.

[27] Pressman, R.S. Software Engineering: A practitioner's approach. *4th Edition. European Adaptation by Darrel Ince. McGraw Hill. USA*, 1997.

[28] Sommerville, I. Software Engineering. *6th Edition. Addison-Wesley. Essex. UK*, 2001.

[29] Riel, A. J. Object-Oriented Design Heuristics. *Addison-Wesley. Reading. Massachusetts. USA*, 1996.

[30] Littlewood, B & Strigini, L. Redundancy and Diversity in Security. *DOTS/DIRC Technical Report. Centre for Software Reliability, City University. Lovdon.*, 2003.

[31] Leveson, N & Turner, C. S. An Investigation of the Therac–25 Accidents. *IEEE Computer.*, 26(7):18–41, July 1993.

[32] Hoffman, D. M & Weiss, D. M. Software Fundamentals: Collected Papers by David L. Parnas. *Addison Wesley. New Jersey. USA.*, 2001.

[33] Brooks, F.P. No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer*, pages 10–19, April 1987.

[34] Brooks, F.P. The mythical man month: Essays on software engineering. *25 year anniversary edition. Addison-Wesley. NY. USA.*, 1995.

[35] Jackson, M. Software Requirements and Specifications: a lexicon of practice, principles, and prejudices. *Addison Wesley. London. Great Britain.*, 1995.

[36] Kerzner, H. Project Management: A Systems Approach to Planning, Scheduling, and Controlling. *John Wiley and Sons, Inc. Canada.*, 1998.

[37] Constantine., L.L. The Peopleware Papers: Notes on the human side of software. *Prentice Hall. New Jersey. USA.*, 2001.

[38] Demarco, T., & Lister, T. Peopleware: Productive projects and teams. *2nd Edition. Dorset House Publishing. New York. USA*, 1999.

[39] Baker, T. T. Chief Programmer Team Management of Production Programming. *IBM Systems Journal.*, 1:56–73, 1972.

[40] Baker, F. T & Mills, H. D. Chief Programmer Teams. *Datamation*, pages 58–61, December 1973.

[41] Steiner, I.D. Group Process and Productivity. *Academic Press. New York. USA.*, 1972.

[42] Hill, G. W. Group Versus Individual Performance: Are N + 1 Heads Better Than One? *Psychological Bulletin.*, 91(03):517–539, 1982.

[43] Brooks, F. P. The Mythical Man Month. *Datamation*, pages 44–52, December 1974.

[44] Gordon, R. L & Lamb, J. C. A Close Look at Brook's Law. *Datamation.*, pages 81–86, June 1977.

[45] Hallows, J. Information Systems Project Management. *Amacom Press. USA*, 1998.

[46] Glass, R.L. Software Runnaways: Lessons learned from massive software project failures. *Prentice Hall. New Jersey. USA*, 1998.

[47] NATO SCIENCE COMMITTEE. Software Engineering. *Report on a conference by the NATO SCIENCE COMMITTEE. Garmisch, Germany, 7th to 11th October. Editors: Naur, P., & Randell, B.*, 1968.

[48] Jones, C. B. Systematic Software Development using VDM. *2nd Edition. Prentice Hall. Cambridge. UK.*, 1990.

[49] Leveson, N. G. High-Pressure Steam Engines and Computer Software. *Keynote Speech at the International Conference of Software Engineering. Melbourne, Australia.*, 1992.

[50] Jackson, M. Problem Frames: Analysing and structuring software development problems. *Addison-Wesley. Harlow. UK.*, 2001.

[51] Wiegers, K. Creating a Software Engineering Culture. *Software Development Journal*, 2(7):59–66, July 1994.

[52] Wiegers, K. E. Creating A Software Engineering Culture. *Dorset House Publishing. New York. USA.*, 1996.

[53] Duncan, W. R. A Guide To The Project Management Body of Knowledge. *PMI. Maryland. USA.*, 1996.

[54] Ashby, W. R. An Introduction to Cybernetics. *Methuen. London. Great Britain.*, 1956. available free online at: http://pcp.vub.ac.be/IntroCyb.pdf.

[55] Landauer, T. K. The Trouble with Computers: Usefulness, Usability and Productivity. *MIT Press. Massachusetts. USA.*, 1996.

[56] Fitzgerald, J & Gorm Larson, P. Modelling Systems: Practical Tools and Techniques in Software Development. *Cambridge University Press. Cambridge. Great Britain.*, 1998.

[57] Hall, A. Seven Myths of Formal Methods. *IEEE Software.*, pages 11–19, September 1990.

[58] Budgen, D. Software Design. *Addison-Wesley. Guildford and King's Lynn. UK.*, 1994.

[59] Cooke, S & Slack, N. Making Management Decisions. *Prentice Hall. Herts. Great Britain.*, 1991.

[60] Handy, C. Inside Organisations: Twenty Ideas for Managers. *Penguin Books. Middlesex. Great Britain.*, 1990.

[61] Handy, C. Understanding Organisations. *Penguin Books. Middlesex. Great Britain* , 1993. 4th Edition.

[62] Kotter, J. P. What Leaders Really Do. *Harvard Business Review.*, pages 103–111, May–June 1990.

[63] Vroom, V. H & Deci, E. (Editors). Management and Motivation. *Penguin Books. London. Great Britain.*, 1992.

[64] Weinberg, G.M. The Psychology of Computer Programming. *Van Nostrand Reinhold. London.*, 1971.

[65] Paulk, M. C., Weber, C. V., Garcia, S. M., Chrissis, M & Bush, M. Key Practices of The Capability Maturity Model. *Technical Report. Software Engineering Institute. Carnegie Mellon University. CMU/SEI-93-TR-25*, February 1993. Version 1.1.

[66] Herbsleb, J., Zubrow, D., Goldenson, D., Hayes, W & Paulk, M. Software Quality and The Capability Maturity Model. *Communications of the ACM.*, 40(6):30–40, June 1997.

[67] Bach, J. Enough About Process: What We Need Are Heroes. *IEEE Software.*, 12(2):96–98, 1994.

[68] Avizienis, A., Laprie, J.C., Randell, B & Landwehr, C. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, January-March 2004.

[69] Yourdon, E. Death March: The Complete Software Developer's Guide to Surviving Mission–Impossible Projects. *Prentice Hall. New Jersey. USA.*, 1997.

[70] Jones, C.B. Providing a formal basis for dependability notions. *Dept of Computer Science. University of Newcastle upon Tyne. UK*, 2002.

[71] Neumann, P.G. Computer Related Risks. *ACM Press. Addison-Wesley. USA*, 1995.

[72] Viller, S., Bowers, J., & Rodden, T. Human factors in requirements engineering: A survey of human sciences literature relevant to the improvement of dependable systems development process. *Interacting with Computers*, 11:665–698, 1999.

[73] Raymond, E.S. The Cathedral and the Bazaar: Musings on Linux and Open Source by an accidental revolutionary. *O'Rielly and Asscociates Inc. USA.*, 1999.

[74] Anderson, R. "How to Cheat at the Lottery (or, Massive Parrallel Requirements Engineering)". *Proceedings of Computer Security Applications Conference. Phoenix. Arizona.*

[75] Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. Strengthening the Case for Pair Programming. *IEEE Software*, pages 19–25, July/August 2000.

[76] Voges, U. Software Diversity in Computerised Control Systems. *Dependable Computing and Fault-Tolerant Systems. Springer-Verlag. Wien. Austria.*, 02, 1988.

[77] Kelly, J. P. J., & Murphy, S. C. Achieving Dependability Throughout the Development Process: A Distributed Software Experiment. *IEEE Transactions on Software Engineering.*, 16(2):153–165, February 1990.

[78] Littlewood, B. & Strigini, L. A discussion of practices for enhancing diversity in software designs. *DISPO Technical Report. LS-DI-TR-04. Version 1.1d. Centre for Software Reliability. City University. London. UK.*, 23rd November 2000.

[79] Popov, P., Romanovsky, A., & Strigini, L. Choosing effective methods for design diversity - how to progress from intuition to science. *Technical Report CS-TR-666. Computer Science Dept. University of Newcastle upon Tyne. UK.*, March 1999.

[80] Westerman, S. J., Shryane, C. M., Crawshaw, C. M., Hockey, G. R. J. & Wyatt-Millington, C.W. "Cognitive Diversity: A structured approach to trapping human error". *: in : Rabe, G. (Ed). SafeComp 1995: Proceedings of the 14th International Conference on Computer Safety, Reliability and Security. Belgirate. Italy.*, pages 142–155, 11th to 13th October 1995.

[81] Westerman, S. J., Shryane, C. M., Crawshaw, C. M. & Hockey, G. R. J. Engineering Cognitive Diversity. *: in : Redmill, F. & Anderson, T. Safer Systems. Proceedings of the Fifth Safety-critical Systems Symposium, Brighton. 1997. Springer-Verlag. London. UK.* , pages 111–120, October 1997.

[82] Daily, B., Whatley, A., Ash, S. R. & Steiner, R. L. The effects of a group decision support system on culturally diverse and culturally homogeneous group decision making. *Information and Management.*, 30:281–289, 1996.

[83] Hoffman, L. R. Homogeneity of member personality and its effects on group problem solving. *Journal of Abnormal and Social Psychology.*, 58:27–32, 1959.

[84] Hoffman, L. R. & Maier, N. R. F. Quality and Acceptance of Problem Solutions By Members of Homogeneous and Heterogeneous Groups. *Journal of Abnormal and Social Psychology.*, 62(02):401–407, 1961.

[85] Laughlin, P. R. & Blitz, D. S. Individual versus dyadic perfomance on a disjunctive task as a function of initial ability level. *Journal of Personality and Social Psychology.*, 31:487–496, 1975.

[86] Triandis, H. H., Hall, E. R. & Ewen, R. B. Member Heterogeneity and Dyadic Creativity. *Human Relations.*, 18(01):33–55, 1965.

[87] Watson, G., Kumar, K. & Michaelson, L.K. "Cultural diversity's impact on interaction process and performance: comparing homogeneous and diverse task groups". *Academy of Management Journal.*, 36(03):590–602, 1993.

[88] Johnson, S. Emergence. *Penguin Books. St Ives. Great Britain*, 2001.

[89] Merton, R. K. Manifest and Latent Functions. *: in : Demerath, N. J. and Peterson, R. A. (Eds). System Change and Conflict. The Free Press.*, 1967.

[90] Checkland, P. B & Wilson, B. Primary Task and Issue Based Root Definitions in System Studies. *Journal of Applied System Analysis.*, 1980.

[91] Drucker, P. The practice of management. *Heinemann, London, Great Britain*, 1958.

[92] Checkland, P & Scholes, J. Soft Systems Methodologies in Action: Making Sense Of The Field. *John Wiley, Chichester, Great Britain.*, 1990.

[93] Neumann, P. G. The Not–So–Accidental Holist. *Communications of the ACM.*, 34(9):122, 1991.

[94] Currie, W. Management Strategy for IT: An International Perspective. *Pitman ?Publishing, London, Great Britain*, 1995.

[95] Porter, M. E. & Miller, V. How Information Gives You Competitive Advantage. *Harvard Business Review*, pages 149–160, July–August 1985.

[96] Earl, M. Management Strategies For IT. *Prentice–Hall Publishing*, 1989.

[97] Lynne Markus, M & Keil, M. If we build it, they will come: Designing information systems that people want to use. *Sloan Management Review*, pages 11–25, Summer 1994.

[98] Kirby, E. G. The Importance of Recognizing Alternative Perspectives: An Analysis of a Failed Project. *International Journal of Project Management*, 14(4):209–211, 1996.

[99] Lehman, M. M & Belady, L. Program Evolution: Processes of Software Change. *Academic Press. London*, 1985.

[100] Boehm, B. W. Software Engineering Economics advances in computing science and technology.

[101] Smith, D.D. Designing Maintainable Software. *Springer–Verlag. New York. USA*, 1999.

[102] Banker, R. D & Kauffman, R. J. Strategic Contributions of Information Technology: An Empirical Study of ATM Networks. *Proceedings of the 9th Conference on Information Systems*, pages 141–150, 1988.

[103] Ross. J. Anderson. Why Cryptosystems Fail. *Communications of the ACM*, 37(11):32–40, November 1994.

[104] Bryan Clough. Cheating at Cards. *RMDP Publication. Essex. UK*, 1994.

[105] Rebecca Mercuri. Computers, freedom, privacy trip report. *RISKS FORUM*, 11(39), 1st April 1991.

[106] O. J. Makela. PIN Verification. *RISKS-FORUM*, 5(73), 8th March 1987.

[107] Michael McKay. Re: "little pitchers have big ears": Atm risks. *RISKS FORUM*, 10(75), 3rd January 1991.

[108] R Anderson. Card Fraud and Computer Evidence. *RISKS FORUM*, 15(54), 14th February 1994.

[109] David Tarabar. Stolen atm card nets 345,777 dollars. *RISKS FORUM*, 16(81), 12th February 1995.

[110] James. Essinger. ATM Networks: Thier Organisation, Security and Future. *Elsevier International Bulletins. Oxford. UK*, 1987.

[111] John Wodehouse. How to rob a bank the cashcard way. *RISKS FORUM*, 14(56), 28th April 1993.

[112] Chris Summers and Sarah Toyne. Gangs preying on cash machines. *BBC News Online*, 9th October 2004.

[113] Anthony Naggs. Re: How to rob a bank the cashcard way. *RISKS FORUM*, 14(57), 30th April 1993.

[114] Brian Randell. Court case casts doubt on cashpoint credibility. *RISKS FORUM*, 15(63), 7th March 1994.

[115] Antony Upward. Uk atms = legal challenge kpmg. *RISKS FORUM*, 13(62), 2nd July 1992.

[116] Malan, R & Bredemeyer, D. Defining Non–Functional Requirements. *Architecture Resources For Enterprise Advantage. Bredemeyer Consulting. Online at URL http://www.bredemeyer.com*, 8th March 2001.

[117] Shimizu,Y., Fujimaki, N & Hirayuma, M. A Systematic Approach to Domain-Oriented Software Development. *International Conference on Software Engineering. Kyoto. Japan*, pages 499–502, April 19-25 1998.

[118] Dobson, J. Modelling real-world issues for dependable software. *: in : Sennett, C. T. High-integrity software. Pitman Publishing. UK*, pages 274–316, 1989.

[119] J Robertson. Even more atm risks (chisholm, risks-18.47). *RISKS FORUM*, 18(48), 23rd September 1996.

[120] Byrne, M.D., & Bovair, S. A Working Memory Model of a Common Procedural Error. *Cognitive Science*, 21(1):31–61, 1997.

[121] Erling Kristiansen. Cash dispenser fraud. *RISKS FORUM*, 13(89), 2nd November 1992.

[122] Colville, A. Autoteller Problems (Re: RISKS 5.22). *RISKS FORUM*, 5(25), 7th August 1987.

[123] Chiasson, D. A Non-Fail-Safe ATM Failure. *RISK FORUM*, 4(71), 2nd April 1987.

[124] D, Andrews & D, Ince. *Practical Formal Methods with VDM*. McGraw-Hill. Berks. UK, 1991. See chapter 10 pp 143–182.

[125] Minow M. ATM Security (from Usenet). *RISKS FORUM*, 4(90), 24th May 1987.

[126] Townsley, B. An ATM Report by the Building Society Association. *The Building Society Association.*, 1984.

[127] Rogers, W. A., Cabriera, E. F., Walker, N., Gilbert, D. K. & Frisk, A. D. A Survey of Automatic Teller Machine Usage Across The Life Span. *Human Factors*, 38(1):156–166, 1996.

[128] Delvin, K. InfoSence: Turing Information into Knowledge. *W. H. Freeman. New York. USA*, pages 108–109, 2001. see chapter 10: The Art of Successful Conversation.

[129] Ralph Moonen. More atm anecdotes. *RISKS FORUM*, 12(54), 22nd October 1991.

[130] Rogers, W. A., Gilbert, D. K. & Cabrera, E. F. An Analysis of Automatic Teller Machine Usage by Older Adults: A structured interview approach. *Applied Ergonomics*, 3:173–180, 1997.

[131] R. C. Lehman. Risks of carrying a bank card. *RISKS*, 6(94), 1988.

[132] Rodney Hoffman. Massive counterfeit atm card scheme foiled. *RISKS FORUM*, 8(84), 12th February 1989.

[133] D Weir. ATM Gangsters. *RISKS FORUM*, 18(70), 20th December 1996.

[134] D. Curry. ATM Ripoff. *RISKS-FORUM*, 2(23), 6th March 1986.

[135] Warburton, N. Thinking: from A to Z. *Routledge. London. UK*, 1996.

[136] Walton, D, N. Informal Logic: A Handbook For Critical Argumentation. *Cambridge University Press. Cambridge. UK.*, 1989.

[137] BlackBurn, S. Think. *Oxford University Press. Oxford. UK*, 1999.

[138] Polya, G. How to Solve It. *Penguin Books. London. UK*, 1949.

[139] Ennis, R. H. Identifying Implicit Assumptions. *Synthese*, 51:61–86, 1982.

[140] O'Conner, J & McDermott, I. The Art Of Systems Thinking: Essential Skills For Creativity and Problem Solving. *Thorsons. London. UK*, 1997.

[141] Velleman, D. J. How To Prove It: A Structured Approach. *Cambridge University Press. Cambridge. UK.*

[142] Delin, P. S., Chittleborough, P., & Delin, C.R. What is an Assumption? *Informal Logic*, 16:115–122, 1994.

[143] DeBona, E. Lateral Thinking: A textbook of creativity. *Penguin Books Ltd. Harmondworth. Middlesex. UK*, 1970.

[144] Hofstede, G. *Cultures and Organisations: Intercultural Cooperation and Its Importance for Survival.* HarperCollinsBusiness. Hammersmith. London., 1991. see chapter 1 on levels of culture.

[145] Johnson, G & Scholes, K. Exploring Corporate Strategy: Text and Cases. *Prentice Hall.Herts. Great Britain*, 1997.

[146] McLaughlin, P. What Functions Explain. *Cambridge University Press. New York*, 2001.

[147] Mayr, E. Teleological and teleonomic: a new analysis. *: in: Plotkin, H. C. (Ed) Learning, Development, and Culture. John Wiley & Sons. UK.*, pages 17–38, 1982.

[148] Delbruck, M. Aristotle-totle-totle. *: in : Monod, J & Borek, E (Eds). Of Microbes and life. Columbia University Press. New York.*, 1971.

[149] Ulanowicz, R. E. *Ecology, The Ascendent Perspective.* Columbia University Press. New York. USA, 1997. see chapter 2: Causality in The Age of Science.

[150] RosenBlueth, A., Wiener, N. & Bigelow, J. Behaviour, Purpose and Teleology. *Philosophy Of Science.*, 10:18–24, 1943.

[151] Locke, E. A. & Latham, G. P. Building a Practically Useful Theory of Goal Setting and Task Management: A 35 Year Odyssey. *American Psychologist*, 57(9):705–717, 2002.

[152] Shallice, T. Dual Functions of Consciousness. *Psychological Review.*, 79(05):383–393, 1972.

[153] Weinberg, G. M. & Schulman, E. L. Goals and Performance in Computer Programming. *Human Factors*, 16(1):70–77, 1974.

[154] Chung, L., Nixon, B. A., Yu, E & Mylopoulos, J. Non–Functional Requirements In Software Engineering. *Kluwer Academic Publishers, Dordrecht, Netherlands.*, 2000.

[155] Fagan, M. E. Design and Code Inspections to Reduce Errors in Program Development. *IBM Systems Journal*, 15(3):182–211, 1976.

[156] Fagan, M.E. Advances in Software Inspections. *IEEE Transactions on Software Engineering*, 12(7):744–751, 1986.

[157] Laitenberger, O & Debaud, J.M. An Encompassing Life Cycle Centric Survey of Software Inspection. *The Journal of Systems and Software*, 50:5–31, 2000.

[158] Gilb, T& Graham, D. Software Inspection. *Addison–Wesley, Reading. MA*, 1993.

[159] Martin, J & Tsai, W. T. N–Fold Inspection: A Requirements Analysis Technique. *Communications of the ACM*, 33(2):225–232, 1990.

[160] Biffl, S & Halling, M. Investigating the Defect Detection Effectiveness and Cost Benefit of Nominal Inspection Teams. *IEEE Transactions on Software Engineering*, 29(5):385–397, 2003.

[161] Neilsen, J & Molich, R. Heuristic Evaluation of User Interfaces. *Proc. of the CHI Conference*, 1990.

[162] Benkoski, S.J., Monticino, M. G., & Weisinger, J. R. A Survey of the Search Theory Literature. *Naval Research Logistics*, 38:469–494, 1991.

[163] Morse, P. M. Bernard Osgood Koopman, 1900–1981. *Operations Research*, 30(3):417–427, 1982.

[164] Koopman, B. O. Search and Screening: General Principles with Historic Applications. *Pergamon, New York,*, 1980.

[165] Frost, J. R. Principles of Search Theory. *SOZA Technical Report. Online @ http://www.soza.com*, 1999.

[166] Chapagne, L., Carl, R. G., & Hill, R. Search Theory, Agent–Based Simulation, and U-Boats in the Bay of Biscay. *Proceedings of the 2003 Winter Simulation Conference*, 2003.

[167] Chwif, L & Paul, R.J. On Simulation Model Complexity. *Proceedings of the 2000 Winter Simulation Conference*, pages 449–455, 2000.

[168] Robson, A.J. Designing and Building Business Models. *McGraw-Hill. Berks. UK*, 1995. see: Chapter 2 Problem Conceptualization and Data Analysis. pp. 10-33.

[169] Lucey, T. Quantitative Techniques. *DP Publications Ltd, Guernsey. UK*, 1992. see chapter 16 pp. 223–237 incl entitled "Simulation".

Index

Appendix A

Non-Functional Requirements Key

A.1 Existing NFR Framework

In this appendix section a subset of the non–functional requirements notation in Chung [154] et al, that is employed in chapter 8 will be explained. It can be seen from figure A.1 on the following page that the non–functional requirements framework forms a tree in structure. The single top cloud at number (1) is the main soft–goal to be promoted. Such a goal would correspond to a dependability attribute to be promoted — such as availability, security, etc. The single circle at number (2) is an AND operation and shows that both soft subgoals at number (3) are required to be promoted to promote the main soft goal at number (1).

At number (7) is the operationalisation goal. The reason why the goals and subgoals at numbers (1) and (3) are called soft–goals is that they represent, in principle, non–functional goals. These soft–goals capture the desired properties or purpose(s) of the software system, but they require some functionalisation in order to implement them. The heavier lined clouds at number (7) represent a functional ascription to promote the soft subgoals immediately above them. The *tick* inside the operationalisation subgoal at number (8) shows that this particular operationalisation was actually implemented, while the *cross* inside the operationalisation subgoal at number (9) was not actually implemented. The double circle at the apex of the two lines at number (4) is an XOR operation and shows that either of

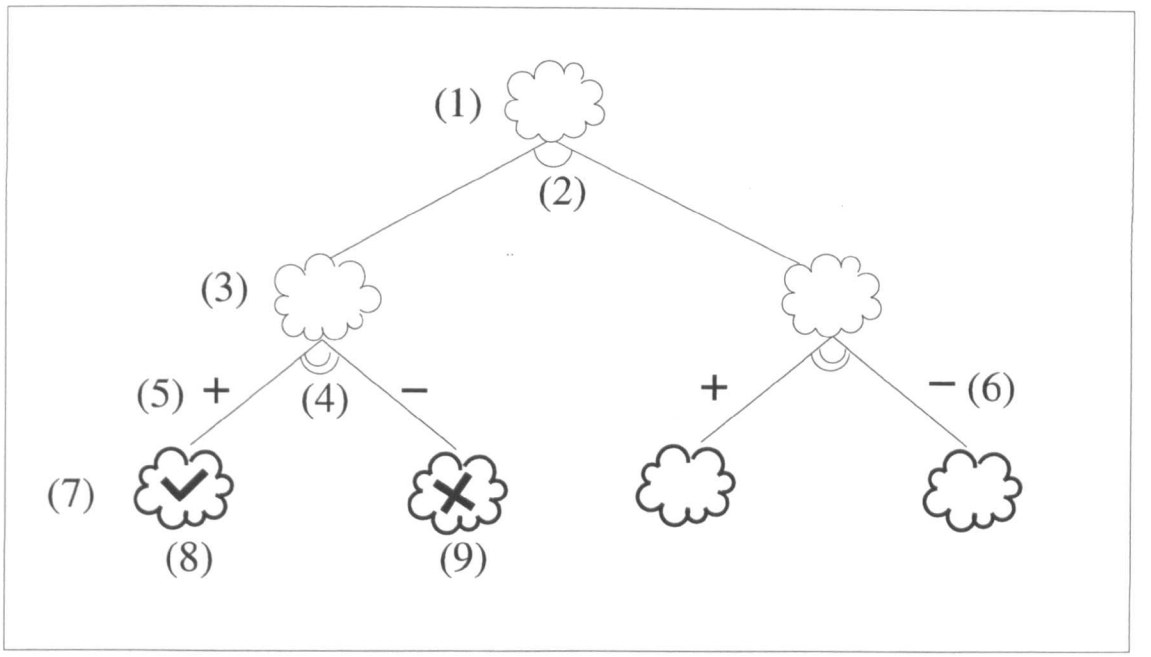


Figure A.1: Non–Functional Requirements Framework Key One

the operationalisation subgoals could be implemented to promote the soft subgoal at number (3).

The '+' symbol on the line at number (5) is used to show that this particular operationalisation subgoal is judged to fully promote the soft subgoal at number (3). Conversely, the '-' symbol on the line at number (6) is used to show that this particular operationalisation subgoal has been assessed to to potentially undermine promotion the soft subgoal immediately above it.

There is also a number of other notions used in chapter 8. These are shown in figure A.2 on the next page. The dashed line clouds at number (10) and (11) represent priority claims made during analysis of a software system. Such claims often represent a belief or assertion about some aspect of the software system. It can also be seen that they can contain a tick or a cross. The meaning of these is the same as before and shows whether the claim was accepted or rejected, respectively.

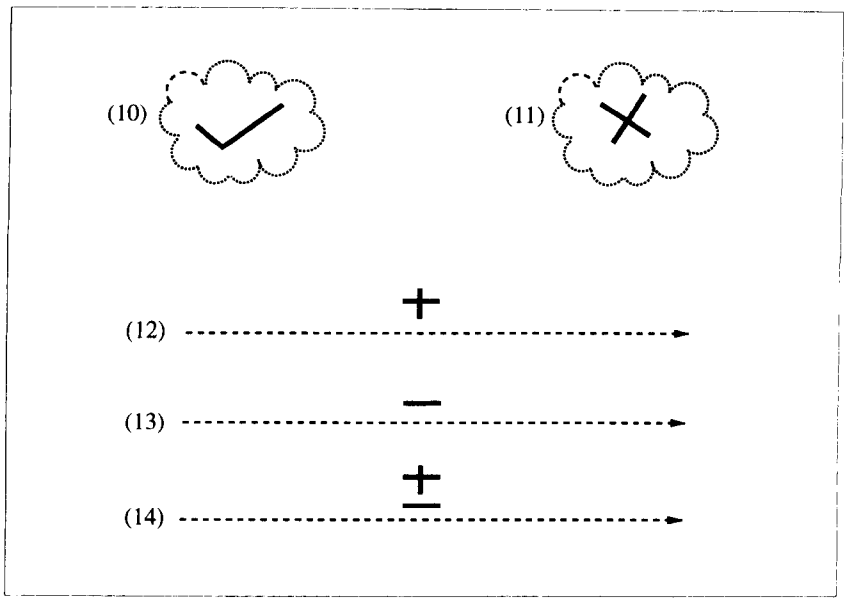


Figure A.2: Non–Functional Requirements Framework Key Two

The dashed lines at numbers (12), (13), and (14) represent interdependency relationships between soft goals, soft subgoals, and/or operationalisation subgoals. If they contain a '+' symbol on the line like number (12) this means that the interdependency relationship is considered to be a positive and reinforcing one. If they contain a '-' symbol on the line like number (13) then the interdependency relationship is considered to be a negative, conflicting, or antagonistic one that can undermine the goal or subgoal it points toward. Lastly, if they contain both a '+' and '-' symbol on the line like number (14) then the interdependency relationship is considered to be a compatible one that neither reinforces or conflicts with the goal or subgoal it points toward.

A.2 Additions to NFR Framework

In addition to the established non–functional requirements framework notation discussed in appendix section A.1, adapted notation was specifically introduced in chapter 8 which is not part of the established non–functional requirements notation. These additions are illustrated in figure A.3 on the following page. The rectangle box is used to represent the suggested assumptive reasoning that subse-

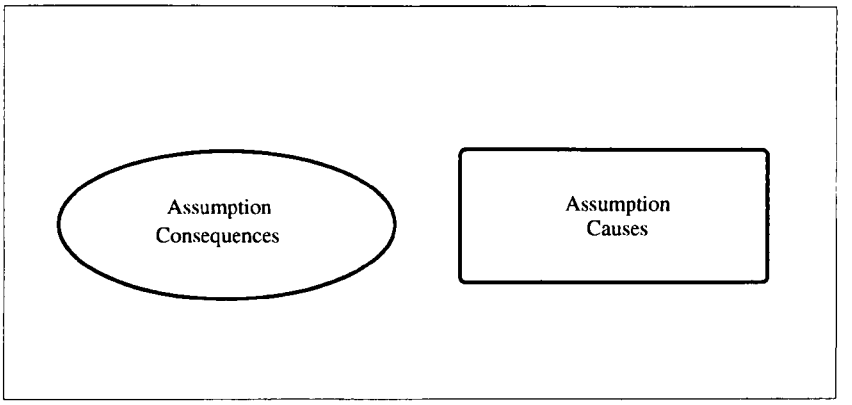


Figure A.3: NFR Additions

quently gives rise to certain priority claims and functionalisation subgoal preferences. The ellipse, on the other hand, is used to represent the suggested fault and vulnerability consequences subsequently experienced through such assumptions being made.

In both cases, the two boxes will be labelled with computer–based system contexts that: a) give rise to the assumption; and b) experience the consequences of the assumption. Furthermore, both boxes will have arrows pointing towards various priority claims and/or operationalisation subgoals to indicate how they correspondingly influence subgoals through the vulnerabilities and faults they tend create.

Appendix B

Statistical Significance Test

B.1 Overview

In figure B.1 on the next page the search simulation model has been configured to use the Diverse Coverage and Diverse Sensitivity (i.e. DC and DS) search strategy to compare search performance on a further two dimensions. These are 1) Number of searchers added (i.e. in rows); and 2) Number of search space locations (i.e. in columns).[1] The figures in each row/column represent the search simulation model's output with this search strategy under additional dimensions 1 and 2 above

Each searcher represents a further fifty resource units of search effort allocation up to (a theoretical) 100% search effort allocation coverage of five hundred resource units. Each addition of a searcher is predisposed to one search location type and the other remaining predispositioning is divided up against the remaining search space location types.

The purpose of this simulation experiment is to satisfy goal 2 of the thesis as stated in chapter 10 section 10.2.1 to compare the under/over representational influences of non–functional attributes upon assumption detection. In the statistical analysis that follows the expected distribution that will be used will be when there are an

[1] The last column (far right) represents the situation where there are the same number of search locations configured as searchers employed in the search.

Searchers/Locations	2 Loc.	3 Loc.	4 Loc.	5 Loc.	6 Loc.	7 Loc.	8 Loc.	9 Loc.	10 Loc.	Same No.
2 Searchers	*39*	40	40	37	36	34	34	34	37	*39*
3 Searchers	49	*59*	59	58	56	54	55	55	55	*59*
4 Searchers	70	68	*80*	75	77	76	76	76	72	*80*
5 Searchers	88	87	91	*100*	97	96	96	95	86	*100*
6 Searchers	106	103	111	109	*117*	113	113	113	107	*117*
7 Searchers	117	116	119	123	127	*137*	133	131	119	*137*
8 Searchers	132	139	140	135	136	145	*153*	148	151	*153*
9 Searchers	148	148	151	159	155	159	167	*171*	167	*171*
10 Searchers	160	162	167	168	169	171	180	182	*184*	*184*

Figure B.1: Searchers to Locations Comparisons

equal number of location types and searchers. The observed distributions of interest will be the two search situations when: a) there is an under representation search situation where there are more location types than searchers employed to predispose them to; and b) where there is an over representation search situation where there are more searchers employed than location types to predispose them diversely to. By analysing both these searching situations statistically, it is possible to get a significance indication of the negative/positive assumption detection effects when non–functional goal attributes are under/over represented.

B.2 More Locations Types Than Searchers

As explained in section B.1 on the preceding page this section shows the statistical significance tests of the search simulation model's output when there are more location types than there are searchers. These are compared to the (ideal) search situation when there are an equal number of searchers to location types to indicate if there is any assumption detection effectiveness losses when a search space location type is under promoted or represented. The null hypothesis "H0" is that there is no significant detection effectiveness difference when there are more location types than searchers to predispose to them. Hypothesis "H1" is that there is a significant detection effectiveness difference when there are an equal number of location types and searchers to predispose them to.

Figures B.2 on the next page, B.3 on the following page, B.4 on the next page, B.5 on the following page, B.6 on the next page, B.7 on the following page, B.8 on page 387 show the Chi squared tests on the search simulation model's output when there are more location types than searchers. They show that when com-

Searchers	10 Locations & 2 to 9 Searchers	Same Number	O – E	(O – E)^2	(O – E)^2/E
2 Searchers	37	39	-2	4	0.1
3 Searchers	55	59	-4	16	0.27
4 Searchers	72	80	-8	64	0.8
5 Searchers	86	100	-14	196	1.96
6 Searchers	107	117	-10	100	0.85
7 Searchers	119	137	-18	324	2.36
8 Searchers	151	153	-2	4	0.03
9 Searchers	167	171	-4	16	0.09
				X^2 =	6.47
				X^2 < 14.067	Accept H0

Figure B.2: 10 Location Types and 2 to 9 Searchers

Searchers	9 Locations & 2 to 8 Searchers	Same Number	O – E	(O – E)^2	(O – E)^2/E
2 Searchers	34	39	-5	25	0.64
3 Searchers	55	59	-4	16	0.27
4 Searchers	76	80	-4	16	0.2
5 Searchers	95	100	-5	25	0.25
6 Searchers	113	117	-4	16	0.14
7 Searchers	131	137	-6	36	0.26
8 Searchers	148	153	-5	25	0.16
				X^2 =	1.93
				X^2 < 12.592	Accept H0

Figure B.3: 9 Location Types and 2 to 8 Searchers

Searchers	8 Locations & 2 to 7 Searchers	Same Number	O – E	(O – E)^2	(O – E)^2/E
2 Searchers	34	39	-5	25	0.64
3 Searchers	55	59	-4	16	0.27
4 Searchers	76	80	-4	16	0.2
5 Searchers	96	100	-4	16	0.16
6 Searchers	113	117	-4	16	0.14
7 Searchers	133	137	-4	16	0.12
				X^2 =	1.53
				X^2 < 11.070	Accept H0

Figure B.4: 8 Location Types and 2 to 7 Searchers

Searchers	7 Locations & 2 to 6 Searchers	Same Number	O – E	(O – E)^2	(O – E)^2/E
2 Searchers	34	39	-5	25	0.64
3 Searchers	54	59	-5	25	0.42
4 Searchers	76	80	-4	16	0.2
5 Searchers	96	100	-4	16	0.16
6 Searchers	113	117	-4	16	0.14
				X^2 =	1.56
				X^2 < 9.488	Accept H0

Figure B.5: 7 Location Types and 2 to 6 Searchers

Searchers	6 Locations & 2 to 5 Searchers	Same Number	O – E	(O – E)^2	(O – E)^2/E
2 Searchers	36	39	-3	9	0.23
2 Searchers	56	59	-3	9	0.15
3 Searchers	77	80	-3	9	0.11
5 Searchers	97	100	-3	9	0.09
				X^2 =	0.59
				X^2 < 7.815	Accept H0

Figure B.6: 6 Location Types and 2 to 5 Searchers

Searchers	5 Locations & 2 to 4 Searchers	Same Number	O – E	(O – E)^2	(O – E)^2/E
2 Searchers	37	39	-2	4	0.1
3 Searchers	58	59	-1	1	0.02
4 Searchers	75	80	-5	25	0.31
				X^2 =	0.43
				X^2 < 5.991	Accept H0

Figure B.7: 5 Location Types and 2 to 4 Searchers

Searchers	4 Locations & 2 to 3 Searchers	Same Number	O – E	(O – E)^2	(O – E)^2/E
2 Searchers	40	39	1	1	0.03
3 Searchers	59	59	0	0	0
				X^2 =	0.03
				X^2 < 3.841	Accept H0

Figure B.8: 4 Location Types and 2 to 3 Searchers

Searchers	2 Locations & 3 to 10 Searchers	Same Number	O – E	(O – E)^2	(O – E)^2/E
3 Searchers	49	59	-10	100	1.69
4 Searchers	70	80	-10	100	1.25
5 Searchers	88	100	-12	144	1.44
6 Searchers	106	117	-11	121	1.03
7 Searchers	117	137	-20	400	2.92
8 Searchers	132	153	-21	441	2.88
9 Searchers	148	171	-23	529	3.09
10 Searchers	160	184	-24	576	3.13
				X^2 =	17.45
				X^2 > 14.067	Accept H3 @ 5%

Figure B.9: 2 Location Types and 3 to 10 Searchers

pared against a search situation where there are an equal number of searchers and location types then there is no statistical significance in the compared distributions. Therefore the null hypothesis "H0" is accepted.

B.3 More Searchers Than Locations

As explained in section B.1 on page 384 this section shows the statistical significance tests of the search simulation model's output when there are more searchers than location types. Again, these are compared to the search simulation model's output when there are an equal number of location types and searchers to predispose to them. The purpose for doing so is to indicate the assumption detection effectiveness in both cases — to see which is potentially more effective. The null hypothesis "H2" is that there is no difference in detection effectiveness when there are more searchers than location types. The alternative hypothesis "H3" is that there is an increase in detection effectiveness when there are an equal number of searchers and location type predispositions.

Figures B.9, B.10 on the following page, B.11 on the next page, B.12 on the following page, B.12 on the next page, B.13 on the following page, B.14 on the next page, B.15 on the following page show the Chi squared tests — based on the search simulation model's data output when there are more searchers than location types compared to when there exists an equal number of searchers to location

Searchers	3 Locations & 4 to 10 Searchers	Same Number	O – E	(O – E)^2	(O – E)^2/E
4 Searchers	68	80	-12	144	1.8
5 Searchers	87	100	-13	169	1.69
6 Searchers	103	117	-14	196	1.68
7 Searchers	116	137	-21	441	3.22
8 Searchers	139	153	-14	196	1.28
9 Searchers	148	171	-23	529	3.09
10 Searchers	162	184	-22	484	2.63
				X^2 =	15.39
				X^2 > 12.592	Accept H3 @ 5%

Figure B.10: 3 Location Types and 4 to 10 Searchers

Searchers	4 Locations & 5 to 10 Searchers	Same Number	O – E	(O – E)^2	(O – E)^2/E
5 Searchers	91	100	-9	81	0.81
6 Searchers	111	117	-6	36	0.31
7 Searchers	119	137	-18	324	2.36
8 Searchers	140	153	-13	169	1.1
9 Searchers	151	171	-20	400	2.34
10 Searchers	167	184	-17	289	1.57
				X^2 =	8.5
				X^2 < 11.070	Accept H0

Figure B.11: 4 Location Types and 5 to 10 Searchers

Searchers	5 Locations & 6 to 10 Searchers	Same Number	O – E	(O – E)^2	(O – E)^2/E
6 Searchers	109	117	-8	64	0.55
7 Searchers	123	137	-14	196	1.43
8 Searchers	135	153	-18	324	2.12
9 Searchers	159	171	-12	144	0.84
10 Searchers	168	184	-16	256	1.39
				X^2 =	6.33
				X^2 < 9.488	Accept H0

Figure B.12: 5 Location Types and 6 to 10 Searchers

Searchers	6 Locations & 7 to 10 Searchers	Same Number	O – E	(O – E)^2	(O – E)^2/E
7 Searchers	127	137	-10	100	0.73
8 Searchers	136	153	-17	289	1.89
9 Searchers	155	171	-16	256	1.5
10 Searchers	169	184	-15	225	1.22
				X^2 =	5.34
				X^2 < 7.815	Accept H0

Figure B.13: 6 Location Types and 7 to 10 Searchers

Searchers	7 Locations & 8 to 10 Searchers	Same Number	O – E	(O – E)^2	(O – E)^2/E
8 Searchers	145	153	-8	64	0.42
9 Searchers	159	171	-12	144	0.84
10 Searchers	171	184	-13	169	0.92
				X^2 =	2.18
				X^2 < 5.991	Accept H0

Figure B.14: 7 Location Types and 8 to 10 Searchers

Searchers	8 Locations & 9 to 10 Searchers	Same Number	O – E	(O – E)^2	(O – E)^2/E
9 Searchers	167	171	-4	16	0.09
10 Searchers	180	184	-4	16	0.09
				X^2 =	0.18
				X^2 < 3.841	Accept H0

Figure B.15: 8 Location Types and 9 to 10 Searchers

types. Overall it is indicated that when there is significantly more searchers — such as 3 location types with 4 to 10 searchers and when there is only 4 location types with 5 to 10 searchers, the homogeneous predispositions results in statistically significant reductions (at 5% confidence level) in detection effectiveness as searchers duplicate on target object detections. However, although the simulation model's output indicates that even with higher numbers of location types and more searchers (i.e. from 5 location types and 6 to 10 searchers and above) that there is generally less target objects detected than when there is an equal number of searchers and location types, the statistical tests on these distributions do not indicate a statistically significant difference. Therefore, we can accept the hypothesis "H3" for search situations where there are only a small number of location types and increasingly large searchers employed, but at higher numbers of location types to searchers (i.e. 5 location types and above) we must accept "H2" that there is no statistically significant detection effectiveness loss than when there is an equal number of searchers and location types

B.4 Overall Comment

Relating the statistical significance tests, within the wider scope of the thesis, indicates that there are reductions in inspection process loss during defect detection if each inspector is predisposed to gain a wider coverage and assumption/fault sensitivity of the defect space by being predisposed to promote a unique goal. However, over promotion of a specific goal, by multiple inspectors (especially at large numbers of overlap), results increasingly in them being sensitised to cover the same defect space and increase the chance of them duplicating detections. Finally, the simulations and statistical analysis of under promotion or representation of a particular goal, whilst reducing the chance of detecting related defects, does not, in itself, directly result in significant inspection process loss.