Using Requirements and Design Information to Predict Volatility in Software Development

Claire Ingram

20th July 2011

This thesis is submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

School of Computing Science,
Newcastle University
NE1 7RU
UK

This thesis is dedicated to Grant Ingram.

Abstract

We hypothesise that data about the requirements and design stages of a software development project can be used to make predictions about the subsequent number of development changes that software components will experience. This would allow managers to concentrate time-consuming efforts (such as traceability and staff training) to a few at-risk, cost-effective areas, and may also allow predictions to be made at an earlier stage than is possible using traditional metrics, such as lines of code. Previous researchers have studied links between change-proneness and metrics such as measures of inheritance, size and code coupling. We extend these studies by including measures of requirements and design activity as well. Firstly we develop structures to model the requirements and design processes, and then propose some new metrics based on these models. The structures are populated using data from a case study project and analysed alongside existing complexity metrics to ascertain whether change-proneness can be predicted. Finally we examine whether combining these metrics with existing metrics improves our ability to make predictions about change-proneness. First results show that our metrics can be linked to the quantity of change experienced by components in a software development project (potentially allowing predictions to take place earlier than before) but that best results are obtained by combining existing complexity metrics such as size, or combining existing metrics with our newer metrics.

Acknowledgements

This work is supported by a CASE studentship awarded by BAESYSTEMS and the U.K. Engineering and Physical Sciences Research Council.

I'd like to thank personnel from the CARMEN project for additional support and access to project data and development team members for this study. I'd also like to thank my supervisor Steve Riddle for his help throughout the work towards this thesis.

Conversations with David Greathead, Zoe Andrews and Igor Mozolevsky were also very helpful, so my thanks go to the individuals concerned for their patience and suggestions.

Finally, thanks go to all the individuals in the CSR, who provide such fun and interesting discussion at tea times.

Contents

Chapter 1

Introduction

Change has traditionally been perceived as highly problematic for a software development project. Past studies have correlated high levels of requirements volatility with cost or time overruns [140, 159], for example, whilst changes arising post-development can result in a long-term decline in the reliability or maintainability of a piece of software [15]. The total cost impact of requirements change alone on industrial projects has been estimated by various researchers at anything between 40 and 90 percent of the total project costs [76, 92, 36, 88, 14].

However, despite these problems, change is common and unavoidable for most projects [16]. For example, a single survey of industrial projects within one company revealed that on average 73 percent of a project's requirements changed over the course of development [50], whilst the first of Lehman's widely-cited 'Laws of program evolution' stipulated continuing change [91]. As has been widely remarked upon, the development and installation of a new software system itself often changes user behaviour, resulting in new user needs to be met [13, 22, 88, 91, 114, 130, 138] - such that the installation of a new system in itself sparks the need for changes.

Much research concentrates on requirements change specifically, but this is not the only reason why changes are carried out. Changes are also motivated by errors in the requirements [114] or in the code; the 'I'll know it when I see it' phenomenon [43, 18, 50] (referring to the difficulty users experience in visualising and communicating their expectations); the availability of new hardware or technologies [11, 91, 144, 43]; new or amended legal requirements [88] regulations or standards [43, 135]; social changes [91]; amended business goals and policies [114] and expanding markets [11, 135]; difficulties in 'designing' certain non-functional requirements, such as 'usability'; or organisational policies or financial considerations [43]. Changes may also be necessitated by the presence of errors in the software. And finally, the well-documented phenomenon of 'code decay' can also necessitate code re-factoring. Code decay typically becomes a problem during the maintenance phase although designs can be 'damaged' before release, too [95].

In fact, three major categories of change were suggested by Swanson in 1976: *corrective* maintenance (resolving failures); *adaptive* maintenance (responding to alterations in the operating environment); and *perfective* maintenance (improvements even when no actual failure has occurred). All of these types of changes can cause both problems and benefits to software projects. On the plus side, software change ensures that software meets its purpose, which inevitably change over time [91]; it can result in higher quality software as errors are detected and resolved and system performance is enhanced; and the life of the system is extended as refactoring ensures code remains maintainable for longer. On the negative side, however, any type of change which is originally unplanned for - fixing errors, adapting to new requirements, making the decision to refactor - has the potential to eat into project time, cause new errors and contribute to code decay.

Table 1.1: Summary of our hypotheses

Summary of hypotheses
H_1: Existing complexity metrics can be used to predict change-proneness.
H_2: Our metrics can be used to predict changes.
H_3: A predictive model using our metrics (combined with other types of existing complexity metrics if necessary) to predict change-proneness outperforms a similar model using the existing complexity metrics alone.

Our work rests on the notion that change tends to be most problematic when it is unexpected (and therefore not planned for). If a software designer could estimate which components in an early system would become change-prone then steps could be taken to minimise the ill effects of change in this area. For example, fine-grained traceability techniques could be applied to these areas, to aid with change management.

1.1 Hypothesis

Our hypothesis is as follows:

> *Assessing the extent of requirements and design activities can lead to improvements in our ability to predict change-proneness in a software system.*

By 'change-proneness', we mean the amount of change to which a software component (such as a Java class file) is subjected over its life. Our predictions centre on detecting *which* components will experience higher levels of change, rather than estimating how much change that might be. By 'requirements and design activities' we mean to include any of the following: modelling and elicitating requirements; developing use cases; identifying design problems; considering available options and selecting from among them; making arguments for and against requirements inclusion and/or design choices; recording requirements and/or design rationale; and identifying external sources of rationale such as standards or regulations. We will attempt to develop a metric for assessing the 'extent' of these activities.

We need to test a number of separate hypotheses to reach this conclusion. Table 1.1 summarises these and we present the reasoning behind each in this section.

As a benchmark, we intend to test how well existing metrics perform against our case study data. Previous studies (summarised in Chapter 2) have suggested that a link may be found between change-proneness in the development process and code complexity metrics (also summarised in Chapter 2). This leads to the first question to test: do any existing metrics show a robust correlation with change-proneness?

> *H_1: Existing complexity metrics can be used to predict change-proneness.*

We intend to test H_1 by gathering data on change-proneness from a case study, generating values for existing metrics and attempting to create a predictive model. Failing to find a sufficiently strong corroboration between metrics and change-proneness could indicate that the metrics cannot be used to make predictions, or that results from previous studies with a different conclusion cannot be generalised to all projects. Alternatively, it may mean the particular algorithms we use do not produce data that can be used for our purpose. Results of our analyses are presented in Chapter 5.

Although some previous studies have examined links between metrics and change-proneness, we are not aware of any studies that examine design and/or requirements activity together with change data. Our next step is thus to develop a new set of metrics (outlined in Chapter 3) which are designed to assess requirements and/or design activity on a project. Our predictions are summarised in 3.2.

> *H_2: Our metrics can be used to predict change-proneness.*

As with existing complexity metrics, we intend to test this by gathering data from a case study project. We examine whether any of these metrics shares a relationship with change-proneness in Chapter 6. Finally, we then turn to our final hypothesis H_3:

> *H_3: A predictive model using our metrics (combined with other types of existing complexity metrics if necessary) to predict development change-proneness outperforms a similar model using the existing complexity metrics alone.*

We will test H_3 by combining metrics from analyses for H_1 and H_2. This will help us determine whether any metric or group of metrics performed better in a predictive capacity. Results are presented in Chapter 7, and evaluations of overall predictive capacity for metrics in Chapter 8.

1.1.1 Why is it useful to predict change-proneness?

Here we present rationale behind our study, which we believe is useful on a practical level for the following reasons:

- Some software engineering techniques - traceability in particular - bring many potential benefits to a project, but still many projects fail to capitalise on them (discussed further in Sections 2.1.1 and 2.1.2). In many cases this is because project planners find it difficult to justify the effort involved [35]. If planners can predict change-prone areas, data capture can be concentrated on components most at risk, maximising the return on investment and allowing more projects to reap the benefits.

  This is a key principle underpinning 'value-based' software engineering, which emphasises prioritisation of artifacts and selective application of time-consuming techniques [39, 52]. For example, Heindl and Biffl [69] describe a 'value-based requirements tracing' strategy which involves selectively reducing the granularity of trace data. Similar, priority-based strategies are described by Egyed *et al.* [51, 52]; and by Huang and Boehm [20, 73]. The criteria for prioritisation will vary from project to project, but predictions of change-proneness may be a useful ranking criteria for many projects.

- If it is known in advance which areas are likely to be volatile, steps could be taken to ensure that more developers are familiar with the design and implementation of potentially change-prone areas. This could help to protect vulnerable areas of the project against knowledge loss that occurs when personnel leave the team or are otherwise unavailable.

- A component that changes frequently is likely to suffer from code decay, the phenomenon whereby code generally becomes increasingly difficult to 'maintain'. There are a number of causes, including increased complexity and eroded structure after code has been altered many times [13, 15, 92, 90] 'until the system becomes so brittle that even simple changes become infeasible' [132]. The value of maintaining well-structured code may not be considered by the pressured maintenance programmer [13, 92], who is often inexperienced or lacking system

knowledge [14, 115]. Re-factoring regularly can help the situation, by re-imposing structure and decreasing complexity. Re-factoring can have implications for the design, with changes potentially necessary to neighbouring components as well – for example, if the component is split into smaller components or is re-organised. An awareness of which components are likely to be change-prone can help with planning as the project progresses, because managers have early warning that some components may require refactoring, can estimate time needed and possibly schedule time in the plan. Ratzinger *et al.* have conducted research specifically on predicting refactoring; we discuss this in Section 2.4.5.

- Documentation is a task rarely prized by development teams and tends to be assigned a low priority if a project is running late. With some warning, time-consuming maintenance documentation for areas predicted to be change-prone could be prioritised, so that areas where it is most likely to be needed are documented first.

1.1.2 Why use early stage data?

There are a number of reasons why we believe that using information about requirements and design activity may prove fruitful:

- As mentioned previously, much software volatility is driven by external factors, such as: business environment; regulations; hardware/software platforms; or organisational culture. A simple metric to incorporate notions of design/requirements rationale and externally-maintained standards will help to identify hidden dependencies between components and external factors (see Section 2.1 on hidden dependencies). Dependencies on external sources could potentially indicate code modules which are more likely to change, as changes may be propagated from outside.

- System soft goals (such as 'usability' or 'security') or are known to be particularly problematic from a design point of view (discussed in Section 2.2.2). Components linked to soft goals could be at greater risk of becoming change-prone.

- An awareness of requirements and design activities can help to develop a clearer picture of a component's relative 'importance'. For example, components implementing key pieces of high-level functionality are likely to have a different pattern of change than more low-level components, and also a different pattern of design activity and requirements links. This type of data could lead to a clearer picture of components' relative importance compared to a model based on software complexity alone.

- Elevated levels of activity at early stages in the project could indicate areas where there has been a particularly extensive dialogue with users or extra time spent on researching options, both of which could result in fewer changes at later stages. Or, on the other hand, an extended period spent on design activities could equally indicate that the component is particularly difficult or complex or that more compromises are being forced onto the design, which could also have an impact on future instability. We discuss this further in Section 3.4.3.

Existing methods for predicting where volatility will occur tend to rest on complexity metrics that describe code structure (such as counts of the number of methods or instances of inter-class coupling) or size (such as lines of code). This type of information may sometimes be available towards the end of the design stage - for example, an approximation of the number of methods. However, some information - such as lines of code, or coupling measures - will not be available

until the code is actually written. The types of information we are proposing to use - interactions between requirements/design and components - is likely to be available towards the end of the design stage or earlier. This would mean that predictions could be made at an earlier stage, when preparatory action can still be taken.

Metrics which assess coupling between components pose some problems for predictive models. The values calculated initially will almost certainly change over time as other components are added to/removed from the system, so metric values cannot be known for certain until quite late in the development. For example, it is possible a code module may initially appear to be prone to volatility based on its coupling or inheritance patterns with nearby modules, but if another module is added nearby, these coupling patterns may change. In contrast, our metrics are unlikely to be affected by the addition of new unplanned components to the system.

On the other hand, a disadvantage posed by using early stage data to make predictions derives from the fact that the finished system is commonly refined and expanded over time, and often has differences from the very first design. Thus, data from the earliest stages of the project is increasingly *less* likely to reflect accurately the current state of the system than is data taken from later stages. For this reason, our expectation is that we will achieve better results by mixing our metrics with existing code-based metrics than we will by using our metrics alone. This is supported by our results, which show that our new metrics can correctly predict more components which are volatile when mixed with existing metrics. Despite this a more precise prediction is achieved by predictive models which use existing metrics only (see Chapter 8).

1.1.3 Criteria for success

Different projects have different priorities, cultures, funding models and resourcing. A prediction of change-proneness could be useful to a project manager who would like to benefit from these strategies and tactics, but does not feel able to justify the cost of documenting the system in full. We are not suggesting that projects should necessarily limit traceability or documentation efforts (for example), but we are suggesting that change-proneness predictions may help a project which does not currently benefit from traceability to begin employing it (although predictions may also be useful to projects which already use all the strategies already mentioned, too).

In order to be useful for this purpose, we should ensure that any predictive model is:

- as accurate as possible. We discuss means of evaluating 'accuracy' in Chapter 7.
- sparing in its predictions (a model which predicts change-proneness for vast quantities of components is not very practical as a prioritising tool). We discuss this in Chapter 7.
- low effort in terms of gathering the data needed.
- generalisable to a variety of projects.
- using data which is available by the end of the design stage.

We return to this list of criteria in our evaluations, presented in Chapter 8.

1.2 Methodology

Several techniques are available to test our hypothesis, including an experiment, a case study, or a survey. Literature on techniques and methodologies for testing hypotheses in a software engineering context is sparse. Basili *et al.* published an influential framework in 1986 for defining experimental work in the field [8]. A more detailed and holistic framework was produced in 1995 by Kitchenham

et al. [82]. The latter redefine Basili *et al.*'s categories, so that single-project experiments are instead re-classed as case studies.

Our work is exploratory in nature, as we intend to test the initial feasibility of using requirements- and design-related data to make predictions about change-proneness. An acceptable technique for this type of work is a detailed study from a single project; this is defined as a case study by Kitchenham *et al.* [82]. Case studies can generate useful data, but it's important to aware of their limitations. In particular, it can be very difficult to generalise from a case study, unlike a multi-project survey or controlled experiment [82]. We discuss the generalisability of our case study in particular in Chapter 4, where we also address Kitchenham *et al.*'s proposed 'checklist' of important considerations for conducting case studies.

Our case study is an observational case study (as described, in a separate paper, by Kitchenham *et al.* [84]). Data capture for this thesis is conducted post-hoc by researchers without detailed project knowledge. For this reason it is important that project personnel validate the data [8]. Several validation exercises are undertaken to ensure that data gathered is reliable and accurate. These are described in Chapters 4 and 6. In addition, it must be possible to verify results across other projects and studies, and so the data being studied must be defined clearly [8, 96]. We provide clear definitions of our metrics in Chapter 3 and explain how we generated values for them and all other data used in our study in Chapter 4.

Our analytical technique centres on generating a number of predefined data sets and searching them all for a useful relationship. This is a valid technique, but it must be recognised that the study is exploratory in nature, and that any results should be verified in subsequent studies before being accepted as definitive conclusions. This is because, by generating a large number of results, there is a chance that we unearth statistically significant findings purely by chance [82]. We explain how we have attempted to contain this risk in Section 5.2.2, and briefly discuss further work (including further confirmatory studies) in Chapter 9.

1.3 Contributions

This thesis makes the following original contributions.

- We show that most of our newly-proposed metrics (based on requirements or design data) can be used to detect significant differences in the number of changes experienced by components, indicating that there are relationships between early-stage development activities and the later number of changes.
- We show that some aspects of predicting change-proneness can be improved by combining our new metrics with existing metrics.
- Previously published studies examining the accuracy of existing metrics in predicting change-proneness have generated different conclusions (we discuss this further in Chapter 2). We present in this thesis new data from a previously unpublished case study, which supports the notion that certain metrics can be used to make statistically significant predictions about change-proneness. However, our results cannot be used to support previous studies which have found linear relationships between metrics and change-proneness.

1.4 Structure of this thesis

This rest of this thesis is laid out as follows.

Chapter 2 provides background material on some concepts underlying this thesis. We examine a selection of systems used to capture and structure requirements or design-related information (key features seen in these systems are built upon in Chapter 3). We survey some existing metrics for characterising software, and methods for validating those metrics, and we discuss some previous studies looking at these metrics in relation to software quality indicators such as changes or defects.

In Chapter 3 we present a new set of metrics for capturing design- and requirements-related activities. We follow the Model-Order-Mapping methodology [64] for doing so, which involves building a model of the data we wish to capture (requirements- and design-related information) and then developing metrics from those models. We build on the requirements and design systems surveyed in Chapter 2, drawing out key elements for our models. We validate our new metrics using existing criteria developed for validating software metrics.

Chapter 4 introduces our major case study, the CARMEN project. We characterise the problem domain and project culture as well as describing how we captured data from the project to populate the models described in Chapter 3. Once the models are populated we can generate values for our new metrics. We also select a tool to generate some existing metrics (such as size, coupling and inheritance metrics) for comparison.

We present some initial analysis in Chapters 5 (examining existing metrics) and 6 (examining our new metrics). We determine thresholds where a statistically significant relationship between change-proneness and our metrics can be demonstrated, and distill this into a set of observational rules.

We evaluate the overall performance of all metrics - existing and new - in Chapter 7 using standard information retrieval evaluation techniques and recommend the 'best' performing predictive models using a variety of criteria. Chapter 8 presents our evaluation of the study in general, including any threats to validity.

Finally, in Chapter 9 we present some conclusions and some directions for further research. In general, results from our case study study demonstrate the feasibility of employing requirements and/or design data to enhance predictions about change-proneness. The performance of the majority of our metrics in detecting significant numbers of changes for components (see Chapter 3 for descriptions of metrics and Chapter 6 for results) demonstrates that there are relationships between early stage data and subsequent numbers of changes which could be leveraged for predictive use.

Chapter 2

Background

In this chapter we introduce some concepts underlying the work in this thesis. As explicitly stated in our hypothesis in Section 1.1 we are specifically interested in improving our ability to predict from an early stage of development which components will be most affected by change-proneness.

We begin with a discussion of traceability and software dependencies in Section 2.1, since these are key issues underpinning our research. We include a survey of several existing schemes which may be employed to capture and structure requirements or design information in Section 2.2; we will return to these in Chapter 3. In Section 2.3 we summarise some previous studies which rely on mining the change logs for data to detect project dependencies. In Section 2.4 we examine previous attempts to predict volatility in software development, many of which rely on the development or reuse of metrics to assess software complexity and/or design. Again, we will be building further on these issues in Chapter 3 when we begin to develop our own metrics for including assessments of requirements and design activity.

2.1 Project dependencies and traceability

Our research is predicated on the notion that project artifacts are linked by inter-dependencies, and that the number and type of these inter-dependencies create recognisable patterns that in some cases may be correlated with patterns of change. There are several ways in which dependencies may affect system volatility:

- Dependencies between project entities may create pathways along which change can propagate (a 'ripple' effect).
- Looking at numbers of links between artefacts may identify those in key positions. A component's position may determine how frequently developers need to, or are willing to, modify it.

The Cognitive Dimensions Framework developed by Thomas Green [63, 62] points out how *hidden* dependencies affect the ability to modify an information system. This framework enables non-specialists to evaluate information-based artefacts, discussing facets of human-system interaction which should be considered. One facet is the presence of hidden dependencies, as these can result in unexpected propagation of changes between artefacts.

Green and Blackwell suggest that hidden dependencies arise from two sources: 'by being one-way or by being local' [63]. In reality, the complexity of information available to requirements engineers, designers, managers, testers, users, developers and maintainers during software development can lead to dependencies that are not necessarily hidden, but which are difficult to uncover,

in part because of stark differences in notation and granularity commonly seen in different phases of the development lifecycle[1]. Traceability is an important tool here, and in particular can tackle the problem of one-way dependencies by recording links which can be searched in 'both a forwards and backwards direction' (to quote a widely-accepted definition) [61].

Traceability is concerned with the relationships between objects (artefacts) created during the development process. Traceability originally was perceived simply as a tool for demonstrating to the customer that every requirement has been satisfied and that all parts of the system can be linked to a requirement, as described by Stehle[123]. It has since evolved, however, and by the mid-1990s many researchers were urging the importance of both pre- and post-requirements traceability (i.e., traceability between requirements and their the sources; and between requirements and artifacts developed subsequently) [61, 74, 120].

Kotonya and Sommerville [86] describe four types of traceability relationship (first classified by Davis):

- backward-from traceability: links between development objects (such as requirements) and their sources
- forward-from traceability: links development objects to subsequent, dependent artifacts (e.g., requirements to design components)
- backward-to traceability: for tracing all those artifacts resulting from a given source
- forward-to traceability: for tracing antecedents of a given artifact

Conceivably, these types of traceability may be given differing priorities, depending on the intended aims of a given traceability scheme.

2.1.1 Benefits of traceability

Traceability is touted as a cure for many ills. In addition to supporting impact analysis (see Section 2.3.1), depending on how it is implemented and practised, traceability can be used for:

- facilitating reuse (by enabling requirements of components to be compared to new systems) [120]
- identifying missing use cases [120]
- enabling the tracing of requirements and design rationale (avoiding the need to rework previous design decisions) [61, 120, 122]. Sources could include people [59]
- enabling process-based process improvement [120]
- linking test cases to requirements, design decisions and components [122]
- proving to the customer that all requirements are satisfied [4, 122]
- tracking project status [122]
- allowing inspection teams to understand design decisions made [120]
- identifying which parts of the design are required to be changed to produce a design variant [4]
- checking consistency of models [69].

[1] A software engineering textbook such as Ian Sommerville's *Software Engineering* [138] provides more details on software process methodologies.

2.1.2 Failure to benefit from traceability

Although many projects and companies have benefitted from extensive and very useful traceability schemes, relatively few projects successfully run fully traceable projects [61, 3]). There are many potential reasons:

- A completely traceable system requires the dutiful recording (and rigorous updating) of very many complex relationships between development artifacts. For example, Bohner [22] shows that a system with a non-trivial number of artifacts can quickly develop an unmanageable web of dependencies. Maintaining information on all potential relationships will be prohibitively time-consuming, and many project managers do not feel able to justify the cost [35]. For this reason, it is important to prioritise traceability needs [86]. The time commitment required means that traceability is frequently the first task to be squeezed from schedules [74].
- Technical staff may fear that the data will be used to apportion blame, or fail to register the importance of traceability, particularly when technical staff themselves are rarely the ones to benefit from traceability [3, 120]. Tellingly, developer resistance is considerably lessened on those projects where developers are able to make use of traceability information for their own benefit - for example, to control requirements creep [4].
- Stakeholders throughout a project have differing expectations from traceability systems and a different understanding of their purpose [3, 61, 122, 123].
- Some project staff dislike the clumsiness of entering traceability data into CASE tools and the way traceability data is handled [77].

Some researchers have concentrated on improving the efficiency and usability of traceability searches, replacing the traditional traceability matrices and lists (as described by [86]) with partially automated traceability systems, although a fully-automated system is unlikely to be possible due to the unstructured nature of the source information [120]. For example Cleland-Huang *et al* [37] propose a partially automated system employing event-based triggers and traceability information to manage evolutionary change.

2.2 Traceability structures and processes

Managing such an extensive quantity of data is no trivial task and requires both a cohesive process and suitable organising structures. Ramesh noticed that traceability is most effective in organisations where rich data structures and dynamically updating (often customised) tools are employed, and an organisation-wide methodology exists, supported by senior management who appreciate the importance of traceability [124, 123].

The choice of structure for modelling traceability data is important. In the next few sections we examine a range of systems for capturing and structuring data from requirements and design stages of the lifecycle. We refer back to these systems in Chapter 3, where we draw upon them for the development of our own metrics assessing requirements and design activity. The selection of schemes presented in Sections 2.2.2 and 2.2.3 is not comprehensive, and is intended to be a reasonably broad sample.

2.2.1 An example dataset

A series of systems for documenting and capturing data from the design process and/or requirements analysis is discussed below. An example of each is presented, using an example dataset

taken from a real-world project. The CARMEN project[2] aims to develop a collaboration tool and repository for neuroscience datasets and services, to be used by a widely geographically distributed community of neuroscience researchers [41]. Neuroscientists who are prepared to share their data with potential collaborators can upload data (or tools) to CARMEN, tagging them with metadata. Metadata serve to describe data uploads accurately and completely, thus enabling other neuroscientists to search for datasets and/or tools that meet their needs. In both cases the end-users (both uploaders and searchers) benefit from finding potential collaborators.

This case study is described in further detail in Chapter 4, but the case study modelled here is (superficially) a relatively simple one: the choice of data format(s) to be supported by the CARMEN system. During requirements elicitation the team ascertains the actual range of formats currently in use (or likely to be in use in the foreseeable future) and the end user's expectations for CARMEN's performance, interoperability and metadata. During the design phase the team explicitly consider a range of solutions in light of the data they have already gathered.

The decision itself is driven by the need to ensure that users are able to share and re-use the data stored by the CARMEN system. Neuroscience research presents a vast range of possible (and incompatible) data formats and potentially very large file sizes. Metadata support is important. Possible solutions considered by the team include selecting from amongst: providing support only for existing standards/formats used for neuroscience research; creating a new format to suit CARMEN's specific requirements; or leaving data in the original format and simply attaching metadata to describe it.

Arguments have been put forward to support or oppose these options. For example, leaving data in the original format has the advantage that a virtually unlimited set of formats can be uploaded to the system by researchers, and no converters are needed. However, CARMEN then cannot guarantee to users that they will be able to read or employ all data stored in the system, since formats used may differ substantially. Selecting an existing format may provide a platform giving access to new potential users who already use it, but inevitably there will be some types of data which cannot be converted into the requisite format.

The entire decision process was originally documented by CARMEN team members in free text, and is presented in a simplified form in the appropriate graphical notation for each scheme. Some of the following notations are more crowded than others, necessitating the omission of some information on some diagrams. However, the same basic scenario is worked through in each notation. The case study data is provided here merely to demonstrate how each scheme handles a complex requirements-modelling and design-making process.

2.2.2 Capturing requirements data

The first stage of substantial analytical work for many projects, requirements analysis is not a single activity but is composed of several disparate stages, frequently iterative in nature. The earliest stages of requirements analysis often orient around understanding the major driving force(s) behind the project, such as the key business objectives that make the system development necessary. Models may be constructed to illustrate and study how the relevant business areas currently operate, before any attempt is made to begin designing a new system to enhance/replace current processes. Once the system's environment and goals are understood, later requirements work begins to focus on producing the requirements statement.

[2]CARMEN stands for 'Code Analysis, Repository & Modelling for E-Neuroscience'. Work commenced 1st October 2006 and project duration is 4 years. http://www.carmen.org.uk/

Goal-Oriented Requirements Engineering

Goals are now recognised as a key concept for understanding the problem domain and driving requirements analysis forward. A goal is defined as 'an objective the system under consideration should achieve' [147]. Capturing the goals a system should meet provides an important insight ultimately into requirements rationale, as each goal should have a clear purpose which is obvious to both technical personnel and non-technical end-users. This helps to ensure that erroneous requirements are avoided [147]. Goals are particularly useful at a very early stage of analysis; they 'drive the identification of requirements to support them' [147], as well as providing the right level of abstraction for reasoning about solutions and varying options.

Many requirements engineering techniques exploit the notion of a goal. Perhaps the most well-known example of a model which is explicitly goal-oriented is the KAOS meta-model [45].

Non-Functional Requirements modelling

A particularly difficult area of requirements analysis is the uncovering, recording and specification of Non-Functional Requirements (NFRs or 'soft goals'). NFRs are generally related to quality aspects of the proposed system (e.g., 'the system will be reliable...') rather than with specific functionality it is required to provide. Often NFRs only become apparent at a relatively late stage of development, perhaps when users receive early versions of the system and realise that certain aspects of its performance or behaviour do not meet implicit expectations. Unlike functional requirements, NFRs tend to be assessed on a scale of measurement (e.g., responsiveness may be measured in numbers of seconds) rather than labelled 'completed' or 'uncompleted'. This makes advance specification - and later acceptance by users - difficult to manage as expectations of what constitutes acceptable compliance may be subjective.

Mylopoulos *et al* proposed a system for modelling NFRs more explicitly [111] in the late 1990s. This system explicitly models 'soft-goals' (NFRs), which can be decomposed into a series of sub-goals. Subgoals can be alternative options (i.e., sharing an 'OR' relationship) or complementary (sharing an 'AND' relationship). Relationships in the NFR framework carry semantic information: they indicate either a positive or negative contribution of a sub-goal towards an NFR, or how sub-goals can contradict or reinforce each other. Dependencies (e.g., where subgoals require all their sibling subgoals to be present) can be explicitly recorded. This system of contributing positively or negatively towards a goal echoes many systems which are used to model design decisions (see Section 2.2.3).

i*

i* is a system for modelling very early requirements analysis work, proposed by Eric Yu in the 1990s [155, 156], and as such it tends to present extremely high-level views of the system and organisational processes, rather than individual conflicts and issues. Models depicting current working practices are created and analysed for the effects on business practices of automating processes. There is an emphasis on softgoals and on the user's point of view: what does each participant hope to gain from the system, and how do they interact with it currently? Capturing participants' points of view is key for building a system to which users are happy to commit. As with NFR modelling, solutions can have either a positive or a negative contribution towards any soft-goals which have been identified.

Unlike other systems discussed here, relationships in i* show the dependencies between parties; each relationship thus consists of a depender and a dependee for each transaction. The roles may invert, so that a depender in one transaction becomes a dependee in another. Dependencies can

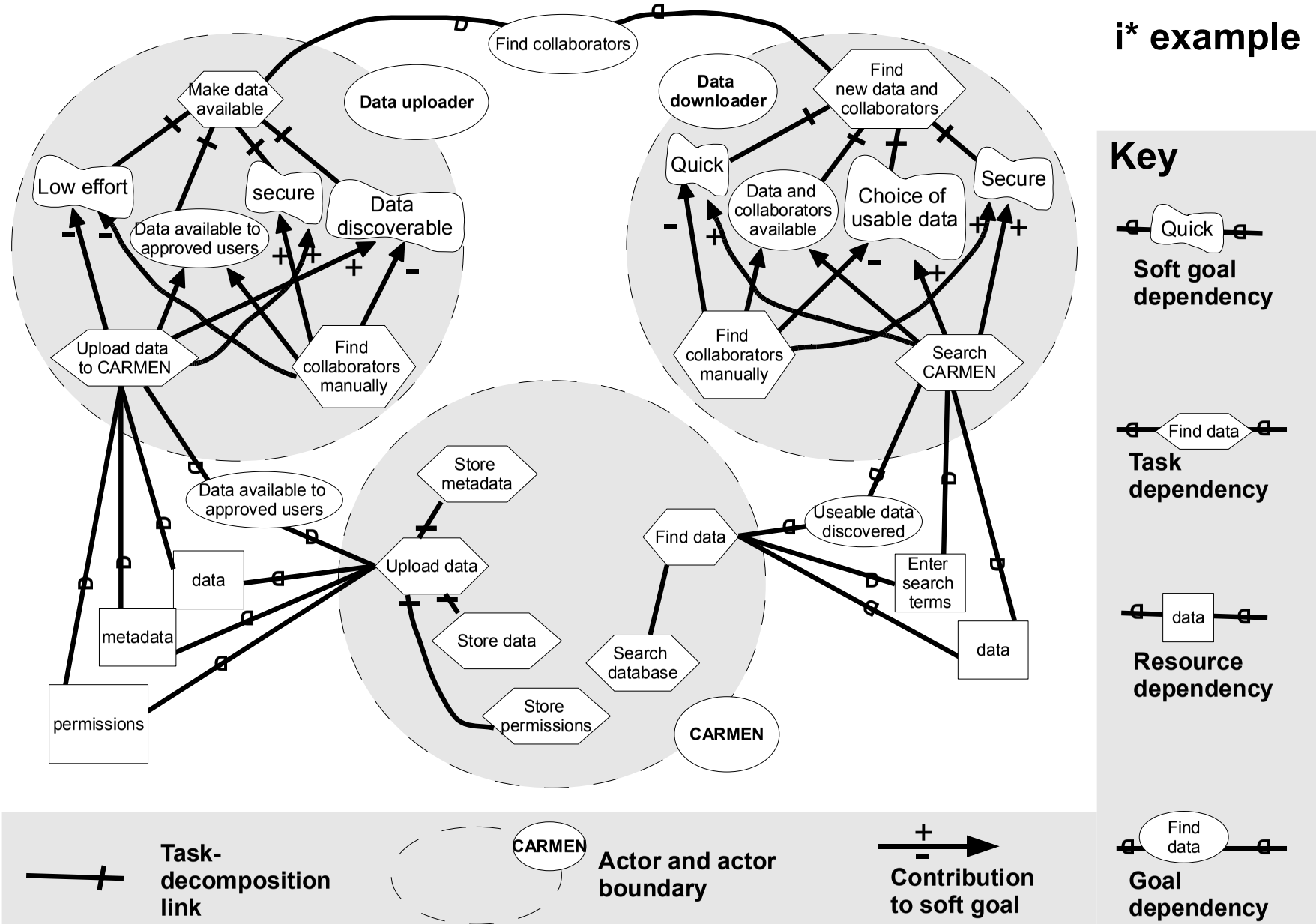

Figure 2.1: An example from the CARMEN case study presented in i* notation

be centred on resources, tasks, goals or soft-goals. Yu provides an example of a bank's processes, amongst which are a customer depending upon a branch manager for 'good service' [155], and a legal department in turn depending upon the branch manager for the goal of 'fundsRecovered' [155].

Figure 2.1 suggests how a high-level initial analysis of the CARMEN project might appear when presented in i* notation. There are two groups of users involved: uploaders, and downloaders. The motivation of both groups is to find a collaborator; each group has goals and soft goals to be satisfied. This particular diagram considers the respective contributions towards these goals/soft-goals made by two possible solutions: using an automated system to share data; or searching for collaborators manually[3]. Each of these solutions affects the range of soft goals differently. For example, for the uploaders, uploading data to an online repository carries some overheads. Large quantities of data may transfer slowly, and some work must be carried out by the uploader to ensure that all datasets are fully described with metadata. However, the chances that a collaborator will be found are increased, as the data, once uploaded, is easily discoverable by potential collaborators.

Contribution Structures

Research interest in capturing traceability links and rationale was particularly strong during the 1990s. A different approach was put forward by Gotel and Finkelstein in 1995 in contribution structures, described in [59] and extended in [60]. This model aimed to record the people behind key requirements decisions, with the aim of improving the traceability of key requirements rationale. The system differentiated between a *principal* (person who motivates decisions); an *author* (person who selects an artefact's form and semantics); and a *documentor* (person responsible for recording the artifact). Different status levels can be assigned with each contributor; for example, the artifact may have different approval stages with the principal, and the documentor can record different levels of confidence in the artefact's accuracy. Relationships captured by this system centred exclusively on levels of contribution towards an artifact.

Goal Structuring Notation (GSN)

The field of safety case construction has resulted in several systems which illustrate the steps taken to ensure that safety-critical goals have been fully considered (e.g., see also DRCS in Section 2.2.3) for an example of structures used to model safety cases from a design point of view). GSN is a system based on design patterns, intended for constructing safety cases ([80, 81]). The notation is not intended to 'prove' that safety goals have been met, but rather to improve the readability of safety cases.

Each safety case orients around a high-level, safety-related goal, which can be linked to alternative or complementary strategies for fulfilment. Strategies are linked to 'contexts', which may provide extra data needed to follow the safety case. Strategies deconstruct further, from higher-level strategies directly linked to goals, down to lower-level strategies, and finally solutions. There are notations for identifying goals which are yet to be developed. GSN does not present an explicit system for capturing or structuring possible negative aspects of the strategies. Which allows emphasis to fall upon what the solution should accomplish and establishing ranking between goals.

Inter-artifact relationships in GSN are dominated by relationships linking higher level strategies to lower-level ones, as goals and strategies are deconstructed. Unlike systems which model soft-goals, the links themselves do not carry any semantic information.

[3] A similar comparison between an automated system and a more manual one is presented by Yu in [156], in the form of a proposed system for scheduling meetings. The two alternatives considered on the part of the meeting initiator are 'schedule meeting' and 'let scheduler schedule meeting'.

2.2.3 Capturing design data

A varied collection of systems for modelling data generated during the design stage has been proposed. These commonly focus on the notion of an *issue, question* or *decision* which examines a range of possible solutions. Some schemes aim specifically to support the designer in their decision-making capacity by modelling relationships and dependencies between the options. Others focus largely on the importance of recording design rationale.

Whilst it is generally agreed that storing the details of decision rationale and any assumptions made is greatly beneficial for the project, capturing such data is not simple and is generally imperfect, if it is captured at all. Design rationale can be particularly useful for a project with a long duration. After a few years personnel may have left the project or simply not remember why a particular design was chosen, or which options were considered and the reasons for their rejection. Ramesh *et al* [122] cite an extreme example, of a project forced to back-hire engineers in order to reconstruct design rationale.

Data on decisions and their reasoning tends to be unstructured and informal and, due to different notations and levels of granularity, can be difficult to link accurately to previous or subsequent work. Rationale behind a decision may be straightforward, or it may be a network of trade-offs and judgement calls which does not translate well to a formal structure. Decisions may even sometimes be made implicitly, without the designer's realisation that other options also exist. Assumptions in particular can be be very difficult to identify and document, since by its very nature an assumption may not have been made consciously.

Some of the problems inherent in recording the design process may be alleviated by the adoption of a model which captures and presents design decisions, rationale and key traceability links in an understandable format. In the process of populating the models engineers are prompted for additional data such as supporting evidence or alternative versions. We survey here a broad range of representative design capture systems.

Questions, Options and Criteria

Questions, options and criteria (QOC) are the component elements of 'Design Space Analysis', as proposed by MacLean *et al* in 1991 [101]. For each decision, the system captures and records the range of options available and sub-decisions dependent upon the options. Figure 2.2 illustrates QOC using data taken from the CARMEN example. The options are considered in the light of 'criteria' attached to various options, which summarise its positive or negative aspects. MacLean *et al* argue that this places the focus on criteria [101], allowing for more considered selection of an appropriate solution. The criteria may be partially or fully ranked to aid the decision making process.

It has been noted that QOC is easy to use, particularly in reverse-engineering a completed or partially-completed system for future documentation [128]. Being a semi-formal, graphically-presented system of data presentation, QOC does aid the transition between informally recorded rationale and more formal data [128]. However, QOC does risk over-simplifying decisions. In reality, some design decisions do not present very neat trade-offs between, for example, an option which is intuitive but effort-intensive, and an option which is the converse (unintuitive but low-effort). Where a decision has many options, some may have negative impact on some criteria and others have a positive impact whilst others are (comparatively speaking) neutral, whilst some criteria may only apply to one option. This creates a messy diagram, and renders a simple ranking of criteria difficult.

QOC also lacks an ability to model separately any assumptions which criteria depend upon. For

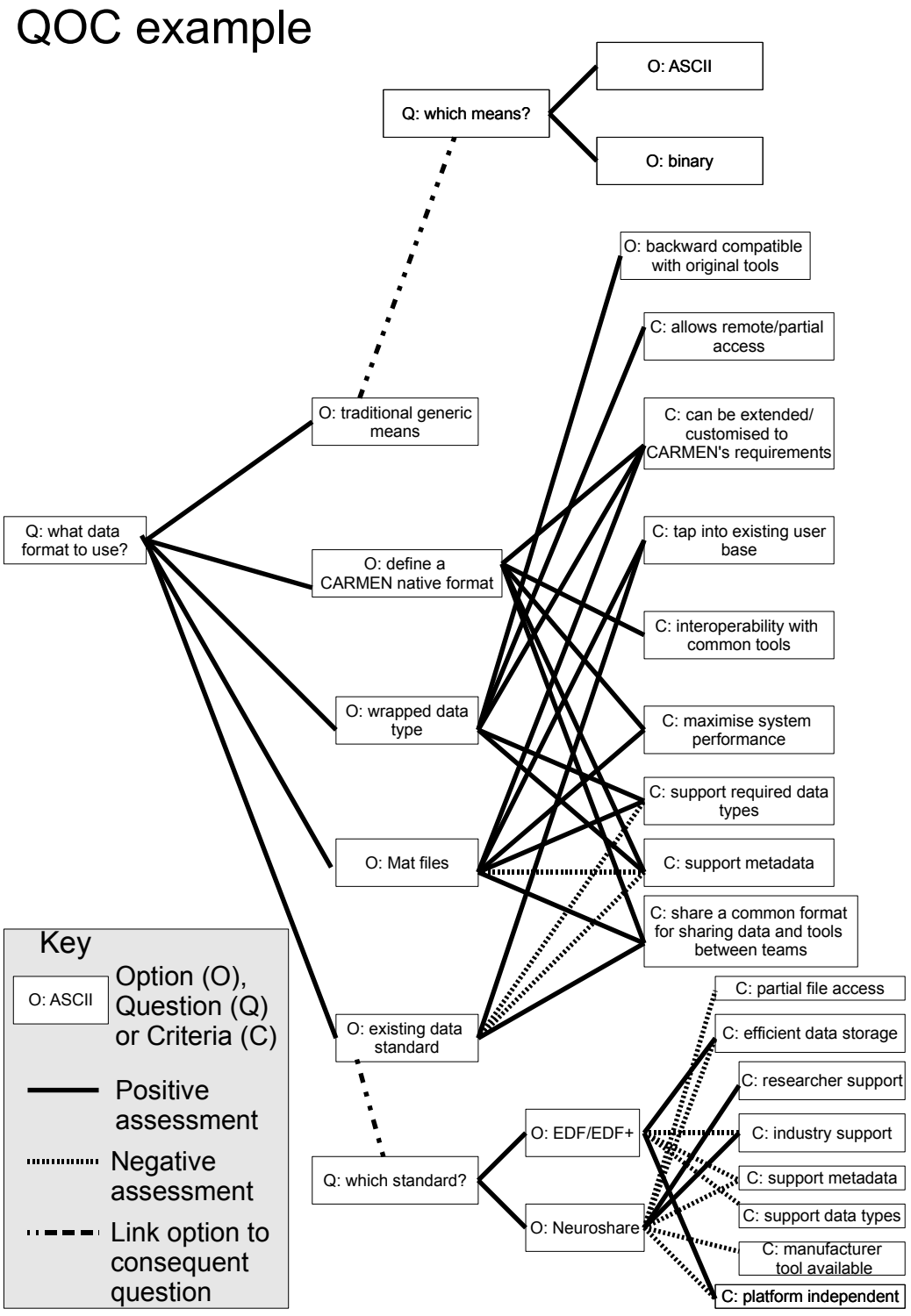

Figure 2.2: Diagram showing an example from the CARMEN case study presented in QOC notation

example, in CARMEN's case, a lack of industry support for a data format is significant. But this situation could change if new manufacturers release new software versions. Modelling assumptions such as this separately would greatly facilitate a reconsideration of the decision at a later date.

Redux

Redux is a system using classical AI planning techniques in a scheme for supporting a decision-making process, first suggested by Petrie [116]. Figure 2.3 shows an example of the CARMEN decision process presented through the Redux scheme. High level options in Redux can be further associated with lower-level goals and constraints (such as 'time' or 'budget'). Decisions are recorded, but if the status of an option changes in the future there is provision to 'backtrack' the decision, and re-plan if necessary. In an example, Petrie suggests that the option of flying between two cities is not possible due to budgetary constraints, although we may wish to revisit this decision if costs change. Redux bridges the step between informal project data (such as free

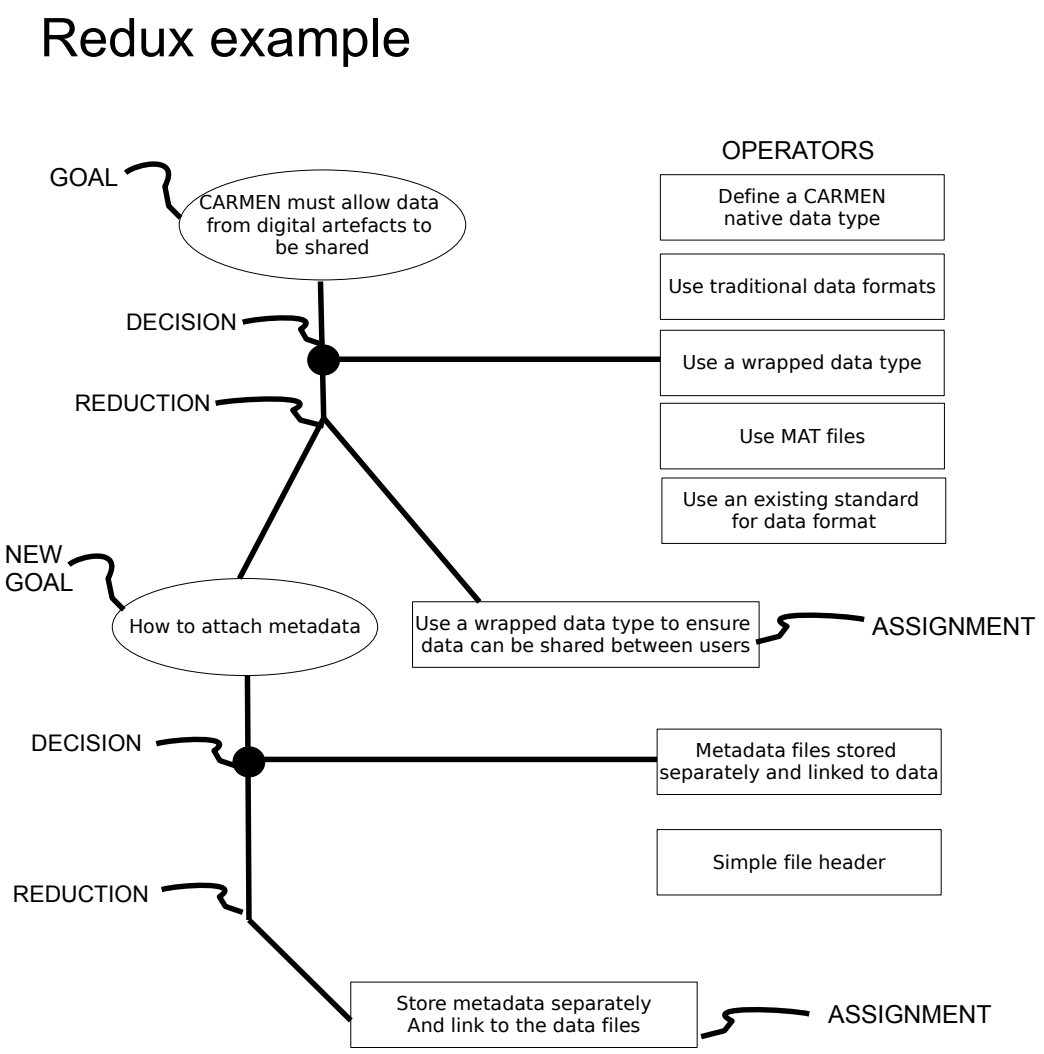

Figure 2.3: Diagram showing an example from the CARMEN case study presented in Redux notation

text describing an issue and its options) and a more formal presentation of arguments. Once data has been recorded, backtracking and replanning can be performed automatically [141]. Like QOC, IBIS and its extensions and DRCS (all discussed elsewhere in this chapter), Redux includes relationships to model the links between an issue/question/problem and a range of mutually-exclusive options. However, unlike these schemes, Redux has a slightly more complex approach, modelling a

decision as: a goal; operators (representing possible solutions); and the resulting assignment/new goal.

Issue Based Information Systems (IBIS)

IBIS is a system centred on *issues* and was proposed by Kunz and Rittel in 1970 [87] (see also [128]). Issues are linked to *positions* which respond to the issue, and are supported or objected to by *arguments*. IBIS itself has been extended and incorporated into other schemes quite extensively. KBDS, for example, merges IBIS with a representation of design history specifically for modelling decisions in a chemical plant [112]. KBDS could be translated to software decisions in principle, since both a physically engineered plant and a software system share KBDS's concepts of a design, areas of functionality, components and values. Design evolution over time is also common to both. However, KBDS's schema is a very specialised one (designed for reconfiguring equipment in a chemical plant) and it doesn't model other types of decision (such as our example) very well.

gIBIS [40] is a graphical browser, also based on IBIS. gIBIS presents a relatively easy to follow scheme, although unlike QOC it separates decisions from issues and introduces a link from the decision to any resulting requirements. This type of link is particularly helpful for documenting requirements rationale, making it clear the process of steps that lead to its inclusion.

REMAP is a well-known system which is also based on IBIS, proposed by Ramesh and Dhar [126]. Whilst IBIS provides 'primitives' for capturing information from the design process, it does not link these to the resultant products, or to the context surrounding supporting or opposing arguments [126]; the REMAP scheme is intended to extend IBIS correspondingly. Figure 2.4 shows a decision process from CARMEN presented using the REMAP scheme. Like IBIS, REMAP also features links from decisions back to requirements that are generated or modified as a result, as well as links from decisions to resulting constraints. A 'decision' is stored separately from the range of positions which may be considered, which allows an emphasis on the decision's effects on entities such as requirements, constraints and issues.

Ramesh and Jarke's reference model for requirements traceability

Redux is moving towards a more comprehensive picture of the early development stages, with the inclusion of goals and requirements as forces which drive design decisions. A scheme with a more incorporated view of the early phases of development can be found in the reference model for requirements traceability proposed by Ramesh and Jarke [125] in 2001. This model aims to incorporate high-level elements from requirements and design stages into a comprehensive framework modelling key, early development traceability links. REMAP forms an important pre-cursor to this framework (they share an author in Ramesh). The framework itself is presented in a series of partitioned submodels:

- A 'low-end' traceability model is a simple structure illustrating links commonly captured by organisations with a 'low-end', simple traceability approach. (Other submodels model data captured by organisations with a more sophisticated approach.)

- A requirements traceability submodel details high level needs and objectives, user scenarios, constraints, mandates (such as standards to meet) and requirements themselves.

- A rationale submodel details decisions with their rationale, issues and/or conflicts and arguments and assumptions that interact with them.

Figure 2.4: Diagram showing an example from the CARMEN case study presented in REMAP notation

- A design allocation submodel details how requirements, designs, areas of functionality and resources are allocated to system components, which might themselves link to external systems.

- A compliance verification submodel illustrates dependencies between Compliance Verification Procedures, the 'mandates' (standards, policies and methods) upon which they are based, the requirements they were developed for and the change proposals they generate. Our work concentrates on the data available at the design phase (we will discuss this in Chapter 4), and excludes dependencies to/from entities created as part of a testing/compliance phase.

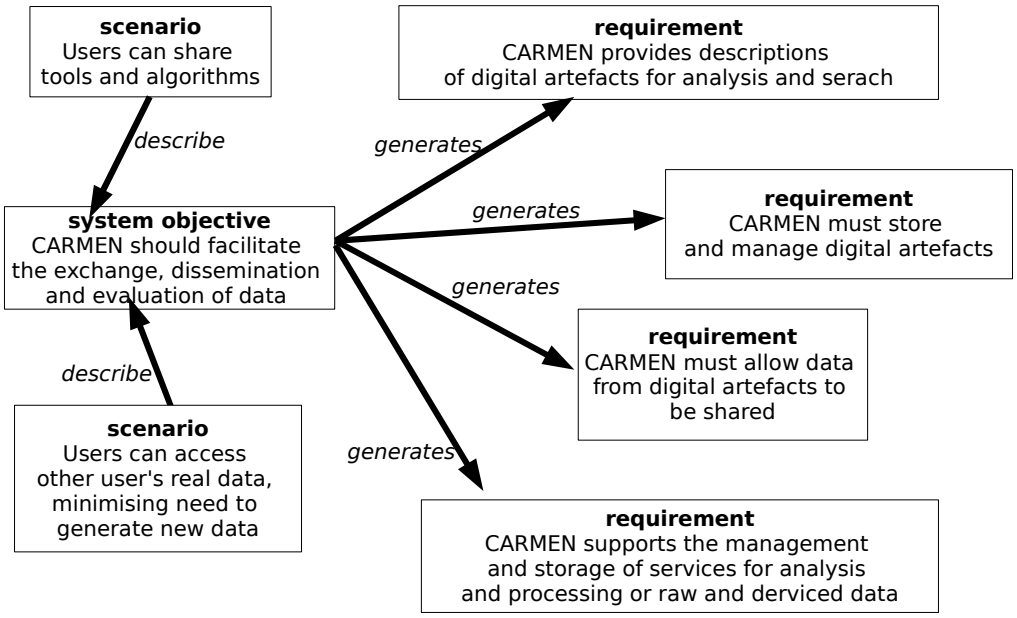

Figure 2.5: Diagram showing an example from the CARMEN case study presented in Ramesh and Jarke's reference framework (requirements submodel)

Figures 2.5, 2.6 and 2.7 show the example drawn from CARMEN presented using the reference model's structures. This is a very flexible model which can be adapted to many situations and problem domains. There is a strong focus on recording rationale and, in particular, identifying key business reasons for the system being modelled, in a goal-oriented approach. For example, there is a high-level structure of operational needs and/or strategic needs which justify key system objectives, which in turn generate requirements. Scenarios are also used to support the case for requirements (via the link 'scenario *describes* requirement'). Change proposals which might affect requirements are also linked to system objectives. In the 'allocation' submodel, requirements can be allocated to components and linked separately to functionality which is to be implemented. Thus all areas of the system are fully justified by a key business goal.

Design rationale is also thoroughly explored in this model. Decisions are linked to a range of alternatives, each of which can be supported or opposed by arguments. The rationale behind a decision is captured separately, unlike any of the other design structures discussed here which generally expect readers to construct the actual rationale behind a decision from an examination of the arguments for/against the presented alternatives. Assumptions underlying each or any of the arguments, decisions, requirements, designs, components or rationale are explicitly captured.

Whilst it is comprehensive and very thorough, this reference model is potentially too complex for some situations. Some relationships and/or entities preserved in the framework are perhaps redundant. For example, it is not clear what the difference is between the 'create' relationship and the 'define' relationship, both of which link designs to components. Similarly the relationships 'depends on' and 'based on', both of which link decisions with rationale, perhaps overlap somewhat. It is important to remember that a framework is intended to fit in with a variety of other schemes, however, so some flexibility and overlap is expected. The links connecting requirements to areas of functionality and/or system components are potentially a little over-simplifying, on the other hand; the presence of a 'requirement R *allocated_to* system component C' or 'functions F *address* requirement R' relationship might be assumed to mean that R has been *fully* addressed by F or allocated *in entirety* to C. In practice, it is not unusual for one requirement to be implemented by the interworking of several system components. This problem can generally by alleviated by ensuring that requirements are decomposed to a relatively fine-grained, low level, although it is still possible that several components are needed to implement what amounts to a single low-lying area of functionality.

Design Rationale Capture System

An intuitive and flexible example of a design rationale scheme can be found in the Design Rationale Capture System (DRCS) [85]. Like the framework described above, DRCS presents a more holistic picture of the design process, although it is somewhat weaker on the requirements aspects than Ramesh and Jarke's framework.

The scheme was originally proposed as a tool to support the preparation of safety cases for a safety-critical engineering system, and as such its focus is on creating easily understood structures of logical arguments. Unlike the other rationale capture systems discussed above (excepting the similarly-expansive model proposed by Ramesh and Jarke), a model of the system's structure is included along with design decisions and their rationale.

DRCS outlines a series of overlapping data structures to represent various design artifacts, and the relationships between them. Systems are represented as a series of entities, allowing personnel to assert claims about those entities; claims can represent rationale to support the entities or can, for example, rank entities in terms of importance or describe interrelationships between them. The substructures include:

- the synthesis component defines project artifacts and plans;
- the evaluation model structures information on design aims and how well these have been achieved;
- the intent component connects assertions and design actions with high level strategies;
- the 'versions' model allows designers to document alternative designs which have been considered;
- and the 'argumentation' model allows designers to document evidence supporting (or disqualifying) claims.

Figure 2.8 shows the CARMEN decision presented via a DRCS traceability structure. The resulting diagram is an intuitive but flexible notation. DRCS's strength (unsurprisingly) lies in its simple representation of the design process. The emphasis is on presenting a series of options (like most design rationale representations) linked to a decision problem. Decision problems are raised by 'assertions'; since 'assertion' does not appear anywhere else on the DRCS structures we

Figure 2.6: Diagram showing an example from the CARMEN case study presented in Ramesh and Jarke's reference framework (rationale submodel)

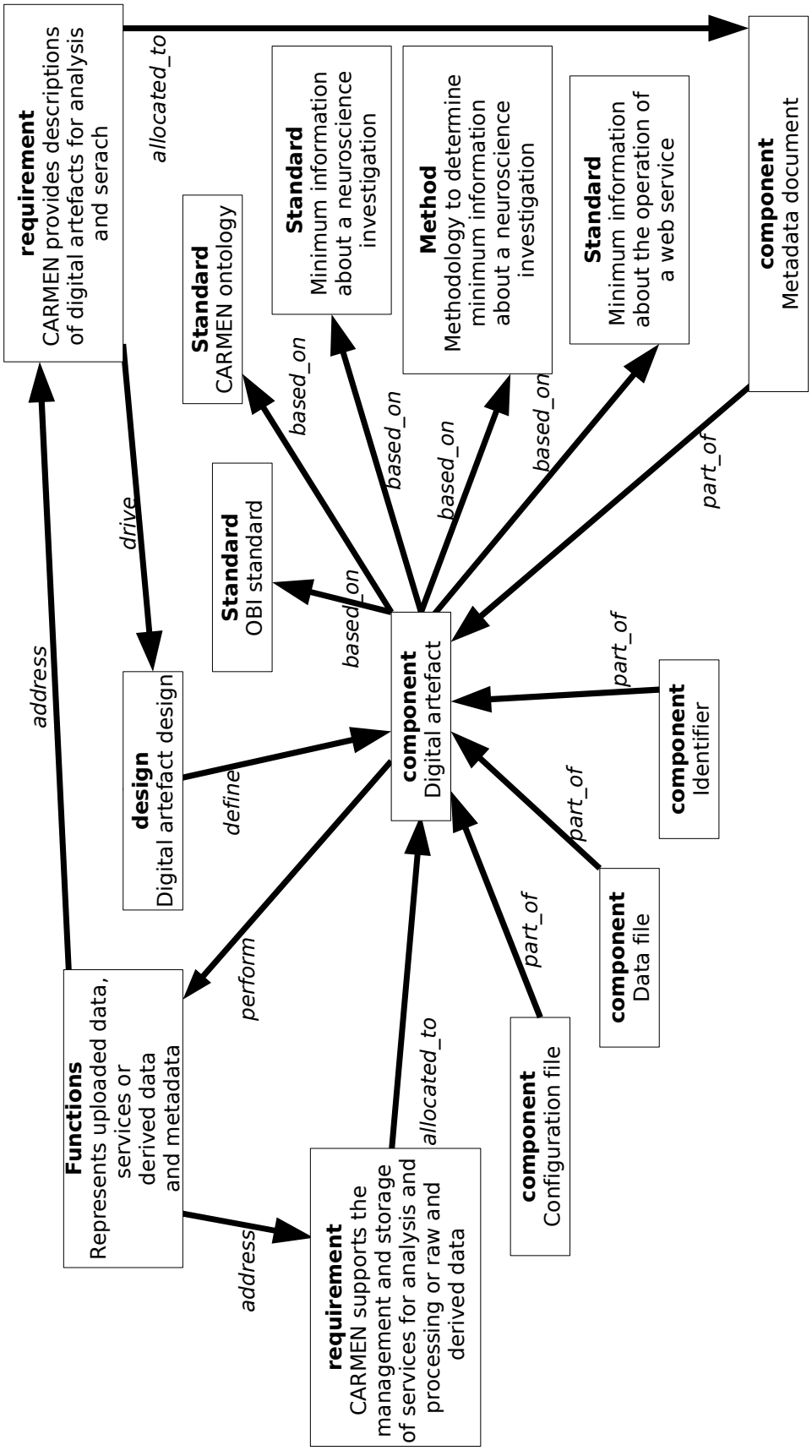

Figure 2.7: Diagram showing an example from the CARMEN case study presented in Ramesh and Jarke's reference framework (allocation submodel)

Figure 2.8: Diagram showing an example from the CARMEN case study presented in DRCS notation

assume that this can represent any other entity which can logically be linked. The different versions considered as solutions can be ranked explicitly using the *has-priority* and *has-greater-priority-than* relationships. Versions may also be explicitly linked to a status such as 'abandoned', 'suspended', or 'conflict' (which in turn may be linked to another version by the *resolved-by* relationship). Versions can be linked to claims that support, qualify, deny or presuppose them, and claims in turn may raise questions or provide answers to them.

Unlike the other systems examined above, DRCS also includes a planning structure. Decision problems can be associated with high-level plans via the *has-strategy* relationship, whilst components (modules) are linked to plans via *has-plan*.

The components themselves are modelled as modules which have attributes. Specifications and values can constrain the attributes. Some of the entities of the artifact synthesis structure are less relevant to software than to mechanical engineering contexts. Klein's worked example illustrates an aeroplane. The body has the interface *body mount*, whilst the wing has the interface *wing mount*, leaving the relationship: wing mount *is-connected-to* body mount. These features (module connections, module interfaces) do not translate easily to a software engineering system, however[4].

Unlike several other models discussed here, DRCS is not an attempt to bridge the gap between free text generated by users and early requirements analysis and a more formal notation. However, as a method for piecing together arguments about the design decisions which are both easily followed and easily checked, its flexibility is a key asset. DRCS can be extended to suit other problem domains (for example, a software engineering project rather than a mechanical one) by adding new entities, or introducing new names for relationships between entities.

DRCS does have some weaker areas. Like the earlier framework proposed by Ramesh and Jarke, some of the relationships presented in the structures seem redundant, and some of the entities are not clearly defined. For example, there is no apparent link between the argumentation structure and the other DRCS structures. Although design rationale is thoroughly explored (in the argumentation and versions structures), and the structure of components themselves (in the artifact synthesis structure) linkages between the two are not well developed, and rationale that drives decisions is largely absent.

In summary, there is a wide range of modelling structures available for structuring data gathered during the requirements and design stages of a project. In the requirements elicitation phase schemes are not intended to be used in isolation, but are often aimed at supporting different activities and several may be employed on the same project. Data structures and models play an important role as they help personnel determine what data need to be collected to complete their task. We return to look at these modelling structures in Chapter 3, as we draw from them to create new models for our own purpose.

In the next sections we move on to look at studies examining data collected from projects through mining the change repository.

2.3 Mining the change repository for historical data

Traceability is concerned with representing dependencies between project entities. Some dependencies may be difficult to capture, however, if project staff are not explicitly aware that they exist. Several researchers have taken an interest in uncovering dependencies that might otherwise be missed by developers. One such approach begins with mining the rich data on project evolution from records maintained by change control systems such as Subversion [145] or CVS [121].

[4]Whilst 'interfaces' exist in object-oriented languages in Java they do not perform the same function.

At the file level, an interesting method for uncovering dependencies between files involves determining whether pairs of files tend to be checked in to the source control at the same time. Files checked in together may have been modified in response to the same change request, indicating a dependency or at least a common area of interest. A suggested list of potentially co-impacted files (based on check-in history) can then be presented to the maintenance developer working on a given file. This approach has a distinct advantage that the automated system does not need to understand the nature of the change, simply that it was made to a particular file, and that historically other files have tended to change with that file. Issues to consider when developing such a technique include: setting a minimum threshold so that files which have on occasion coincidentally changed together in the past are not classed with files which regularly change together; and the statistical method used to detect a correlation or a probability of co-changing.

Bevan and Whitehead [17], for example, use static dependence graphs to identify co-changing files, using a minimum threshold to ensure that coincidental groups of co-evolutions are excluded. On a similar theme, Hassan and Holt [68] use a variety of data sources (such as changes made by the same developer or changes made at the same time) to predict co-evolving files, whilst Shirabad *et al* [137] suggest use of an induction algorithm on mined data. Xing and Stroulia [153] model a software development as a series of UML class models, employing Apriori rule association to elicit co-evolving files. Zimmerman *et al* [158] have developed a tool to recommend possible dependencies to a software developer modifying a file; the tool is fine-grained enough to break files down into shorter entities such as methods. Bouktif *et al* [23] use Dynamic Time Warping, a method used in pattern recognition, to discover co-evolving files, whilst German [57] shows that modification coupling graphs can be employed to visualise files modified together and Ball *et al* [5] use cluster graphs, employing metrics derived from version control history to develop a probability that two classes are modified together.

Many of these techniques do not discriminate in terms of type of file returned to the developer. Often no attempt is made to uncover the nature of the hidden dependency (if any) between the entities; any such investigation is left to the developer. The aim of these techniques is to narrow the maintenance developer's search by recommending 'most likely' candidates for inspection. Our research has a different focus in that we aim to predict change using data from an early stage of development where a sufficient history of co-evolving files has not yet been amassed.

A system with a similar aim but no attempt to mine the change history is proposed by Cleland-Huang *et al* [36] in the 'Event-Based Traceability' system. This application uses a rule-based system to identify an event as any one of a series of potential atomic changes to a requirements set. If a requirements change is detected, appropriate notifications can then be forwarded to subscribers who have an interest in a particular requirement's evolution. The system aims to improve intra-project communication and to reduce the risk that different, dependent components diverge in consistency.

2.3.1 Impact analysis

Impact analysis and effort prediction are heavily underpinned by traceability. Impact analysis is concerned with estimating the areas of the project which are likely to be impacted by a given change, and may be employed at any time in the project. This overlaps to some extent with the work outlined above, but the two applications are not entirely coincident. Impact analysis and/or effort prediction may be called for during the initial development (such as during the release planning stage [97]), such that the change repository may not yet be populated with sufficient historical data to make recommendations using the techniques outlined in Section 2.3. The techniques outlined above recommend files to be inspected when a known file is modified. Impact analysis generally

centres on identifying which files which change when some *requirement* is altered, so identifying links between requirements and final components is key.

Effort prediction obviously overlaps with attempts to predict which components are involved in a change. Effort can be calculated using information on the relative complexity and/or size of the involved components, or by comparing the components and change type to archived change/effort data. A brief summary of some extant techniques for predicting effort is provided in [136]. Previous work has also found that each change carried out bears an overhead (of understanding the problem, locating the relevant code, updating documentation, etc.) such that productivity declines for smaller changes (productivity was assessed using SLOC altered per person day and the total size of the change by total SLOC altered) [70]. Our work is more closely aligned with impact prediction (identifying high-risk files or components) than effort estimation.

Ideally an impact analysis would be able to predict automatically the likelihood of a change propagating from file to file. Hence the importance of traceability: many impact analysis techniques rely on a record of the dependencies between files as part of a general algorithm for calculating where change is likely to propagate. Bohner has shown that simply recording the presence of a link is insufficient as the number of links can quickly become unwieldy and unproductive [22]. Adding semantic information to the link (for example, giving it a named 'type', recording a probability that it may propagate change, noting its direction and/or its level of indirection) can help to keep the focus on the most important dependencies or relationships. Prediction models such as those described in Section 2.4 rely on this data.

Many impact assessment schemes involve estimating and representing the impact in at least two dimensions. For example, the range of the impact (e.g., the quantity of artifacts potentially affected) may be calculated alongside the criticality of the proposed change (ie, how critical it is that errors are not introduced). This allows differentiation between a cosmetic change affecting many components, and a change to a single critical code module. Other multi-dimensional models are available: Chapin *et al* map their definitions of change types against both the impact on the software and impact on the business process [28], whilst other researchers such as Henry and Henry take a number of factors into account when deriving a combined 'impact' value [70]. Chaumun *et al* consider the type of change and the type of links a class shares; the underlying assumption that a specific type of change (eg, altering the scope of a method) is likely to be propagated along certain types of links only. The total value of an impact is normally presented as a combined value reflecting both of these factors.

Clarkson *et al* [34] outline a detailed change prediction method, identifying components which are the most influential or susceptible to change. They draw up a matrix of change dependencies within a subsystem and calculate a 'risk' of change for each component; 'risk' is the product of the likelihood of a change propagating to neighbouring components and the average proportion of re-work for a change. Data is derived from archived data where available and/or the views of experienced designers. The system was originally proposed for mechanical engineering projects but could be extended to software systems. Eliciting accurate 'expert' knowledge could be challenging on a complex software project, however, where hidden dependencies are more likely to exist than on a mechanical engineering project.

Cleland-Huang *et al* [37] have proposed a framework for managing traceability amongst Non-Functional Requirements (NFRs). The framework models a series of subgoals which contribute positively or negatively towards the satisfaction of the NFR. Traceability links are extracted automatically using a probabilistic network to analyse vocabulary links from documents. Once the links are established, related components and goals can be identified using search algorithms. Many authors attribute many projects' failure to benefit from traceability fully to lack of time and under-

standing amongst project personnel, so automating these processes potentially has a great impact on success rates.

2.4 Change prediction models

Change prediction models, unlike impact analysis models and co-changing file predictions, do not try to predict the response to a specific change but instead aim to characterise some feature of system evolution. This area of research involves identifying system features and characteristics which can be associated with a high level of volatility or stability.

Lam and Shankaraman [89] propose a Change Maturity Model (mirroring the well-known Capability Maturity Model[5]) for improving techniques for handling changes. One facet of the model emphasises facilitating change estimation, by taking into account the size of a proposed change and its type; the current development stage; developer ability; and archived information on the deviation between previous estimates of effort and the actual effort involved. The model does not go into detail on the practicalities of the generic change process itself; low-level details are left for individual organisations to resolve.

More detailed change prediction models, used for impact analysis (discussed in Section 2.3.1) and change prediction, take into account hidden dependencies which are recorded using traceability techniques. Our own research aims to discover whether the types of cross-phase relationships recorded by traceability schemes can be used to improve change prediction.

2.4.1 Software metrics

Software metrics are concerned with quantifying some characteristic of a system or component for purposes of analysis and/or comparison[6]. If a metric (or combination of metrics) could be correlated with a high degree of change it could be extracted automatically and used to predict the anticipated volatility of individual components of the system. Mens and Demeyer [107] briefly outline possible uses for metrics in software evolution, dividing predictive research in this field into three areas:

- assessing which parts of a system will need to be evolved
- assessing which parts of a system are particularly prone to changes
- assessing which parts of a system will be particularly sensitive to changes (for example, areas which could create problems).

Our work fits squarely into the second category. Mens and Demayer note that coupling and cohesion metrics, which can be used for this purpose amongst others, are areas of contention in the literature, with several proposed metrics suites suggested [107]. We outline below a selection of some of the more-widely used metrics.

Measuring software is not a straightforward task. Measurement can be defined as mapping attributes of empirical objects (such as objects in the real world, or, in our case, software) to a formalised representation [160]. For physical objects measurement is much more intuitive - since obvious dimensions spring to mind - than for software. This has led to a plethora of differing schemes to assess varying dimensions of the software. Traditional software metrics employed for

[5]For an explanation of the Capability Maturity Model see a standard requirements or software engineering textbook, such as [100] or [138].

[6]Definitions of metrics mentioned in this chapter can be found in the glossary.

procedural systems include measures of complexity (such as McCabe's cyclomatic complexity measure [105] and size (such as the widely-used *lines of code* (LOC, sometimes referred to as *source lines of code* - SLOC) measure [55]). LOC as a measure of size has received some criticism, partly because of a lack of clear agreement on its definition. Lehman argues for the use of number of modules as a size metric rather than SLOC, noting that LOC is particularly sensitive to individual developer habits whilst the number of modules is generally less so [90].

As the object-oriented programming paradigm gained in popularity, new metrics more suited to describing non-procedural software have been developed. For example, a suite of metrics suitable for object-oriented systems was proposed by Chidamber and Kemerer in 1994 [31]. These metrics are termed 'design' metrics, since much of the information describes system structure and/or complexity and can in many cases be calculated from reasonably detailed designs. Six static code measures describing different aspects of a system's complexity and structure are proposed by Chidamber and Kemerer (hereafter referred to as C&K metrics):

- Weighted Methods Per Class (WMC). A count of the number of methods in a class, which can be optionally weighted by another value if necessary.
- Depth of Inheritance Tree (DIT). Number of ancestor classes.
- Number of Children (NOC).
- Coupling Between Objects (CBO). 'Coupling' here measures the relationships created when objects of one class act on another, for example, accessing a member variable or method.
- Response for a Class (RFC). This is a set of all methods which 'can potentially be executed in response to a message received by an object of that class' [31].
- Lack of Cohesion of Methods (LCOM) is designed to assess the commonality of design of a class. LCOM is 'a count of the number of method pairs whose similarity is 0... minus the count of method pairs whose similarity is not zero' [31]. 'Similarity' here is assessed by use of a shared instance variable between a pair of methods. LCOM will be high if methods do not share instance variables (suggesting that they have separate, non-overlapping functionality and perhaps do not belong in a class together) and low if methods operate on the same instance variables.

Calculating values for the various metrics are not necessarily straightforward. For example, as Churcher and Shepperd point out, when counting the methods in a class, should inherited methods be included? These will presumably be included in any similar count for the parent class where they are defined [33]. Should both visible and hidden methods be included [33]? There are potentially a series of assumptions and variations involved so authors re-employing metrics for their own study need to make it clear how they have calculated metrics.

In their original proposal, Chidamber and Kemerer discuss the implications of each metric, suggesting that, for example, a high LCOM increases complexity and may result in a higher rate of errors, or that a large response set can increase complexity, faults and testing time [31]. Another article [44] briefly summarises how these metrics may be put to practical use, summarising a number of studies using C&K metrics to evaluate designs or maintainability. We discuss similar studies in Section 2.4.4.

Abreu *et al* also propose a suite of software-based design metrics (the MOOD set of metrics) [48], consisting of:

- Attribute hiding factor. The percentage of attributes which are private or protected, out of all attributes in a class.

- Method hiding factor. The percentage of methods which are private or protected, out of all methods in a class.
- Method inheritance. The percentage of methods inherited, out of all methods in a class.
- Attribute inheritance. The percentage of attributes inherited, out of all attributes in a class.
- Polymorphism factor. The percentage of overriding methods out of the number of new methods in a class when it is multiplied by the number of direct descendants of a class.
- Coupling factor. The number of links which are not internal and not inherited, out of all possible inter-class couplings minus the maximum number of couplings due to inheritance. For each each pair of classes, this is only counted once, even if there are many links between the same two classes.

The authors examined a series of applications in the public domain which they considered were likely to exhibit excellent design features. The metrics described above were extracted for all systems and the possible implications of extreme values for each metric are hypothesised, although, as of yet, the hypotheses have not been compared to projects which are not considered to be well-designed (and the assumptions of good or poor design in these systems are not compared to measures such as understandability, testability, etc.). Theoretical validation of these metrics was carried out by Harrison *et al* [67]; see Section 2.5 for more on validation of metrics.

Unlike the C&K metrics, the MOOD metrics describe properties of the system as a whole rather than features of individual classes. As Harrison *et al* point out, the C&K metrics and the MOOD metrics can be seen as complementary for this reason.

Li and Henry [94] used the C&K metrics but replaced the CBO measure. They identify three possible means of inter-class communication which could be measured:

- Coupling through inheritance.
- Message Passing Coupling (MPC). The number of send statements in a class.
- Data Abstraction Coupling (DAC). The number of abstract data types (ADTs) defined in a class. The authors note that 'A class can be viewed as an implementation of an ADT'[94].

They note, however, that coupling through inheritance is already assessed through the DIT and NOC measures. This leaves two new metrics to replace CBO: MPC and DAC[7]. In addition, three other metrics suggested by Li and Henry:

- SIZE1. The number of semicolons in a class
- SIZE2. The number of attributes + number of methods in a class
- NOM. The number of local methods in a class.

Like C&K metrics, L&H metrics describe class features rather than system properties. Studies conducted to assess the predictive powers of C&K, L&H and MOOD metrics are summarised in Section 2.4.4.

Some metrics aim to quantify system properties at an even earlier stage in development. For example, Loconsole and Börstler summarise some metrics used to characterise UML diagrams [98], whilst Piattini *et al* develop some metrics to describe entity-relationship diagrams [117]. Some early phase metrics are discussed in Section 2.4.5.

[7]The combination of C&K metrics (with CBO replaced) plus new measures proposed by Li and Henry are hereafter referred to as L&H metrics.

A framework to unify the plethora of proposed coupling metrics is outlined by Briand *et al* [25]. The framework summarises important points to consider in choosing a suitable metric:

- deciding what constitutes a coupling relationship
- inclusion of both/either import and export relationships
- granularity
- stability of the server class
- inclusion of indirect relationships
- whether inheritance relationships should be considered/distinguished from other links.

Answers to these questions, conclude Briand *et al*, are not clear cut for every situation, even after considering a body of empirical case studies [25].

Metrics for characterising software are therefore a well-charted field, but techniques for assessing change and maintenance are more difficult. The next section summarises some potential methods of assessing maintenance and volatility.

2.4.2 Predicting requirements volatility

Volatility in requirements can be difficult to predict and relatively few empirical studies have been conducted. This is partly because requirements volatility is dependent on a very individual combination of development environment and culture, staff and user experiences and political, legal and business pressures.

Harker and Eason developed a classification scheme for describing varying states of requirements stability [66]. Within this scheme, requirements can be either stable or changing; changing requirements are further subdivided into types: mutable; emergent; consequential; adaptive; and migration. Classifying requirements under this scheme helps to identify those which are known (from past experience) to be at risk of changing. The authors suggest some strategies for handling or adapting to the different types of changes.

An empirical study by Nurmuliani *et al* examined change request (CR) forms completed on an industrial case project [114]. A taxonomy of change was developed from the data provided on the forms.

Strens and Sugden [143] have published proposals for a framework which captures and handles risks of requirements volatility. The framework requires identifying requirements which could change, beginning with identifying factors to which requirements may be sensitive. The sensitivity of each requirement to these factors is examined. Some areas are known to be high-risk areas already - such as areas affecting machine interface or speed- and time-critical components.

The lack of readily-available metrics for assessing requirements volatility is an issue. Nurmuliani *et al* express volatility as the ratio of requirements changed to the total number of requirements within a time period [114]. A range of metrics from available literature is briefly summarised by Zhang *et al* [157], whilst the authors themselves propose the measure 'mean time between enhancements' (MTBE) as a means of characterising volatility. Statistical analysis on archived data from an industrial COBOL project confirms the hypothesis that the independent variables of complexity (assessed by cyclomatic measurements), complexity per SLOC and system age differ according to the MTBE [157]. Higher levels of complexity are found to be correlated with higher levels of software volatility in the same study. However, this study focuses on non object-oriented

legacy software development; this limits the ability to generalise to more recent systems, which have a very difficult structure and complexity.

Stark *et al* [140] define requirements volatility more formally as:

$$\text{Requirement Volatility} = 100 \frac{\text{Added+Deleted+Changed}}{\text{\#of Requirements in VCN}}$$

where *VCN* is the version content notice (a document containing the agreed requirements); and *Added, Deleted* and *Changed* refer to requirements added, removed or altered, respectively, after the initial release of the VCN. The authors have developed a regression model for forecasting the actual schedule duration using the expression:

$$\text{Percentage Planned Schedule} = 0.97 + 0.41\sqrt{\text{Volatility}} + 0.23 \text{ Risk}$$

Risk is defined as the number of requirements delivered by type divided by the staff days charged to the type of requirement. Whilst our research overlaps with requirements volatility, however, we are not measuring requirements volatility itself, but looking for detectable patterns associated with changes in system components (which may be motivated by requirements changes).

2.4.3 Measuring maintainability

Defining 'maintainability' is of enormous interest to researchers as well as industrial teams. Change accounts for such a large portion of development resources that even small improvements in change management and implementation can lead to a noticeable gains[8]. Maintainability has been defined as a collection of desirable qualities concerned with ease of comprehension, avoidance of code scattering and improved testing. These are desirable characteristics of software throughout its life, and are not restricted to the maintenance phase alone. ISO/IEC9126 (addressing standards for software quality) breaks down 'software maintenance' into sub-goals such as: understandability; simplicity; analysability; modifiability; stability; and testability [117]. Such qualities are thought to expedite the process of understanding, implementing and testing a change without introducing new faults to the system. Measuring them is not straightforward, however, and studying sub-goals such as 'simplicity' or 'understandability' can potentially lead towards studies that focus on a surrogate measure (such as whether users can understand a diagrammatic representation more easily) rather than on how that measure can contribute towards improving the actual maintenance experience itself.

A selection of possible size and maintenance metrics were suggested by Swanson [144] and by Lehman and Belady in the 1970s, including: number of instructions/modules per release and average instructions per release [12]; and number of modules handled or changes made, cumulatively, per day or as a fraction [93]. Such metrics were largely concerned with characterising the evolutionary process itself. Early work using these types of metrics lead to revelations which are now widely accepted as common knowledge, such as the inevitability of change and code decay. More recent work has tended to focus on identifying particularly change- or fault-prone areas of a system. For example, several studies focus on the potential impact value of a class by calculating the number of components a given change made to class C/method M may propagate to [78, 29].

Many studies have analysed 'change', which is sometimes quantified as the number of lines deleted/added per class [94, 146, 2]. Lines added/deleted per class can be extracted automatically using a tool such as *diff* to compare two archived versions of the system. However, it is important

[8]The impact of change on industrial projects has variously been estimated at anything between 40 and 90% of the total project cost [76, 92, 36, 88, 14].

to realise that such tools are not always accurate, and tend to count a single modified line as a line deleted and a new line added.

We discuss our approach to change measurement in Chapter 4.

2.4.4 Change prediction using software metrics

A number of researchers have searched for correlations between software metrics and quality measures such as the number of changes made or errors detected. The number of errors detected and the quantity of changes made are not entirely unrelated, since the detection of a fault usually results in a change to resolve it, although there are many other sources of change also. We include here brief summaries of studies examining relationships between complexity metrics and either maintainability or fault-proneness (we assume that a correlation detected between a metric and fault-proneness is worth examining further for those with interests in predicting maintainability). A summary of studies looking at software metrics or other properties is provided in Table 2.4.5.

2.4.5 Can software change be predicted at all?

A superficial consideration of the subject might lead us to expect that unplanned changes cannot be predicted at all, since by its very nature it is *unplanned*. There are certain predictions that can be made, however. It is known, for example, that changes from external sources are likely to affect a project, and that the likelihood of this happening tends to increase as project duration rises. For example, Ebert and De Man have suggested that change rates on a project may be expected to affect 1-3% of requirements per month [50]. We have already summarised some studies presenting techniques for analysing and representing the likelihood of requirements changes (see Section 2.4.2), which may be used to make more detailed predictions about changes to those components which implement the requirements.

Certain features of software components may work together to render some components more likely to experience change than others, perhaps because of their key position in the software structure; their (lack of) encapsulation; their connections to other components (making them susceptible to change ripples). In general, numerous previous researchers have discovered that predictions regarding change-proneness can be made that outperform basic techniques, such as looking at module size or history.

For example, Tsantalis *et al* produced a framework for predicting the probability of a change in an object-oriented system, although it does require a minimum number of previous versions of a system. Three 'axes of change' are identified: inheritance; reference and dependency. The probabilities of change are automatically calculated by parsing a program to examine the potential axes of change and examining past history of changes. Calculated probabilities take into account that a single component may have relationships (and propagated changes) on more than one axis. Tests using this system with two open source Java applications found that there was a correlation between the probabilities produced and that it outperformed a simple examination of the files' histories.

One possible avenue of enquiry leads into the field of dynamic code metrics, suggested by Arisholm *et al*, who note that not all types of coupling are visible from static code [2]. This is in contrast to the C&K metrics, which are static. Distinguishing between 'import' and 'export' coupling links (ie, the direction of the link) the researchers count the total number of messages, methods invoked and classes used/used by a method, for both classes and objects, in each direction. Using an open-source case study, the dynamic metrics are compared to the amount of change detected between two versions of the system using a *diff* tool. Dynamic and static metrics achieved

slightly different results and appear to capture slightly different (but overlapping) data. Export coupling measures are shown to be more significantly related to change proneness than import, and a combination of dynamic measures with static measures and size metrics account for more variance in this particular case study than a model employing static measures and size metrics alone.

Whilst these are promising, models developed by Tsantalis *et al* and Arisholm *et al* can only be employed once certain data is available (respectively, a minimum history of changes, or actual, executable code). We aim to develop a predictive model for use at an early stage of development, when it can be acted upon productively, and where design or management changes can be effected at the lowest cost.

Ratzinger *et al* [127] develop a model for predicting which components are likely to need refactoring in the next two months. The model relies on change data from the previous three months, and was tested against two open source Java projects. Metrics used included:

- size (numbers of lines modified, added or deleted)
- number of developers working the same file
- additions, modifications and deletions per author and per file
- change activity rate and lines activity rate
- bug fixes and lines added, modified and removed for bug fixes
- coupling, assessed by number of co-changing files, and lines added, modified or deleted for an entire transaction
- average days between changes and average days per line

The study using data mining algorithms and grouped files into classifications: not refactored; refactored once; refactored more than once. Results suggested that classes which were ultimately refactored or not in the next two months could be identified (although distinguishing between those refactored once and those refactored more often was more difficult). The most useful attributes were the 'linesActivityRate' ('number of lines of code relative to the age of the file in months' [127]); coChangeNew (how often the file was checked-in alongside a brand new file); and coChangedFiles ('number of co-changed files relative to the change count of the learning period' [127]). Overall, this result is very encouraging, although the development style of the projects under study may limit the generalisability of the results. The projects studied were open source projects adopting agile development methodologies that emphasise frequent refactoring. We would suggest that caution should be employed before attempting to generalise this to projects with a different style of development.

Using early-stage project data to predict changes

Our hypothesis relies on employing data from the requirements and design stages to make predictions about the later software. Relatively little work has been conducted to examine the feasibility of using data from an early stage of the project to make predictions about change. This is partly because it is difficult to locate projects with the required information readily available.

Piattini *et al* propose employing metrics from the requirements stage for use in predicting maintainability of software systems. A series of metrics is proposed which may be drawn from early-stage entity-relationship diagrams [117]. The extant testing on these metrics is very exploratory, however: an experiment in which students and professors are asked to evaluate entity-relationship

diagrams for qualities such as understandability, simplicity, analysability, modifiability, stability and testability. The same entity-relationship diagrams are then parsed to gather metrics and the two results - metrics and ratings by humans - are compared. The study found that three metrics could be correlated with the given quality characteristics: the number of attributes in the diagram; the number of one-to-many relationships; and the number of binary relationships. Whilst interesting, the human assessments and the metrics were arguably measuring the same properties, since the only information that users have available to them is the number and type of attributes and relationships. Without the original entity-relationship diagrams for a live project and a subsequent history of changes it is impossible to tell whether users' initial predictions relating to maintainability based on early design work actually translate to genuine project characteristics later on.

Related more closely to our own research aims, Han *et al* also proposed a metric, Behavioural Dependency Measure (BDM), which can be calculated based on information drawn from UML 2.0 models early in the development process [65]. The researchers concentrate on propagated changes, constructing an Object Dependency Graph for each sequence diagram based on all direct and indirect links between objects. The number of weighted reachable paths is summed for all pairs of classes. Tests were conducted on two open source Java projects. The authors handled the lack of appropriate case study material by deriving UML models from the source code. Step-wise multiple regression was used to build a model to predict change-proneness; a source code counter calculated the lines of code added and deleted for all classes which were included in the original version of the applications. The results suggested that including the BDM metric in a model which already employed C&K metrics improved its predictive ability. The authors point out that the fitness of the model is low relative to code-based metrics, but that this is inescapable when basing a model on the limited amount of data available at the design stage. However, the design data was reverse engineered by the researchers from existing code. This possibly results in UML models which are quite different to those which might have been created at design time.

However, whilst it can be very difficult to locate an appropriate data set for this type of study, initial results from exploratory studies like these suggest that the area may feasibly be studied in more depth.

Studies employing software design metrics

Whilst little research has been conducted on early-stage data for prediction purposes, much more has analysed the relationships between change-proneness or maintainability and complexity metrics.

For example, a number of studies have examined change propagation and CBO, making the assumption that changes are propagated along links which can be identified by the CBO metric. Several studies have found this not to be entirely true. For example, Wilkie and Kitchenham conducted a study into data from a commercial C++ application to examine the relationship between CBO and the existence of change 'ripples' [152]. The authors concluded that CBO could identify the most change-prone classes, but that these were not necessarily the classes most at risk of *propagated* changes, suggesting that the lack of attention paid to inheritance relationships by the CBO metric makes it less useful for predicting areas likely to be hit by change ripples. Another explanation could be that a high value for CBO indicates that the component occupies a key position in the design. We discuss this hypothesis further in Chapter 4.

Table 2.1: Summary of previous studies examining correlations between software metrics and software quality indicators

Metric	Test system	Lang.	Stats technique	Dependent var	Results
C&K	Systems developed by student groups matched on experience [7]	C++	Stepwise logistic regression	Fault-proneness	All metrics significant except LCOM
	Industrial case studies. Sample change data selected by researchers [78]	C++	Pearson's correlation coefficient	No. of classes a given change propagates to	LCOM not correlated with change
	Industrial case study. Only one sample change considered due to resource constraints [29]	C++	ANOVA. Classes are organised into groups (high, medium or low) based on WMC and samples drawn from change averaged per class each group	'Change impact' (for a type of change, whether a method can propagate that)	There is a relationship between WMC and change impact
	Industrial case study. [152]	C++	Non-parametric tests (particularly Kruskall -Wallis) on means of groups (e.g., changed and	'Change ripples' (change that affects > 1 class)	Explicit dependencies uncovered. CBO identifies change prone classes, not necessarily change-ripple

Continued on next page

Table 2.1 – continued from previous page

Metric (s)	Test system	Lang	Stats technique	Dependent var	Results
			unchanged classes		prone. CBO outperforms no. of public functions as predictor.
	Industrial case study [53]	C++	Logistic regression.	Presence of a fault.	No correlations detected after size taken into account as confounding factor.
	Industrial case studies [32]	C++, Objective C	Stepwise regression.	Correlations with productivity, rework & design effort	High CBO & LCOM associated with lower productivity, greater rework & greater design effort.
L&H	Industrial case studies [94]	Classic -Ada	Stepwise multivariate regression	'Change' (no. lines added/	All variables significant except size.
	Industrial case study (re-using L&K data) [146]	Classic -Ada	Bayesian network; regression tree; stepwise & backwards linear regression	'Change' (no. lines changed during maintenance)	No Bayesian models are 'accurate'. For one system regression achieves better results than Bayesian; for others regression is better.
Briand metrics (C&K,	Industrial case study [26]	C++	Univariate logistic regression on 200 sample sets,	Presence/absence of a common change between a	CBO; method invocations; and 'indirect aggregation coupling'

Continued on next page

Table 2.1 – continued from previous page

Metric (s)	Test system	Lang	Stats technique	Dependent var	Results
L&H, & new metrics)			ranking independent variables	pair of classes	are significant but many change ripples not explained by them
C&K, & Behavioural Dependency Measure (BDM)	Two open source projects [65]	Java	Step-wise multiple regression	Change-proneness Lines of code added/deleted	BDM improves predictive ability of C&K model (although fitness of model is low)
13 metrics for E-R diagrams	Student and professor questionnaires [117]	E-R diagrams	Fuzzy rule system.	N/A	No. attributes; no. 1:N relationships; & no. binary relationships are possible predictors
Dynamic coupling metrics	Open source case study system (Velocity) [2]	Java	Multiple linear regression on (SLOC) as a confounding factor ordinary least squares regression	SLOC added or deleted	Dynamic *export* metrics 'clearly related to change'
Complexity metrics	Experiment - student participants perform corrective or perfective maintenance	Not stated.	ANOVA; correlation analysis; multivariate linear regression.	Minutes needed to understand, develop & implement requirements	Each metric useful as predictor, but very little variation in maintenance time is explained

Continued on next page

Table 2.1 – continued from previous page

Metric (s)	Test system	Lang	Stats technique	Dependent var	Results
	tasks on high or low complexity systems [6]				
Range of data drawn from change repository	Two open source case studies: ArgoUML and Spring [127]	Java	data mining algorithms: J48[2]; LMT; Rip; NNge. Model evaluation by 10-fold cross validation, using *precision, recall* and *F-measure*	number of refactorings; identified by commit messages; 100 random modifications checked to evaluate related to file age	Data from previous three months can be used to predict refactorings in the next two months. Data on: files added/modified at the same time and lines most useful

On a similar theme, coupling measures from a variety of sources are used to identify classes with common changes (i.e., classes that are checked into source control together) in a study by Briand *et al* [26] which looked at a commercial project[9] While common changes tended to occur more in classes which experienced high coupling, a substantial number of 'ripples' were not covered by the most highly coupled classes. 'Changes' in this study were extracted from change logs that recorded a change ID and a list of all classes affected by the change.

Approaching a similar position from a different angle, Kabaili *et al* [78] made the assumption that high cohesion usually accompanies low coupling, and since it's easier and cheaper to examine cohesion metrics, high cohesion could be identified and assumed to indicate low coupling. They used a variety of measures of cohesion, including a version of C&K's LCOM and two metrics for assessing cohesion developed by Biemang and Kang [78]. Changes were quantified using an already-published impact model, although tests were limited to six changes. Analysis on the results showed that no cohesion metric was statistically significant as a predictor for change; the authors postulate that none of the cohesion metrics captures all aspects of cohesion perfectly, rather than that CBO does not explain all change ripples. Cohesion metrics have also been shown not to be significant predictive factors for faults. Looking at detection of faults, Basili *et al*[7] studied systems developed by groups of students and tested by software professionals. Statistical analysis using multivariate logistic regression to identify correlations between C&K metrics and a binary value representing the presence of a fault revealed that all the C&K measures were significant (particularly DIT, NOC and RFC) except for LCOM, which was shown not to be significant.

On the other hand, research to examine other aspects of CBO has uncovered other correlations. For example, a study by Chidamber *et al* [32] looked at three commercial banking systems, calculating C&K metrics for the three systems. In addition to C&K metrics, four new metrics were used: size (LOC including comments); effort (developer time in hours); productivity (size/effort); and variables to identify specific developers - used as control variables. The hypothesis underlying the study is that productivity, rework and design effort are all likely to be explained in part as a function of complexity. The researchers gathered the required data from source control and from management documents (such as programmer reports and timetables) and used regression analysis to search for evidence of any correlations. They found that higher levels of CBO and LCOM were associated with higher productivity, greater rework and greater design effort, and also that inheritance was not widely used (resulting in low values for NOC and DIT). This suggests that classes that are highly coupled and also incohesive are more 'difficult' or complex than others. The issue of identifying classes that are 'difficult' is potentially useful. We return to this issue in Chapter 6, where we examine links between design activity (captured via our metrics, which are described in Chapter 3) and change-proneness, hypothesising that components which pose more design difficulties tend to be more volatile than those which do not.

A study by Chaumun *et al* on a commercial test case system divided classes into three groups: those with a 'high' WMC (WMC $>$ 29), with a 'medium' WMC (3 $<=$ WMC $<=$ 29) and those with a low WMC (WMC $<$ 3). A 'change impact' figure is calculated for each possible type of change, by identifying categories of relationships between entities which are known to be capable of propagating a known type of change. Four categories of links in total are identified: association; aggregation; inheritance; and invocation; plus an extra link type for C++ applications: friendship. A mean 'impact' figure for all methods is a measure of a class's ability to propagate change. ANOVA analysis on the results suggested that the the WMC metric and the calculated change impact are correlated - i.e., that WMC could be used to predict classes which are more likely to propagate changes. This study looked at risk of change propagation rather than simply likelihood

[9]For a summary of studies with a similar aim see Section 2.3.

of change (which is our aim). However, our results do suggest that WMC is linked to the quantity of change; we discuss this in Chapter 7.

Some studies have provided evidence to suggest that links may be drawn between change-proneness and software metrics. For example, as mentioned previously, Li and Henry have conducted a study to test correlations between all C&K metrics, with CBO replaced by L&H metrics, and 'maintainability' [94], defined as number of lines added/deleted per class (with modifications classed as an addition and a deletion). Regression analysis on two commercial projects written in Ada showed that maintenance could be predicted using software metrics. A final regression model included: DIT; NOC; MPC; RFC; LCOM; DAC; WMC; and NOM. The two proposed size metrics were discarded. The finding that size is not a predictive factor is an interesting one and contradicts several other studies. We discuss this further in Section 2.4.6.

Li and Henry's findings, however, were not completely supported by a subsequent study by Van Koten and Gray, who re-use commercial case study data published by Li and Henry to test three different statistical models. The models employ the proposed L&H metrics (i.e, five C&K metrics - excluding CBO - and with five additional measures) [146]. The three model types are a Bayesian network, a regression tree and two multiple linear regression models, one constructed using backwards regression and one constructed using stepwise regression. A Bayesian network (the main focus of this study) is 'a directed acyclic graph (DAG) whose nodes represent events in a domain' [146]. Links between nodes represent causal relationships, and are defined by a conditional probability. Probabilities can be calculated either using input from an expert user or from analysis of former cases. When used as a predictive model, the Bayesian network produces a list of all possible outcomes for a variable and their associated probabilities. None of the models in this study met criteria for an accurate prediction model, although the authors note that software maintenance effort prediction models generally have low accuracy. The Bayesian network outperformed regression-based models for one dataset and performed as well as other models on the other.

A slightly different goal is undertaken by Antoniol *et al*: estimating the *size* of changes. The proposed technique relies on a history of changes being available. An abstract representation of each version of the system is created, differences between each pair of versions calculated, and added, deleted and modified classes/methods counted. Linear models are used to detect a correlation with the size of a change. The method is tested on a public domain C++ system, finding that a simple model based on the number of modified methods can achieve 'extremely good predictions' [1], although there is no relationship found between the modified *classes* and the size of a change.

2.4.6 Issues in the use of software metrics

There is a substantial variation in the conclusions drawn by different researchers as a result of predictive studies using software metrics. This is perhaps partly a result of variation between different projects; it's inevitable that a few projects will not follow trends seen on others. The primary means of analysing software characteristics and quality empirically lies in case studies. Kitchenham and Pickard [83] define a case study as a study of a single real-world project; a study of several real-world projects is classed as a survey. Experiments are more unusual in software engineering fields but do occur. Some differences in conclusions between studies can perhaps be attributed to different statistical and experimental techniques applied by different teams. We discuss these issues below.

Validity of empirical studies

Although many studies have concluded that there *is* a predictive relationship between some subset of metrics and either fault- or change-proneness, there are varying results for the efficacy of individual (or combined) metrics and disagreement. For example, [78] and [7] find no evidence of a predictive correlation between the cohesion C&K metric LCOM, whilst [94] found sufficient correlation to include LCOM in their refined multivariate regression model. This may be partly explained by the differing nature of the dependent variables used in different studies. Some studies, for example, represent 'maintenance' by a count of the lines added/deleted between versions of the system, employing linear regression to test for correlations, whilst others use a binary variable to indicate the absence/presence of a fault/change, and logistic regression techniques to search for correlations.

Research in this area involves balancing the value of being able to study a real-world project with both predictable and unpredictable real-world events and obstacles (sudden changes to environmental requirements, staff departures, etc., which are very difficult to model or manufacture in artificial data) and the need to remove or control the influence of confounding factors which could bias results, such as the experience of project staff and events specific to the problem domain. Basili and Weiss recommend gathering data as the project progresses in order to ensure that information is not lost to memory [9].

Unfortunately real-world data is not always available. A relatively common approach for computer scientists is to design an experiment to be undertaken by teams of students (as noted by [6]). An experiment allows several treatments of a situation to occur in an environment where confounding factors can be monitored and/or controlled (unlike case studies or surveys of real-world projects). More generalisable conclusions may be drawn from an experiment like this than from a case study [83]. However, students are rarely as experienced in coding and general software engineering practice as professional teams, and due to time constraints, student projects are rarely as large and complex as many real-world problems. Bandi *et al* [6] argue that despite this, any correlations detected in a student-participant 'toy' project could still suggest avenues which could produce interesting real-world results. Another issue is that studies with a small number of participants are highly sensitive to differences in ability and/or experience. Where participants work in teams, this could be resolved by ensuring that the most and least capable and/or experienced students are distributed evenly amongst the teams (a stratified sampling approach). Another way to resolve this is to ask each team or individual to try all approaches in turn, and compare results. However, participants have a natural tendency to improve when asked to repeat a similar task several times (the 'order effect') [42, 131]. Other problems include participant fatigue (participants may invest less time and energy on work as it progresses); and the motivation of participants should be carefully considered (although often this is quite high if the study forms part of a student's assessed body of work).

For these reasons, when comparing the results of various studies in Table 2.4.5 we include notes on the statistical model used, the nature of the study (industrial case study, experiment with teams of students, etc.) and the language of the system analysed.

Size metrics

Size is known to be an important factor when examining quality attributes such as change- or fault-proneness. There are a number of reasons why a larger component might change more frequently:

- If we assume that changes can affect any lines of code in the project with equal likelihood (not a very safe assumption), then a larger component would naturally be affected by more

changes.

- A larger component probably implements more requirements. As we discussed in Chapter 1, requirements changes are one major source of changes to a project, so a component which implements more requirements is more likely to become subject to more requirements-related changes. We discuss this further in Chapter 6.

- A larger component may be larger because it occupies a key position, perhaps linking to a higher number of other components. This situation could result in change 'ripples' propagating to the large component disproportionately more than other components.

- Larger components may genuinely be more difficult and complex to create and understand, with more errors and re-work.

- Larger modules are perhaps less likely, by definition, to contain easily encapsulated functionality - such that developers are disproportionately more likely to perform refactoring.

There are a number of studies to demonstrate that size is, so far, one of the better metrics for predicting change-proneness. For example, in a regression test using lines added/deleted as a dependent variable and the two size metrics SIZE1 and SIZE2 as predictors, Li and Henry showed that a significant portion of the total variance in the lines added or deleted is accounted for by size [94]. Arisholm *et al* tested specifically 'whether the dynamic coupling measures [used] are significant *additional* explanatory variables, over and above what has already been accounted for by size' [2]. An industrial case study looking at the size of a use case model and the size of changes made to it also found a strong correlation [98].

This is an issue addressed by El Eman *et al.* [53], who argue that code size is a confounding factor which should be accounted for in any statistical analysis purporting to provide empirical validation for software metrics. An example test study conducted by the authors using data from a commercial telecommunications system in C++ showed a statistically significant correlation between software metrics (the study used C&K metrics plus two metrics defined by Lorenz and Kidd [53]) and fault-proneness. However, when the study controlled for size factors the results were not statistically significant. The study employed multivariate logistic regression, with 'fault-proneness' represented by a binary variable indicating the presence or absence of a fault. The tests controlled for size using a regression adjustment, where the confounding variable is included in the study as another independent variable. The authors point out that the technique widely employed by most researchers (such as Arisholm *et al.*, mentioned in the previous paragraph), is to perform regression testing on each of the independent variables and a size variable to search for significant correlations; they suggest that this is insufficient, however.

This conclusion is controversial. For example, Evanco [54] argues that each variable must have independent effect on the dependent variable (such as fault-proneness). Since size metrics such as LOC are not independent from software metrics such as WMC [10] accounting for size as a confounding factor does not provide useful statistical information.

We believe that size itself is a type of complexity metric - arguably, in the form of LOC, the earliest (and perhaps most crude). A complexity metric can only give a general indication of which parts of a system are likely to be more difficult to understand than others. For example, a high rate of coupling with other classes can indicate where complicated code structures *may* lie; a high number of code paths can indicate where complex code *may* be located. This is not going to be a cut-and-dried case for any complexity metric. Cyclomatic complexity, for example, which counts

[10] For example, it is not possible to add or remove methods - i.e., change the WMC measure - without also affecting LOC, unless other changes are also made to compensate [54].

possible code paths, can be high in cases where switch statements with many options are used, although many programmers do not find such switch statements complex to follow. Our belief is that size metrics like LOC behave in the same way as other complexity metrics in this respect; a module with a high value for LOC *may* be more difficult to understand.

Many other researchers also tend to treat LOC as type of complexity metric. For example, Briand *et al.* produce a framework for evaluating various types of complexity metric (discussed in more detail in Section 3.6.1) in which they divide 'complexity' metrics into five categories or types; one of these categories is size.

We agree that a study examining relationships between some complexity metrics and any other dependent variable is stronger for examining the effect of size as well. Some studies examining links between defects or change requests and complexity include size as at least one predictor variable (this is the case, for example, for Li and Henry [94]. This can introduce another problem in the form of multicollinearity[11] in multiple regression tests (if regression is the technique used). Size is frequently correlated, as Li and Henry discovered [94], with complexity. We examine size as a confounding factor for our metrics in Section 6.7.1.

Overall accuracy

As mentioned previously, there is some discrepancy in the final accuracy of the results achieved with similar predictive models. For example, Li and Henry claim that their 'predictions are reasonably accurate' when generated from a compact model using multivariate linear regression [94]. However, models developed by van Koten and Gray (including linear regression models) and tested on the same data 'do not satisfy the criteria of an accurate prediction model' [146].

Fenton and Neil argued in 2000 that 'existing models are incapable of predicting defects accurately using size and complexity metrics alone' [55]. Models which aim to use metrics alone to provide guidance don't provide the necessary causal information which designers and developers need in order to make decisions, they claim. They also point out the dangers of using surrogate measures (such as numbers of faults detected during the testing phase). They recommend the use of Bayesian belief networks (BBNs) to improve a model's representation of causal links. A BBN represents events (nodes on a graph) connected by causal relationships (directed edges on the graph). In a study to compare the prediction of software maintainability using a BBN, a regression tree, and multivariate linear regression on two separate datasets, the BBN only outperformed regression-based models on one dataset [146], although the authors concluded it performed 'reasonably' competitively. Whether a BBN is more useful than a regression model perhaps depends ultimately on the intended purpose of the study results. For example, an application which presents a list of files to a maintenance developer as possible candidates for co-changing might benefit from a Bayesian model, since information on the type and direction of the link between two files is likely to be more useful than a blank list of possible suspects. Attempts to recover the rationale behind changes made to the system automatically have made some progress, however (for example, see [108]), and could prove useful here.

None of the researchers investigating the relationship between software quality indicators and software metrics claims that the correlations detected so far suggest a causal relationship. However, Fenton and Neil's argument that data is missing is important. It is clear that although a relationship may exist between coupling and/or complexity metrics and the level of volatility a system experiences, the relationship is not perfect and a certain amount of change - quite a large portion, in some studies - cannot be explained by the coupling metrics alone. Briand *et al.*, for

[11] where a number of input variables (such as different complexity variables) that are used in a test are strongly correlated with each other

example, note that 'other important dependencies are clearly not measured or accounted for, and may not be measurable from code alone' [26], whilst Bandi *et al.* concluded that '...because software maintenance is such a complex task, there are likely many issues that come into play besides the complexity of the design' [6].

2.5 Metric validation

In this thesis we develop a new set of metrics designed to 'measure' design and requirements activity. We discuss in Chapter 3 how we interpret this and present the metrics themselves. Measurement of software and activities surrounding its creation is not a straightforward topic and many metrics have been proposed in the past. Accordingly, research attention has been focused over the past few decades onto the subject of creating and validating software metrics. Validation efforts fall into two complementary branches, both of which should be considered in a rigorous study, although they are named differently by different authors. Kitchenham *at al*, in the paper 'Preliminary Guidelines for Empirical Research in Software Engineering' [84], describe the two types of validation as follows:

> Internal validity relates to the extent to which the design and analysis may have been compromised by the existence of confounding variables and other unexpected sources of bias. External validity relates to the extent to which the hypotheses capture the objectives of the research and the extent to which any conclusions can be generalised [84].

In a separate paper, Kitchenham *et al.* [82] refer to 'theoretical' and 'empirical' validation. 'Theoretical' validation 'confirms that the measurement does not violate any necessary properties' [82], whilst 'empirical' measurement echoes external validation by providing corroboration that the measured values and the predicted attributes are consistent. Poels and Dedene [119] characterise external validity as referring to 'the usefulness of a measure', whilst Briand *et al.* term this the 'theoretical soundness of a measure, i.e., the fact that it really measures what it is supposed to measure' [24]. Regarding external validation, Kaner and Bond prefer the term *construct validity*: 'This is the *construct validity* question. How do we know that thesis metric measures that attribute?' [79].

In Chapters 5 and 6 we will investigate the 'usefulness' of our measures by evaluating their ability to predict information in which we are interested. We will also consider potential threats to the validity. In this section, however, we assess the theoretical validity of our proposed metrics.

A suggested measure should exhibit some of the basic mathematical properties associated with measurement. A number of authors have proposed guidelines on this subject with specific reference to software complexity metrics. Unlike measurements in the physical world, where objects have visible proportions and properties, 'measuring' software is not a simple issue[12]. Unfortunately no single set of properties has been universally agreed upon, although several suggested sets of properties have been widely cited and, in many cases, employed. We consider several authors' proposals on the subject in the following sections.

It is worth noting that validating metrics using lists of desirable properties does not demonstrate that they are meaningful. This was demonstrated in a short paper by Cherniavsky and Smith, for example, who defined a (largely meaningless) 'complexity measure' but which they claimed nevertheless satisfies all nine of Weyuker's properties [30]. A similar point is made by Poels and Dedene

[12]Poels and Dardene paraphrase Zuse to summarise the problem: 'The fundamental problem of software measurement is that for many software attributes, including software complexity, it is not known what the empirical relationship system looks like' [119] (we discussed the 'empirical system' in Section 3.1.

[119] (although not specifically about Weyuker's properties), who argue that most property-based validation exercises produce lists of properties which are necessary but not sufficient. Therefore, whilst a failure to satisfy the properties can be a useful proof of invalidity of a proposed metric, satisfaction of the properties is not proof of validity.

2.5.1 Property-based validation as proposed by Weyuker

One of the most widely discussed and adopted set of properties for software metrics is that proposed by Weyuker in the late eighties [151]. Weyuker combines several properties associated with measures in general with some others that characterise software complexity specifically. The first three (of the nine) properties, for example, require that metrics are capable of distinguishing between different entities, be not too coarse-grained and not too fine-grained.

Many researchers who have proposed complexity measures have evaluated metrics using these properties (for example [6, 31]). However, although Weyuker's properties have been influential and widely referenced, they are not without criticism. Horst Zuse, for example, has suggested that Weyuker's properties are contradictory. This is contested by Morasca *et al.* [110], who argue that the metrics are not contradictory 'in the usual mathematical sense of being incapable of being satisfied at the same time'[13].

Many of the metrics proposed by Weyuker are specific to a procedural program's structure and are not easily or logically extendable to modular developments or metrics which, like ours, attempt to extend the empirical world beyond the bounds of the program code itself. Property 7 of Weyuker's properties, for example, requires that 'there are program bodies P and Q such that Q is formed by permuting the order of statements of P, and $|P| \neq |Q|$ [151] [14]. This property is highly specific to software in particular, and does not translate easily to metrics assessing requirements and design links. Property 6 addresses the notion that concatenation of two program bodies does not always affect the complexity of the resultant program in a uniform way [151]. This particular property has met with controversy, particularly with regard to object-oriented programming, for which it is not best suited. Chidamber and Kemerer [31] suggest disregarding it for modularised development projects.

2.5.2 Property-based validation as proposed by Briand *et al.*

Another framework for validating software metrics was proposed by Briand *et al.* [24] in the mid-nineties and has been employed to validate metrics by, for example, Piattini *et al.* [117]. This framework discriminates between different *types* of metric, with the properties tailored to the specific requirements of different types. The types of metrics are: size; complexity; a subtle concept which the authors term 'length'; cohesion; and coupling. Researchers can determine a metric's type by establishing whether it complies with all the suggested properties for that type. Metrics can fall into two groups at the same time (e.g., a metric can be said to be measuring both/either the size and complexity of software), if it satisfies all the properties for each types. We discuss these metric types further in Section 3.6.1.

2.5.3 Measure properties proposed by Kitchenham *et al.*

Kitchenham *et al.* [82] in 1995 discussed validity in the general field of measurement in rather broader scope, arguing that we must separately consider the validity of attributes, units, instruments and protocols employed. Some of the properties suggested are drawn from general principles

[13] Comments on this topic from both from Zuse and Weyuker are summarised in [160]
[14] Weyuker writes $|P|$ to mean *the value of a metric for program P.*

of measurement (such as the condition that '*Each unit of an attribute contributing to a valid measure is equivalent*'. There is a subset of overlap between Weyuker's properties and those proposed by Kitchenham *et al.*, such as the suggestion that any measurement system must allow two entities to be distinguished from each other (this is Weyuker's first property) and that two entities must be capable of achieving the same value where appropriate.

Kitchenham *et al.* further suggest that a measure must follow the Representation Condition ('i.e., it must preserve our intuitive notions about the attribute and the way in which it distinguishes different entities' [82]. The authors argue that this renders some of Weyuker's properties unnecessary.

Kitchenham *et al.* also propose that:

> Since an attribute may be measured in many different ways, attributes are independent of the unit used to measure them. Thus, any definition of an attribute that implies a particular measurement scale is invalid [82].

This results in a rejection of some of Weyuker's properties, because they violate the principle of not implicitly implying a particular scale type. In total, Kitchenham *et al.* reject six of Weyuker's nine properties.

However, the principle that attributes may not imply a particular measurement type is disputed by Morasca *et al.* in [110], who argue that this requirement places such a restriction on measurement that it limits the ability to define adequate properties to characterise interesting attributes. They point to measures employed in physics, such as temperature, which imply scale types in the definitions of their properties.

An indirect measure is a measure derived by examining direct measures. For example, productivity is a commonly-cited example of an indirect measure, as it combines direct measurements such as lines of code and elapsed time, which can be directly measured, to produce a new metric. In addition to the general principles described above, some further properties are added by Kitchenham *et al.* specifically for indirect measures, which must:

- be based on a model concerning the relationship amongst attributes
- not show unexpected discontinuities in value
- use units and scales correctly
- be based on a dimensionally-consistent model (all taken from [82]).

2.5.4 Axiomatic approaches to software measurement as proposed by Poels and Dedene

Poels and Dedene [119] argued in 2000 for a 'distance-based' approach to software measurement, re-employing axiomatic approaches from mathematics (credited as first suggested by Kitchenham and Stell).

In mathematics a 'metric' is defined as a measure of distance. The approach adopted by Poels and Dedene is to define attributes of a software product as distances (between the software products and reference points used for measurement): 'If software attributes are defined as distances, then the measures of these distances can be used as software measures' [119]. This results in an axiomatic model of software measures. The authors argue that other property-based validation exercises produce lists of properties which are necessary but not sufficient. By this, they mean that the list may be used to prove the invalidity of metrics by a failure to satisfy some properties, but not to

prove their validity by satisfaction (see Section 2.5.1 above). Poels and Dedene thus propose a set of axioms to apply to distance-based metrics, the sufficiency of which is guaranteed by measurement theory. We discuss these properties further in Section 3.6.2.

2.6 Our approach

We intend to build our approach on existing studies, which have generally concluded that there is some sort of predictive relationship or correlation between software metrics and change-proneness. As we discussed in Chapter 1, our underlying assumption is that much change is driven by factors that can be identified with certain patterns in software links. Whilst much of this change is largely unpredictable in itself, we hypothesise that recording links between requirements and design elements rather than just between software components will create a more holistic model that can take into account the hidden dependencies between entities with similar or linked requirements. Some of these links will not be clear to a person restricted to studying coupling between software components and their relative complexity.

We intend to develop a new set of metrics from a traceability system that records a holistic set of relationships between entities at different stages of the lifecycle, including requirements as well as design features. We outline the development of these new metrics in the next chapter.

Chapter 3

Developing descriptive metrics

In Chapter 1 we introduced the hypothesis behind our research: to improve prediction as to which components in a software development project are most likely to be change-prone. We hypothesise that data about requirements and/or design can be used to improve predictions.

To test this hypothesis, we will develop metrics to characterise the requirements or design involvement for each component. This chapter documents the development and validation of our metrics. The rest of this chapter is laid out as follows. We first explain the methodology we will follow in Section 3.1. We survey existing schemes for capturing useful data from the requirements and design stages of the lifecycle in Section 3.2. In Section 3.3 we put this information to use in developing models which we believe capture the dependencies we need from the requirements and design stages. We use the models to develop some metrics in Section 3.4 and finally we validate these metrics in Section 3.6.

3.1 Methodology

Measuring activities such as design work or qualities of intangible entities such as volatility of software components is not a straightforward task. Metrology (the study of measurement) is a discipline that has attracted considerable interest over many years. Writing about metrology and software, Horst Zuse has argued that metrology must assume the presence of entities in the empirical world and of empirical relations between those entities [160]; the 'empirical world' contains those objects we wish to measure. Zuse then conceptualises measurement as creating a mapping between the dimensions of entities in the empirical world, and a formalised definition.

We will assume that software components, requirements and design concepts can be represented as a set of entities which are linked by relationships. For example, we assert that a relationship exists between component c and the requirement r that it implements. Our research centres on the development of metrics to measure these relations; our aim is thus to develop a clearly-defined mapping between characteristics of objects in the empirical world and a numerical value to represent them.

We follow the model-order-mapping (MOM) approach proposed by Gustafson *et al* [64] to develop our metrics. This approach begins with the concept of a set of documents from the empirical world; in our case, these are code documents. The documents display observable characteristics, and so a model is created to represent the document and its characteristics of interest. The mapping between them is defined precisely. A partial ordering to define how the objects of the model are related is provided. And a further definition is provided to describe mappings between the model and an 'answer set' of 'real numbers' [64]. Such an approach 'leads to a *quantitative* mea-

sure based on a *qualitative* assessment of the document's attributes' [64]. Figure 3.1 illustrates the process we follow (based on a diagram of the MOM process in [64]).

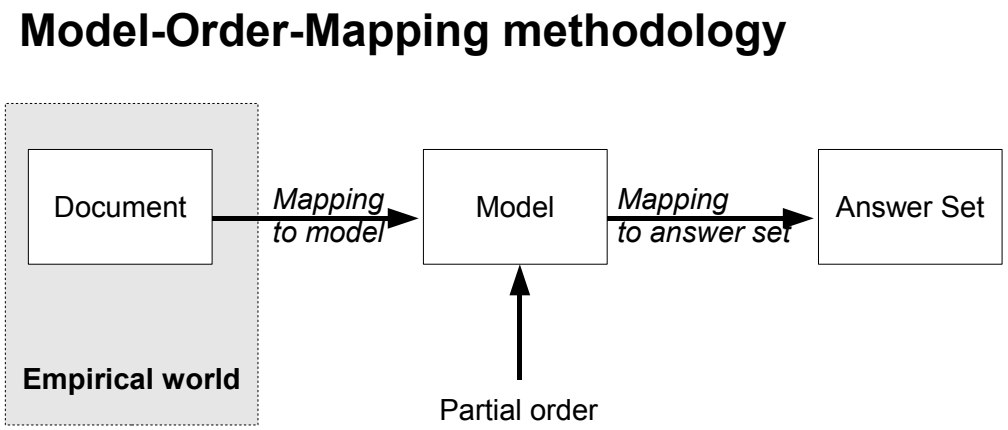

Figure 3.1: MOM methodology employed in the development of new software metrics. Taken from [64]

In the next section we consider a range of existing schemes for modelling and capturing data from requirements elicitation and design stages, identifying key entities and relationships. We will use this information to develop models (as suggested by the MOM methodology) which represent the impact of the requirements and design stages on the software components.

3.2 Traceability structures

Our process requires us to determine which objects and relations from the empirical world are important. We then develop a model of these objects and relations.

3.2.1 Types of data available

Figure 3.2 shows the data that may be gathered from early stages of many projects. We wish our predictions to be usable by projects where a reasonably complete design has been developed. We therefore restrict our metrics to data gathered from requirements, specification and design stages, by which time much should be known about the software's aims, rationale, decision-making and high-level software structure.

We consider a series of existing structures and models for capturing this data in the following few sections. We then draw on these models to recreate our own simple model of key requirements-stage dependencies, presented in Section 3.3.2.

3.2.2 Key concepts from the requirements stage

In general, techniques and data structures employed during the requirements analysis phase vary widely (reflecting the wide disparity of activities in this stage) and have very different aims. In Tables **??** and **??** we presented a summary of key features of existing modelling schemes, noting the different types of entities and dependencies captured by various modelling schemes and picking

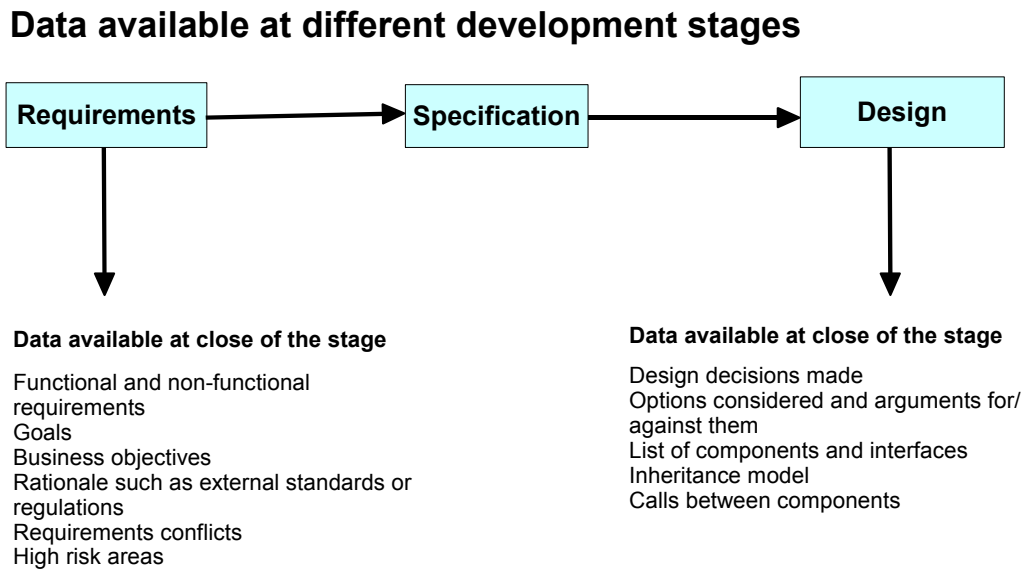

Figure 3.2: Data drawn from different stages of the lifecycle

out some in particular that we believed could be relevant for predicting change-proneness. Some of the key concepts captured by various requirements structures include the following:

- Requirements rationale. Rationale can be approached in a variety of ways. i* [155, 156], for example, models the emergent new system alongside the existing process, showing clearly the motivations for creating the new system, and the benefits (or even disadvantages) it may present. Such a model is, however, too complex for our purposes, as we are aiming for a set of metrics which can be calculated relatively simply. Gotel and Finkelstein [59] propose recording people who have played a role[1]. However, we do not include links to contributors in our models, as we suspect that it may not contribute usefully to our predictions. Many projects will have a limited number of key decision makers (as well as - potentially - authors and documentor), meaning that there is unlikely to be enough information to differentiate between entities based on this type of link, although details of the original author would be helpful for other purposes, such as impact analysis. Finally, we do not think that links between individual contributors and change-proneness are likely to be generalisable between projects.

  Many changes are imposed on projects by external sources. On the other hand, adherence to a stable, externally-controlled source such as a standard may insulate a component from change, since there is much pressure not to change widely-accepted standards lightly. We should like to document any dependencies on an external resource such as external regulations or standards, since they may act as a source of changes or as a stabilising influence.

  A common source of information for many projects - and a common way of viewing requirements - is via a scenario or use-case model. Where use-cases are not included in documentation, interviews with end user groups often serves to document, from a user's point of view, what they expect the system to accomplish. Since these data are likely to be readily available on most projects in one form or another, user scenario could form a simple part of

[1] All the schemes mentioned in this section are discussed in more detail in Section 2.2.2.

requirements rationale in our models.

- High level strategies and/or goals. Goals or strategies are not dissimilar to rationale, as they can be seen as a central driving force behind the requirements themselves. Examples can be seen in Goal-Oriented Requirements Engineering (GORE) techniques [147] or in the Goal Structuring Notation (GSN) which addresses safety case strategies [80, 81]).

  Organising requirements around the principle of goals or high-level strategies provides an alternative view of requirements, which have historically been organised around functionality. Documenting high level strategies and/or goals allows requirements which address different but cross-cutting functionality to be grouped together. This is important in our aim of capturing hidden dependencies that might aid in the prediction of change-proneness. Entities which contribute towards the same objectives or goals may tend to change together, even though they may not have any overt shared areas of functionality. However, for the time being we elect not to include a 'strategy' or goal entity. If a project has followed GORE techniques for the requirements analysis, the data will be readily available for research. However, many projects do not employ such techniques for requirements gathering, and attempting to impose a goal-oriented or strategic view onto an exercise which was not goal-oriented in nature is likely to introduce a considerable experimental bias into our results.

- Non-functional requirements. NFRs (or 'soft goals') are notoriously difficult to 'design' and in many situations rely on repeated user acceptance testing and refinement, resulting in many changes. Both i* and the NFR modelling systems described above represent this type of data. Soft goals can be a significant source of changes in a project. For this reason we will incorporate soft-goal dependencies in our model. We discuss argumentation structures (to represent positive/negative contributions) in Section 3.2.3.

3.2.3 Key concepts from the design stage

There is arguably less variation in the design stage modelling systems than in requirements stage modelling system. The design-stage models and techniques discussed Section 2.2.3 generally centre on a decision with one or more options. Our design-stage model minimally requires an entity to reflect design choices/issues and entities to represent options being considered (all of the systems described in Section 2.2.3 include these).

Ideally it also includes a structure for considering the positive and negative points associated with each option. Such a structure echoes to some extent the models illustrating trade-offs affecting NFRs, as modelled in the NFR framework [111] (see Section 2.2.2) and in i* [156] (see Section 2.2.2). We choose for the time being simply to make reference to the fact that a conflict could be involved when listing arguments supporting and opposing a potential solution.

Our system for representing the decision-option structure strongly echoes DRCS [85] (see Section 2.2.3), which has the advantage of simplicity. In DRCS, an issue is raised; it has one or more options; options are supported or opposed by claims. We wish to provide space for arguments to be made either for or against each option, so that a series of points can be connected to each option independently from any other options. The DRCS style of argumentation structure permits this, unlike the more clumsy QOC [101] structure (see Section 2.2.3). We do make some changes to the DRCS model, however:

- We rename 'Modules' to 'Components'.

- We remove the entities used to represent system structure (in the artifact synthesis structure). This is for several reasons: the artifact synthesis structure is rather too high-level for our

purposes; it is designed to represent a mechanically engineered system is sits a little uneasily on a software system; and we intend to use existing code complexity metrics as a method for assessing software structure, so it is not needed.

- We drop the DRCS entity, Specification, and add a new one, Requirement. DRCS 'has-specification' and 'has-subspecification' relationships to represent links between specifications and modules. This relationship was altered, so that Requirements are 'allocated-to' Components, instead of Components 'having' Requirement. We feel that reflected better how a developer views the flow of project information. Correspondingly, the 'has-subspecification' relationship became 'has-subrequirement'. Requirements in both new and old structures raised issues.

- Entities related to planning have been dropped, as have system structure (such as modules and submodules). Superficially, it is tempting to hypothesise that greater attention paid to planning activities may have an impact on change-proneness. However, in reality it is difficult to define exactly what is intended by entities such 'task' or 'strategy' (these are the planning entities employed by DRCS). There's a risk that 'tasks' for many projects have a one-to-one relationship with components (e.g., each component is linked to a single task: *develop component*) and this is unlikely genuinely to aid our predictive efforts. We would also expect that actual planning efforts would be updated frequently throughout the project, making it difficult to decide at what point planning data is stable enough to consider using in predictive efforts.

- The relationship 'denied' (used by Claims which oppose Options) was renamed to 'opposed', to express better the idea that opposing arguments do not necessarily veto the Option.

- DRCS uses the 'is-best-option-for' relationship to identify the selected or preferred solution from a range of options. However, other design capture systems, such as the framework suggested by Ramesh and Jarke [125], or Redux [116], model the outcome of a decision separately to the decision itself and its options. We include a separate decision outcome element, too, as it is a tidy way to abstract the outcome of the issue (which may change in the future) from the argumentation structures created around each of the options. This makes it possible to attach reasons to the decision outcome that were not necessarily about supporting or opposing individual options

- We drop the DRCS structure for raising a question and providing an answer, which does not link rationale closely enough to requirements for our purposes. It also risks duplicating the 'raises-issue'/'has-option' structure.

- DRCS uses relationships ('is-better-than') to rank different options. This is not very scalable, because logically to rank many options we would have to create relationships between each option and all others ranked 'below' it. Instead we use the Decision Outcome entity and supporting Claims to represent ranking.

3.2.4 Requirements for our metrics

Here we summarise the qualities we wish our metrics to exhibit, before we move on towards developing them. These requirements are based on our surveys of existing modelling structures for requirements and design-related data, which were summarised in Tables **??** and **??**.

1. They should draw on data that are available to the designer who has completed an advanced design.

2. Metrics should be capable of identifying those entities with the most burdens placed upon them by the requirements, or by soft goals.

3. Metrics should examine potential sources of external changes, such as regulations or standards outside the control of the project team.

4. We are interested in entities which have been subjected to more detailed scrutiny during the design phase, so metrics should be capable of identifying entities which are the result of a design issue.

5. Metrics should capable of identifying components which have been associated with the most difficult and/or time-consuming design issues.

6. Metrics should be simple to understand and rely, where possible, on data that a project has available without special instrumentation or data-gathering stages specially introduced.

7. Metrics should be based on models that are natural and intuitive. Drawing information from a real project to populate a traceability structure is a time-consuming and unattractive activity. Ideally, the traceability structures should be easily translated to the types of data that most projects naturally store.

8. The metrics should be expressed always in relation to an entity which represents a system component (for example, the code modules for a software system) as we aim to predict change-proneness specifically for these entities.

3.3 Developing models to structure necessary data

In this section we describe the models we have developed to capture pertinent information from the requirements and design stages and link it to components themselves. These models are for the explicit purpose of capturing a minimum of project data from the requirements and design stages that we believe may help us predict change-proneness.

We do not devote efforts to modelling extra information drawn from specification stages. This is for the following reasons:

- We believe that links between requirements and specifications, redor between specifications and code, are unlikely to reveal large quantities of hidden dependencies which will prove useful for change ripple prediction, partly because many of these links are likely to be similar to the original requirements-to-code links.

- It is assumed that many projects do not necessarily carry out a formal specification stage, and this may be merged with an iterative design stage. There is little focus on specifications in terms of entities or relationships between entities in the design- and requirements-oriented models discussed earlier in this chapter, for example. Anectotally, a number of smaller and medium-sized projects may reduce the scope of specification stages, or merge it with neighbouring activities such as an iterative design. This is the case, for example, for the CARMEN project described in Chapter 4.

Our models consist of a set of entities of various types. Entities are linked by relationships. Relationships do not carry any semantic information aside from their type (i.e., there is no ordinal information associated with a relationship and relationships are not grouped into 'AND' or 'OR' groups). As in DRCS [85], our models represent a dependency between two entities as a labelled relationship. Like DRCS, we use descriptive labels to differentiate between different relationship types.

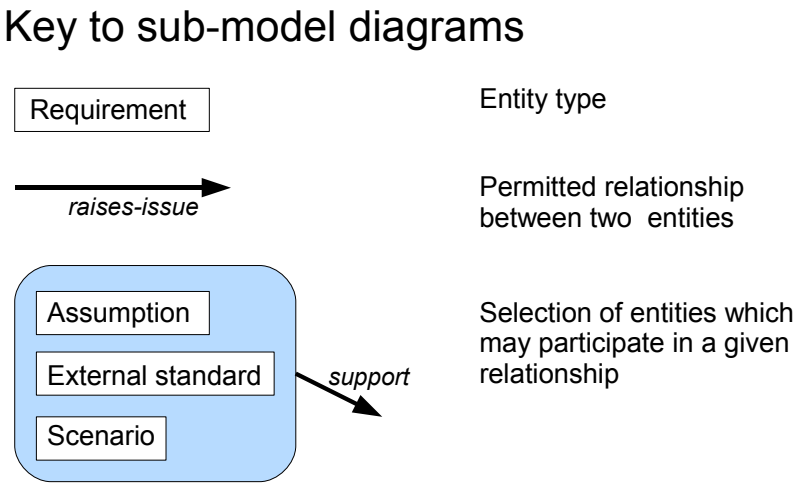

Figure 3.3: Key for models

3.3.1 Relationship direction

Relationships on these models are directional. The direction of the relationship aims to follow a logical progression, from requirements rationale towards requirements, from requirements towards issues and options, and from both requirements and design decisions towards components. This sense of progression is familiar and intuitive to software developers, even if development on a real project is sometimes more 'messy' or iterative than linear.

3.3.2 Requirements model

Figure 3.4 shows the proposed structure for modelling requirements data from a project. Entities include:

- *Requirement.* Used to represent a standard Requirement, although it may also (optionally) be specialised as a *Soft Goal.*
- *Component.* Represents a software component, such as a code module.
- *Claim.* A Claim is an argument that supports or opposes a particular Requirement. Claims are also an important part of the design model (below).
- *Decision problem.* Represents a situation where an explicit decision has been made and recorded. Further details of the Decision Problem are captured in the design model (below).
- *Scenario.* Draws upon the user scenario entity employed by Ramesh and Jarke's requirements traceability framework. The Scenario itself is envisaged as a simple user statement of their needs from the finished system, and is intended to form a small part of requirements rationale.
- *Assumption.* Represents an assumption made by the requirements analyst or the designer.
- *External standard.* Represents any standard published by a third party (eg, for data formats, transmission, etc.) which affect the finished system. We distinguish between an External Standard, which is applied to project artifacts as a constraint, and an Internal Standard (described in the design model below) which is developed by the team themselves. They differ

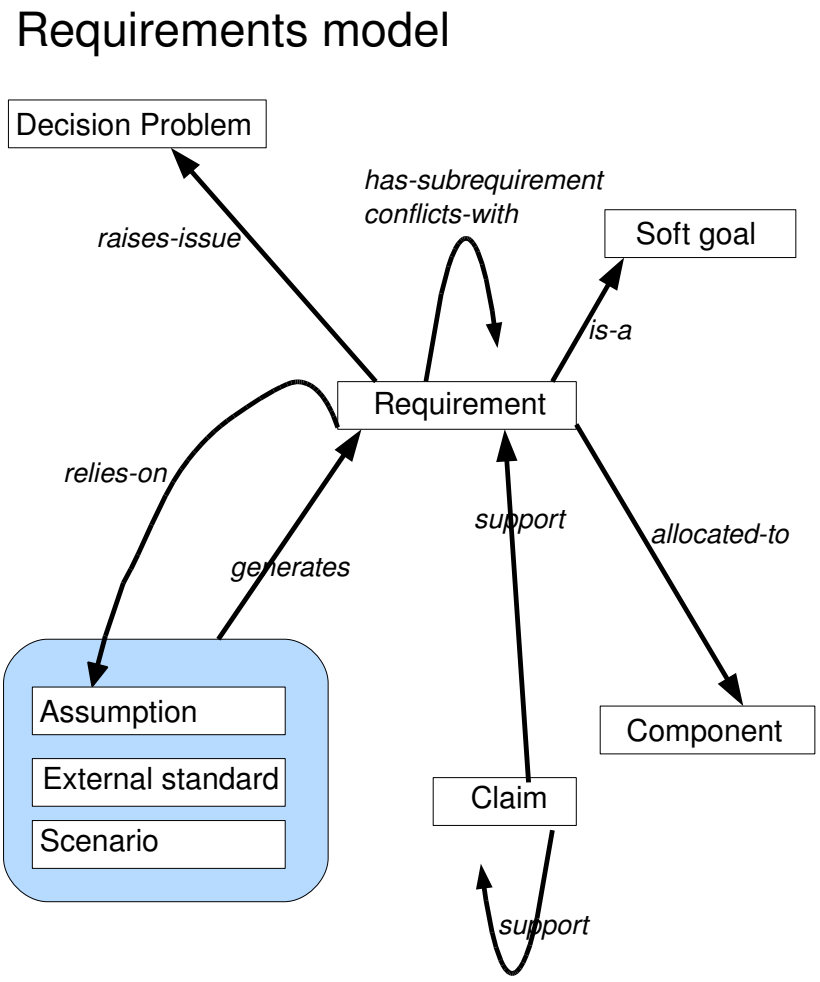

Figure 3.4: Diagram showing proposed model of entities and relationships for the requirements phase

in the amount of effort required. We believe that the adoption of an External Standard (which we'd expect to reasonably stable) and the creation of a new internal one (which presumably will undergo much work before it becomes stable) will have substantially different impacts on volatility.

Requirements rationale is envisaged as series of Assumptions, External Standards and user Scenarios. The *generates* relationship is used to show that an Assumption, Scenario or Standard has played a part in the derivation of a new requirement. Additional rationale may be added in the form of Claims which *support* a requirement.

Requirements may be specialised (as noted earlier) into soft goals. They may also be specialised by decomposition into subrequirements. All types of requirements are attached to Components using the *allocated-to* relationship, to indicate the component tasked with implementing the given requirement.

We are particularly interested in capturing the external rationales and standards that drive requirements, since (as discussed in Chapter 1) many changes may be imposed as a result of changes in the external world - such as amendments to regulations or standards, hardware and software platforms and the business environment. It is therefore useful to identify where requirements share

a dependency on observed external phenomena, as this could be a potentially helpful source of previously hidden dependencies through which changes may ripple.

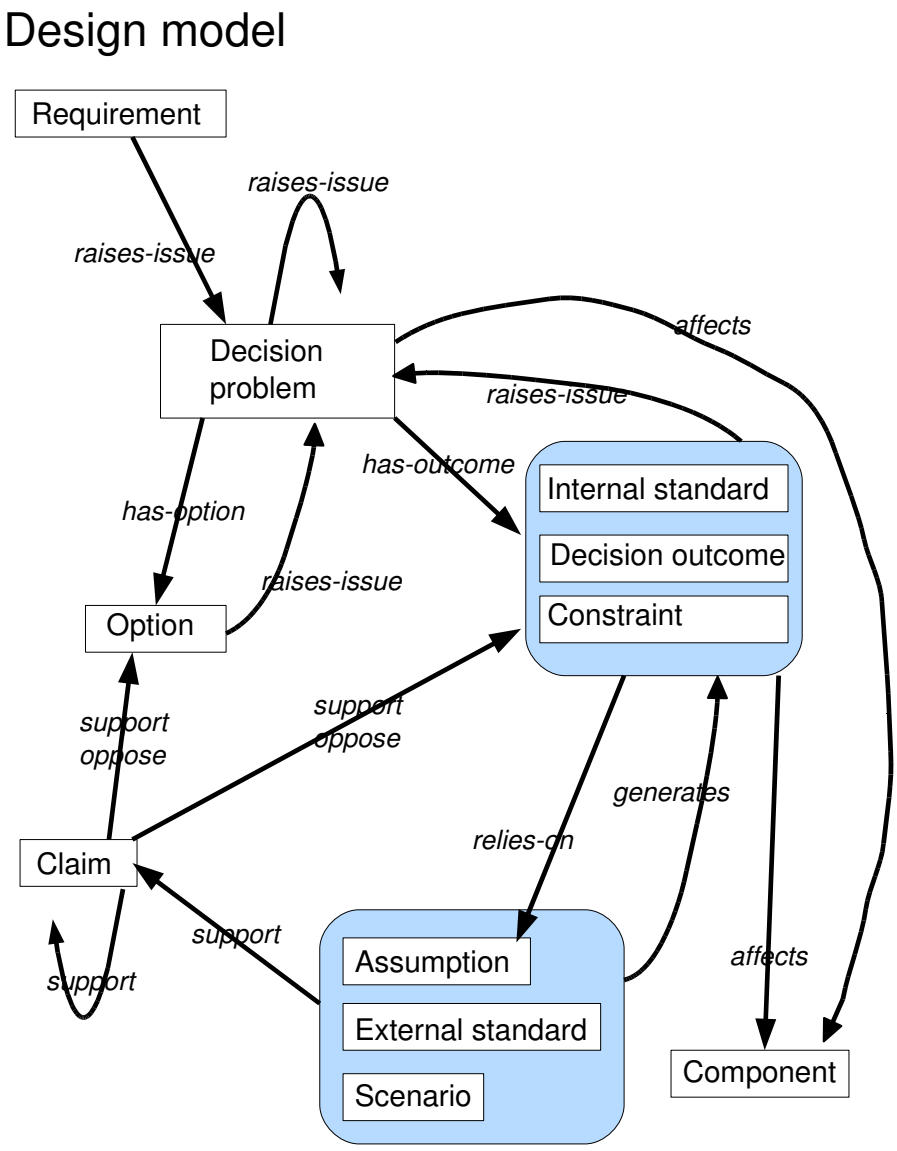

Figure 3.5: Diagram showing proposed model of entities and relationships for the design phase

3.3.3 Design model

Figure 3.5 shows our proposed structure for modelling design data from a project. The two models overlap slightly, so some entities from the requirements model reappear in this structure. Entities which are new include the following:

- *Option.* Some design problems will be associated with a range of possible Options, whilst others may only consider one.

- *Claim.* Claims appeared in the requirements model (above). In the design model, they may support or oppose an Option under consideration or the outcome of a Decision Problem.

- *Constraint*, *Internal Standard* and *Decision Outcome.* It is assumed in our models that a Decision Problem is an activity that results in the generation of some additional entities. We model these as either a Decision Outcome, a Constraint which further narrows the potential range of behaviour, or an Internal Standard (discussed above). All the decision problem outcomes (Decision Outcomes, Constraints and Internal Standards) are linked to the components they affect.

Addressing a Requirement may result in an issue that requires some decision (represented by a Decision Problem). Decision Problems can, in turn, raise further Decision Problems, and can be associated with one or more Options. Each Option may be accompanied by supporting or opposing Claims, which may, in turn, be supported by user Scenarios, External Standards or Assumptions. We include a relationship between a Decision Problem and an Internal Standard, suggesting that a Decision Problem may result in the creation of a new Internal Standard. Decision Problems and their Outcomes can affect Components. Many Decision Problems may not have, at the time of gathering data, a definite outcome as of yet. However, in many cases, it may already known which components will be affected by the outcome of the decision; in this case the decision structure can still be linked to components directly using the 'affects' relationship. We expect that a proportion of links will be missed, because the links cannot be anticipated at this stage of development, but our expectation is that major links may be known, and this is what we wish to 'measure'.

Explicitly recording those Assumptions upon which Claims and Requirements are based is an important step towards managing changes; if the Assumption becomes - or is subsequently found to be - invalid, then those Requirements or design solutions that were associated with it can be traced easily. DRCS and Ramesh and Jarke's reference model both represent assumptions explicitly, and since our interest lies in the study of changes we replicate this.

As with Requirements, Assumptions, External Standards and user Scenarios have a role to play in generating Constraints, Internal Standards or Decision Outcomes. We represent this using the *generates* relationship. As before, we believe that since many changes may be motivated by changes in the external world, decisions with dependencies on external entities such as standards may exhibit dependencies with similarly dependent decisions, which may not be obvious from examining the relevant code alone.

In this model, there are some types of entity that we expect should always have at least one 'parent' (that is, they should always participate in a relationship which explains their origin): they are Decision Problems (we expect that they should always be raised by another entity); Options (we expect that they should always be attached to at least one Decision Problem); and any of the three entity types we have grouped together as 'Outcomes' (we expect that they should originate from a Decision Problem). These are summarised in Table 3.1.

3.4 Developing metrics from the models

Following the MOM methodology discussed in Section 3.1, our models aim to capture characteristics of software components that we are interested in: in this case we wish to capture the extent of activity requirements and design stages of the project development. The MOM methodology draws on the models to develop a mapping to an 'answer set'. We now define a series of metrics which allow us to assign numerical values to the characteristics of interest from our model. The metrics defined in this section are summarised in Table 3.2 on page 64.

Table 3.1: Summary of the mandatory relationships that apply to some entities in our models

Entity type	**Mandatory relationship**
Decision Problem	should be raised by an entity: a Requirement, a Claim or another Decision Problem
Option	should be linked to a Decision Problem via the 'has-option' relationship
Outcome (Internal Standard, Decision Outcome, or Constraint	should be linked to a Decision Problem via the 'has-outcome' relationship

3.4.1 Notes on notation

We use standard mathematical set building notation to define our metrics. Defining metrics in this way has the following benefits:

- it reduces ambiguity in our meaning (ambiguity in metric definition has been a problem in the past, as noted in [96])
- formal specifications lead easily to executable statements - facilitating production of automated tools to generate figures

Using our models, we represent both the system and all the data describing its development (such as requirements, assumptions, etc.) as a directed graph, G, which contains entities (x, y, z ...). Edges connect those entities to indicate the presence of a relationship.

Where we specify a particular type of relationship we write (x, y, l) to refer to entities x and y linked by relationship with label l. In some cases, we don't mind what label is attached to the edge between x and y. In that case, we simply write (x, y) to indicate a linked pair of entities.

The set of entities G contains several meaningful subsets of entities, as below:

- *Reqs* is the set of all Requirements.
- *Cmpts* is the set of all Components.
- *SG* is the set of all Soft Goals.
- *Optns* is the set of all Options.
- *Claims* is the set of all Claims.
- *Genrs* is the set of all Scenarios and Assumptions and External Standards (we refer to this group as 'Generators').
- *Assmpns* is the set of all Assumptions.
- *DP* is the set of all Decision Problems.
- *Outcomes* is the set of all Internal Standards and Decision Outcomes and Constraints (we refer to this group as 'Outcomes').
- *Externals* is the set of all External Standards.

We don't make assumptions about disjoint-ness of subsets. For example, it's possible for Claims to be part of a set of Claims that support requirements-related entities such as Requirements and Soft Goals, or part of a design-related set of Claims that support Options or Outcomes. We

don't assume that these are necessarily disjoint sets[2]. We use a lower-case letter to refer to an individual instance of an entity (eg, *Cmpts* is the set of all components; c is a single component where $c \in Compts$.

We count distinct relationships only; where duplicate relationships exist between *(x,y)* we count only one. This follows the standard semantics of a set.

The requirements- and design-oriented metrics we propose are described in the following sections. In the following definitions, we write $M(c)$ to refer to a metric M calculated for component c.

3.4.2 Requirements metrics

In this section we define the metrics we're using to assess requirements-related activity, using the relationships illustrated in Figure 3.3.2.

- Number of Requirements (NR). NR(c) is defined as the number of requirements which have been allocated to a component c.

  $$\mathrm{NR}(c) = |\ \{\ x \mid (x,y) \in G\ :\quad y = c \wedge x\ \in\ Reqs\}\ |$$

  where *Reqs* is the set of requirements. In other words, we are interested in all those relationships (*x,y*) in the graph G which connect component c (indicated by $y{=}c$) with any entity from the set of Requirements (indicated by $x \in Reqs$). We do not specify a label for the edge between x and y since only one label is possible (the 'allocated-to' relationship).

  We expect that a greater number of requirements linked to a component will have some effect on the component's volatility, but we do not make predictions about the direction of this relationship. A greater quantity of allocated requirements could result in more changes propagating to the given component as a result of requirements change; we would expect a positive correlation if this is the case. On the other hand, a component with an unusually high number of linked requirements may be an 'important' component - perhaps implementing key business objectives - which is less likely to change than more peripheral functions. This would result in a negative correlation between change-proneness for c and NR(c). An equally plausible position is that the component with many requirements lies at the intersection of several functional areas and is therefore more vulnerable to changes impacting on different areas of functionality.

- Number of Soft-Goals (NSG). NSG(c) is defined as the number of requirements which are soft-goals allocated to component c.

  $$\mathrm{NSG}(c) = |\ \{\ x \mid (x,y) \in G\ :\quad y = c \wedge x\ \in\ SG\}\ |$$

  where SG is the set of soft goals.

  We hypothesise that a greater quantity of soft-goal links will result in a greater number of changes made to a component. This is because, as discussed in Section 2.2.2, soft-goals are notoriously difficult to identify, quantify and communicate before implementation starts, and difficult to 'design' even when they have been clearly identified. Soft-goals thus tend to emerge or evolve as the project progresses, resulting in repeated adjustments. A positive correlation between change-proneness for c and NSG(c) could indicate this is the case.

[2]In practice, however, on our case study (introduced in Chapter 4) these do turn out to be disjoint subsets.

- Strength of Requirements Rationale (SRR). SRR is intended to examine to what extent requirements linked to a particular component have a recorded rationale. 'Generators' are considered to form requirements rationale; this could be a user Scenario, an External Standard or an Assumption. We also consider Claims to be part of the rationale for a requirement where they are directly linked to a requirement via the *supports* relationships.

  Formally we can define SRR for component (c) as follows:

  $\text{SRR}(c) = \mathit{ReqsGenerators}(c) + \mathit{ReqsClaims}(c)$

  where

  $$\text{ReqsGenerators}(c) = |\ \{\ x \mid (x,y,l) \in G\ :\ y \in\ \mathit{Reqs}(c) \wedge x\ \in\ \mathit{Genrs} \wedge l = \text{'generates'}\}\ |$$

  and

  $$\text{ReqsClaims}(c) = |\ \{\ x \mid (x,y,l) \in G\ :\ y \in\ \mathit{Reqs}(c) \wedge x \in\ \mathit{Claims} \wedge l = \text{'supports'}\}\ |$$

  and

  $$\text{Reqs}(c) = \{\ x \mid (x,y) \in G\ :\ y = c \wedge x\ \in\ \mathit{Reqs}\}$$

  and where *Reqs* the set of all requirements. Less formally, if r is a requirement linked to c, $\text{SRR}(c)$ is the total sum of External Standards, Assumptions and Scenarios linked to r by the relationship 'generates' plus the Claims linked to r by the relationship 'supports'.

  We anticipate that this metric may have some relationship with change-proneness, but we do not attempt to predict the nature of that relationship. A requirement with a larger number of recorded sources - particularly external ones - may be more susceptible to change than others. Changes which enter the system from external sources (such as changes to regulations or business operating environments) are perhaps more likely to impact upon a component with many high-level sources than on one with few or none. We would expect to see a positive correlation between change-proneness for c and $\text{SRR}(c)$ if this is the case. On the other hand, the presence of many sources may also play a stabilising role, essentially identifying those issues for which users are not empowered to request changes; this would result in a negative correlation with change-proneness.

  Where one component, c, is linked to two requirements, R_1 and R_2, and both R_1 and R_2 are linked to a single entity (a standard, a scenario, or an assumption), S_1, then we only count S_1 once (this hypothetical data structure is illustrated in Figure 3.6.

- Average Strength of Requirements Rationale (ASRR). This metric is based on the SSR but a mean figure of rationale recorded per requirement is calculated, so that we may compare the relative weighting of requirements rationale between entities with different numbers of requirements.

  $ASRR(c) = \frac{\text{SSR}(c)}{\text{NR}(c)}$

- Number of Conflicting Requirements (NCR). This metric is intended to provide some measure of the amount of conflict amongst requirements linked to component c. This is calculated as follows:

  $$\text{NCR}(c) = |\ \{\ x \mid (x,y,l) \in G\ :\ y \in \mathit{Reqs}(c) \wedge x \in \mathit{Reqs} \wedge l = \text{'conflicts-with'}\}\ |$$

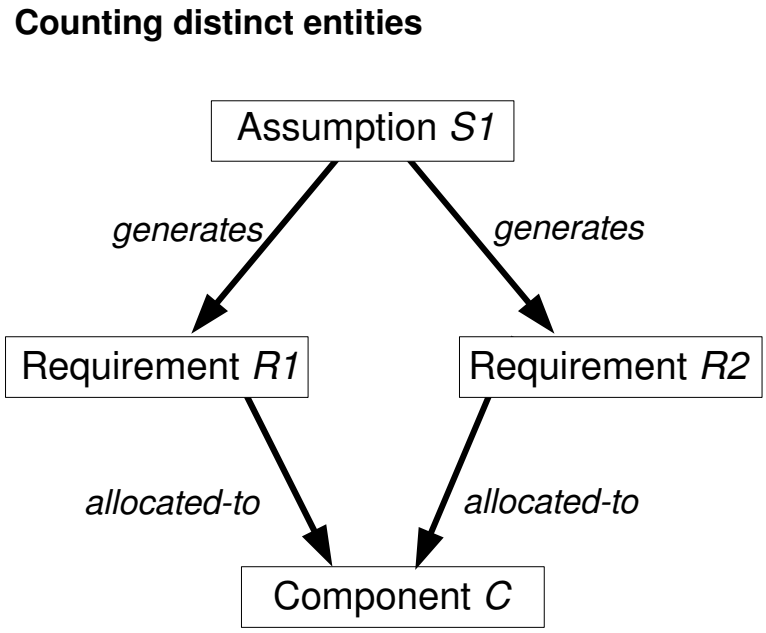

Figure 3.6: Diagram showing hypothetical structure in which one standard may be linked twice to the same component. We count each distinct entity only once.

where *Reqs(c)* is - as has already been defined earlier - the set of requirements allocated to component c, and *Reqs* is simply the set of all Requirements.

An important note about this metric: if r is a requirement linked to c, NCR(c) counts the number of target requirements conflicting with r. This is different to the number of requirements linked to c which experience conflicts. Consider, for example, that components *c1* and *c2* are each linked to a separate requirement (*r1* and *r2*, respectively) that conflict with other requirements. This hypothetical example is shown in Figure 3.7. Requirement *r2* conflicts with many other requirements whilst *r1* only conflicts with one other requirement. The NCR values for *c1* and *c2* will be different: NCR(*c1*) = 1 and NCR(*c2*) = 3. NCR is thus weighted, returning a higher value for requirements which conflict frequently than for requirements which conflict with only one other requirement.

We only count direct relationships; i.e., we don't continue to count transitive relationships - requirements which conflict with requirements which conflict with r. We ignore the direction of the relationship for this metric - if it participates in a 'conflicts-with' relationship, then a requirement is counted as conflicting with one other requirement, irrespective of whether it is the 'source' or 'target' of that relationship (indicated by the direction of the arrow on the model in Figure 3.3.2).

We hypothesise that a higher figure here is likely to lead to a higher rate of change for a component, given that conflicting requirements are fertile ground for change requests, design compromises and iterative 'tweaks' and adjustments to the system. This would manifest itself in a positive correlation between change-proneness and NCR[3].

[3]It is tempting to assume that conflicts should not be present in the requirements specification, but conflicts do occur in real-world scenarios [49]. Easterbrook argues that the uncovering and communication of conflicts (such as requirements conflicts) can even serve a purpose; he proposes a framework for recording and resolving such conflicts

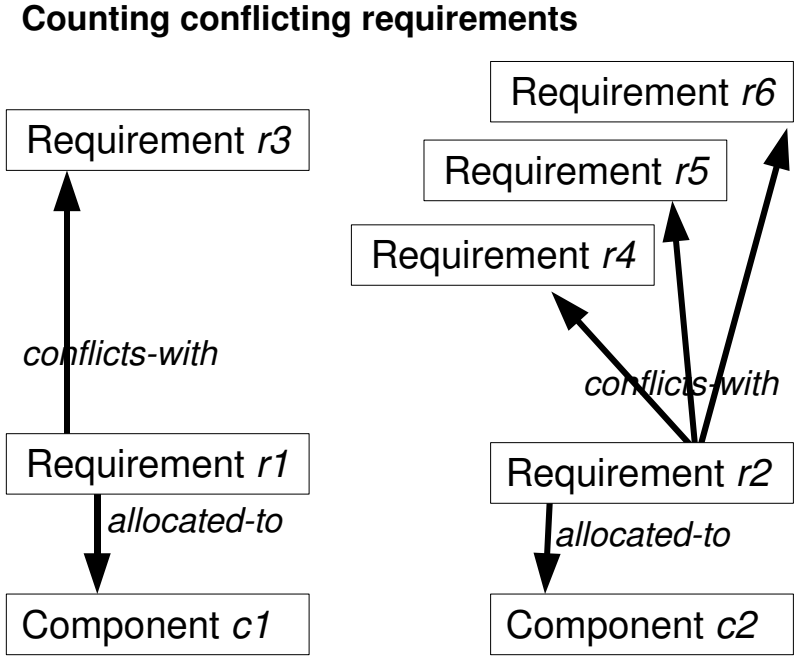

Figure 3.7: Diagram showing hypothetical structure in which two components with the same number of requirements return different values for NCR.

- Requirement Importance(RI). It is assumed that a requirement with sub-requirements has greater impact ('importance') than a requirement with no sub-requirements. This metric is defined as follows:

  $$\mathrm{RI}(c) = |\ \{\ x \mid (x, y, l) \in G\ :\ y \in Reqs(c) \wedge x \in Reqs \wedge l = \text{'has-subrequirement'}\}\ |$$

  In the interests of simplicity, we count only direct sub-requirements, and do not include sub-requirements of the sub-requirements.

  We predict that this metric will have an effect on the number of changes a component experiences but do not hypothesise about the nature of that relationship. A requirement with many sub-requirements is more likely to be high-level, reflecting key business objectives, and less likely to experience changes as a result. A negative correlation between change-proneness and RI could imply that this is the case. On the other hand, an important requirement is likely to affect a greater number of components, with a greater potential to spread change ripples. A positive correlation between RI and change-proneness could imply that this is the case.

3.4.3 Design metrics

We hypothesise that a greater quantity of design activity will have an effect on component change-proneness, but (except where noted for specific metrics below) we do not predict what the effect may be. Investment in design work is traditionally seen as protective against some types of changes later in the project. An increased level of design activity represents more detailed investigation

clearly.

of available solutions, a greater proportion of time allocated to research and and extended consideration of the impact on non-functional requirements. Or it could indicate the presence of an extended dialogue with users which improves the understanding of both developers and users of the finished system's required capabilities. All of these activities may help to reduce the overall number of changes later in development, since there are likely to be fewer adjustments due to unforeseen design incompatibilities or user-developer misunderstandings. If so, we expect to see negative correlations between the design metrics below and change-proneness.

Table 3.2: Table summarising our proposed metrics for assessing design and requirements activity

Metric	**Description**	**Predicted effect**
Number of requirements (NR)	Number of Requirements allocated to c	No prediction as to direction of correlation
Number of soft goals (NSG)	Number of Soft Goals allocated to c	We predict this leads to increased changes
Strength of requirements rationale (SRR)	Number of Claims that support, plus Generators that generate, Requirements allocated to c	No prediction as to direction of correlation
Average strength of requirements rationale (ASRR)	SRR for c divided by NR(c)	No prediction as to direction of correlation
Number of conflicting requirements (NCR)	Number of Requirements that conflict with Requirements allocated to c	We predict this leads to increased changes
Requirement importance (RI)	Number of sub-Requirements for Requirements allocated to c	No prediction as to direction of correlation
Number of outcomes (NOut)	Number of Decision Outcomes affecting c	No prediction as to direction of correlation
Strength of outcome rationale (SOR)	Number of Claims supporting, plus Generators generating, Outcomes affecting c	No prediction as to direction of correlation
Average strength of outcome rationale (ASOR)	SOR for c divided by NOut for c	No prediction as to direction of correlation
Number of External Standards (NES)	Number of External Standards which can be linked to c, via a Requirement or a Decision Outcome	No prediction as to direction of correlation
Number of Decision Problems (NDP)	Number of Decision Problems linked to Outcomes affecting c	No prediction as to direction of correlation
Number of Assumptions (NA)	Number of Assumptions upon which Outcomes affecting c rely	We predict this leads to increased changes
Number of Options (NOpt)	Number of Options linked to Decision Problems for c	No prediction as to direction of correlation
Average No. of Options per Decision Problem	NOpt for c divided by NDP(c)	No prediction as to direction of correlation

Continued on next page

Table 3.2 – continued from previous page

Metric	Description	Predicted effect
(ANODP)		
Strength of Option Support (SOS)	The number of Claims that support Options for c	No prediction as to direction of correlation
Average Strength of Option Support (ASOS)	SOS for c divided by NOpt(c)	No prediction as to direction of correlation
Strength of Option Opposition (SOO)	The number of Claims that oppose Options for c	No prediction as to direction of correlation
Average Strength of Option Opposition (ASOO)	SOO for c divided by NOpt$c$	No prediction as to direction of correlation

On the other hand, perhaps an elevated level of visible design activity results from difficulty in finding a solution that fully meets requirements, resulting in more compromises forced onto the design. If this is the case, higher levels of design work could indicate components which will experience *more* change requests. This would result in a positive correlation between change-proneness and design metrics.

We are hypothesising that 'design activity' could affect specifically design-related changes. Conventional expectations are that extra work in the design phase should help to alleviate such problems, by researching the options in greater depth and exploring user needs in more detail. Other types of changes (such as those forced on the system by external environments/platforms/markets) and faults in the system, are unlikely to be affected by increased effort in design activities.

Within this scheme, we define 'design activity' as activity involving the investigation of design decisions, various solutions to issues, claims to support or oppose solutions and the rationale for those claims. We plan to characterise this activity using the metrics outlined below:

- Number of Outcomes (NOut). This metric aims to express the number of decision outcomes that affect a given component. Formally, we define NOut for component c thus:

  $$\text{NOut}(c) = |\ \{\ x \mid (x, y) \in G\ :\quad y = c \wedge x \in \mathit{Reqs} \wedge x\ \in \mathit{Outcomes}\}\ |$$

  We do not specify the relationship label here since there is only one type of relationship linking components to 'Outcomes'. As defined earlier, 'Outcomes' is the set of Internal Standards, Decision Outcomes and Constraints. This is the simplest measure of the level of design activity relevant to a component.

  We hypothesise that a higher value for NOut suggests a component has been subjected to more conscious design activity and decisions than other components. A negative correlation between NOut and change-proneness might suggest that this effort reduces design-related changes. Conversely, a higher value for NOut could indicate that more and more constraints placed on the component as a result of design decisions have lead to compromises and trade-offs.

- Strength of Outcome Rationale (SOR). SOR expresses the total quantity of rationale recorded for an outcome o. We define rationale in this case as 'Generators' that are linked to 'Outcomes' via the 'generates' relationship and Claims that support 'Outcomes'.

  $$\text{SOR}(c) = \mathit{OutsGenerators}(c) + \mathit{OutsClaims}(c)$$

where

$$\text{OutsGenerators}(c) = |\ \{\ x \mid (x,y,l) \in G\ :\ y \in\ Outcomes(c) \wedge x \in\ Genrs \wedge l = \text{`generates'}\}\ |$$

and

$$\text{OutsClaims}(c) = |\ \{\ x \mid (x,y,l) \in G\ :\ y \in\ Outcomes(c) \wedge x \in\ Claims \wedge l = \text{`supports'}\}\ |$$

and

$$\text{Outcomes}(c) = \{\ x \mid (x,y) \in G\ :\ y = c \wedge x \in\ Outcomes\}$$

A larger number of recorded sources may make an outcome more susceptible to change than others, which would be indicated by a positive correlation with change-proneness. Alternatively, the presence of many sources may also play a stabilising role; this could be indicated by a negative correlation.

- Average Strength of Outcome Rationale (ASOR). This metric is based on the SOR but a mean figure of rationale recorded per outcome is calculated, so that we may compare the relative weighting of rationale between components with different numbers of outcomes.

  $ASOR(c) = \frac{\text{SOR}(c)}{\text{NOut}(c)}$

- Number of External Standards (NES). This metric quantifies the total number of external standards with which a component is associated. External Standards may play a role in 'generating' either Requirements or Decision Outcomes.

  $\text{NES}(c) = ReqsExternals(c) + OutcomeExternals(c)$

  where

  $$\text{ReqsExternals}(C) = |\ \{\ x \mid (x,y,l) \in G\ :\ y \in\ Outcomes(c) \wedge x \in\ Externals \wedge l = \text{`generates'}\}\ |$$

  and

  $$\text{OutcomeExternals}(C) = |\ \{\ x \mid (x,y,l) \in G\ :\ y \in\ Requirements(c) \wedge x \in\ Externals \wedge l = \text{`generates'}\}\ |$$

  This metric spans both requirements- and design-related entities. More specifically than assessing 'rationale', this metric identifies only those entities which can be shown to have a dependency on a standard or regulation outwith the team's control. External sources can be a significant motivator of change (we discussed reasons for changes in Chapter 1). However, many standards, once widely adopted, are under considerable pressure *not* to effect changes for fear of 'breaking' or rendering obsolete previous work which adheres to it. Adherence to an external standard may thus confer stability, or be the source of extra changes.

- Number of Decision Problems (NDP). This is a simple count of the number of Decision Problems which can be linked to a component c. This is an indirect relationship; Decision Problems are linked to components via the Outcomes they generate. Here is the formal definition:

  $$\text{NDP}(c) = |\ \{\ x \mid (x,y,l) \in G\ :\ y \in\ Outcomes(c) \wedge x \in\ DP \wedge l = \text{`has-outcome'}\}\ |$$

where *Outcomes(c)* is - as has already been defined - the number of outcomes linked to component c.

Many decisions during the design phase may be made implicitly, perhaps without the designer even consciously being aware that a range of solutions exist. The fact that a decision is being consciously recorded and considered implies increased design activity for this component and is likely to trigger a research phase to investigate options and user needs more thoroughly. We may result in fewer change requests later (signalled by a negative correlation with change-proneness), due to initial efforts on selecting the best solution.

On the other hand, some designs will force an increasing series of constraints on the design of the system and raise more issues in turn. This could suggest that the original decision was chosen as the best available, rather than ideal, choice. A positive correlation with change-proneness could indicate that this is the case.

- Number of Assumptions (NA). The NA is a metric expressed for Internal Standards, Decision Outcomes or Constraints which rely upon an Assumption. NA for an component c is calculated as follows:

$$\begin{aligned} \mathrm{NA}(c) = |\ \{\ x \mid (x,y,l) \in G\ :\quad & y \in\ Outcomes(c) \\ & \wedge\ x \in\ Assmpns \\ & \wedge\ l = \text{`relies-on'}\} \mid \end{aligned}$$

  We suspect that a greater reliance on assumptions in the decision-making process could increase propensity to change. The implication is that any change in the status of the assumption renders the Outcome invalid, making change ripples likely. This would be indicated by a positive correlation between NA and change-proneness.

More complex design metrics

The following set of metrics builds further on information gathered about the design process. We can characterise each decision problem to express the extent of options considered (i.e. the number of Options) and how well supported (or opposed) those options were. These metrics are gaining in complexity for the designer who would like to use them to predict where the project's volatility hot spots will arise. However, we believe that a simple characterisation of whether decision problems relevant to a particular component were accompanied by a wide or a narrow choice, and many or few supporting arguments, are still intuitive enough for our aims.

- Number of Options (NOpt). We are interested in the amount of activity associated with a given decision problem. This metric expresses the number of options that can be linked to a component via the decision problems that have affected it. It is re-used in other metrics. NOpt for a component c is calculated as follows:

$$\begin{aligned} \mathrm{NOpt}(c) = |\ \{\ x \mid (x,y,l) \in G\ :\quad & y \in\ DP(c) \\ & \wedge\ x \in\ Optns \\ & \wedge\ l = \text{`has-option'}\} \mid \end{aligned}$$

  where

$$\begin{aligned} \mathrm{DP}(c) = \{\ x \mid (x,y,l) \in G\ :\quad & y \in\ Outcomes(c) \\ & \wedge\ x \in\ DP \\ & \wedge\ l = \text{`has-outcome'}\} \end{aligned}$$

  There could be several reasons why a particular decision problem may be attached to a particularly wide range of options. We suggest a few reasons below:

1. A higher figure may indicate high-risk (i.e., most complex or least understood) areas, which many methodologies recommend are tackled early (for example, the spiral model proposed by Boehm [21] recommends this). Likewise decision problems with a narrow range of options may be better-understood. As has been observed earlier, a decision which has absorbed a disproportionate quantity of research and design effort may experience fewer changes later, particularly if the decision was recognised as a high-risk area and successfully tackled early on (there would be a negative correlation with change-proneness if this is the case).
2. The functionality in question is quite complex and the designer is struggling to find a suitable solution that adequately meets all requirements. This might imply that more trade-offs are necessarily made and the later change requests will be increased. A positive correlation with change-proneness could indicate that this is the case.
3. For this particular business functionality there is a large number of third party components that can complete the job, and the designer is matching up their functionality to the project's functional and non-functional requirements.

- Average Number of Options per Decision Problem (ANODP). This metric is expressed for a component c as follows:

  $ANODP(c) = \frac{\text{NOpt}(c)}{\text{DP}(c)}$

  where both *NOpt(c)* and *DP(c)* have been defined earlier. Less formally, this is the total number of Options which can be linked to a component divided by the total number of Decision Problems linked to the same component.

- Strength of Option Support (SOS). We are interested in the quantity of rationale recorded for each option. This metric expresses the quantity of support behind an option which can be linked to decision problems relevant to component c. This metric is re-used in other metrics. Here is the formal definition:

  $$\begin{aligned}\text{SOS}(c) = |\ \{\ x \mid (x,y,l) \in G\ :\quad & y \in\ Optns(c) \\ & \wedge\ x \in\ Claims \\ & \wedge\ l = \text{`support'}\}\ |\end{aligned}$$

  where

  $$\begin{aligned}\text{Optns}(c) = \{\ x \mid (x,y) \in G\ :\quad & y \in\ DP(c) \\ & \wedge\ x \in\ Optns\}\end{aligned}$$

  DP(c) has already been defined as the set of Decision Problems which are linked to a component (via the 'Outcomes' set).

  Below we suggest some reasons why a decision problem might be accompanied by a particularly high number of supporting arguments:

  1. The decision problem under consideration is particularly important one from a design/business point of view and is given extra attention at design time.
  2. As suggested above (in discussions accompanying the NOpt metric), difficult areas may be more likely to be tackled first and to be allocated a disproportionate amount of design time. When choosing between a variety of different solutions that can perform the required functionality, decisions are increasing likely to be swayed by wider arguments that, for example, consider the impact of a given solution on the non-functional requirements. In this case, we might expect to see that change requests fall as the quantity of support ranged behind decision problems increases.

3. The design decisions are the subject of disagreement. This is likely to lead to larger numbers of arguments put forward by competing groups, and perhaps also more change requests as disappointed groups find that their priorities have not been addressed as they hoped. If there is disagreement in this way, this suggests that, again, no ideal solution exists and trade-offs are being forced onto the project. A positive correlation between change-proneness and SOS could imply that this is the case.

- Average Strength of Option Support (ASOS). This metric expresses how much support was lent to options considered as part of a Decision Problem which can be linked to a component. Formally, we define ASOS for a component c thus:

  $ASOS(c) = \frac{\text{SOS}(c)}{\text{NOpt}(c)}$

- Strength of Option Opposition (SOO). This metric expresses the quantity of opposition behind an option which can be linked to decision problems relevant to component c. This metric is re-used in other metrics. Here is the formal definition:

  $$\text{SOO}(c) = |\ \{\ x \mid (x, y, l) \in G\ :\quad y \in\ Optns(c) \wedge x \in\ Claims \wedge l = \text{`oppose'}\}\ |$$

  where

  $$\text{Optns}(c) = \{\ x \mid (x, y) \in G\ :\quad y \in\ DP(c) \wedge x\ \in\ Optns\}$$

  DP(c) has already been defined as the set of Decision Problems which are linked to a component (via the 'Outcomes' set).

  A greater number of opposers ranged against options could imply that some/many options considered for a particular solution are not a good match - or that their selection is controversial - and design compromises may be necessary. On the other hand, it could imply a particularly thorough exploration of the design space, leading to fewer future requests for change. As for supporting arguments (discussed above), if several solutions match the project's requirements then supporting or opposing arguments may be made regarding the wider context. A negative correlation with change-proneness could indicate this is true, whilst a positive correlation would suggest compromises have been forced onto the project.

- Average Strength of Option Opposition (ASOO). This metric expresses how much opposition was ranged against options considered as part of a Decision Problem (indirectly) related to component c. Formally, we define ASOS for a component c thus:

  $ASOS(c) = \frac{\text{SOS}(c)}{\text{NOpt}(c)}$

3.4.4 How well do these metrics meet our requirements?

In Table 3.3 we evaluate how well this proposed collection of metrics meets the requirements we described in Section 3.2.4.

3.5 Practical requirements for the modelling framework

This modelling framework is intended to represent a minimal amount of project information. As we have mentioned previously in our requirements for the framework, we intend to place the smallest possible burden of data collection on the project, and to utilise where possible information that

Table 3.3: Table evaluating how well our metrics meet our requirements (which were introduced in Section 3.2.4)

Requirement	**Evaluation**
1. Draw on data from an advanced design	The metrics and models require a complete set of requirements, and a reasonably complete design - including consideration of major issues, options available and advantages and disadvantages in each. Proposed solutions for issues should generally be available. A high-level class design - including knowledge of where main functionality will be sited - allows designers to tell where requirements links will fall and which components will be affected by high level decisions.
2. Identify larger requirements or soft goal burdens.	The NR metric is a simple expression of quantity of requirements links. NSG quantifies links with soft goals.
3. Identify external sources	The NES metric specifically counts the number of External Standards associated with an entity, either via a Requirement or via an Outcome.
4. Identify entities resulting from design issues	NDP is a simple expression of the number of design issues linked to a component
5. Identify more challenging design issues	NA, NOpt, ANODP, ASOS, SOS, SOO and ASOO all characterise some feature of a design issue, including the number of available options, quantity of arguments supporting or opposing the options and the general reliance on assumptions. ASOR and SOR give an indication of how many factors were considered in resolving an issue.
6-7. Simple to understand, use existing data	Our requirements-related metrics meet this constraint more effectively than our design-related metrics. Metrics such as NR and NSG are simple counts of information which is usually already known (requirements and soft goals, and how they link to components). Design-related metrics are less simple, and do not necessarily rely on data which is readily available to a designer in numeric form. However, we expect that designers would have an idea of which issues had proved most difficult to resolve, and that any correlations we find can be translated into relatively easy heuristics.
8. Expressed in relation to a component.	All metrics are expressed in relation to a component.

is likely to be available without any special data gathering effort. We envisage predictions about likely change-proneness to be most useful to a project manager wishing to achieve maximum return on investment (ROI) for efforts such as traceability, who would not wish to undermine the ROI by embarking upon additional data-gathering tasks.

Table 3.4: Minimum requirements to be satisifed by an organisation to populate our models

Data Required	**Likely Availability**
Minimal model	
List of components/classes	This information should be automatically generated.
Requirements & the components they affect	Many projects commonly store this type of simple traceability link already. For those that don't, a developer with project knowledge should be able to create these links relatively quickly.
User scenarios motivating requirements inclusion & the requirements they affect	User scenarios are typically available to most projects, as interviews with users and/or use cases frequently form part of requirements elicitation.
Major decisions made, the components they affect & the requirements, decisions or options that generated them	For our case study we extracted this information from free text documentation. Many developers will have this knowledge implicitly but it may not be recorded formally. This can relatively quickly be reconstructed with multiple developers' input during or shortly after the design process (this type of knowledge will likely fade over time). Capturing these data can be very useful for other reasons - e.g., avoiding future re-work if changes are requested and the original reasons for a design choice cannot be recalled.
Claims made by stakeholders to motivate requirements inclusion & the requirements they affect.	These may or may not be captured already by a project. We extracted this type of information from free text design documents. We believe that many projects may have this information available in free text, or not at all. A developer with project knowledge should be able to record many of these relatively quickly, but having an accurate project

Continued on next page

Table 3.4 – continued from previous page

Data Required	Likely Availability
	knowledge is key here; it may require input from several people to ensure that the spectrum of views is recorded. This is likely to be amongst the more time-consuming entities to record.
External standards & the requirements/ claims/ outcomes they affect	We believe that relevant external standards would be known to developers on the project. Our expectation is that most projects will be affected by a relatively small number of external standards, so this data should be relatively easy to capture.
Major assumptions made & the requirements they affect	Assumptions are more difficult to capture, since they are frequently implicit. We extracted this information from free text documents for our case study. This is likely to be amongst the more time-consuming items to collect.
Internal standards developed in-house, the components they affect and the decisions that affected their generation	As with external standards, we expect that most projects will produce a relatively small number of internal standards, that will be known to developers. This information should be quick and easy to capture.
Decision outcomes & constraints, the components they affect & the decisions that affected their generation	Lists of outcomes from decision problems. For our case study, this was extracted from free text design documents. This type of data could probably be extracted from the existing forms (such as technical internal designs, free text ocumentation, etc.) by a developer with project knowledge.
Fuller model	
Options considered for each decision	For many projects this data will not already be captured. For a developer or designer involved in the decisions this information should be relatively easy to record, assuming that it is captured during or shortly after the design process.
Soft goals & links to components	This type of information may already be stored. Soft goals tend to be more abstract and high level than requirements - we estimate is that these links would be easier & quicker to generate for a developer with

Continued on next page

Table 3.4 – continued from previous page

Data Required	Likely Availability
	project knowledge than requirements links.
Major assumptions made & the outcomes or claims they affect they affect	As mentioned before, Assumptions may be difficult to capture because they are often implicit.
Claims made by stakeholders to support options or decision outcomes, the options/ decision outcomes they affect & any assumptions, external standards or scenarios that affect them in turn	Major claims made during the decision making process process may be well known to relevant developers & could be easy to record during or shortly after the design phase is completed. We extracted these data on our case study from free text documentation.
Assumptions, external standards or scenarios that affect decision outcomes, and the outcomes they affect	As with Claims that support decisions, (above), this information is unlikely to be formally captured already and may need to be extracted from another source, or may not be captured at all currently. Best results are likely to be achieved by multiple designers to ensure a broad spread of views and rationale is captured.

Because the entities in our models are very high-level, we believe our approach could be applied by projects employing quite different formalisms to represent design and/or requirements data, and that it may be possible to generate some details (e.g., numbers of requirements linked to a component) automatically, depending on the notations and/or tools in use. An absolutely minimal model should consist of Components, Requirements, Decision Problems and the Outcomes/Constraints/Internal Standards that they generate, and the Assumptions, External Standards, Claims and Scenarios that generate Requirements. We select these entities in particular because we can generate most of the simpler metrics from these data, and additionally these metrics performed better in our evaluation (see Chapter 7). Our more complex metrics require some extra information. We summarise the information required by a project wishing to apply our approach in Table 3.4.

We believe that most of this data can be gathered irrespective of the notation currently used to store project data. For example, user scenarios can be extracted easily from UML use cases or from user interviews. Decision problems can be extracted from free text, or on projects where this data is already captured, from more formal representations and models. In some cases the data required may not currently be captured. We assume that, if a sufficiently accurate prediction can be obtained, a project may be motivated enough to capture it.

Our model could be employed on projects utilising a variety of lifecycle models. Our case study demonstrates that the we can employ it productively on a project with an iterative or waterfall development model (since the case study project begins with one model, and continues with another). An agile development that featured a series of short design stages followed by short development stages could also use our models, although the burden of data gathering may be higher as each design stage could presumably result in more data to be added to the model.

As a minimum requirement, however, a project wishing to apply our approach needs requirements and awareness of the source of those requirements and/or any external dependencies they have, and a design or plan with awareness of major decisions that have been taken, what options were considered and reasons for the outcome. A project which does not produce such data before coding begins in earnest (perhaps just for a subset of the system functionality) would not be able to use our system.

3.6 Metric validation

In Section 2.5 we introduced the topic of metric validation and discussed several competing systems for validating newly proposed software metrics. Here we present a summary of the validation of our metrics using some of these systems. By this we hope to demonstrate that our new metrics behave in an objectively logical manner consistent with other metrics.

3.6.1 Validating our proposed metrics with Briand *et al*'s properties

For validating our metrics we employ the framework of properties for software measures as proposed by Briand *et al* [24]. This work is more recent than Weyuker's [151], which has been so far the most widely-used set of validation metrics. Unlike Weyuker's proposals, the framework is more suited to a modular style of development. The authors have explicitly attempted to list properties that are flexible enough to apply to entities other than code modules. This makes it suitable for validating our metrics, which assess the complexity of relationships between modules and other project artifacts. In the interests of completeness, however, we also consider our metrics in the light of properties suggested by Poels and Dedene [119].

The guidelines and properties discussed by Kitchenham *et al* are rather broader in scope, considering potential sources of error in a wide-ranging consideration of measurement, scale, units and instrumentation as well as study validity. We consider issues of measurement, instrumentation and potential sources of error and bias in Chapters 4, 5 and 6.

For convenience, in discussion below we separate our metrics into groups that share certain structural features (also described in Section 3.7). NR, NSG, NCR, RI, NOut, NDP, NA, NES and NOpt are all simple counts of the number of relationships affecting a given entity. SRR, SOR and SOS are all summed counts, where the values of several different metrics are added together. ASRR, ANODP, ASOS, ASOO and ASOR are averages achieved by dividing the value of one metric by another.

All properties discussed below are drawn from [24]. The authors define the system and modules upon which all their properties are based as follows:

> *A system* S *will be represented as a pair* <E, R> *where* E *represents the set of elements of* S *and* R *is a binary relation on* E (R $\subseteq$ E x E) *representing the relationships between* S*'s elements.*
>
> *Given a system* S = <E, R>, *a system* m = <E_m, R_m> *is a module of* S *if and only if* $E_m \subseteq E$, $R_m \subseteq E_m$ x E_m, *and* $R_m \subseteq R$. *As an example,* E *can be defined as a set of code statements and* R *as the set of control flows from one statement to another. A module m may be a code segment or a subprogram.* [24]

As described above, we also model our system as a series of entities connected by binary relationships, and so the definition above is compatible with our models. Whilst our Component entities are code modules, we redefine E to refer to the set of all entities. Code modules are a

subset of all entities. We redefine R as the set of edges connecting our entities. Briand *et al.* refer to code 'module' m; we map this to to a Component in our structures: $m \subset E$.

Below we list the five different categories of metric suggested by Briand *et al* and consider our metrics in light of the associated properties for each category.

Size metrics

The properties Briand *et al* suggest for size metrics are as follows:

- Nonnegativity. $Size(S) \geq 0$. All our metrics meet this requirement; counts and totalled counts have a minimum value of 0, as do averages.

- Null value. This property requires that a value of zero is returned if there are no relationships. This is also true for all our metrics.

- Module additivity. This property requires that the size of the system should be the sum of sizes of disjoint modules, such that any element in the system belongs to one of these modules. This holds true for simple counts and also for summed counts.

  It does not hold true for averaged metrics, however. We use ASRR as an example of an average here. Consider a hypothetical case where component c has 10 requirements, with a further 5 entities generating a subset of those requirements. This gives an SRR of 5, and ASRR of 0.5. Component d, contains all other requirements and generators: 1 requirement with 3 standards supporting it. This gives an ASRR of 3. If we add these two components together, the logical answer should be a total of 11 requirements and 8 supporting claims, giving an SRR of 8 and an ASRR of 0.73. However, adding the two ASRR figures together would give us 3.5. This property does not hold for averaged metrics, and so we do not consider averaged values further in relation to the properties for size.

- From the above metrics, it follows that the size of a system S is not greater than the sum of the sizes of any pair of its modules *m1* and *m2*, such that any element of S is an element of either *m1* or *m2* or both. This holds true for simple and totalled counts, which may be larger than system size (eg, entities may participate in relationships with more than one component, and therefore be counted more than once) but not smaller.

- Monotonicity is a property that follows from the above properties. Briefly, adding elements to the system should not result in a decrease in its size. This property clearly holds true for simple counts, and also for totalled counts.

- Finally, it also follows that the size of a system built by merging such modules cannot be greater than the sum of the sizes of the modules due to the presence of common elements. This holds true for simple and totalled counts.

Length

Briand *et al* [24] proposed 'length' as a new system characteristic. The overall value for the system is the highest value for any single component. Length metrics satisfy the following properties:

- Nonnegativity. Holds for all our metrics.

- Null value. Holds true for all our metrics.

- Non-increasing monotonicity for connected components. Let S be a system and c a component within that system which can be represented by a connected component on the graph representing S. Briand *et al* explain that two elements of S belong to the same connected component if there is path from one to the other in the non-directed graph that we get if we remove directions in the arcs from the graph that represents S. In our system, we interpret 'connected component' to be the structure composed of all components which can be linked to component c, irrespective of relationship direction. Adding relationships between modules of the connected component should not increase the length of S as a whole.

  This property does not necessarily hold true for simple counts and for totalled counts. It does hold true for NR, NSG, NOut, NDP and NOpt. This is because the only way to increase these counts is to add a relationship to a new entity outside the existing connected component (e.g., the only way to increase NSG is to introduce a new Soft Goal to the connected component, since all Soft Goals within the connected component have already been counted).

  However, it does not hold true for metrics that count relationships between entities of the same type: NCR and RI. This is because it is possible to introduce a meaningful new relationship that increases the value of the metric between entities already inside the connected component. For example, consider the case that component c is linked to two requirements, *r1* and *r2* (see the 'before' section in Figure 3.8). All three entities thus belong to the same connected component. Neither requirement currently participates in the *has-subrequirement* relationship or the *conflicts-with* relationship. If we add a *has-subrequirement* relationship between *r1* and *r2* (as in the 'after' section of Figure 3.8, the RI metric for c increases from zero to a non-zero value. If the maximum value of RI throughout the system as a whole was zero (i.e., there were no other *has-subrequirement* relationships, this would increase the system length, violating the property. Likewise, if we add a *conflicts-with* relationship between *r1* and *r2*, the NCR metric also increases from zero to a non-zero number.

  Similarly, it is also possible to increase the values of SRR, SOR, NA, NES, SOS and SOO without introducing new entities to the connected component. This is because these metrics assess relationships featuring entities which may participate in more than one type of relationship. The entities may already be present in the connected component as the result of another relationship. For example, consider again the hypothetical case shown in Figure 3.8. External Standard *es1* and Decision Outcome *do1* already both participate in the connected component, but it would be possible to add a relationship between them. Doing so would increase the SOR metric from zero to a non-zero value.

  Thus this property does not hold true for RI, NCR, SRR, SOR, NA, NES, SOS and SOO; we don't consider these metrics any further for length-related properties.

  Since this property does not hold true for metrics SRR, SOR, SOS and SOO, it also does not hold true for metrics based on these: ASRR, ASOR, ASOS and ASOO. For example, adding a link between a claim that was already present in the connected component and an outcome already in the component would increase ASOS from a potential value of zero (if the option did not already have any support) to a non-zero value.

  However, the property does hold true for ANODP - there are no relationships we can add between entities already present in the connected component that would result in an increase (or a decrease, admittedly) in the ANODP figure.

- Nondecreasing Monotonicity for Non-connected Components. This property dictates that if we add relationships between entities of two different, disjoint connected components the

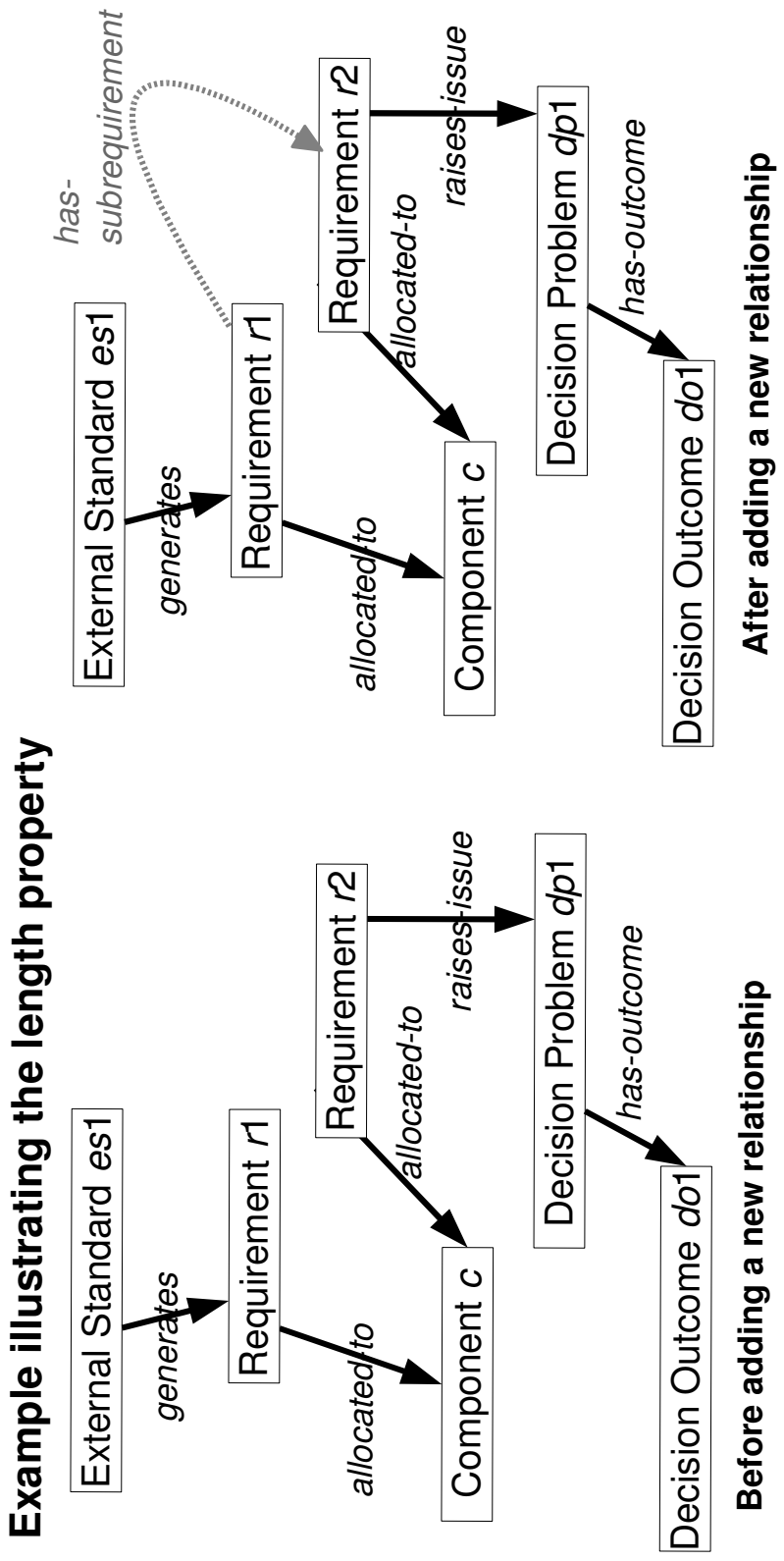

Figure 3.8: Hypothetical scenario in which adding a relationship between two components violates the third property for the length metric.

total length of the system S should not decrease. This property holds true for the remaining simple and summed counts in our metric list: adding new relationships from one connected component to another can only increase the counts of NR, NSG, NOut, NDP and NOpt.

It does not hold true for ANODP. Consider the hypothetical case illustrated in Figure 3.9. In this case, Component *c1* in connected component *cc1* has an ANODP value of 2 (two options divided by one decision problem). If we add a link between decision outcome *do2* in connected component *cc2* and component *c1*, *dp2* and its options are now linked to component *c1* and affect the value of ANODP. Previously ANODP(*c1*) was 2. Now, however, *c1* is linked to two decision problems and three options. This produces a new ANODP(*c1*) of 1.5, a decrease, and thus ANODP does not satisfy the property.

- Disjoint Modules. This final property for the length type of metrics requires that the length of a system made of two disjoint modules is equal to the maximum of the lengths of the two modules. It is not logical to state that the number of requirements in the system (for example) is equal to the single largest value of NR for for any disjoint connected component in our system. This applies to all of the remaining simple and summed counts in our list of metrics, and thus none of our metrics satisfy all the properties for a length-related metric.

Complexity

Briand *et al* [24] propose that a metric exhibiting the following properties could be considered a 'complexity' metric:

- Nonnegativity. This holds for all our metrics.
- Null value. This also holds for all our metrics.
- Symmetry. This property requires that the complexity of a system does not depend upon the convention chosen to represent the relationships between its elements[4]. This holds true for: NR, NES, NSG, and NOut. However, it does not hold true for SRR, NCR, RI, SOR, NDP, NOpt, SOS and SOO, because these metrics rely on the direction of the arc between entities and/or the label attached to the relationship. For example, there are two possible relationships between Decision Problems and Options. The NOpt metric is only interested in counting one of these relationships, so we specify the label of the relationship we are interested in. The RI metric takes the direction of the arc into account, since the direction indicates which requirement is the parent and which the subrequirement which should be counted.

  Averages such as ANODP, ASRR, ASOR, ASOS and ASOO rely upon direct metrics which pay credence to the direction and label of a relationship, so the property does not hold for these metrics, either.

- Module monotonicity. The complexity of system S is no less than the sum of complexities of any two of its modules which have no relationships in common. This holds true for all the metrics which have not been ruled out as complexity metrics thus far. For example, using NOut as an example, if we take two disjoint connected components the total number of outcomes in the system will be at least as many as appear in these two connected components. Similar logic applies to NR, NES and NSG.

[4]This property is related to Weyuker's property 9, which requires that the metric does not depend upon the naming conversion chosen.

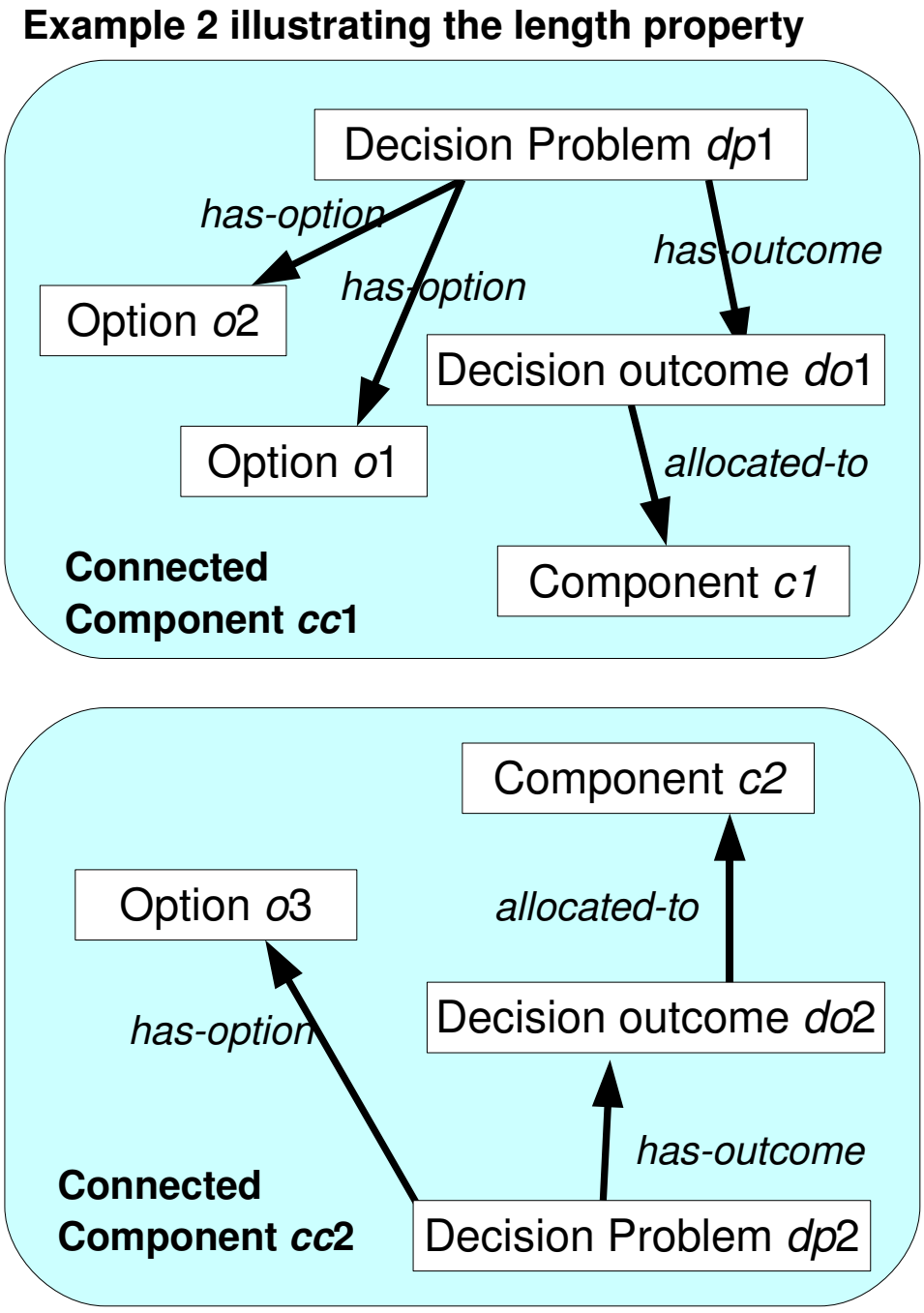

Figure 3.9: Diagram showing hypothetical scenario in which adding a relationship between two components violates the fourth property for the length metric.

- Disjoint module additivity: the complexity for a system composed of two disjoint modules is equal to the sum of complexities of two modules. This property holds true for the remaining metrics: NR, NSG, NES and NOpt.

Cohesion

Our modules do not attempt to assess cohesion, so it is not surprising to find that the properties for cohesion are not satisfied by any of our metrics. The first property for cohesion requires that the resultant values are normalised, and thus fall on a scale between an upper and a lower bound. This does not hold true for any of our metrics; even averaged values hypothetically have no upper bound if the denominator is 1.

Coupling metrics

In Briand *et al.*'s framework, coupling metrics take relationship direction into account, such that inbound and outbound relationships are considered separately. Properties for coupling metrics are discussed below.

- Nonnegativity. True for all metrics.
- Null value. If there are no outbound relationships, then the value of the metric should be zero. This property does not hold true for any of our metrics, since we rely on two directional (inbound) relationships (*affects* and *allocated-to*) to link all our entities either directly or indirectly to components. The concept of 'coupling' does not translate easily to models of component's interactions with designs and requirements. Whilst a component does interact with requirements and design outcomes, these tend to have the effect of imposing data (such as constraints) on components rather than allowing the component to import data for consumption.

We do not consider coupling properties further here.

3.6.2 Validating our proposed metrics with Poels and Dedene's properties

Poels and Dedene's properties [119] are less extensive and tailored than those suggested by Briand *et al.* [24]. However, they argue that, unlike other proposed metrics, their properties are both necessary *and* sufficient, which not only ensures that proposed metrics can claim validity in terms of internal logic, but goes some way towards proving that metrics are meaningful or useful.

Poels and Dedene argue that any function $\delta : X \text{ x } X \rightarrow \mathbb{R}$ is a 'measure of distance' if it satisfies the following basic properties:

- Non-negativity. As described for Briand *et al.*'s metrics above, all our metrics satisfy this property.
- Identity. $\forall x, y \in X : \delta(x, y) = 0 \leftrightarrow x = y$ This property does not logically hold true for all our metrics. Whilst a 'distance' can be determined between any two entities which exist inside the same connected component[5], we have no means for determining 'distance' (i.e., a function for determining the value of some metric) between any two metrics from disjoint components. Where no path exists to link two given entities the we have no metric to describe this.

[5]We borrow the definition of 'connected component' from Briand *et al.* here [24]. See 3.6.1 for a definition.

Table 3.5: Table summarising types of metrics (as proposed by Briand et al) and whether those types are fully satisfied by the metrics proposed here

Briand *et al.*			
Type of metric group	Simple counts (NR, NSG, RI, NOut NDP, NA, NES, NOpt)	Totals (SRR, SOR, SOS)	Averages (ASRR, ANODP, ASOS, ASOR, ASOO)
Size metrics	Yes	Yes	No
Length metrics	No	No	No
Complexity metrics	satisfied only by NR, NSG, NES & NOpt	No	No
Cohesion metrics	No	No	No
Coupling metrics	No	No	No
Poels and Dedene			
N/A	No	No	No

- Symmetry. $\forall x, y \in X : \delta(x, y) = \delta(y, x)$ Again, this property does not hold true for any of our metrics, because the values for our metrics are determined by relationship direction and are therefore not symmetrical.

- Triangle inequality. $\forall x, y, z \in X : \delta(x, y) \leq \delta(x, z) + \delta(z, y)$ This property also does not logically hold true for our metrics. Our metrics represent all values in relation to individual components, such that we have no meaningful way of comparing values between a triumvirate of components.

Summary of metric validation

Table 3.5 summarises the validation described above. For convenience, metrics which are very similar have been grouped together.

The simple counts and the totals can be considered to be measuring system size according to Briand *et al.*'s collections of properties. A subset of our simple counts (NR, NES, NSG and NOpt) can also be argued to be assessing system complexity. The metrics are counts of the numbers of entities related to given components, so size is a reasonably logical definition. We have argued previously (in Section 2.4.6) that size (such as LOC) can be considered a crude form of complexity metric, and other metrics commonly considered to be complexity metrics focus on counts and sizes (such as WMC, which returns an optionally weighted count of methods [31]).

Disappointingly, however, none of our averaged metrics satisfy a complete set of properties from the framework. Briand *et al.* have criticised previous researchers for proposing metrics, determining that the metrics fail to satisfy fully some basic set of desirable/necessary properties, and not subsequently concluding that the metrics are not suitable measures [24]. They also, however, acknowledge that, as the field of software engineering lacks a single, agreed-upon set of properties or standards for assessing complex software metrics, few concrete conclusions can be drawn. We intend to proceed collecting and examining the values for our averaged metrics, but their primary use will be for visualisation and identification for humans of those components with unusually high values rather than for the detection of correlations.

Unfortunately, none of our metrics meet the basic properties required by Poels and Dedene, which means that our metrics do not benefit from satisfying properties which are both necessary

and sufficient. According to Poels and Dedene, doing so would prove that our metrics can make a claim to usefulness which is not automatically accorded by other properties.

The fact that we can satisfy a subset of one proposed set of properties, but fail to satisfy the basic requirements of another set perhaps reveals quite how little agreement there is within the field of software metrics as to how metrics ought to be validated. We believe that validation with Briand *et al.*'s properties is sufficient that we can have reasonable confidence our metrics behave logically and consistently, according to some objective standard. In Chapter 4 we investigate whether our metrics can be claimed to measure anything meaningful.

3.7 Scales of our metrics

Before attempting any statistical analysis, it is important to have an understanding of the scale types of the metrics we have proposed. All of the metrics we propose above are represented by values on the ratio scale, as they are numeric and can represent the point at which the attribute being measured vanishes with zero.

3.8 Summary

In this chapter we developed models onto which entities and relations from the empirical world can be mapped. We followed the MOM methodology, which requires the development of a model to represent the characteristics of interest and a partial ordering. These should allow us to map between objects in the 'empirical world' (i.e., the actual objects we are interested in measuring) and an 'answer set' of, in our case, numerical values.

Accordingly, models representing requirements and design activity on a project were developed by extracting relationships and entity types from existing schemes, following clearly-laid out criteria. After describing the models, we map them onto a series of metrics to measure the relative complexity of interactions with requirements, soft goals and their rationale as well as, design decisions and rationale, options considered, and claims made to support them. Finally, we examined issues of theoretical validity in relation to our metrics, considering a series of necessary and desirable properties for software metrics to possess.

In the next two chapters we move on to external validation of our metrics, by putting them to work in capturing data from a live case study project.

Chapter 4

A case study

In this chapter we describe the case study from which we populate our metric models. The rest of this chapter is laid out as follows: in Section 4.1 we describe our minimum and ideal criteria for a potential case study, and in Section we consider some possible sources of case study data. In Section 4.3 we characterise the case study that was selected, and in Section 4.4 we discuss issues relating to the design of our case study investigation. In Section 4.5 we move forward to describe how data for our project was gathered from the project. This includes requirements data, information on requirements and the generation of complexity metrics discussed in Chapter 2. In Chapter 5 we will present analyses of the data we have gathered.

4.1 Identifying a Case Study

The case study project selected must meet the following minimum (mandatory) and ideal (optional) properties to ensure that we have all the information needed to conduct a rigorous and meaningful study:

4.1.1 Minimum requirements

Change logs

Access to change logs, so that accurate and complete lists of changes may be collected.

Documentation

Access to project documentation, such as: requirements; design and architecture documents; the software itself; any feasibility studies and/or advisory reports; feedback from users; project management documentation; and development team discussions. This is needed so that the requirements and design models presented in Chapter 3 can be populated with project data.

4.1.2 Ideal requirements

The case study is more likely to generate useful, scalable results if the following, ideal, requirements can also be met:

Design decisions

Project data such as major design discussions and issues will ideally have been documented

Timing

The ideal project would already have delivered some code, or be close to delivering this, to

ensure that a sufficient number of changes have already been logged. The current development stage of the project will determine whether the study characterises change during the maintenance cycle or during pre-release development.

Complexity
The project should be sufficiently complex as to provide generalisable project data, but not so large that data-gathering becomes prohibitively time-consuming.

Users
The project needs to have contact with real-world users to provide feedback, as this is a major source of change for many projects.

Technical personnel
Developers and managers need to be available, as personal contacts may reveal discussions and decisions that have not been formally documented.

4.2 Case studies considered

Several avenues were pursued in the search for a suitable case study. We discuss below several potential sources of case study data.

4.2.1 Commercial software firms

A commercial case study offers a number of advantages to a researcher. Data are usually generalisable to a large subset of other commercial projects. There is sometimes even the possibility to study a new technique/process as it is trialled by an organisation, comparing to similar projects elsewhere in same organisational culture.

However, it can sometimes be very difficult to locate a willing commercial partner. Some firms may suspect that the result will be a reputation-damaging 'critique'. Sometime tight deadlines mean staff are unavailable to co-operate with a research project. Furthermore, many firms involved in bespoke developments come into contact with customers' data and systems, and are contractually obliged to maintain confidentiality.

4.2.2 Student projects

Undergraduates on computer science degree programmes sometimes become participants in software engineering experiments (see, for example, Basili *et al.* [7]). The advantage is the availability of groups of programmers who can be set upon the same task, using perhaps different techniques or processes. However, undergraduates are rarely as experienced as professional programmers, and time constraints mean that assignments rarely reach the level of complexity of a 'real' project. Student projects lack the accompanying pressures of change requests, maintenance considerations and user-reported errors, which limits generalisability of results.

4.2.3 Open source project

Open source projects have formed case studies for much previous research[1]. The term 'open-source' covers a wide range of very different styles of development (just as the term 'commercial' does).

[1]See, for example, Ratzinger *et al.* [127], who studied refactoring in ArgoUML and Spring; Pinzger *et al.* [118] (evolution in Mozilla); Vasa and Schneider [148] (evolution of cyclometric complexity in Tomcat, JDictionary, JasperReports, JEdit and Hibernate); Han *et al.* [65] (change-proneness in JFreeChart); Arisholm *et al.* [2] (dynamic coupling in Velocity); Bouktif *et al.* [23] (change patterns in Mozilla); Hassan and Holt [68] (change propagation in NetBSD, FreeBSD, OpenBSD, Postgres and GCC); Zimmerman *et al.* [158] (co-evolving classes in Eclipse, GCC,

Whilst some open source projects are still created by volunteers contributing code from disparate locations, others are developed by students and/or academics (we address academic projects in Sections 4.2.2 and 4.2.4) or professional programmers employed in a commercial enterprise (we assume this type of project is similar to those addressed in Section 4.2.1). The initial stages of an open source project may be quite different to other projects. Developers are often not co-located; communication is commonly via email and/or forums. Volunteer programmers may be more interested in producing code and less interested in producing documentation; on some open source projects (though not all) design and requirements documentation are not much used. Feature planning may be determined not by a specification that was drawn up at inception, but by the interests and/or skill-sets of the programmers, a 'to-do' list and/or 'bug-list'. A project's unique development style may lead to limited generalisability for us, because we are primarily interested in requirements and design data.

4.2.4 Academic development

Typically an academic project is funded by a grant and evaluated periodically and/or at conclusion for success. This type of project is likely to have quite a lot in common with a commercial project or an in-house support team: experienced developers are employed; and requirements, specification and design stages are undertaken to scope out the project at an early stage. Personnel involved are likely to be immersed in a research environment and more sympathetic to researchers aiming to gather data.

Being based at a university, academic developments are available to us and one project we approached proved a good match with our requirements. The following section characterises the case study which was selected.

4.3 CARMEN Case Study

Our investigation will concentrate on CARMEN (Code Analysis Repository & Modelling for e-Neuroscience) as a case study. This is an e-Science Pilot Project funded by the Engineering and Physical Sciences Research Council (EPSRC) (UK), which aims to create a 'virtual laboratory for neurophysiology' [41].

When complete (at time of gathering data, development is three years into its four-year life-cycle), CARMEN will enable 'sharing and collaborative exploitation of data, analysis code and expertise' [41]. CARMEN will act as a repository for neuroscientists to upload datasets and data-processing tools, which are ultimately available for sharing with other researchers. In Table 4.2 we summarise how well CARMEN meets our requirements as a case study and in the sections below (and in Table 4.1) we characterise aspects of the project's size and complexity in more detail. Graphs depicting some of the information are presented in Section 4.6. We present a pie chart showing the relative proportions of entity types in our models of CARMEN in Figures 4.1 and 4.2.

4.3.1 CARMEN's Problem Domain

The problem domain of CARMEN (neurophysiology) is highly specialised. Data requirements in particular are exacting: a wide variety of file types and tools for generating and processing data are in use and no single standard for data formats is accepted across the field. Files generated can be large and optimisation of system performance is very important.

GIMP, JBoss, JEdit, KOffice, Postgres and Python); Wermelinger and Yu [149] (evolution of Eclipse plug-ins) and Wermelinger *et al.* [150] (social hierarchies of developers on Eclipse).

Some general properties of CARMEN	
Language	Java
Maximum developer team size	7
Total number of unique classes over all snapshots	254
Average number of classes in one snapshot	73
Average class size	144 (in LOC, generated by the tool CCCC)
Total number of requirements	152
Number of soft goals	7
Project duration	Originally 4 years (subsequently extended)
Development methodology	Initially waterfall, refined into iterative cycles
Properties of the most recent snapshot (approx. one year after first release to users)	
Number of classes that we count (see Section 4.5.2)	184
Average LOC per class	178
Total LOC	32933.83
Average CBO for each class	6.86
Average number of methods per class	7.54
Average McCabe cyclomatic complexity per class (figures for each class are sums of values for all methods in the class)	16.92
Properties of our models, populated with CARMEN data	
Total number of entities in our model of CARMEN	856
Total number of relationships in our model of CARMEN	2095
Number of Decision Problems in our model of CARMEN	51
Number of 'Outcomes' (Internal Standards, Decision Outcomes, and Constraints	98
Number of 'Generators' (External Standards, Assumptions, and Scenarios	58
Number of Options	49
Number of Claims Claims linked to Requirements Claims linked to Design-related entities (Options, Outcomes and Claims that support them)	187 9 178

Table 4.1: Table summarising characteristics of CARMEN project

Data uploaded may have widely varying characteristics (eg, format, means of gathering, software used, experimental conditions) which must be captured accurately via metadata to enable searching and collaborations. This requirement initiates a major development effort to create a standard for the minimum metadata necessary to adequately describe each dataset.

A major design decision was the choice of data format for CARMEN (a simplified version of this decision was modelled in a variety of ways in Sections 2.2.2 and 2.2.3). This potentially has substantial repercussions for CARMEN's future use.

Different user groups have different requirements for data sharing, with some happy to share data straight away and other groups needing to guarantee that their data and/or tools cannot be accessed by unauthorised personnel until a later date. For this reason, security of the data and users' confidence in the system's ability to deliver this is key.

4.3.2 Project size and structural complexity

We present some details of CARMEN's size in Table 4.1. Since CARMEN is undergoing development throughout the period of our study, size and complexity are not constant. We provide details

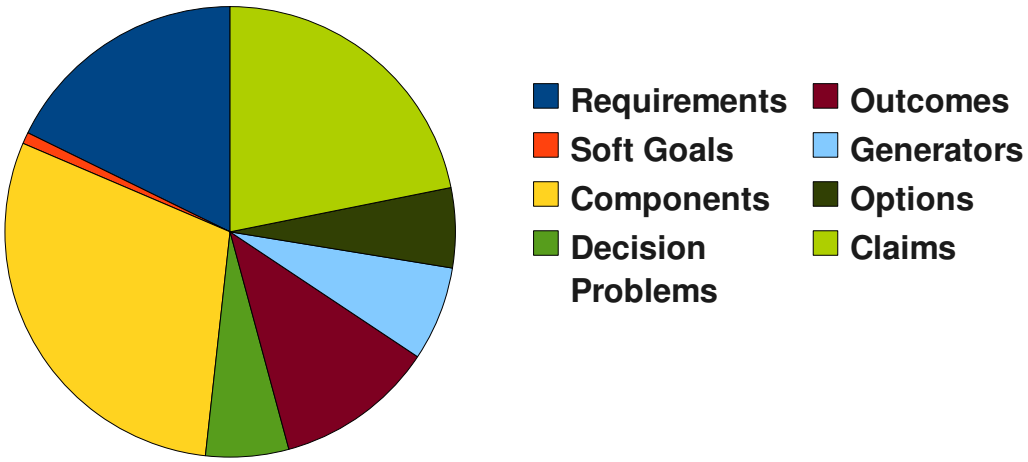

Figure 4.1: Pie chart illustrating relative proportions of different entity types in our models of CARMEN

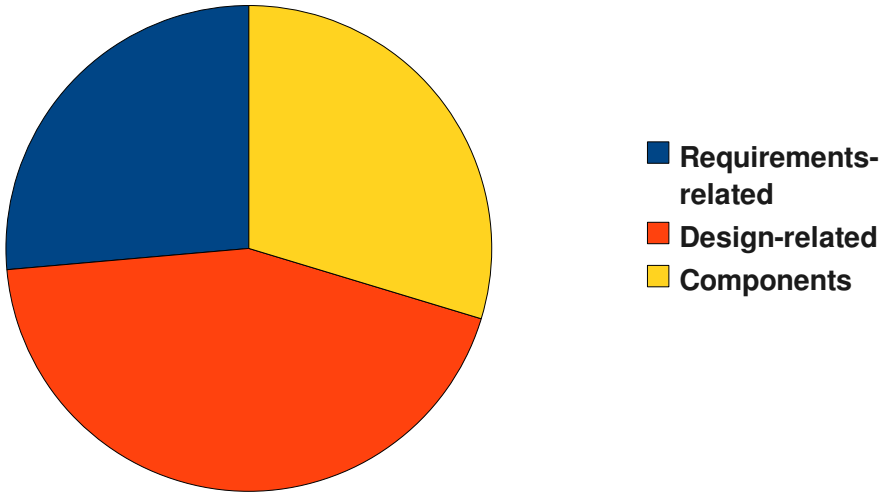

Figure 4.2: Pie chart illustrating relative proportions of requirements- and design-related entities and Components in our case study models

of the average LOC per class, methods per class and cyclomatic complexity over the study lifespan as well as figures for the most recent snapshot we have taken, to provide an idea of growth. CARMEN is a mid-sized project, absorbing the time of a small team of technical staff over a number of years and totalling around 33,000 lines of ode in the most recent snapshot we have studied, spread over 184 classes. We discuss in Section 4.5.2 how we determined which classes should be counted in our study. The 'total LOC' figure in Table 4.1 is not an integer, because a small number of classes exist in two forks of the code simultaneously. To deal with this eventuality, we average the two values together (discussed in Section 4.5.2.

4.3.3 Team Structure

CARMEN's development effort is divided into a number of teams; the team handling Work Package 0 (WP0) are responsible for delivering data uploading and sharing capabilities, user interfaces and a platform for installation of separately developed data analysis tools. The activities of this team are the focus of the case study.

A set of services which will provide data analysis tools are also currently undergoing parallel development by separate teams (work packages 1-6). WP0 developers themselves are geographically distributed with bases at two universities.

Like many projects running over a number of years, CARMEN has experienced some personnel turnover, with several of the technical personnel leaving the project, and others joining, over the course of its development. At maximum size, WP0 is supported by seven developers, a project manager and clerical staff. Technical expertise from developers of third party components (who may be based in the same university) and end-user researchers has also been available to the development team.

The WP0 team is generally an experienced team. Many members of the team have PhDs in relevant subjects.

4.3.4 Development Model

Initially WP0 adopted a waterfall-style process model. This has been gradually restructured into a series of iterations over a spiral-style model, in order to elicit and act upon user feedback as early as possible. Iterations are long, with around two iterations completed in the first two years of the project. Iterations are shorter later in the project, with regular deliveries of updates in CARMEN's third and fourth years.

A full requirements elicitation exercise was conducted early in the project, which involved structured interviews conducted with all the main user groups to determine how they expect to incorporate CARMEN into their research. A single, high-level architecture and design document was produced that drew together specification and design work for major aspects of the system.

A number of sub-projects are subsumed in CARMEN's main development, including developing standards for metadata and specifications for CARMEN's data format. Although design issues and specifications are both addressed in one architecture document for the main development, separate specification documents were produced in the course of development for several of the sub-projects. The requirements, design and specification stages overlap somewhat, both with each other and with some very early implementation work, which suggests that these stages were iterative and feeding into one another. See Section 4.5.8 for more discussion on this.

4.3.5 Architecture

The CARMEN platform produced by WP0 is written in Java. The finished system will make strong use of existing components targeted at similar problem domains and will employ a distributed architecture.

4.3.6 End Users

User groups have been identified in advance. Groups are widely distributed geographically, meaning that face-to-face communications (such as gathering feedback or eliciting requirements) can be time-consuming. Developers from all work packages and end users meet at regular 'all-hands' meetings.

Requirement	**Comparison with CARMEN**
Project documentation, change log and (ideally) personnel	CARMEN's documents are stored in Subversion[145]. The contents of Subversion, CARMEN's web-based project and available management tool, email discussions and development team members and all available for the study.
Project data is well-documented	Requirements elicitation exercises, specifications, design decisions and architecture phases are well-documented and available from Subversion
The ideal project will have delivered some code, or be close to delivering it	CARMEN is already 3 years through a 4 year lifecycle. Initial release (with subset of full functionality gradually being expanded) has been in use for around a year.
The project is of a sufficient size and architectural complexity	WP0 is a relatively small team, with a maximum of seven developers, a project manager and clerical support. There is a potentially large group of users (widely geographically located), a distributed architecture, computationally-intensive processing, stringent security needs and a highly-specialised problem domain.
The project needs to have contact with real-world users	User groups have been identified in advance and are known to the development teams. User feedback is gathered via structured usability sessions. A small number of expert users are more readily available to the development teams for queries. The project's core functionality has been in use by users for around a year (at time of writing).

Table 4.2: Table summarising how CARMEN meets our requirements for a case study

CARMEN is a novel application within its domain, so usability is an important element of the software's design. Feedback on the system's user interfaces is elicited actively through structured usability testing sessions with actual user groups.

4.3.7 How generalisable is CARMEN?

Because we examine a single project only, results from case studies are particularly difficult to generalise [8, 82]. In this section we consider how far any results from a case study of the CARMEN project may be generalised to apply to other projects, and where limitations lie.

Architecture

Results are unlikely to be readily comparable to projects employing a procedural paradigm. Links between components and the design elements or requirements are likely to be substantially different for projects implemented in object-oriented and procedural languages, although further research would be necessary to confirm this.

Development methodology

The methodology followed is important, because different methodologies impose different structures on design and requirements process (our focus of interest). For example, the 'traditional' waterfall lifecycle, the spiral model [21, 19], evolutionary model [58, 104] and the 'eXtreme programming' paradigm organise requirements design activities very differently[2] .

Although CARMEN's development is iterative, the initial iterations are very long and were preceded by high-level, full-scale requirements and design documents. In this respect, CARMEN's development initially resembles that of a linear methodology, although subsequent work is more iterative. We believe that this means CARMEN's methodology is comparable to that of many other projects. Many projects prefer to begin with large-scale requirements and design work so that the full project can be scoped and costed, even if they do not intend to proceed further with a linear model during implementation and testing. And even a 'traditional' linear model rarely experiences an entirely linear progress during development, with no need to return and refine elements which have already been implemented. For this reason we believe that results from the CARMEN case study are representative of a wide variety of projects, as long as their development begins with a relatively full scoping exercise.

On the other hand, we do acknowledge some limits, particularly with regard to 'agile' development models, (such as the 'eXtreme Programming' (XP) paradigm [10]). Because the requirements and design phases are organised very differently for CARMEN and for a project following the XP paradigm, we can't guarantee that our results will be generalisable to this type of project; further studies would be necessary to confirm whether this is possible.

Development phase

The development phase of our case study project has a substantial effect on the generalisability of our results. In particular, there are important differences between projects which are still working through initial development and projects in the 'maintenance' phase. This binary division is not necessarily clear cut. The maintenance phase is traditionally dominated by user requests (to resolve errors, make alterations or add new features). However, many projects see early versions of software released to users before the initial development is fully complete, which tends to result

[2] A software engineering textbook such as Ian Sommerville's *Software Engineering* [138] provides more details on software process methodologies.

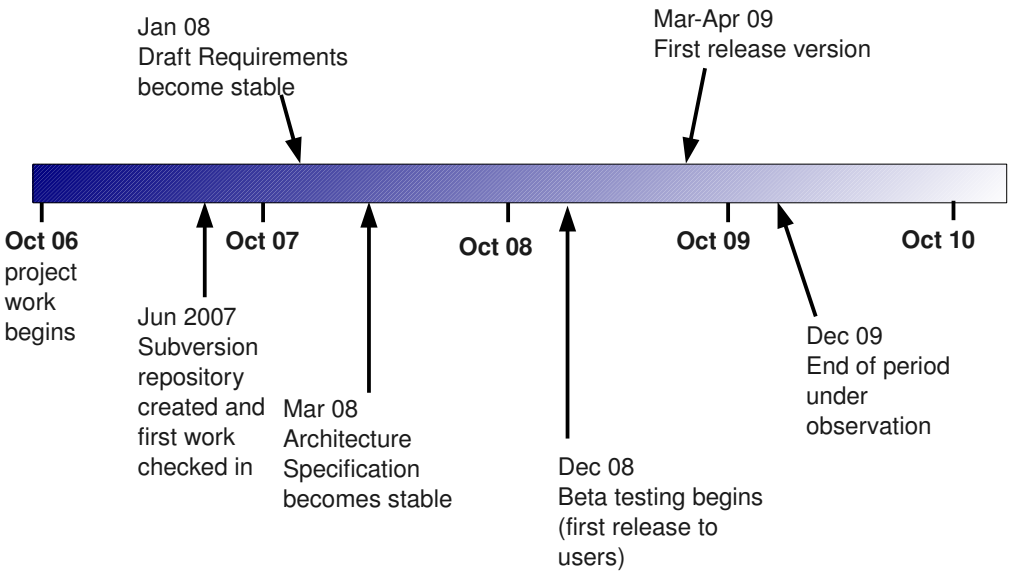

Figure 4.3: Timeline showing major milestones in the CARMEN project

in requests for changes alongside original planned development. Meanwhile maintenance phases frequently involve adding new features to a system as well as 'traditional' maintenance activities.

This means that 'maintenance' phases are not simply confined to responding to user reports of errors, and the initial development phase is not confined solely to adding planned features to the system. However, there are differences between young projects (still in the pre-release stage, or only recently released to users) and projects well into the maintenance phase. These include:

- an older project is increasingly likely to need refactoring [11]
- requests for new features because less frequent as attention is paid to the possibility of a successor
- different principles of prioritising may be applied to a project in initial development (and under time pressure) and a project in maintenance, where contractual obligations are often different
- new features added in the maintenance phase are likely to have a smaller learning curve to understand the problem domain and much less in the way of requirements elicitation than a brand new system
- designers adding new features during maintenance must work within parameters already determined by the existing system (e.g., platform and architecture are already fixed), which do not affect designers at an earlier stage
- refactoring and complexity are much more of an issue during maintenance, as the system is repeatedly patched and altered

CARMEN is a four-year project; a timeline for the project is shown in Figure 4.3. Data gathered from CARMEN covers CARMEN's initial, pre-release development until the release of version 1.1.32, approximately three years into development. Major milestones include a 'beta' release, usable by the target user group, released approximately two years into the project. This means that CARMEN's team have not quite finished adding all the originally-planned features

to the system at the time of writing. However, the system has been exposed to users for over a year (as a usable beta release, through structured feedback sessions and via demonstrations at all-hands meetings). We believe that this mixture makes CARMEN comparable to many projects at a relatively early stage: projects which are either still in their initial development or recently completed. We do not attempt to claim that the results of our study here can necessarily be generalised to projects well into their maintenance phase.

Personnel

CARMEN has experienced some personnel changes. Personnel turnover is not unusual in projects with a duration over several years, and so this does not mean that CARMEN cannot be compared to similar projects.

The WP0 team is an experienced and highly-qualified one. Many members of the team have a PhD in a relevant area and all members have several years of experience. Experience is an important factor in eliciting requirements and designing a system successfully. We do not suggest attempting to extrapolate from the results of this case study project and applying the same results to a case study which may have a less experienced team without further research.

The number of developers actively working on and checking in Java files is relatively small. Previous researchers (e.g., Ratzinger *et al.* [127]) have suggested that a larger number of developers operating the same files is more likely to be associated with re-work and errors. The results of their study did not show that this was an important factor in identifying classes likely to be refactored [127]. However, the size of a development team in itself may have an effect on a project's culture and organisation. Therefore we should exercise caution before attempting to generalise results from a relatively small project like CARMEN to a team which is very large.

Problem domain

As we have mentioned before, CARMEN's industry is a challenging one technically. We do not believe that CARMEN's results cannot be compared to results of studies from other industries, however. Large quantities of data can be found in the routine business applications as well as more specialist software adapted for specific industrial requirements. A mixture of incompatible proprietary and open data formats are not unusual in many industries. So whilst CARMEN is an extreme example in this regard, we believe that results from CARMEN may still be applied to other industries.

Project funding

CARMEN is an academic project, with a funding model which is different to that of many commercial projects. Whilst economic models for commercial projects vary enormously, a relatively common business model sees a software company contracting to deliver the functionality outlined in the specification or requirements document. Failure to deliver functionality on time can result in significant problems (in extreme cases, withholding of payment or legal wrangles over contracts, although perhaps more likely are: extra work; less profit; and/or no possibility of future work from the client).

Thus the software company is motivated to produce a requirements document which is as tightly-specified as possible, because it will potentially form the basis of contractual obligations and final profit margins. The aims are to ensure that it is clear when requirements have been fully discharged; and to make sure that the client cannot request extra features/changes without there

being an impact on the final price. Thus a very carefully-worded requirements document is an investment worth the extra time taken to produce it.

In an academic project, there may be less focus on the contractual obligations imposed by the original requirements document, since funding is not likely to be predicated on this alone. This is not specific to the academic environment, but may arise in other contexts as well. For example, an IT department which completes work for other departments within the same organisation may not employ a watertight requirements specification for each project. The ultimate purpose is to ensure that organisational needs are met rather than return a profit for the IT department. Similarly, developers of boxed products often need to be flexible about the requirements they adopt, as final sales (and job security) depend not on how well the product meets the original concept but on how well the finished product actually meets customers' (constantly-evolving) expectations at time of release. In these situations, specifying a detailed contract at the outset is not the best use of developer time if it is not strictly necessary for the business model.

We are not suggesting that developers in any project are at liberty to respond to user requests indefinitely. Nor are we suggesting that thorough requirements elicitation and initial planning is not important for a project like CARMEN. For any style of project, a failure to implement a system that ultimately meets user needs, preferably on time and/or to budget, will threaten the chances of obtaining further funded work. The difference here is that some funding models are likely to result in a different focus in the early stages of requirements design work. This may affect the values we obtain for our requirements- and design-derived metrics. More research is needed before we can claim that the results from the CARMEN case study can be applied to projects where the funding model is different to CARMEN's.

4.4 Case study design

In this section we develop further how we intend to test our hypothesis, which was outlined in Section 1.1. We divided our hypothesis into three sub-hypotheses:

- H_1: Existing complexity metrics can be used to predict change-proneness.
- H_2: Our metrics can be used to predict change-proneness.
- H_3: A predictive model using our metrics (combined with other types of existing complexity metrics if necessary) to predict development change-proneness outperforms a similar model using the existing complexity metrics alone

We tackle each of these sub-hypotheses in Chapters 5, 6 and 7.

Kitchenham *et al.* [82] have proposed a 'checklist' of steps and issues to consider at each step when conducting a case study, pointing out that all of these issues should be in turn in any study that aims to produce valid results. Table 4.3 presents these issues and how we address them.

4.4.1 Nature of the case study

This study is an exploratory study. We have used a single case study to derive a relatively large number of separate results, and examined them to discover which (if any) is relevant. This is a valid approach, but it is important to be aware of its limitations. Only weak conclusions may be drawn from such a study without further case studies and/or experiments to back up specific results [84]. We discuss this further in Chapter 8.

Ours is an observational study and not an experiment: we are not able to control the variables under investigation, and test the results with variables set to different levels. A well-designed

experiment produces results upon which strong conclusions may be based, unlike an observational case study, which can only point out interesting trends for further analysis and/or corroboration in other studies.

Table 4.3: Details of our case study planning, re-using the checklist proposed by Kitchenham *et al.* [82]

Kitchenham *et al.*'s checklist for case study planning	
What are the objectives of your case study?	To determine whether our metrics can be used to predict change-proneness.
What is the baseline against which you will compare the results of the evaluation?	We will compare the results of our metrics to results from existing metrics which have been tested in previous studies.
What are your external project constraints?	Time constraints.
What is your evaluation hypothesis?	See Section 1.1.
How do you define, in measurable terms, what you want to evaluate (that is, what are your response variables and how will you measure them)?	We are interested in change-proneness. We discuss options for assessing this in Section 4.5.7.
What are the experimental subjects and objects of the study?	Subjects are CARMEN's WP0 team and their end-users (who have had active input into the system). Objects are the components making up the CARMEN platform developed by the WP0 team, their requirements & design artifacts.
When in the development process or lifecycle will the method be used?	Design and requirements data is gathered as soon as they are stable. See Section 4.5.8 for diagrams of data on components.
When in the development or life cycle will the response variables be measured?	Number of changes is assessed throughout the lifecycle.
Can you collect the data you need to calculate the selected measures?	Yes. See Table 4.2.
Can you clearly identify the effects of the treatment you want to evaluate and isolate them from other influences on the development?	We observe the project and gather data without introducing any treatments. Confounding influences that could affect our results, however, arediscussed in Chapters 5 and 6.

Continued on next page

Table 4.3 – continued from previous page	
Kitchenham *et al.*'s checklist for case study planning	
Have you taken adequate procedures to ensure that the method or tool is being correctly used?	N/A
If you intend to integrate the method or tool into your development process, is the method or tool likely to have an effect other than the one you want to investigate?	N/A
Which state variables or project characteristics are most important to your case study?	Change proneness (see Section 4.5.7).
Do you need to generalise the result other projects? Is so, is your proposed case study project typical of those projects?	We discuss issues of generalisability in Section 4.3.7
Do you need a high level of confidence in your evaluation result? If so, do you need to do a multi-project study?	Our study is exploratory only.
How are you going to analyse the case study results?	We intend to use two statistical techniques: regression allows us to search for a linear relationship and Mann-Whitney tests allow us to detect non-linear relationships. See Section 5.2.
Is the type of case study going to provide the level of confidence you require?	Yes.

4.5 Gathering data from the CARMEN project

In this section we explain how we collect sufficient data from the case study to test our hypotheses. This involves determining what is a component, a requirement, a decision problem, etc., and then populating our models with information that replicates CARMEN's components, requirements and design activities.

4.5.1 Technical details of data gathering

All our data are stored in a relational Postgres database. A schema diagram is presented in Appendix A.

A note on CARMEN's source control

CARMEN's development team use Subversion [145] as a version control system. Using this system, files are placed under Subversion's control, and subsequent changes are committed by 'checking in' the files. Subversion tracks the modifications made and the dates they were committed. Each

time a check-in occurs, Subversion increments the *system* version number. This differs to some other version control systems, such as Concurrent Versions System (CVS) [121], which maintains a separate version number for each individual file. Also unlike CVS, Subversion maintains a list of all files that were modified/checked in together[3].

At the time of research, CARMEN's system version number was 1588, meaning that 1588 check-in events had occurred. Each check-in event consists of one or more files (sometimes very large numbers of files may be checked in together).

4.5.2 Components

The first stage is to determine what forms a 'component'. This is the key element of our models, and all of our metrics are expressed in relation to a component.

We have some criteria that a component should meet:

- it should form part of the CARMEN solution
- it should be a source code file (e.g., a .java file) from which we can generate complexity metrics (such as the C&K metrics)
- the list of components should be as complete as possible

CARMEN's log lists 1588 system versions, together totalling 18,764 files checked in. The life of this log spanned CARMEN's development from creation of the original project in Subversion (almost one year after CARMEN's official inception in 2006), covering two years of development. Core functionality had been released as a 'beta' release after approximately one year. At system version 1588, version 1.0 of the system was readily available.

Next we searched the log for any files which did not meet our criteria: i.e., any files which are not code, or that do not implement part of CARMEN's functionality. We included .java files only and excluded others. Files that had been created before the inception of the CARMEN project (looking at date-created comments in file headers) and had not been modified since were excluded from our study, as they did not represent effort on the part of the development team. Files which were clearly marked as having been modified since CARMEN's inception were included.

This left a total of 2616 files in the Subversion logs which we thought were potentially valid components. However, this includes duplicates; if a file has been modified ten times, it will appear in the log ten times. In some cases identification of such duplicates is easy, because path and the file name are the same for both instances. In other cases, the file has been renamed or moved, or its container folder has been renamed or moved, so identifying pairs of duplicates is more difficult. We can't assume that two files with the same name residing in different folders are the same file. First we identify all files with the same name *and* location as duplicates. Next we search for further duplicates using the following strategy:

1. Search for records indicating that a file has been directly moved or renamed within Subversion (Subversion's log indicates this using the syntax 'newfilename.java *from* oldfilename.java').
2. Sometimes a whole parent folder is moved and all the files beneath it are moved recursively, resulting in a single entry in the Subversion log for the parent folder only. We search for revisions where any parent directories have been renamed or moved (using the same *from* syntax), and then check that the files in the new and old locations can be matched up as two versions of the same file.

[3]Efforts to replicate, for research purposes, lists of files that were checked in together under a CVS-controlled system have absorbed researcher time in the past. See, for example, [23] or [158]

3. Sometimes changes to a folder or a file are not noted explicitly by Subversion. For example, in some cases the user deleted a location and checked in a new one, instead of instructing Subversion to perform a move operation. Subversion would not be aware that there is a relationship between the 'old' and 'new' locations and no trail would be produced in the log.

   To locate duplicates resulting from this, we compare files with similar names which either reside in different versions of the same folder, or have the same name in different locations. We examine the headers, where structured comments document file properties and history. If these match, we assume that the file is the same file. In a small number of early files without the structured comment headers we look at the code, looking for matching variable names, method names and comments.

   This matching exercise is purely manual - meaning that some files may have been missed. Only a relatively small number of files with similar names and/or locations were detected.

At the end of this exercise, we have a total of 254 unique components, all of which are Java files implementing some feature of CARMEN's functionality and implemented by CARMEN's development team.

4.5.3 Requirements and soft goals

CARMEN's requirements elicitation exercise centred on structured interviews with user groups, with results compiled into a requirements document. We simply imported all of the requirements from the document.

In a few cases, the requirements document itself presented some user requirements as a single high-level requirement with a series of sub-points. We translated these in our system into a requirement and its sub-requirements.

The requirements document did not explicitly document soft goals. However, the design documentation made frequent reference to key soft goals (particularly usability, security, performance and platform independence), often using them as rationale for decision making. Discussions with the developers revealed a short list of soft goals (including all of the above) that the team thought were important. These were added to our models. The total number of requirements was 152, with an additional 7 soft goals.

4.5.4 Linking requirements and components

In our models requirements are linked to components via the *allocated-to* relationship. CARMEN's team do not maintain a documented mapping to connect requirements to components.

To uncover these links, we examined each of the components in turn, examining comments, method names and code to determine the component's function. We then manually created *allocated-to* links between the requirements and components. This is a relationship with optional N:M participation. In general, we created a link only where a component actually implemented, in full or in part, the required functionality. For example, Requirement r is allocated to Component c. Where a container panel d simply provides a link to display c, we did not create an *allocated-to* link between the r and the container d. However, if the container d performed some work (for example, a permissions check) first before displaying c, then we deemed that the container d had added to the functionality provided by c and we added an *allocated-to* link between r and d.

In addition, we followed some general guidelines:

- We allocated the requirement that CARMEN must provide a web portal to any portal component that exhibits awareness of web functionality, such as generating HTML, handling

URLs and setting HTML styles.

- A soft goal requiring a high level of usability was allocated to any component providing functionality designed to make users' lives easier but not mandated by requirements, such as: bookmarks for favourite services; most recent data lists; help; templates for pre-populating metadata, etc.
- Requirements relating to data storage and/or use of SQL were allocated to any component that explicitly connects to a database or storage module and executes commands or SQL statements.
- We allocate the requirement that 'a user MUST be able to exercise his rights' to any component which explicitly involves carrying out a task which is normally restricted to users with appropriate rights.
- We allocate the requirement that CARMEN provide a GUI to any component addressing the cosmetic appearance of the page, e.g., setting styles or positioning, outputting HTML, etc.
- We assume that any component generating user interface should be allocated to requirements governing 'look and feel'

4.5.5 Design entities

Design-related entities (such as Decision Problems, Options, Outcomes etc.) were populated by reading the design documentation and extracting relevant data. Documents used as sources included:

- CARMEN System Architecture Specification. CARMEN's main design document, drawing together high-level implementation details, options and rationale for decisions and rationale for some requirements.
- Carmen Requirements. Summarises the requirements gathered from users as a series of implementable statements, drawing on data from the Draft Initial Requirements.
- CARMEN: Draft Initial Requirements. Transcripts of user responses to structured questions and interviews.
- Data Formats in CARMEN, and Data Format in CARMEN - V2.0. Two versions of technical reports which address the issue of which (if any) data format should be supported and/or implemented by CARMEN.
- CARMEN Metadata: Ontologies and Software. Specifies the CARMEN project metadata architecture.
- Carmen Data Format Specifications. Specifications for the proposed CARMEN common data type.

We tried to identify decisions that had been made and the options available. Sometimes no options were discussed in the document; this was usually because the decision had not yet been made. The design document described how the system was to be implemented, narrowing down the wide field of possible choices to a specific path. These statements were represented in our models as Constraints. The presence of a Constraint was generally taken as a sign that a decision must have been taken, perhaps implicitly, and a Decision Problem was retrospectively added. Some decisions also documented options, which were accordingly added to our models as Options. The final choice

was modelled as a Decision Outcome. Some Decision Problems resulted in the generation of an Internal Standard; examples include standards governing the minimum data required to describe an uploaded dataset or service (tool).

The design documents also revealed requirements rationale - for example, referencing external standards and documenting assumptions. These were added to the design and requirements models. Further rationale for requirements was gathered by reading transcripts of interviews with user groups. Where an argument had been made (eg, for or against a particular option, or for the importance of a standard) these were included as Claims.

Examples of populated design models

Here we present example paragraphs extracted from the CARMEN System Architecture document, together with the finished and populated structures, to illustrate how information was extracted from free text documents and translated to our structures.

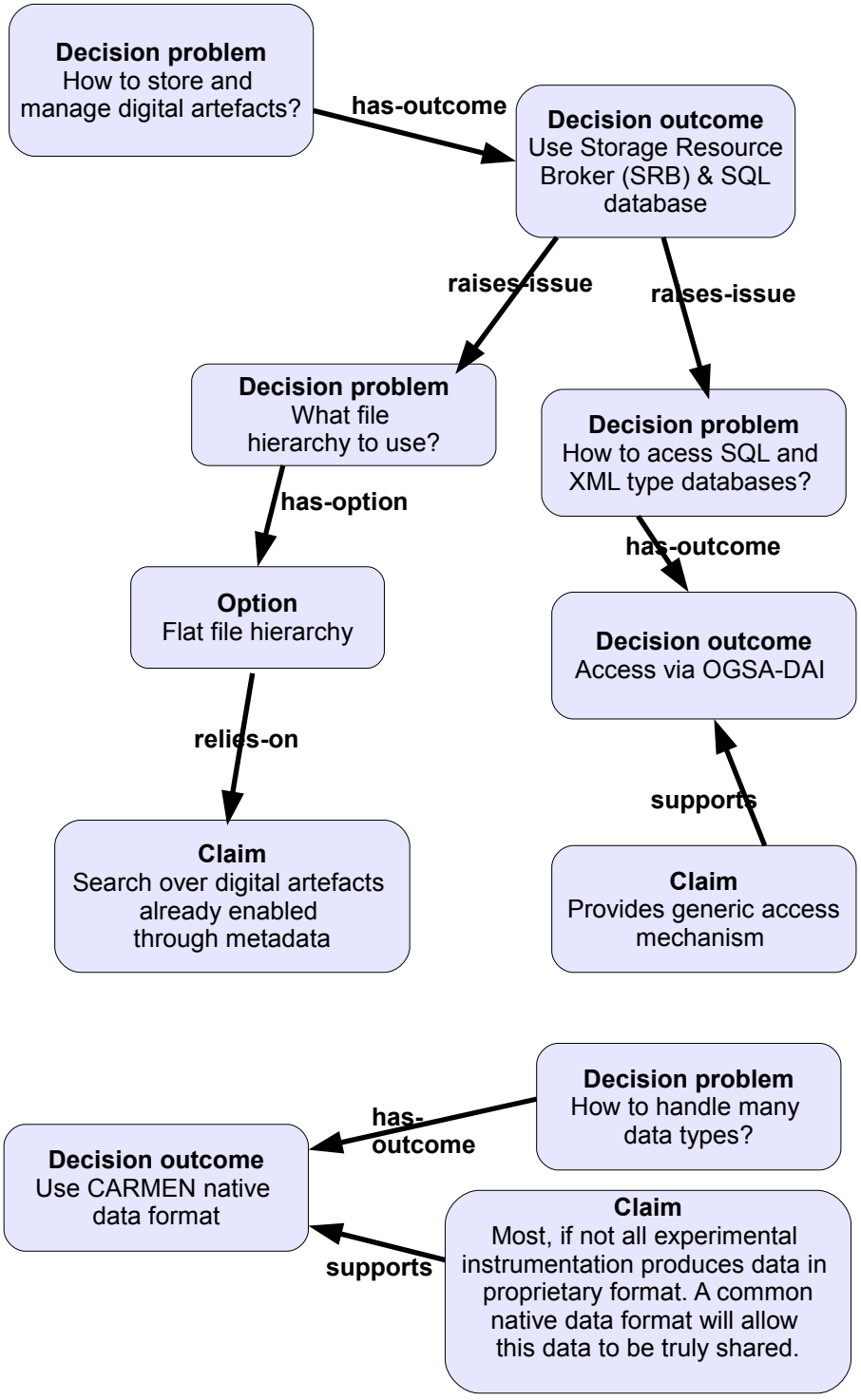

Figure 4.4: An example of design entities created using data from Section 4.3 of the CARMEN System Architecture Document

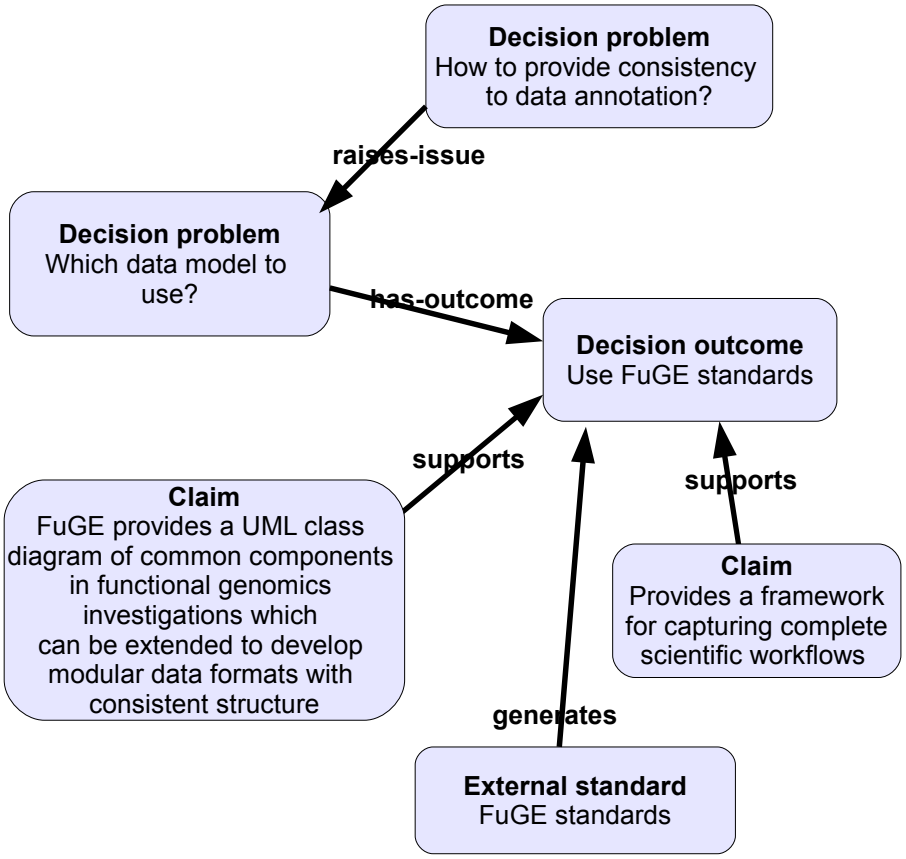

Figure 4.5: An example of design entities created using data from Section 4.6.2 of the CARMEN System Architecture Document

Section 4.3 of the System Architecture document contains the following paragraphs:

> 'The Storage Resource Broker (SRB) 8 and SQL databases such as Postgres 8 have been identified as pre-requisite technologies for storage and management of Digital Artefacts. Most, if not all experimental instrumentation produces data in proprietary format. In order for this data to be truly shared, a common, native data format will be deployed.
>
> 'Since SRB represents a file system, a file hierarchy design is required. Given that the CARMEN System will contain metadata enabling search over Digital Artefacts, this hierarchy could be completely flat. SQL and XML type databases will be accessed via OGSA-DAI 8, which encapsulates database access in web service form, providing a generic access mechanism.'

The design entities created (and linked together) using this information are shown in Figure 4.4. We envisage four decisions being made implicitly:

- A decision to use SRB 8 and SQL databases. No options are discussed in the text, so we do not list any and simply provide the Decision Outcome.
- A decision on file hierarchy design. Only one outcome is discussed (a flat file), although it is mentioned that this is a possibility because searching is already enabled through metadata. We add the latter point as an Assumption upon which the Decision Outcome depends.
- The other decision raised is not stated explicitly. It is explained that the database will be accessed using OGSA-DAI. The fact that this constraint is introduced suggests that other options are available for accessing databases, and that a decision has been made at some point to use this option. We thus add this as a Decision Problem and Decision Outcome. A reason supporting this decision is mentioned (OGSA-DAI provides a generic access mechanism).
- The fact that a constraint is introduced ('a common, native data format will be deployed') and a reason to support this ('in order for this data to be truly shared') shows us that other options were available, and that a decision has been made to select this one. We introduce entities to model the Decision Problem, the Decision Outcome and attach the Claim to the Outcome. The data format is a major design issue for CARMEN, and is touched upon in other paragraphs of this document and in other documents, too. We only show here entities relevant to the extracted text above.

Section 4.6.2 of the CARMEN System Architecture document contains the following text addressing the issue of the data model:

> 'To enable annotation of digital entities, a data model is required to structure the corresponding data and metadata. The CARMEN System will use the FuGE data model. FuGE provides a set of UML classes representing common components in life-science studies, such as materials, data, protocols, equipment and software. These can be extended consistently by creating new modules to describe further levels of granularity. FuGE provides a framework for capturing complete scientific workflows, enabling the integration of pre-existing data formats; i.e. FuGE enables capture of additional metadata that gives formats a context within the complete workflow.
>
> 'Many of the FuGE models can be repurposed to represent the neuroscience workflow with the addition of appropriate ontology. The FuGE bio.package can be extended to represent the laboratory based workflow. The FuGE data.package can be extended

to represent the in silico based workflow. The specific extensions required for the CARMEN System will be identified through case studies. To date, several specific protocols have been identified.'

Figure 4.5 shows the resultant design model based on this data. The entity 'How to provide consistency to data annotation' is partially based on information from elsewhere in the document, which also addresses a similar question. This raises a sub-question which is addressed here: which data model to use? The statement that a data model is needed and the description of the one selected indicates that other options were available (although not discussed here). We add a Decision Problem and a Decision Outcome to our model. Other comments in the paragraph amount to claims that support FuGE's data model (it provides a framework for capturing scientific workflows, it can be expanded). These are added to the model as well. And, since FuGE is an externally-produced set of standards, we add a link to the External Standard to which CARMEN's data model will adhere.

After completing the above stages, the total number of entities in our databases, including components, design entities and requirements, is 852, with 2095 relationships linking these entities.

4.5.6 Generating complexity metrics

We build on the work of previous researchers (see Section 2.4.1) by re-using existing complexity metrics.

Using 'snapshots'

CARMEN's Subversion repository covers a period of over two years. This raises the question: at what stage should the complexity metrics be generated? Some of the components we are working with have been added, modified, and then replaced at various stages with newer versions that better meet the project's needs. Generating metrics from a specific point in time would mean that some components would be excluded, either because they have already been removed, or because they have not yet been added. Furthermore, the complexity of different components alters over the life of the project. Ideally we want to be able to represent the full extent of system complexity over time.

In order to represent this state of flux, we generated a series of complexity metrics for the system over its lifetime. There were over 1588 versions of CARMEN available. Generating metrics for all versions is prohibitively time-consuming. Instead, a copy of every nth version of the system (we call this a 'snapshot') was obtained and metrics generated from it. This approach has several advantages: we can track the evolution of complexity in CARMEN over time. And we avoid the need to pick an arbitrary point in time from which to generate our metrics. However, it still does not guarantee that all components appear in at least one snapshot. For example, if we created snapshots at version 200 and at version 240, a component which was added in version 211 and removed in version 234 would not appear in any of our snapshots (and therefore we would have no complexity data for it).

We can attempt to reduce the number of excluded entities to an acceptable level by choosing an appropriate value for n. Too high and the data to gather becomes increasingly time-consuming; too low and we have insufficient data for analysis. We decided to set n at 50, for the following reasons:

- Setting n at 50 results in 31 separate snapshots, which is sufficient to give us a general picture of CARMEN's evolution[4] and fits within a reasonable time scale.

[4]This is comparable to previous studies of complexity evolution. A study by Gall *et al.*, for example, looked

- The number of entities excluded from the study because they don't fall into a snapshot is 31 (out of a total 254). This strikes a reasonable balance between minimising the time involved and minimising excluded entities. Setting n at 40, for example, would result 39 snapshots and a total 23 entities excluded (i.e., we'd include another 8 entities in our study). Setting n at 30 would result in 52 snapshots, and exclude 22 entities (adding another 9 entities). Lowering the value of n generally produces quite a lot more work and increases the number of available entities only very slightly.

Components excluded from snapshots are excluded from analysis of complexity metrics.

Choosing metrics tools

Our ideal tools should provide the following features:

- generate a range of design metrics, which have published definitions. Ideally we want to generate C&K or L&H metrics, because previous studies have linked change-proneness to C&K or L&H metrics (although [7] and [78] found no correlation between change-proneness and LCOM) and we wish our results to be comparable.

- be easily available.

- operate on Java source code. Should adequate tools to analyse source code not be available, we should then consider tools which analyse compiled code.

We surveyed a range of tools before making a selection, including the following[5]:

- Analyst4j[6], a commercial product, although licenses with restricted duration/functionality are available. Generates a wide range of complexity and design metrics from Java source code.

- Chidamber and Kemerer Java Metrics[7], an open-source tool which operates on compiled classes.

- CCCC (C and C++ Code Counter)[8], an open-source project originally developed to support PhD research. The project is officially dormant as of 2005, although it is still available and has achieved a reasonable level of maturity. Capable of generating a range of complexity and design metrics from Java source code.

- CodePro Analytix[9], a commercial product integrating with Eclipse. A time-limited evaluation license is available. Produces a very wide range of complexity, inheritance and dependency metrics, but many of these are not 'standard' implementations of C&K or L&H metrics (assuming a standard exists, bearing in mind the points made by [96]).

- DependencyFinder (DepFind)[10], an open source tool. Accepts compiled Java source code, and does not produce a very wide range of complexity/design-related metrics, which makes it unsuitable for our research.

at 20 versions of the system produced over 2 years [56]. Wermelinger and Yu examined 47 releases in total of the Eclipse system [149].

[5] note that many of these tools are also briefly surveyed in Lincke *et al.* [96]

[6] http://www.codeswat.com

[7] http://www.spinellis.gr/sw/ckjm

[8] http://cccc.sourceforge.net

[9] http://www.instantiations.com

[10] http://www.depfind.sourceforge.net

- Eclipse Metrics plug-in 1.3.6 by Frank Sauer[11], an open source tool generating a range of complexity and design metrics, although this does not include CBO.

- Eclipse Metrics plug-in 3.4 by Lance Walton[12], also an open source tool. Does not generate CBO, although other metrics can be extracted.

- Essential Metrics, a commercial tool generating design and complexity metrics for Java[13]. Free evaluation licenses are available. Generates a very wide range of metrics, including CBO, although not the full range of C&K or L&H metrics.

- OOMeter, a research tool generating metrics from Java source code.

- SD Metrics[14], a commercial tool with a demonstration version available. Produces a very wide range of metrics, although not quite the full set of C&K or L&H metrics. Coupling metrics are assessed through other means than the CBO metric, which makes it more difficult for us to compare to previous research.

- Understand for Java[15], a commercial tool. Does generate a wide range of metrics for a range of languages, although documentation points out that the Java version is incomplete, making it sub-optimal for us.

- VizzAnalzyer[16], a commercial tool, although free for non-commercial use. Produces a range of complexity and design metrics, although not CBO. However, it is one of the few tools to implement any L&H metrics (DAC). The tool provides alternative coupling-based metrics, although this makes it more difficult for us to compare to previous research.

In addition to the tools listed above, other metrics-generating tools were considered, but either these did not generate design/complexity metrics of the type we need for this particular research, or they were unavailable for licensing reasons. This includes: CMTJava[17]; CodeAnalyzer[18]; Resource Standard Metrics [19]; Java Source Code Metrics[20]; JDepend[21]; jMove[22]; CyVis[23]; JHawk[24]; jMOOD[25]; Mccabe IQ[26]; JStyle[27]; and SonarJ[28].

We noted that few of the tools examined above generate a complete set of either the C&K or L&H metrics. CCCC, VizzAnalyser, OOMeter and Analyst4j, and Essential Metrics seem to be the tools which most closely match our requirements (primarily, operate on Java source code and generate a range of metrics). Table 4.4 displays the selected tools and their coverage. In the first instance, we use CCCC to gather metrics as it is the most readily available.

[11] http://metrics.sourceforge.net
[12] http://eclipse-metrics.sourceforge.net
[13] http://www.powersoftware.com
[14] http://www.sdmetrics.com
[15] https://www.scitools.com
[16] http://www.arisa.se
[17] http://www.verifysoft.com
[18] http://www.codeanalyzer.teel.ws
[19] http://www.msquaredtechnologies.com
[20] http://www.semdesigns.com
[21] http://www.semdesigns.com
[22] http://jmove.sourceforge.net
[23] http://cyvis.sourceforge.net
[24] http://www.virtualmachinery.com
[25] http://sourceforge.net/projects/jmood
[26] http://www.mccabe.com
[27] http://www.codework.com/JStyle/product.html
[28] http://www.hello2morrow.com

Tool	CBO	DIT	LCOM	NOC	RFC	WMC
Analyst4j (commercial)	•	•	•	•	•	•
CCCC (open source)	•	•		•		•
Essential Metrics (commercial)	•	•		•		•
OOMeter (commercial)	•	•		•		
VizzAnalyzer (commercial)		•	•	•	•	•

Table 4.4: Table summarising coverage of selected metrics-generating tools

Generating the complexity metrics

We took the following steps to generate the metrics. Firstly we checked out of Subversion every 50th version of the system, starting with version r1[29], then version r51, and so on, to create a regular 'snapshot' of the system as it develops. The first two of our snapshots contained no code. Thus our code generation begins with a small version of the system at r101.

For each version of the system we generated the metrics using our chosen tool for all valid components present in the system. We set the file permissions to read-only to avoid the possibility that a tool could make any changes.

Handling forked versions

In a number of cases, the system was 'forked' (e.g., as a particular version intended for a release or a demonstration was separated from the main development trunk). This resulted in the presence of two or more copies of the same component within a single snapshot, potentially with different sizes and/or complexity values.

When this occurred we first checked to see whether the different copies had different complexity values (using the same metrics-generating tool); if they are exactly the same (i.e, an exact duplicate exists) then we simply use any single copy and exclude the others.

If there is a difference, this means that two (or more) versions of the same file are potentially under current active development, and we should like to reflect this. We could attempt to identify the most recently-modified copy and use this to generate our metrics. This approach assumes that the older version is likely to be discarded by the team and the most recently modified copy will be the one intended for long-term use. This is not a valid assumption. For example, the most recently-modified copy could be an evaluation, demonstration or prototype version being prepared for release to users, with the intention that it will be discarded once user feedback has been gathered. Alternatively, two or more copies of a file may be under active development at once: one copy being prepared for release, whilst its sibling copy continues with main development. We should like the metrics values to reflect the fact that several different versions of a file may exist at once, and that any could be considered the 'true' version. Where more than one version of the file exists in a single snapshot we average the values together.

Several exceptions exist to this:

- Several demonstration copies of the system were created early in the lifecycle, checked into separate directories, and never removed. They thus exist for the duration of the project as archived demonstration copies, but are never modified after their demonstration purpose is complete. Because these are very early copies that never change trying to combine them with more recent versions applies a considerable 'drag' to the metric results. We decided to

[29] We follow Subversion's convention, naming system versions rn, where r represents 'revision' and n the system version number.

exclude these early demonstration directories from our study once they become dormant. We define a demonstration directory as 'dormant' after the last modifications have been included in a snapshot. For example, if the last modification made to any file within a demonstration directory was in version r143, then we *included* results from the directory in the next snapshot (r151), and then classed the directory as dormant for subsequent snapshots (r201 onwards). Figure 4.6 illustrates this example. This strategy ensures that all modifications are included in a snapshot at least once, but that forgotten demonstration copies do not skew results.

Directories falling into this category in CARMEN are named 'SFNDemo', 'April_Consortium_Target' and 'AFM2007LiveDemo'.

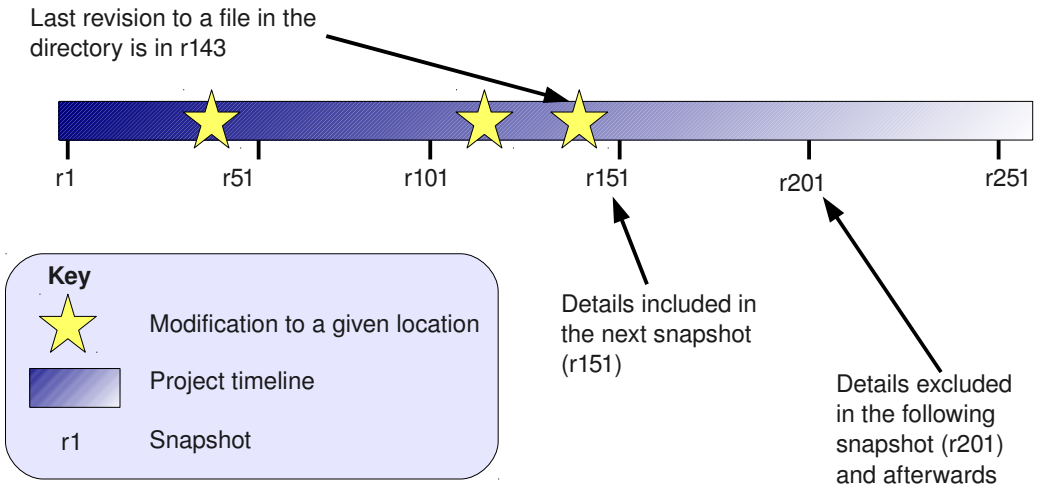

Figure 4.6: Hypothetical project timeline showing a directory which is modified several times, most recently in r143

- Later during a refactoring exercise, folders were reorganised and many existing files were copied into a directory named 'carmen-old'. There are no modifications to files in this folder after it is created (although some files are later copied back into the main development trunk). Following the same principle outlined above, we include the 'carmen-old' directory in one snapshot after it is created, but then exclude it from our study for future snapshots since it is never modified again.

When deciding whether a directory was dormant or currently active, we took into account changes to files which we did not consider 'valid' components (see 4.5.2). For example, if all development on Java files in a demonstration copy ceased in r143, but changes were made to images (for example) in the same directory until r221, we did not class the directory as dormant until after snapshot r251.

The idea of a 'dormant' directory was applied *only* to directories which were:

- intended to become an archived copy and not under active development; and
- not modified at all after a relatively early stage in the project.

No other directories other than those detailed above were classed as 'dormant'. Metrics were gathered and included for all other components. Any other code outside the directories mentioned above which is not modified for a long time is regarded as a stable part of the project (and included)

rather than dormant. We used Subversion's logs to determine which files and folders had changed, not values of metrics.

Generating metrics from CCCC

Using the CCCC package, we generated eight separate metrics for each entity:

- lines of code (LOC)
- lines of comments
- NOC
- DIT
- CBO
- McCabe's cyclomatic measures (labelled by CCCC as MVG)
- WMC (with 'unity' weighting, making this a simple count of methods per class.
- A separate value was also generated for WMC with 'weighting = visible' (i.e., the count is weighted by method visibility) but this did not result in a broad spread of values (in fact, all components had the same value) and so it was not used.

As discussed in Section 4.5.6, we should like to determine the methods used by each tool to assign values to these metrics, since they may vary between tools. CCCC's documentation provides a very broad explanation of the counting methods used for some metrics, but little specific information on others (such as CBO). For example, the user manual states that LOC is counted as followed:

> This count follows the industry standard of counting non-blank, non-comment lines of source code. Preprocessor lines are treated as blank. Class and function declarations are counted, but declarations of global data are ignored. There may be some double counting of lines in class definitions as the algorithm treats the total over a module as the sum of lines belonging to the module itself and lines belonging to its member functions (the declarations and definitions of member functions in the body of the class definition will contribute to both counts) [27].

Lines of comment are defined: 'any line which contains any part of a comment for the language concerned is treated as a comment by CCCC' [27]. Initial testing on a small selection of random CARMEN files suggests that CCCC excludes header comments (providing details on authors, changes, dates and purpose before the class declaration) when counting lines of comments. This is useful for us: if we include header comments then shorter files appear to have more comments per line than longer files, so excluding header comments removes this bias. Similarly, many files in CARMEN contain a list in the header comments of the changes made to the file, the changer and the date. Files which had changed often would therefore have a skewed number of comment lines if headers were counted.

In addition to some of the design level metrics proposed by Chidamber and Kemerer and size related metrics (such as lines of code or comments) CCCC also generates a value for Cyclomatic Complexity Metric (v(G)) - sometimes referred to as MVG. This measure was first proposed by McCabe in the 1970s [105], and was primarily oriented towards assessing the multiplicity of paths in procedural software. Typically the metric is calculated on a method-level. CCCC produces values for methods, and also sums those values to produce a total for a class. Whilst MVG per class is not frequently used, we include it in our study since it is widely-known and easy to generate from a variety of existing tools.

4.5.7 Measuring change-proneness

By 'change-proneness' we generally mean the amount of change to which a component is subjected (see Chapter 1).

There are number of different ways to assess change-proneness, none of which is perfect:

1. We could count the number of times a component/file has been checked in to Subversion. This figure may be distorted by individual programmer behaviour; for example, some developers may habitually check in files once a day (or more) whilst others prefer to work offline and only check in files once a major change has been completed and tested. Many projects have some policies regarding the frequency with which developers commit changes to Subversion, which minimises extreme differences.

   This approach has the disadvantage that all changes of any size are flattened and counted as one change, so this does not necessarily give us a very accurate picture if we are interested in actual change *quantity*.

2. On a post-release project, we could count the number of change requests or bug reports raised and/or dealt with. The data we have for CARMEN straddles pre-release development and initial releases to users, and as a result change request mechanisms available do not generate large amounts of useful information.

3. We could count the number of lines changed between different versions of the files. This indicates the size of the change between two check-in events, and has been used by previous studies (e.g., [146, 94]. Like option 1 above, this approach is also susceptible to distortion if individual programmer check-in habits vary wildly. Whilst automated tools exist for comparing text files (such as source code) and counting the changed lines, such tools are not perfect, and may confuse a line which has changed for a line deleted and another line added.

We need to decide whether we should like to gather data on how *much* or how *frequently* files are altered. A single large change that doesn't happen often is perhaps less likely to result in code decay, a confusing structure or an increase in the propagation of change ripples than a file which has been the subject of smaller, repeated changes, which are perhaps even completed by different developers with different understandings of the design. We do not know of any research which has suggested that particular types, sizes or frequency of changes are more likely to contribute to long-term code decay. As a result, we assume that any type of change may contribute to code decay, but that frequency is more likely to be of interest than type or size of change, and we count the number of check-in events.

In an ideal world, we should like to be able to categorises types of changes. For example, we could test whether extra design work results in fewer bug fixes, fewer user requests etc. later. However, CARMEN's a log of types of changes made by developers is not maintained, and attempting to catalogue changes retrospectively is prohibitively time-consuming. It could be argued that it is less useful to know where the majority of changes occur during the initial release, since these are planned. However, we know that the changes studied in this project definitely include:

- initial research efforts
- planned developments
- responding to user comments - changes that Swanson calls 'adaptive' [144] (e.g., the structured usability session carried out shortly before beta release resulted in extra work on the user interface, and discussions with developers revealed that they worked closely with specific users immediately after initial beta release)

- corrective changes (there are references to 'bug fixes' in log comments)
- refactoring efforts and other perfective changes (see Section 4.6 for more on this).

As we have mentioned in discussions in Chapter 3, we suspect that some metrics may share a relationship with a specific type of change (e.g., extra work on design might reduce user requests later). This type of relationship may not be easily detectable since these specific types of changes form a subset of all changes. However, we believe that major relationships which are useful to know about will be visible.

To gather this data, we counted the number of times that each component is referenced in Subversion's logs. We discount the first, initial check-in, so that only subsequent changes are counted.

This will only count the times that a file is individually modified, however. There are cases where a file may be modified as a side-effect of another action, and this will not show up explicitly in Subversion's logs. For example, if a parent folder which contains a file is moved or deleted, then the log records that the move or deletion of the folder took place, but does not list individual entities inside the folder. Such changes need to be identified manually.

We do not wish to count a change to a file each time that its parent folder is renamed or moved. Counting this single, easy change as a change against each file inside the folder risks giving the renaming or move of a folder disproportionate significance. However, we are interested in folder deletions. In order to detect folder deletions, we produced a list of all folder locations from the log which contain valid components. Next we searched the logs for check-in events that may have resulted in a folder deletion, either for the direct parent folder, or another folder higher up the directory tree. When we found that a folder containing valid components had been deleted we manually recorded an 'extra' change against each of the components affected.

4.5.8 At what stage were requirements and design data collected?

Documents produced as part of the the design and requirements stages - and used to populate our submodels and thus to generate values for our metrics - are shown in Table 4.5.8.

We can divide CARMEN's requirements, specification and design phases into several stages. An initial effort running roughly from r42 until r872 sees firstly the stabilisation of requirements information (relatively early, in r366). During this phase the design also becomes stable (in r607), alongside the design of the metatadata architecture (stable from r683). Finally, recommendations on the decision as to what - if any - data format should be mandated and/or supported by CARMEN become stable in r872. We can characterise this phase as establishing the major plans and design decisions for CARMEN.

A second major phase of requirements and design work runs roughly from r757 to r1198. This sees a re-focussing of attention on the presentation of the requirements into a series of testable statements; this becomes stable from r1013. The specification of the CARMEN data format becomes stable in r1198. We can characterise this second phase as a consolidation of earlier work requirements and design work; in particular the Data Format Specifications act as a kind of sub-project of the main development and build on work from the earlier design phase.

Implementation work was ongoing throughout both of these phases, although earlier stages did not see the creation of as many components as we see later (Figure 4.8 shows component numbers for each snapshot). The total number of components climbs only very gradually during the first phase of requirements and design, but in r1001 it begins to climb more steeply[30]. r1001 is after the

[30]Note that the apparent dip at r951 is the result of a refactoring exercise that saw many modules which are no longer needed moved into an archive folder called 'carmen-old'. r951 falls into the middle of this activity.

Table 4.5: Table summarising documents used to gather design and requirements information from the CARMEN project

Document	**Added in**	**Last modified**
CARMEN System Architecture Specification CARMEN's main design document, covers high-level implementation details, options considered and rationale for decisions and requirements	r276	r607 (moved location in r911)
Carmen Requirements Summarises requirements gathered from users as a series of implementable statements	r997	r1013
CARMEN: Draft Initial Requirements Contains transcripts of user responses to structured questions during the requirements elicitation exercise. This data was also saved into documents organised by question/topic and by user group, for ease of navigation.	r288	r366
Data Formats in CARMEN and **Data Format in CARMEN - V2.0** Two versions of technical reports addressing the issue of data formats to be supported/ or implemented by CARMEN.	r284 (V1.0) r422 (V2.0)	r872 (V1.0) r424 (V2.0)
CARMEN Metadata: Ontologies and Software Describes the CARMEN project metadata architecture.	r42	r683
Carmen Data Format Specifications Specifies proposed CARMEN project common data type.	r757	r1198

first major design effort has completed, very close to the stabilisation of the requirements document and mid-way through the sub-project to specify details of the CARMEN data format, suggesting that attention is now focussing more closely on the implementation of the system after completion of the first requirements and design phase.

In r651 (after the major design document has been completed, and where reports on metadata are relatively close to completion) 53 components out of the total (over the project's lifetime) 254 are present. 19 of those components have been removed from the project by r1551, suggesting that some of the components, at least, are early versions of the system which are later superseded. This suggests that development effort at r651 was concentrated on requirements, design and specification work as much as on implementation.

Because the implementation of early components overlaps with production of the main System Architecture Document, technical reports and requirements we suggest that the requirements, specification/design and early implementation phases were most likely heavily iterative in CARMEN, with feedback from each stage influencing the others. Once these phases begin to stabilise we see a significant refactoring event (between r851 and r1001) and implementation appears to begin in earnest with a sharp increase in the number of modules.

The design information with which we populated our design submodel was largely drawn from

the System Architecture Document, and, to a lesser extent, the technical reports addressing data formats. These were all stable by r872. The requirements information we used was drawn from the requirements document which was last modified in r1013, and supplemented with details from the architecture documentation and technical reports (for information on rationale and sources). The data with which we have populated our models is thus contemporary with the earliest implementation efforts. The data we've used would be available to a designer who has completed the requirements and design work and also some very early efforts at implementation - even if the implementation work is exploratory and not intended for long-term inclusion in the project.

4.6 Evolution of complexity

Before we begin to analyse correlations between change-proneness and any of the metrics gathered, we present a brief study of the evolution of the software complexity in the CARMEN case study.

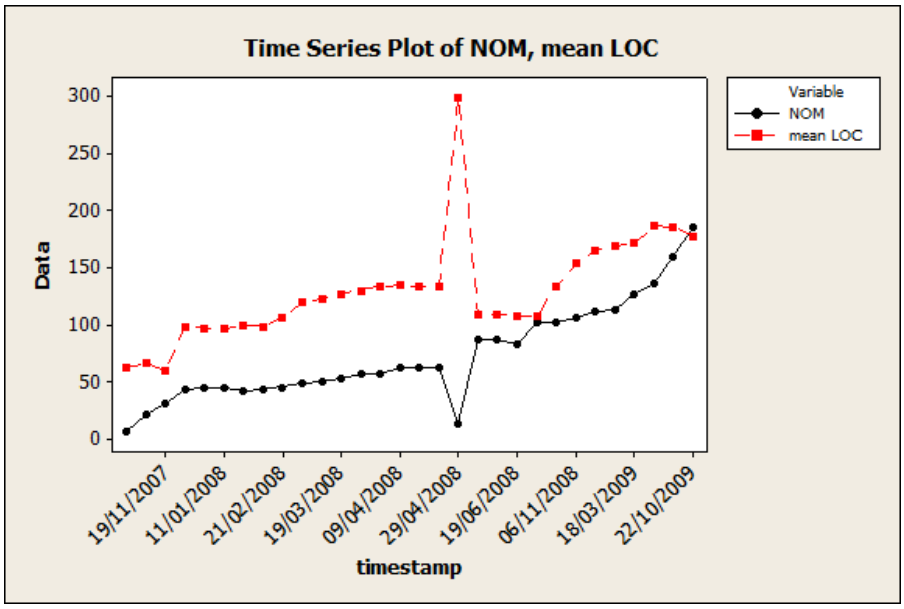

Figure 4.7: Number of modules and mean LOC per module shown against the actual timestamps of the snapshots.

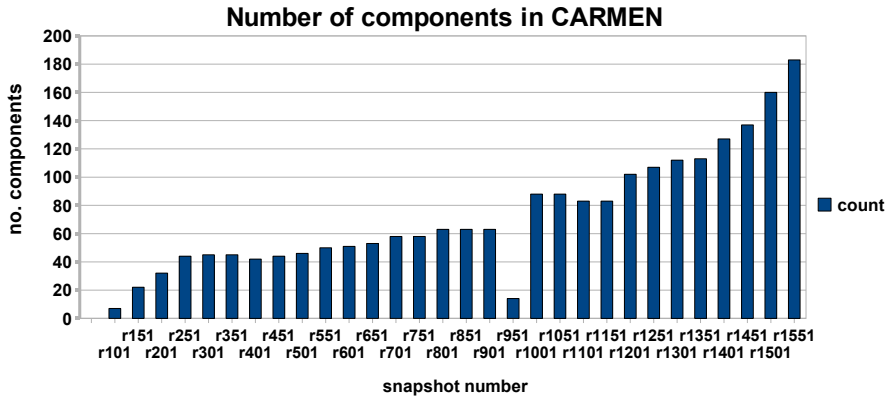

Figure 4.8: Number of modules over time

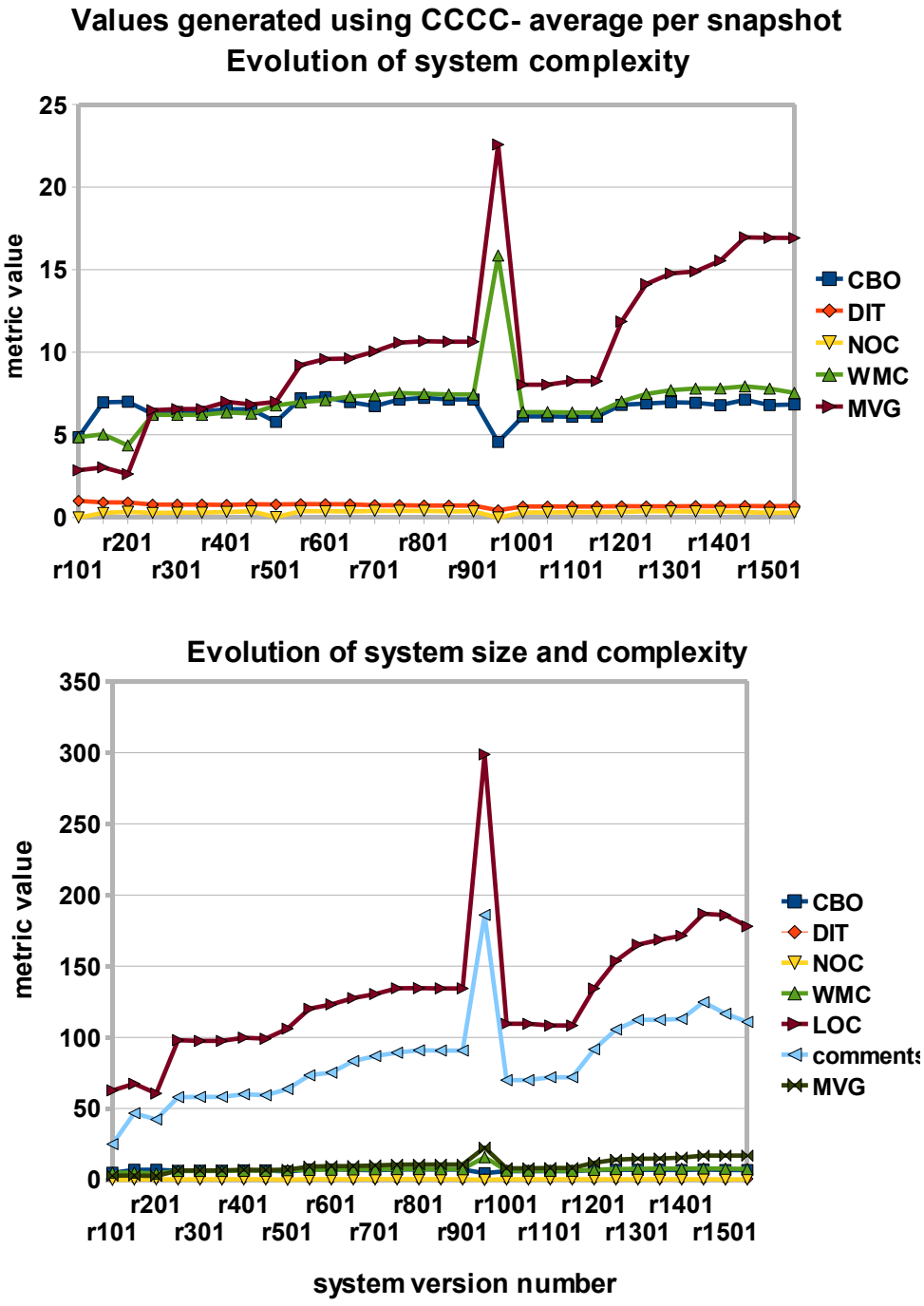

Figure 4.9: The evolution in various size and complexity metrics over CARMEN's development life. The top diagram shows complexity metrics only, the bottom shows the same data with the addition of lines of code and comments. Metrics are generated by the package CCCC

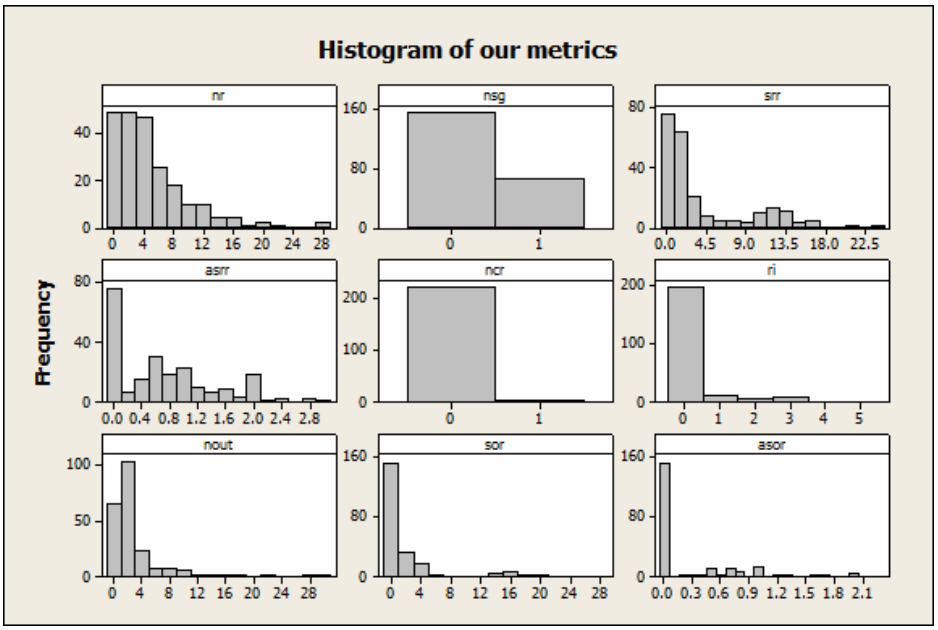

Figure 4.10: Histogram of our metrics

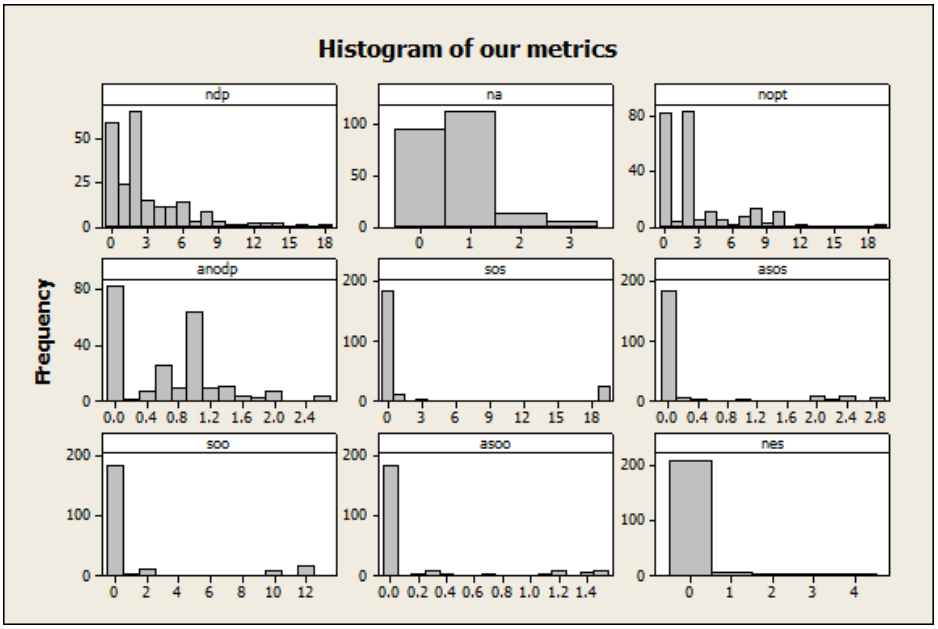

Figure 4.11: Histogram of our metrics

4.6.1 Metrics generated by CCCC

Our first observation is that complexity - and simple size metrics such as lines of code and comments - can decrease over the project life as well as increase. This conclusion matches data from other projects[31]. CARMEN's code tends to increase in complexity over time (with some plateaux), followed by a drop in complexity as a refactoring exercise restructures the software, followed by

[31] Pinzger *et al.*, for example, visualise evolution of complexity on a Kiviat diagram which can illustrate increases and decreases between versions [118].

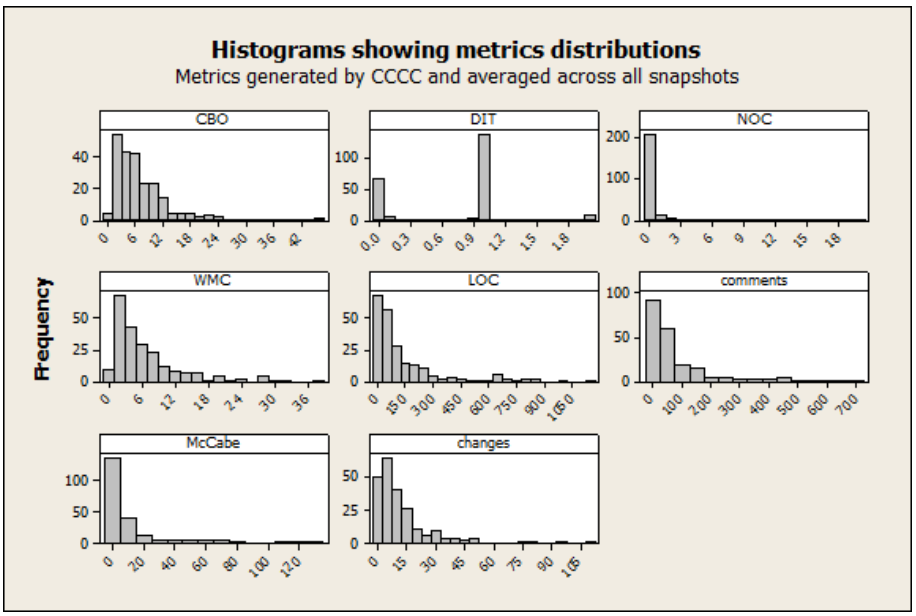

Figure 4.12: Histograms of all metrics generated by CCCC plus the number of observed changes

another gradual increase. In Figure 4.9 a gradual increase is stymied by a major refactoring around r901 - r951.

Figure 4.7 provides further evidence of the refactoring exercises adopted by CARMEN's developers, as we can see that after the major, mid-way refactoring exercise in March 2008 the two measures behave as mirror images of each other, with the number of modules dropping when the mean lines of code per module increases, and vice versa. This implies that code is being moved around the project; the mean LOC increases, and so developers redistribute the code to other classes or move it into new classes entirely to improve system comprehensibility.

Figure 4.9 suggests that refactoring has successfully decreased the system complexity: McCabe MVG and WMC drop in complexity as LOC decreases. CBO, however, shows larger classes decreasing complexity and a larger number of classes increasing it. CBO assesses instances of coupling between classes; and so a smaller number of classes thus tends to result fewer coupling links.

4.6.2 Distributions of our metrics

We need to know whether our data is normally distributed as this affects our choice of statistics techniques later. The statistics package Minitab[32] was used to check whether our data was normally distributed; we executed an Anderson-Darling normality test on each dataset.

The null hypothesis H_0 for the Anderson-Darling test is that data is normally distributed. The test calculates a p value to indicate the probability of a Type I error (i.e., the probability that we falsely reject H_0). We use a commonly-accepted threshold of 0.05 to determine whether we reject H_0; this means that there is only a 5% probability that we do so wrongly. Full results from Anderson-Darling normality tests on the metrics can be seen in Appendix E. We also present in Figures 4.10 and 4.11 histograms of our metrics.

None of our metrics are drawn from the normal distribution.

[32]http://www.minitab.com

4.6.3 Distributions of CCCC generated metrics

Figure 4.12 shows a set of histograms of metrics generated by CCCC, and also the number of changes observed. Anderson-Darling normality tests for metrics generated by CCCC are shown in Appendix E.

None of the metrics, nor changes, meets the criteria for a normally-distributed dataset.

4.7 Summary

In this chapter we have introduced and characterised our case study project. We described the processes involved in gathering data from the case study project to populate our models with design and requirements-related data, as well as how we derived counts of change-proneness. We discussed the selection of tools for generating existing complexity metrics. Finally, based on the values for various complexity metrics generated by these tools, we have attempted to characterise the evolution of complexity over CARMEN's development. In the next chapter we apply a more statistically rigorous study of this data as we search for evidence to support or reject our original hypotheses.

Chapter 5

Analysing existing complexity metrics

In the last chapter we introduced our case study project, CARMEN, and described the processes involved in gathering data from CARMEN to populate our metrics models.

In this chapter we return to the first of our hypotheses (introduced in Section 4.4), and apply statistical techniques to the data as we hunt for evidence supporting its acceptance or rejection.

5.1 Introductory summary

In this section we explore our hypotheses H_1 (first introduced in 4.4):

> *H_1: Existing complexity metrics can be used to predict volatility.*

In order to test this, we take data on existing metrics (generated by CCCC, as described in Chapter 4) and apply some statistical techniques to ascertain whether there are any significant relationships between change-proneness and metric values.

First of all we employ a linear regression technique (the technique is described in Section 5.2) to search for a linear correlation between change-proneness and the value of metric m for CARMEN's components. Linear regression is a well-understood technique and has been widely used in many similar studies. Re-employing the same technique means that our results can easily be compared to the work of other researchers. For example, previous studies of complexity metrics which employ regression include [2, 6, 7, 53, 146, 94] and [103]. A linear relationship - where it exists - is useful for making predictions since it implies quite straightforwardly that an increase in metric m results in a similar increase in change-proneness, and vice versa. However, we find no evidence of a linear correlation between any of our metrics and the number of changes. We discuss the techniques we employed and the results in Section 5.4.

Next we move on to employ a different statistical technique; we gather components into groups based on the value of metric m and compare mean change-proneness for different groups (the statistical techniques are described in greater detail in Section 5.2). It is worth utilising both of these different techniques (comparison of means, and linear regression) because they search for different types of relationship. Using both minimises the chances that useful information is missed.

There are several techniques widely used for comparing means. The t-test is a simple test for comparing two groups, whilst ANOVA ('ANalysis Of VAriance') techniques are commonly used to compare more than two groups. However, as we explain in Section 5.2.2 we use non-parametric

alternatives (the Kruskal-Wallis and Mann-Whitney tests) because our datasets are not normally distributed. These techniques have also been widely used in previous studies. For example, [29] and [6] use ANOVA and [152] use the Kruskal-Wallis test.

The Kruskal-Wallis test allows us to compare the change-proneness for many different groups of components in one single test, but the outcome can only inform us that a significant difference exists between two or more of the groups. It cannot inform us *which* groups these are. The Mann-Whitney test compares change-proneness between two groups only, and can indicate clearly whether a significant difference in change-proneness exists between them or not.

Our approach with this type of test is as follows:

- For each metric m, we partition the components into categories (sometimes called 'bins') based on the value of m. Some groups will be for components with low values of m, some for components with medium values and some groups will be for components with high values of m. When defining the categories, we attempt to ensure that each category spans a similarly-sized range of values and that categories are as evenly populated with components as possible. In some cases it has not been possible to ensure that categories are evenly populated, but the tests we use (Kruskal-Wallis and Mann-Whitney) are still valid for unevenly-populated groups.

- We then execute one Kruskal-Wallis test against all of the categories for m. The Kruskal-Wallis test specifically looks at the values of change-proneness associated with the components in each category. The result of the test indicates whether any significant differences in change-proneness exist between any of the categories - i.e., whether one category tends to see higher/lower values for change-proneness than other categories. Since the components are partitioned based on the value of m this would indicate that some values of m are associated with a higher/lower change-proneness than other values of m.

- If the Kruskal-Wallis test indicated that a significant difference did exist then we need to find which categories this applies to. To do this, we match up every category in the test with every other category and execute a Mann-Whitney test on the pairing. The Mann-Whitney test specifically looks at values for change-proneness in the two groups and determines whether they tend to be significantly different. Conceptually, this is a single collection of tests, even though this stage (pairing each category with each other category) can result in a very large number of individual pairings. We should like to ensure that, across the collection, the likelihood of error is below α. However, if we simply check that p is below α for each pairing, then with a large number of tests the likelihood of an error is compounded and becomes unacceptably large. Failing to control the error rate and 'share' it across all the pairings would increase the chances that 'significant' differences are wrongly detected, simply because there is a very large number of tests being conducted together. For this reason we employ a technique to share the error rate across many tests, explained in Section 5.2.2.

- After we have executed the Mann-Whitney tests, we examine the pairings which were substantially diferent and attempt to distil this into a series of heuristics. For example, the Mann-Whitney tests for CBO show that, of all the significant differences detected, they are between categories with CBO below 4, and CBBO above 4. We therefore determine that components with CBO below 4 are less likely to become change-prone than components with CBO above 4. We complete a similar process for all metrics.

We find that comparisons between groups of components (differentiated by metric value) does allow us to detect significant differences in the number of changes; this is the case for all of the

metrics. In general we find that, for size-related metrics such as LOC, comments and WMC, and also for MVG, there is a generally increasing likelihood of higher change-proneness as the value of metric m increases. However, not all metrics reproduce this pattern. CARMEN does not make heavy use of inheritance, for example, and our studies of the metrics DIT and NOC do not suggest a generally increasing change-proneness as DIT/NOC increases. For both of these metrics, in fact, we can see that higher DIT/NOC values appear to show a lower likelihood of change-proneness, although there are some differences between the two (we discuss the differences in further detail in Section 5.6.3).

CBO produces an interesting result; in general there is an increasing likelihood of of higher changes as CBO increases, but the boxplot that illustrates the changes for each category shows a number of peaks and troughs. Further studies would be needed to determine whether these reflect CARMEN's specific context only, or whether all projects see such peaks and troughs. We discuss CBO and its differences to other complexity measures (such as MVG) further in 5.6.4 and 5.6.6.

It is worth noting, however, that almost all categories for all metrics contain components which have experienced zero changes (as can be seen in boxplots presented in Section 5.5). This could be a reason why linear regression fails to detect useful correlations, since a linear relationship is difficult to detect in this situation.

We present the detailed results and the resultant heuristics in Section 5.5; we accompany each metric with a boxplot to illustrate visually the numbers of changes experienced by components in each category. It is important to realise that detection of statistically significant results alone does not determine that all metrics are equally suited for prediction, however. We do not compare the performance of different metrics in this chapter; our comparative evaluations are presented in Chapter 7.

The rest of this chapter is laid out as follows: Section 5.2 introduces on the statistical techniques we use and notes key information we look for. Sections 5.3 to 5.7 present results from analyses on metrics generated by CCCC (C and C++ Code Counter). Finally, Section 5.8 hosts discussion of our results and their implications for our hypotheses whilst Section 5.9 summarises our conclusions.

5.2 Statistics techniques used in this chapter

As explained in Section 5.1, we use linear regression and ANOVA techniques in analysing our data. Analyses in this thesis are carried out using the statistics package Minitab, unless otherwise stated[1].

5.2.1 Linear regression

Linear regression is commonly used to determine whether a linear correlation exists between one or more predictor variables and a dependent variable. The null hypothesis H_0 in any regression test postulates that there is no significant relationship between the predictor variables and the dependent variables. A regression analysis produces several pieces of data.

- Residuals, which measure the distance between the predicted values and the actual values of the dependent variable (also referred to as the 'errors').
- 'R-Sq%', which is the percentage of variation in the dependent value which can be explained by the model. A perfect model would return 100% , but this is highly unlikely in real-

[1] http://www.minitab.com. More can be read about Minitab in many sources, such as [133]. Details on regression can be found in, general statistics textbooks such as [47] or [109]. More on comparison tests and their non-parametric alternatives can be found in general textbooks such as [109] or non-parametric texts such as [139].

world software engineering data. A low figure could indicate that the model is missing some important explanatory data.

- Probability p, a value between 0 and 1, indicating the probability of obtaining the observed results if the null hypothesis is true [46], and, therefore, the likelihood of a Type I error. A Type I error occurs if we reject the null hypothesis when it is actually true. A commonly-adopted standard (usually denoted α) for rejecting H_0 is $p <= 0.05$, which indicates a 5% chance that H_0 is true, and a 95% probability that it is false.

Multiple regression

Predictor variables may interoperate in ways that are not obvious, so when working with multiple predictors we include or exclude variables from the model one step at a time and re-consider at each step. This 'step-wise' regression can be peformed forwards (ie, start with no variables and add in those with a sufficiently low p value until no more can be added). Or it can be performed in reverse, by including all predictors and excluding those with high p values. The aim is to reach the most sparing but accurate model possible.

Minitab also implements a 'best subsets' or 'all possible regressions' [133] option for regression analysis. This algorithm searches through each predictor, finding the two best one-predictor models ('best' being the model with the highest R-Sq%), then the two best two-predictor models, until all predictors have been used. Different techniques for including and excluding predictor variables can result in different models, so we use both stepwise and best subsets algorithms.

Testing the reliability of regression models

There are several tests to perform on the results of a regression model before it can be accepted. These include:

- Errors should be independent, normally distributed, with zero mean and a constant variance [47, 103].
- Predictor variables should not be autocorrelated (i.e., tending naturally to increase/decrease over time) [133].
- Multiple predictors should not show signs of multicollinearity (i.e. correlate with one another) [133].
- We should make sure we are not attempting to fit a linear model to non linearly-related data [113].

All of these, should they occur, tend to result in an unreliable model. However, since our models all fail to produce normally-distributed residuals (the first point) we do not carry out any further tests.

Autocorrelation could be a problem for studies of software complexity like ours, since we might expect that compexity would increase naturally over time for a developing system. However, the complexity metrics we use are averaged over the component's lifecycle, to achieve a single figure per component.

There is a risk involved in attempting to regress averaged values, as they tend to iron out variation in a population; if these values are then used as input into a regression equation this tends to result in a higher correlation than otherwise [109]. Since we have not detected any linear relationships using regression, however, we do not believe this to be a risk for our study.

5.2.2 Comparing means and medians

Another technique we can use is to divide the data into groups based on some criteria (for example, based on the value of metric m), and compare the values for the dependent variable (i.e., number of changes) in the resultant groups to see if they are significantly different. This technique tells us whether there tend to be significant differences in the number of changes experienced by components with different values of m.

Parametric and non-parametric tests

Both t-tests and ANOVA methods should only be used on normally distributed data. Non-parametric alternatives which may be used on non-normally distributed data instead include Kruskal-Wallis (an alternative to ANOVA) and the Mann-Whitney[2] test (an alternative to the two sample t-test). Non-parametric tests have other advantages: they are suitable for small sample sizes (although a minimum sample size of 5 is needed for a Kruskal-Wallis test [99, 106]); and they are not skewed by unusual values, since they rely on analysing medians. On the other hand, many non-parametric techniques (including the two we use) rank observations and then substitute the ranks for the actual recorded data. This loss of information makes them less powerful than similar parametric techniques, which is why parametric techniques are preferred where possible. As explained in Section 4.6.2, our data is not normally distributed, and so we also use non-parametric tests.

Mann-Whitney test

The Mann-Whitney test compares two groups of observations - which do not have to be the same size [42] - and hunts for a significant difference between them. Observations across all groups are assigned a rank, from smallest to largest. 'Tied data' is assigned an average rank [139, 133]. Output generated by the test includes:

- W: the sum of ranks corresponding to the observations in the first sample. W will tend to be larger if the first sample contains larger values, and smaller if the values in the first sample are generally smaller [133].
- The significance of the test: the probability that two groups are observed with values as separated as these when the populations in fact have the same median (i.e., the probability of falsely rejecting the null hypothesis, or a Type I error). As with regression, we set α at 0.05.
- A confidence interval, indicating a range within which it is estimated the true difference in the group medians lies. We use 95% as a confidence level, meaning that there is only a 5% chance that the actual difference lies outside the range. A difference of 0.00 obviously would mean that the groups are very similar. A confidence interval which does not straddle 0.00 is another way of expressing that a significant difference between the two groups exists.

Kruskal-Wallis test

The Kruskal-Wallis test is very similar to the Mann-Whitney, but generalised to handle large numbers of samples. Observations should be independent (i.e., they should not be paired, such as groups that show values 'before' and 'after' some event for the same individual [133]). The

[2]This test was introduced by Wilcoxon and may be called the Wilcoxon rank sum test, Mann-Whitney U test, Mann-Whitney-Wilcoxon (MWW), or Wilcoxon-Mann-Whitney. We follow the example of Minitab in naming the procedure the 'Mann-Whitney' test [133], although this is not universal in statistics texts.

null hypothesis for a Kruskal-Wallis test is that the samples are drawn from identical populations [106]); the alternate hypothesis is that one or more samples are drawn from a different population.

Like the Mann-Whitney test, data observations are assigned ranks and tied data is assigned an averaged rank. Output generated by the test includes:

- *H*. Calculated using a preset formula[3], H detects cases where lower or higher ranks dominate a particular group. Minitab calculates an 'adjusted' value for H, which takes into account tied values; otherwise the presence of tied values may introduce a bias [133]. A higher value for H indicates stronger evidence that we can reject the null hypothesis [133, 139].

- p: the probability that the samples would show such dissimilar results were the populations actually the same [133] (i.e., the probability of a Type I error). As with other tests in this chapter, our desired maximum α level is 0.05.

Multiple comparison analysis after Kruskal-Wallis

Kruskal-Wallis tests can only prove that a significant difference in the means of the populations is present; it does not indicate which groups these are. If the Kruskal-Wallis test reveals a significant difference, therefore, we complete further multiple comparison tests to identify the differing groups [134, 109]. This is achieved by pairing up each group/category from the Kruskal-Wallis test with each other group/category in the same test, and executing a two-sample test on each pairing. We use the Mann-Whitney test, as it is suitable for comparing two groups of non normally-distributed data.

As with regression testing, we need to ensure that the risk of a Type I error (i.e., rejecting the null hypothesis when actually it is true) is 0.05 or lower. However, if α is 0.05 for *each* individual pairing with the multiple comparisons method, then our overall potential for error can become unacceptably large, since there will be many unique pairings within one test. There are several techniques for controlling α so that it is 0.05 across *all* comparisons. The Bonferroni adjustment [109, 154], is one; this adjustment simply divides the desired α by the number of paired comparisons k [134]. For example, to compare six groups, the multiple comparison testing would result in fifteen separate two-sample comparisons. Using the Bonferroni adjustment method, we would calculate α as 0.05/15 for each individual comparison.

The Bonferroni adjustment is sometimes regarded as overly cautious, and may result in some significant correlations being missed [134]. Holm has proposed instead a sequentially rejective method [71, 134]. Following this method, hypotheses (for pair-wise comparisons) 1 to k are ranked, in descending order of significance, with the most significant (i.e., with the lowest p) ranked as 1. α is calculated individually for each pair-wise hypothesis as: $\alpha = \frac{\alpha}{k-rank+1}$. For the first hypothesis, α is therefore α/k; for the second, α is $\alpha/k-1$; and so on. When a comparison is encountered that exceeds α, that and all subsequent hypotheses must be accepted.

This technique balances a controlled α with the need to ensure that the test is sensitive enough to detect genuinely significant relationships. We use the Holm sequentially rejective method for our analyses where appropriate.

5.3 Analysing metrics generated by CCCC

To test hypothesis H_1, we first of all attempt to establish a correlation by building a regression model. Next we organise components into groups and search for differences in the number of

[3]The actual equation can be found in a general statistics textbook, such as [133] or [139].

changes between groups using Mann-Whitney and Kruskal-Wallis tests. Our assumption is that if a relationship is discovered between some metric and the number of changes, then the metric may be useable in a predictive capacity (discussed further in Section 5.8).

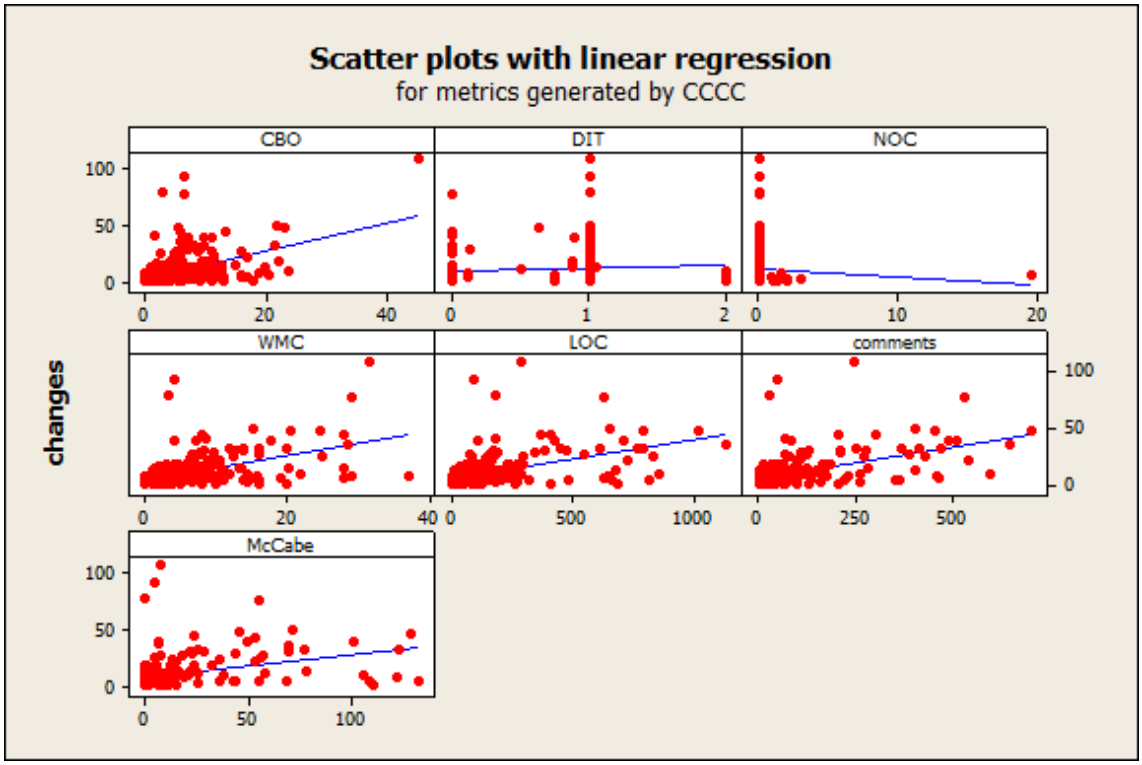

Figure 5.1: Scatter plots illustrating relationships between a variety of metrics generated by CCCC and number of changes

We have chosen to use both lines of code (LOC) and lines of comments as potential predictors. Both of these have the advantage of being well-known and understood as metrics. We have argued (in Section 2.4.6) that size metrics can be employed as a crude type of complexity metric in themselves. Lines of code can be an erratic measurement, however, and highly dependent on an individual programmer's style. To alleviate this somewhat, LOC metrics generally exclude comment lines and may also exclude other types of lines, such as lines which consist only of closing braces. Code written during our period of study for CARMEN is predominantly produced by a small subset of the team, and one programmer in particular produced many of the classes; whilst this means that results from CARMEN must be carefully validated on other projects, it does mean that we have a reasonably consistent style across the project.

We believe that comments may also be a useful complexity metric. Logically, more complex areas of code are naturally likely to be more heavily commented. Comments is potentially a natural indicator of code that is actually 'difficult', because the presence of comments is likely to be tied to a progammer's actual experience of the code's comprehensibility. However, as with other measures of complexity, it is not flawless. For example, programmers commonly comment out sections of code temporarily when making alterations, which would distort measurements. Some programmers may comment all code heavily (or sparsely), regardless of difficulty. We discuss this further in Section 5.6.2.

In general, lines of comments is a metric which straddles issues of size and complexity, because a larger component will probably contain more comments. We also normalise the comments metric to account for size, by calculating an averaged comments-per-line figure which concentrate more

Metric	p value	R-Sq (adj)	Residuals normally distributed
CBO	0.00	19.5%	No
DIT	0.159	0.4%	No
NOC	0.329	0.0%	No
WMC	0.00	23.8%	No
LOC	0.00	23.2%	No
Comments	0.00	25.5%	No
Comments per line	0.513	0.0%	No
MVG	0.00	10.9%	No

Table 5.1: Table summarising results of regression tests on individual metrics, all generated by CCCC

squarely on issues of complexity that might be revealed by extra heavy commenting.

Scatterplots for all metrics plotted against number of changes are shown in Figure 5.1.

5.4 Regression results

Key details from regression results for each metric are summarised in Table 5.1.

Clearly, NOC and DIT cannot be claimed to correlate linearly with number of changes, producing unacceptably high p values. As we noted in Chapter 4.6.1, the values obtained for NOC and DIT do not vary widely, and in fact most components have the same (low) values for these metrics. The inheritance tree of CARMEN's components is thus too granular for our purpose[4].

Although p values for LOC, comments, MVG and CBO are very low, none of the metrics boasts a high R-Sq% figure. The relatively low figures here imply that there may be other, unaccounted-for factors affecting the results. Furthermore, Anderson-Darling tests on the residuals show that the errors are not normally-distributed. As a result, we conclude that none of these metrics can individually be claimed to act as predictors of number of changes.

5.4.1 Multiple regression

Groups of input predictors may collectively explain more of the dependent variable's behaviour, so we also attempt to create a multiple regression model. The stepwise regression algorithm in Minitab suggests a model with four predictors: CBO; WMC; comments; and MVG. (The output from the stepwise regression is shown in Appendix F.)

Output from Minitab's 'best subsets' regression anlaysis is shown in Appendix F. Two versions each of the one-, two-, three-, four-, five- and six-predictor models were produced, and one with all seven predictors. In general, we look for the model with:

- a high R-Sq(adj)%
- as small number of predictors as is possible
- a low value for *Mallows Cp*, which is also close to the number of predictors. This would indicate a more precise model. *Mallows Cp* is a statistic that 'attempts to measure both random error and any bias from fitting a model with too few predictors' [133]

[4]This observation echoes findings of other researchers. Chidamber *et al.*, for example, noted 'generally small values' for NOC and DIT in several case study systems [32].

We believe that the optimal model produced by this method is one of the 4-predictors models, consisting of: CBO; comments; McCabe's MVG figure; and WMC. This model achieves one of the highest values for R-Sq(adj)%; R-Sq% does not increase noticeably with more predictors. It also has one of the lowest values for *Mallows Cp*, which, at 3.4, is also close to the number of predictors. This technique thus reaches the same conclusions as the stepwise regression technique.

An Anderson-Darling test on the residuals of this single model, recommended by both stepwise and best subsets algorithms, produces a p value of less than 0.05, and therefore the residuals of the model cannot be considered normally distributed. As a result, we conclude that regression techniques find no correlation between changes and the complexity metrics here. Boxplots generated for our Kruskal-Wallis tests (see Section 5.5) show that there are still some components experiencing few or no changes irrespective of the value of metric m, which perhaps explains the difficulty in fitting a linear model to this data.

5.5 Kruskal-Wallis for comparing three or more groups

We divided components into a series of 'categories', to see whether higher values for a metric m are more likely to be associated with change than lower values. When dividing components into groups we created some groups with a fixed value (e.g., all components where m=1) and some groups which cover a range of values (e.g., all components where 5<=m<10). We do this to ensure that all groups contain at least five components. Although Kruskal-Wallis tests can tolerate groups of different sizes, groups should be as equal as possible.

Results are summarised in Table 5.2, along with a list of the groups that were created for each metric and the number n of components in each group. The p values for each of these metrics in the Kruskal-Wallis test is below α and the H value is relatively high, showing that there is strong evidence for a significant difference between two or more of the categories for each metric. To discover *which* two or more categories differ, we conduct multiple comparison testing. We pair up each category with every other category for the same metric and execute a Mann-Whitney test on the pairing (with the null hypothesis that there is no difference). As explained in Section 5.2, we use the Holm sequentially rejective technique. Key results from the post-hoc tests are shown in Tables 5.3 to 5.9. The tables show the actual p calculated by Minitab; how we calculate α; and indicates whether we can safely reject H_0. The comparisons are shown in sorted order; for brevity we include only as far as the first comparison for each metric that is not sigificant enough to reject H_0.

Table 5.2: Table summarising results of Kruskal-Wallis tests on metrics generated by CCCC

Metric	***p* (adj)**	**H**	**Groups**	***n***	**Median**
CBO	0.000	80.45	*CBO<=1*	24	1.0
			1<CBO<=2	28	4.0
			2<CBO<=3	28	3.5
			3<CBO<=4	13	4.0
			4<CBO<=5	13	11.0
			5<CBO<=6	25	9.0
			6<CBO<=7	19	9.0
			7<CBO<=8	9	9.0

Continued on next page

Table 5.2 – continued from previous page

Metric	p (adj)	H	Groups	n	Median
			8<CBO<=9	10	10.0
			9<CBO<=10	11	17.0
			10<CBO<=11	12	13.0
			11<CBO<=12	8	11.0
			12<CBO<=13	6	7.5
			CBO>13	17	15.0
DIT	0.000	30.25	*DIT=0*	67	3.0
			0<DIT<1	11	12.0
			DIT=1	137	9.0
			DIT>1	8	2.5
NOC	0.001	14.01	*NOC=0*	206	8.0
			0<NOC<=1	7	2.0
			NOC>1	10	3.0
WMC	0.000	81.68	*WMC=0*	10	3.0
			0<WMC<=1	25	3.0
			1<WMC<=2	34	4.0
			2<WMC<=3	21	4.0
			3<WMC<=4	19	5.0
			4<WMC<=5	24	5.0
			5<WMC<=6	9	8.0
			6<WMC<=7	9	15.0
			7<WMC<=8	16	13.0
			8<WMC<=9	6	18.0
			9<WMC<=10	7	11.0
			10<WMC<=15	19	14.0
			15<WMC<=20	11	9.0
			WMC>20	13	26.0
LOC	0.000	103.18	*0<=LOC<5*	22	2.0
			5<=LOC<10	11	4.0
			10<=LOC<15	13	8.0
			15<=LOC<20	12	3.0
			20<=LOC<30	19	17.0
			30<=LOC<40	14	3.0
			40<=LOC<50	10	13.0
			50<=LOC<75	22	4.5
			75<=LOC<100	18	31.0
			100<=LOC<150	17	8.0
			150<=LOC<200	15	2.0
			200<=LOC<300	21	9.5
			300<=LOC<500	10	22.0
			500<=LOC<750	11	12.5
			LOC>=750	8	32.5
Comments	0.000	92.63	*C=0*	19	2.0
			0<C<=5	25	3.0

Continued on next page

Table 5.2 – continued from previous page

Metric	p (adj)	H	Groups	n	Median
(C)			*5<C<=10*	18	1.5
			10<C<=20	23	5.0
			20<C<=30	18	6.5
			30<C<=50	26	9.0
			50<C<=75	23	10.0
			75<C<=100	11	12.0
			100<C<=150	18	13.0
			150<C<=200	8	23.0
			200<C<=300	13	25.0
			300<C<=500	15	28.0
			C>500	6	38.5
CPL	0.000	58.36	*CPL=0*	19	2.0
			0<CPL<=0.2	18	8.0
			0.2<CPL<=0.4	22	3.5
			0.4<CPL<=0.6	28	17.5
			0.6<=CPL<=0.8	36	12.0
			0.8<CPL<=1.0	41	8.0
			1.0<CPL<=1.5	23	4.0
			CPL>1.5	6	2.0
McCabe's MVG	0.000	20408.85	*0<=MVG<1*	92	4.0
			1<=MVG<2	16	5.0
			2<=MVG<5	28	11.0
			5<=MVG<10	24	19.0
			10<=MVG<15	16	9.5
			15<=MVG<30	17	19.0
			30<=MVG<50	9	9.5
			50<=MVG<100	13	28.0
			MVG>100	8	10.0

5.6 Results of analysis

For each metric, we distil any useful findings into some general observations. Results for metrics are discussed below.

5.6.1 LOC

LOC is primarily a measure of size, which has been correlated with change in the past (for example, see [94]). Our results generally support the idea that higher LOC can be associated with more changes. Figure 5.2 shows some minor peaks and troughs in the median[5] number of changes as LOC increases, but there is a generally positive trend. The very lowest value categories for LOC (where LOC is between 0 and 40) have markedly lower medians and shorter interquartile ranges for number of changes than categories where LOC is above 50. The median and the interquartile range increase sharply again once LOC reaches values of 150 or more, and again at 300 or more.

[5]Medians are indicated on the boxplots by a horizontal line through the solid box.

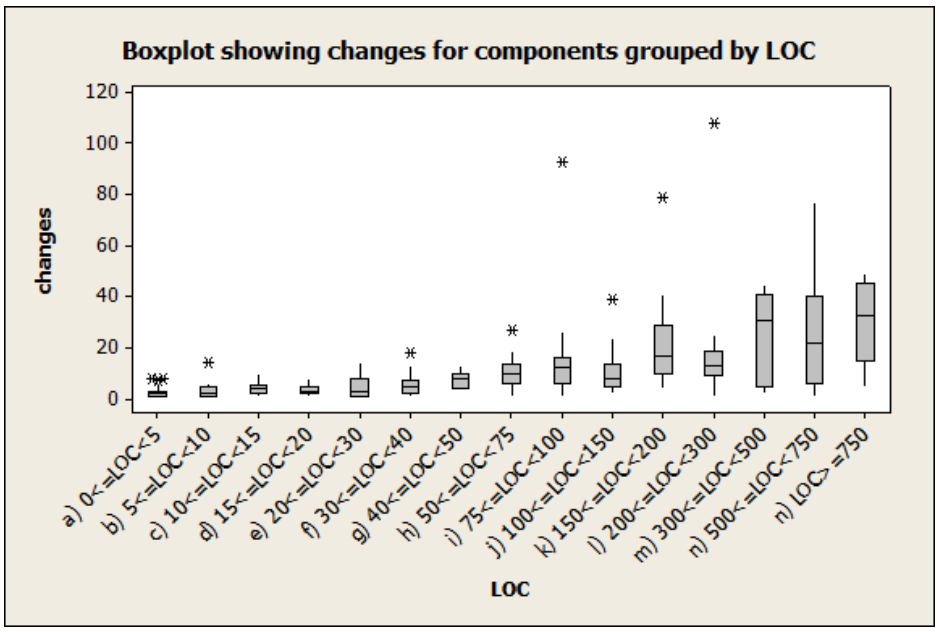

Figure 5.2: Boxplot showing the number of changes experienced by components categorised by values for LOC

Values above 300 do not seem to show much further increase in medians (with a few individual exceptions): the median number of changes - and the interquartile range - is similar for components where *300<=LOC<500* and for components where *LOC>=750*.

Multiple comparisons tests, summarised in Table 5.7, support these observations. All but one of the pairings which show a significant difference involve one category where LOC is below 50, and one where LOC is above 50 (and usually much higher).

We make the following observations:

1. Low values of LOC (below 40) are less likely to experience higher numbers of changes.
2. Components with LOC above 300 are the most likely to high numbers of changes.
3. It is still possible to see no (or very few) changes for any value of LOC.

We have already discussed the reasons why size and volatility may be related in Chapter 2.

5.6.2 Comments and CPL

Like LOC, we can see a generally (but not inorexably) upward trend as number of comments increase. Components with very low numbers of comments are less likely to experience high numbers of changes. The interquartile range begins to climb once the number of comments reaches around 50, and the median number of changes also increases markedly as comments reaches 75. A sharp jump upwards occurs once the number of comments is above 200. Unlike LOC, the boxplot seems to imply that numbers of changes may continue to increase slightly as numbers of comments climb (although further studies would be necessarily to confirm this). We can see that the final category in Figure 5.3 (*Comments>500*) is unusual in that no components in this category had very low numbers of changes, whilst the median, upper limit and interquartile range are higher here than for any other category.

Multiple comparison testing for comments is summarised in Table 5.4. All but two of the tests with a significant difference are comparisons between one of the four lowest-value categories (with comments below 20) and a category where comments is more than 50. Eleven of the 24 significantly

Pairing	W	p (adj)	α	Reject H_0?
0<=LOC<5 and 50<=LOC<75	299.5	0.0000	0.05/105 = 0.000476	Yes
0<=LOC<5 and 150<=LOC<200	258.0	0.0000	0.05/104 = 0.000481	Yes
0<=LOC<5 and 200<=LOC<300	283.5	0.0000	0.05/103 = 0.000485	Yes
15<=LOC<20 and 50<=LOC<200	82.5	0.0000	0.05/102 = 0.000490	Yes
20<=LOC<30 and 150<=LOC<200	209.0	0.0000	0.05/101 = 0.000495	Yes
20<=LOC<30 and 200<=LOC<300	239.0	0.0000	0.05/100 = 0.0005	Yes
15<=LOC<20 and 200<=LOC<300	97.5	0.0001	0.05/99 = 0.000505	Yes
0<=LOC<5 and 75<=LOC<100	303.5	0.0001	0.05/98 = 0.000510	Yes
0<=LOC<5 and 100<=LOC<150	305.5	0.0001	0.05/97 = 0.000515	Yes
0<=LOC<5 and LOC>=750	257.0	0.0001	0.05/96 = 0.000521	Yes
5<=LOC<10 and 150<=LOC<200	74.5	0.0001	0.05/95 = 0.000526	Yes
10<=LOC<15 and 150<=LOC<200	101.5	0.0001	0.05/94 = 0.000526	Yes
10<=LOC<15 and 200<=LOC<300	117.0	0.0001	0.05/93 = 0.000538	Yes
0<=LOC<5 and 40<=LOC<50	271.0	0.0002	0.05/92 = 0.000543	Yes
0<=LOC<5 and 300<=LOC<500	273.5	0.0002	0.05/91 = 0.000549	Yes
0<=LOC<5 and 500<=LOC<750	279.5	0.0003	0.05/90 = 0.000556	Yes
5<=LOC<10 and 200<=LOC<300	89.5	0.0003	0.05/89 = 0.000562	Yes
15<=LOC<20 and 50<=LOC<75	109.0	0.0003	0.05/88 = 0.000568	Yes
30<=LOC<40 and 150<=LOC<200	126.0	0.0003	0.05/87 = 0.000575	Yes
20<=LOC<30 and LOC>=750	198.5	0.0003	0.05/86 = 0.000581	Yes
10<=LOC<15 and LOC>=750	94.5	0.0005	0.05/85 = 0.000588	Yes
15<=LOC<20 and LOC>=750	80.5	0.0005	0.05/84 = 0.000595	Yes
30<=LOC<40 and 200<=LOC<300	153.0	0.0009	0.05/83 = 0.000602	No

Table 5.3: Table showing Holm procedure for adjusting p values on multiple analyses for LOC as generated by CCCC

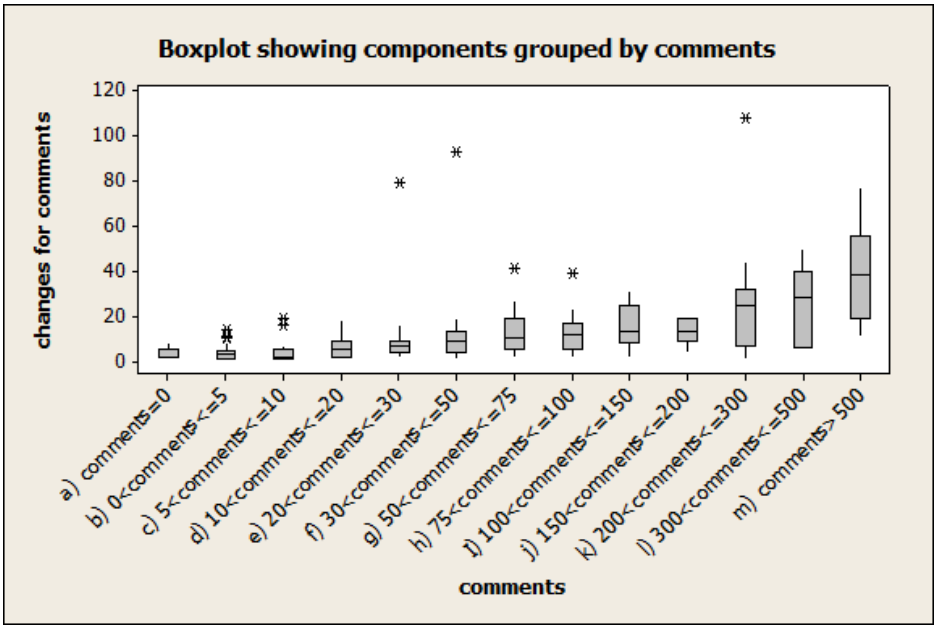

Figure 5.3: Boxplot showing the number of changes experienced by components categorised by values for comments

different comparisons include a comparison between one of the three highest-value categories (where comments>200) and a low-value category. However, higher- and mid-range numbers of comments don't differ from each other, so a very high number of comments (>200) only makes a significant difference to the likely number of changes when compared to a very low number of comments (<50).

Pairing	**W**	***p* (adj)**	α	**Reject H_0?**
comments=0 & 50<comments<=75	240.5	0.0000	0.05/78=0.000641	Yes
comments=0 & 100<comments<=150	209.5	0.0000	0.05/77=0.000649	Yes
comments=0 & 300<comments<=500	206.5	0.0000	0.05/76=0.000658	Yes
0<comments<=5 & 50<comments<=75	410.5	0.0000	0.05/75=0.000667	Yes
0<comments<=5 & 00<comments<=150	365.5	0.0000	0.05/74=0.000676	Yes
0<comments<=5 & 300<comments<=500	349.5	0.0000	0.05/73=0.000685	Yes
5<comments<=10 & 300<comments<=500	193.5	0.0000	0.05/72=0.000694	Yes
5<comments<=10 & 100<comments<=150	207.0	0.0001	0.05/71=0.000704	Yes
comments=0 & 30<comments<=50	274.5	0.0002	0.05/70=0.000714	Yes
comments=0 & 150<comments<=200	195.5	0.0002	0.05/69=0.000725	Yes
comments=0 & 200<comments<=300	217.5	0.0002	0.05/68=0.000735	Yes
0<comments<=5 & 200<comments<=300	367.0	0.0002	0.05/67=0.0007463	Yes
0<comments<=5 & comments>500	326.5	0.0002	0.05/66=0.000758	Yes
5<comments<=10 & 50<comments<=75	237.5	0.0002	0.05/65=0.000769	Yes
10<comments<=20 & 300<comments<=500	323.0	0.0002	0.05/64=0.000781	Yes
comments=0 & comments>500	190.0	0.0003	0.05/63=0.000794	Yes
10<comments<=20 & 100<comments<=150	349.0	0.0004	0.05/62=0.000806	Yes
comments=0 & 75<comments<=100	213.0	0.0004	0.05/61=0.00082	Yes
0<comments<=5 & 150<comments<=200	340.5	0.0004	0.05/60=0.000833	Yes
0<comments<=5 & 75<comments<=100	360.0	0.0004	0.05/59=0.000847	Yes
5<comments<=10 & comments>500	173.0	0.0004	0.05/58=0.000862	Yes
10<comments<=20 & comments>500	279.0	0.0004	0.05/57=0.000877	Yes
5<comments<=10 & 200<comments<=300	201.5	0.0005	0.05/56=0.000893	Yes
0<comments<=5 & 30<comments<=50	467.5	0.0006	0.05/55=0.00091	Yes
comments=0 & 20<comments<=30	256.0	0.0013	0.05/54=0.000926	No

Table 5.4: Table showing Holm procedure for adjusting p values on multiple analyses for comments as generated by CCCC

Based on these results, we make the following observations:

4. Components with fewer than 50 lines of comments are less likely to experience higher numbers of changes.

5. Components with more than 200 lines of comments are the most likely to see higher numbers of changes.

6. It is still possible to see no (or very few) changes to components with any number of comments, until the total number of comments is above 500.

Lines of comments, as a metric, may be skewed by its relationship with size (since a longer component would naturally tend to contain more comments - for example, see Figure 5.6 for a

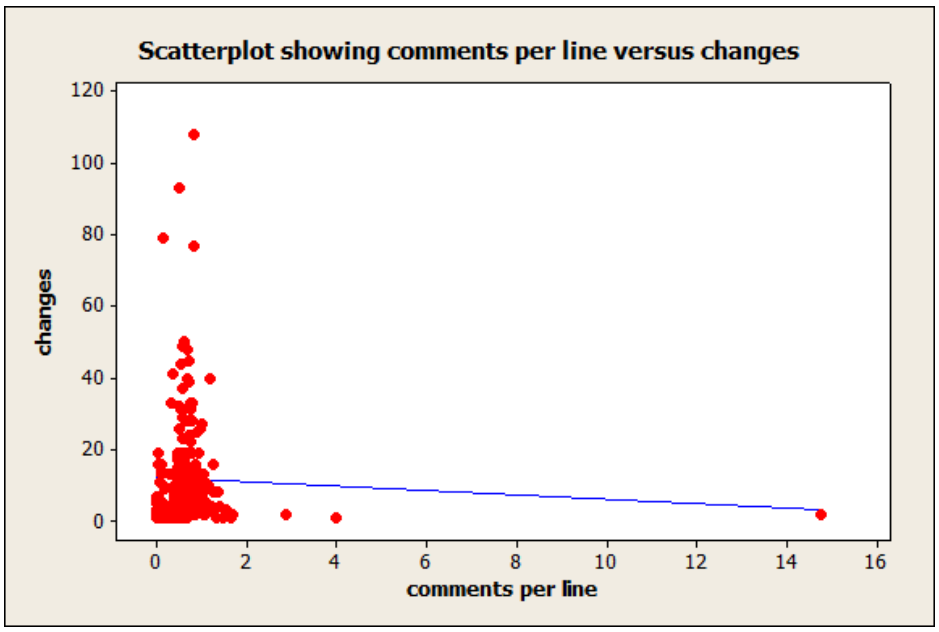

Figure 5.4: Scatterplot showing CPL plotted against LOC (both generated by CCCC)

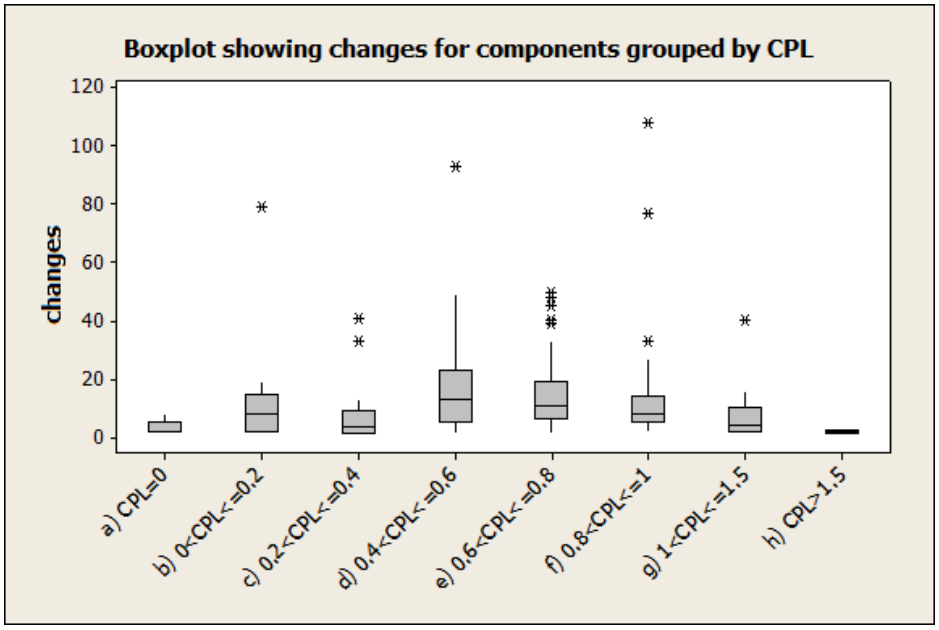

Figure 5.5: Boxplot showing the number of changes for components grouped by CPL

scatterplot showing the relationship between comments and LOC). For this reason we also use Comments Per Line (CPL, calculated as comments/LOC) as a potential metric. Figure 5.4 plots CPL against changes[6].

Figure 5.5 shows a boxplot illustrating changes for components grouped by CPL. These data have an interesting shape: unlike other metrics we analyse, the medians form (with one exception) a bell-shaped curve. This probably explains the very poor performance of the regression model for CPL. This observation is echoed in multiple comparison testing (summarised in Table 5.5), which find the lowest category (*CPL=0*) significantly different to all other categories except

[6]It is tempting to assume that the single point on the extreme right of the graph exerts an undue influence on the result. Outliers on the x axis in regression tests tend to be influential on the result and 'drag' the line towards themselves much more than outliers on the y axis. However, a similar result is achieved even if this outlying point (representing an interface with many comments but little code) is removed from the results.

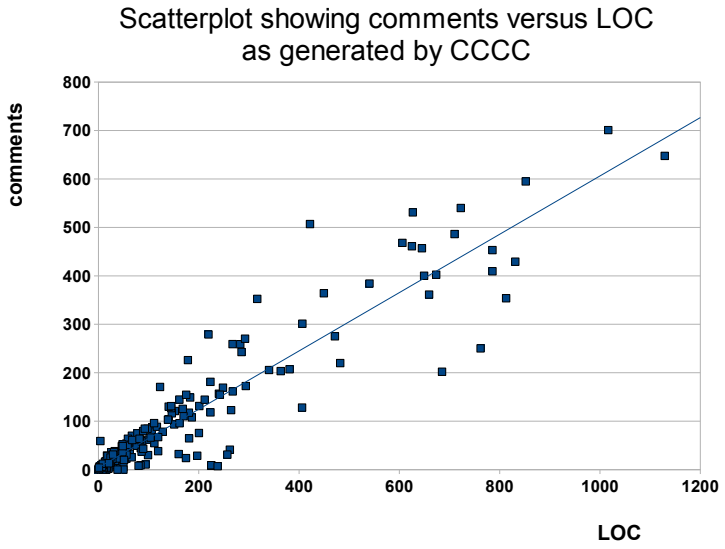

Figure 5.6: Scatterplot showing line of linear regression, calculated by Open Office Calc

Pairing	**W**	***p* (adj)**	α	**Reject H_0?**
CPL=0 and 0.4<CPL<=0.6	331.0	0.0000	0.05/28=0.001786	Yes
CPL=0 and 0.6<CPL<=0.8	296.0	0.0000	0.05/27=0.001852	Yes
CPL=0 and 0.6<CPL<=1	290.0	0.0000	0.05/26=0.001923	Yes
0.2<CPL<=0.4 and 0.6<CPL<=0.8	494.0	0.0001	0.05/25=0.002	Yes
0.8<CPL<=1 and CPL>1.5	1101.0	0.0002	0.05/24=0.002083	Yes
0.6<CPL<=0.8 and CPL>1.5	1616.5	0.0004	0.05/23=0.002174	Yes
0.6<CPL<=0.8 and 1<CPL<=1.5	2204.0	0.0007	0.05/22=0.002273	Yes
0.2<CPL<=0.4 and 0.4<CPL<=0.6	492.0	0.0012	0.05/21=0.002381	Yes
0.2<CPL<=0.4 and 0.8<CPL<=1	480.5	0.0013	0.05/20=0.0025	Yes
0.4<CPL<=0.6 and CPL>1.5	1179.0	0.0016	0.05/19=0.002632	Yes
CPL=0 and 0<CPL<=0.2	286.5	0.0232	0.05/18=0.002778	Yes
0.4<CPL<=0.6 and <CPL<=1.5	1658.5	0.0033	0.05/17=0.002941	No

Table 5.5: Table summarising results of post-hoc multiple analysis on CBO as generated by CCCC

the highest value category *CPL>1.5* and one other which has a low median. The highest value category *CPL>1.5* is found to be significantly different to the three mid-range categories (CPL between 0.4 and 1). Other differences found are between categories close to the extremes (such as *0.2<CPL<=0.4*, a category which bucks the bell-shaped trend, and *1<CPL<=1.5*) and mid-range values.

We make the following observations for CPL:

7. A low value of CPL (below 0.4) or a higher value (above 1) are less likely to experience high numbers of changes

8. A mid-range value (between 0.4 and 1) is more likely to experience a high number of changes.

9. A component can experience few or no changes irrespective of the CPL.

We would expect that a higher proportion of comments per line of code would accompany the most difficult and/or high risk areas of functionality. This would result in a positive correlation

Pairing	W	p (adj)	α	Reject H_0?
DIT				
DIT=0 and DIT=1	4975.5	0.0000	0.05/6=0.008333	Yes
DIT=0 and 0<DIT<1	2458.5	0.0066	0.05/5= 0.01	Yes
DIT=1 and DIT>1	10300.0	0.0096	0.05/4=0.0125	Yes
0<DIT<1 and DIT>1	139.0	0.0179	0.05/3=0.016667	No
NOC				
NOC=0 and 0<NOC<=1	22489.0	0.0053	0.05/3=0.016667	Yes
NOC=0 and NOC>1	22850.0	0.0096	0.05/2=0.025	Yes
0<NOC<=1 and NOC>1	54.0	0.3900	0.05/1=0.05	No

Table 5.6: Table summarising results of post-hoc multiple analysis on DIT as generated by CCCC

between CPL and changes. What we see, however, is a gradual increase until CPL reaches around 0.5, followed by an equally gradual decline. We suspect that the assumption that more comments indicates more complex code holds true up to a certain point, when other factors come into play. These could include:

- Abstract classes or interfaces, that have unimplemented declarations but relatively large quantities of comments to document intended use/functionality. High-level, detail-light components like this are likely to experience fewer changes.
- Components where old code has been commented out. Such a component will have a high CPL - but this figure does not actually measure actual comments which aid understanding.
- Components with licensing information or documentation in the comments.

In actual fact, only six components fall into the category *CPL>1.5* in CARMEN: two are interfaces; and one contained licensing information in the comments, implying the factors above do play a role in CARMEN. Individual programmer style can also vary enormously: in CARMEN's case, representation of individual programmers in the six components with high CPL do not reflect the project norms (i.e., the most prolific programmer is underrepresented in this small group), so this could be a factor. It may be that some programmers naturally comment more profusely/sparsely than others (or that they format their comments differently, resulting in an apparently skewed line count). Or that different programmers handle different areas of functionality, some of which require more comments than others. Samples from other projects would have to be studied to see if results from CARMEN can be generalised. In general, we treat comments and CPL with caution as we compare performance of different metrics in Chapter 7.

Finally, as mentioned in Section 5.2, Kruskal-Wallis tests can tolerate groups with different sizes. However, it is possible that the small number of components in the group *CPL>1.5* is not fully representative and presents a skewed picture. This factor may help to explain the low number of changes associated with the group *CPL>1.5*, but other, higher-value groups (where CPL>0.8) do also show low medians and interquartile ranges, so it is unlikely to be the sole reason.

5.6.3 DIT and NOC

DIT and NOC are concerned with inheritance. In our case study, both rendered very granular results, with a short range of potential values. Despite this, Kruskal-Wallis tests find significant differences between components grouped by DIT or NOC values (accompanying boxplots shown in Figure 5.7 and 5.8). We'd normally expect both DIT and NOC to be inteteger values. However,

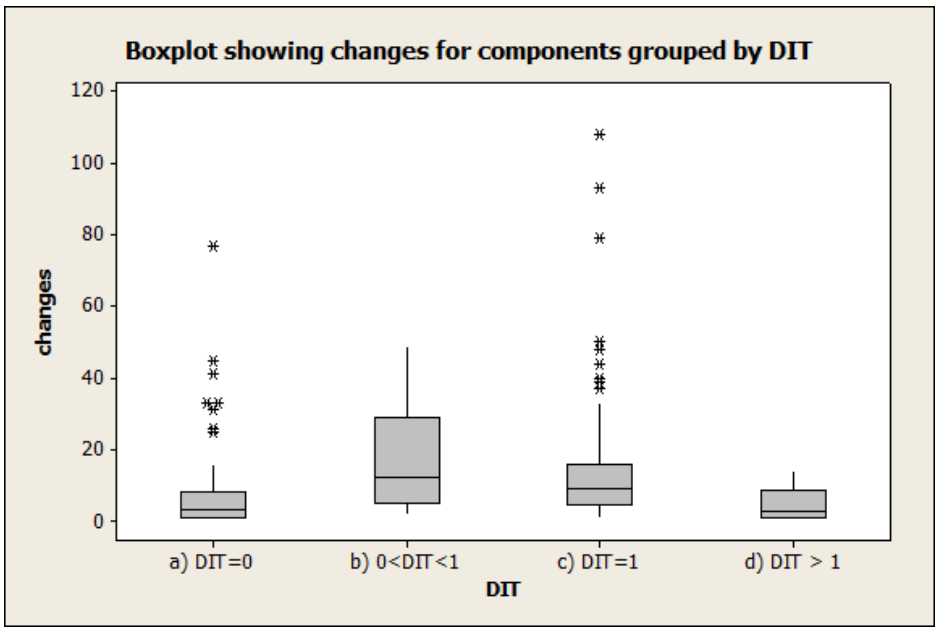

Figure 5.7: Boxplot showing the number of changes experienced by components categorised by values for DIT

we have averaged the values over a series of snapshots, so components for which DIT or NOC has altered at some point have a non-integer value. This effect may distort our results when we create groups that include or exclude integer and non-integer values, because the non-integer grouping automatically includes components which we know must have been affected by at least one change over their life (admittedly, it may be a change taking place in the inheritance tree external to the component itself and not a change within the component).

For example, the boxplot in Figure 5.7 appears to suggest that the category *0<DIT<1* has a higher median - and wider interquartile range - than either *DIT=0* or *DIT=1*. This affects the group *0<DIT<1*, but does not affect the group *DIT>1* to the same extent because the latter only contains one non-integer value. The multiple comparison tests in Table 5.6 indicate that significant differences exist between *DIT=0* and *0<DIT<1*, and between *DIT=0* and *DIT=1*, but *not* between the lowest values in *DIT=0* and the highest values, in *DIT>1*. It seems superficially that values over 0 and up to and including 1 carry the highest risk of higher changes, but then the risk reduces as DIT climbs. Because there are very few non-integer values in *DIT>1* we must be careful before drawing this conclusion.

We make the following observations:

10. Components with DIT greater than 0, but not exceeding 1, are more likely to experience changes than components with many or no children.

11. Components can experience few or no changes irrespective of the value of DIT.

In contrast, a higher value for NOC appears to suggest a *decreased* tendency to change, which can be seen Figure 5.8. This observation is also supported by the multiple comparison testing (see Table 5.6), which indicate that significant differences between the group *NOC=0* and other groups exist, but not between the two non-zero groups.

Unlike DIT, for the NOC metric, non-integer categories are not associated with a higher overall rate of change. The category *0<NOC<=1* only contains one component with an actual non-integer value, though; all other values in the category are 1.

Our observations are:

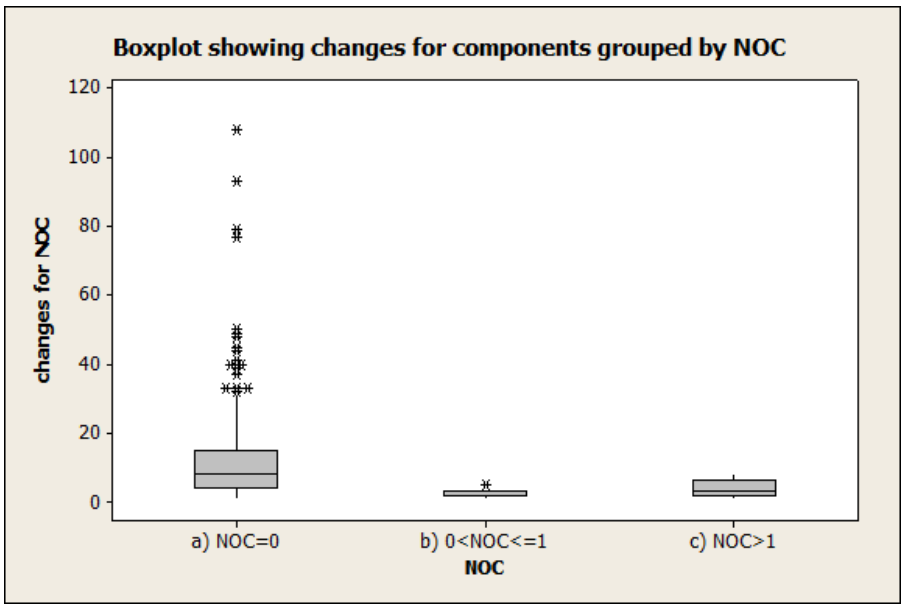

Figure 5.8: Boxplot showing the number of changes experienced by components categorised by values for NOC

12. Components with a value of 0 for NOC are more likely to change than others.

13. Little or no change can be achieved irrespective of NOC.

Our results suggested that components with *no* ancestors (i.e., *DIT=0*) may experience fewer changes. Such components may have fewer links with other classes, and therefore fewer channels through which changes may be propagated. Alternatively, components with DIT of 0 may be abstract or unused classes or interfaces which are less likely to change.

At the other end of the scale, components for which *DIT>1* are by definition located closer to the 'leaves' in the inheritance tree, and are therefore perhaps less 'important', attracting fewer functional changes. These components are also likely to be implementing modular, encapsulated functionality, which is less likely to be closely coupled to other classes and therefore less likely to suffer propagated changes.

Some of the components in the category *NOC=0* must be components which are 'leaves' at the lowest level of the inheritance tree, and therefore, according to results for DIT, we would expect this group to experience fewer changes. Despite this, the results for NOC suggest that this group experience *more* changes. However, because inheritance is not used extensively in CARMEN, a majority of components in the system fall into the groups *NOC=0* and *DIT=0*, so any pattern of changes for 'leaves' components may be subsumed and difficult to see.

5.6.4 CBO

Previous research has found that CBO can be used to identify some types of change-prone classes; for example, [152] found that CBO identified change-prone classes (although not necessarily those prone to 'change ripples' - defined as changes that affected more than one class). Briand *et al.* [26] found that CBO and some other metrics were significant in searching for common changes between pairs of classes, but many change ripples were not explained by CBO metrics.

In our case study, the differences detected by the multiple comparisons (see Table 5.7) tended to occur between components with low CBO values, and those with higher values. More than half of the pairings where a significant difference was found include the category *CBO<=1*, and almost

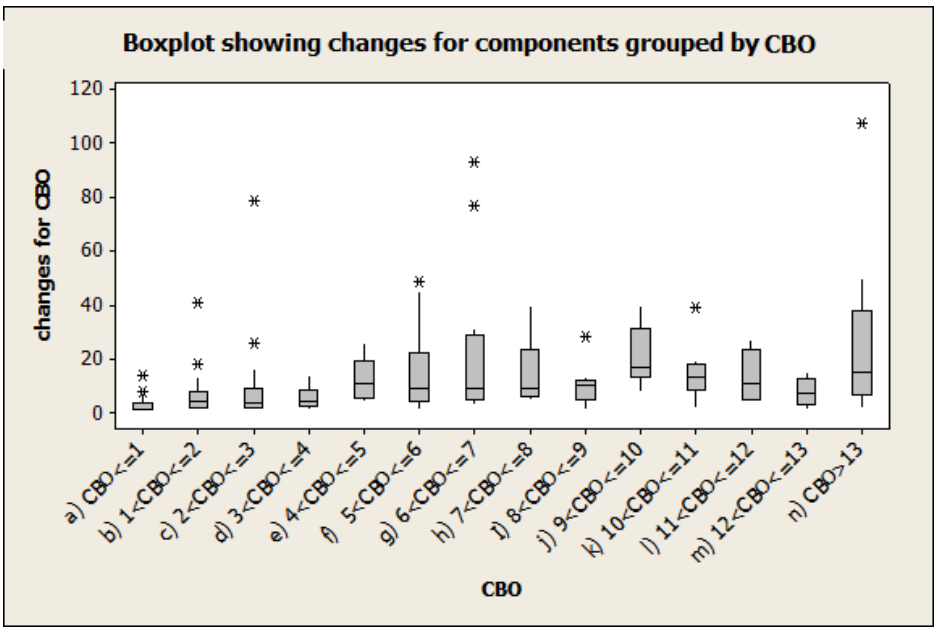

Figure 5.9: Boxplot showing the number of changes experienced by components categorised by values for CBO

half of the pairings also include one of the three groups with the highest values for CBO. The group of components *CBO<=1* differs significantly to almost every other category. The boxplot in Figure 5.9, shows that the interquartile range for this category is much shorter, and the median lower, than for any other category.

Overall, for values of 4 or less, the interquartile range continues to be noticeably shorter, and the median lower, than for most other categories. This observation is backed up by the multiple comparisons: the lowest-value group which is significantly different to *CBO<=1* is *4<CBO<=5*. Although the group *CBO<=1* is significantly different to almost every other group, the groups *1<CBO<=2* and *2<CBO<=3* are also significantly different to other, higher-value groups. A substantial increase in interquartile range occurs when CBO exceeds 13 (the highest-value group), and multiple analysis also highlights this group as having significant differences to components with lower values.

There are several mini peaks in interquartile range and upper limits in the centre of the box plot, but the median is reasonably stable until *9<CBO<=10*. There is then an increase in both interquartile range and median which gradually declines until the final category. *9<CBO<=10* is unusual in that there are no components experiencing no changes at all. Despite this, the multiple comparison testing finds that category *9<CBO<=10* is significantly different to categories for which CBO is below 4, but not to other high value categories.

We make the following general observations:

14. Values of CBO below 4 or 5 are less likely to be highly change-prone.

15. Values of 13 or over are more likely to be highly change-prone.

16. It is possible for a component to experience few or no changes in almost any category.

Components with little to no coupling to other components experience fewer changes. This could be because the components are likely to be smaller and/or less 'important' components, or even abstract classes, which are less affected by changes to requirements. Alternatively, with few coupling links there are few channels to propagate change to components in this group.

Pairing	W	p (adj)	α	Reject H_0?
CBO<=1 and 4<CBO<=5	322.5	0.0000	0.05/91=0.000549	Yes
CBO<=1 and 5<CBO<=6	378.5	0.0000	0.05/90=0.000556	Yes
CBO<=1 and 6<CBO<=7	339.5	0.0000	0.05/89=0.000562	Yes
CBO<=1 and 9<CBO<=10	303.5	0.0000	0.05/88=0.000568	Yes
CBO<=1 and 10<CBO<=11	319.5	0.0000	0.05/87=0.000575	Yes
CBO<=1 and CBO>13	326.0	0.0000	0.05/86=0.000581	Yes
1<CBO<=2 and 9<CBO<=10	429.5	0.0000	0.05/85=0.000588	Yes
CBO<=1 and 7<CBO<=8	315.0	0.0001	0.05/84=0.000595	Yes
1<CBO<=2 and CBO>13	478.5	0.0001	0.05/83=0.000602	Yes
2<CBO<=3 and 9<CBO<=10	438.0	0.0001	0.05/82=0.00061	Yes
CBO<=1 and 11<CBO<=12	314.0	0.0002	0.05/81=0.000617	Yes
3<CBO<=4 and 9<CBO<=10	97.0	0.0002	0.05/80=0.000625	Yes
2<CBO<=3 and CBO>13	493.0	0.0004	0.05/79=0.000633	Yes
CBO<=1 and 8<CBO<=9	332.5	0.0006	0.05/78=0.000641	Yes
1<CBO<=2 and 10<CBO<=11	461.0	0.0008	0.05/77=0.000649	No

Table 5.7: Table summarising results of post-hoc multiple analysis on CBO as generated by CCCC

In Section 5.6.3 we referred to research by Arisholm *et al*, which found that export coupling (i.e., where A *is called by* B) was linked to changes [2], so the direction of the link may also play a role in governing the number of changes. We don't have any data on this from the CARMEN case study, however.

5.6.5 WMC

Pairing	W	p (adj)	α	Reject H_0?
0<WMC<=1 and 7<WMC<=8	368.5	0.0000	0.05/91=0.000549	Yes
0<WMC<=1 and 10<WMC<=15	362.0	0.0000	0.05/90=0.000556	Yes
0<WMC<=1 and WMC>20	336.5	0.0000	0.05/89=0.000562	Yes
1<WMC<=2 and WMC>20	618.5	0.0000	0.05/88=0.000568	Yes
1<WMC<=2 and 10<WMC<=15	681.0	0.0000	0.05/87=0.000575	Yes
1<WMC<=2 and 7<WMC<=8	674.0	0.0001	0.05/86=0.000581	Yes
2<WMC<=3 and WMC>20	257.5	0.0001	0.05/85=0.000588	Yes
0<WMC<=1 and 8<WMC<=9	326.5	0.0002	0.05/84=0.000595	Yes
0<WMC<=1 and 6<WMC<=7	341.0	0.0002	0.05/83=0.000602	Yes
WMC=0 and WMC>20	60.5	0.0002	0.05/82=0.00061	Yes
1<WMC<=2 and 8<WMC<=9	601.5	0.0003	0.05/81=0.000617	Yes
2<WMC<=3 and 10<WMC<=15	300.0	0.0004	0.05/80=0.000625	Yes
1<WMC<=2 and 6<WMC<=7	634.0	0.0006	0.05/79=0.000633	Yes
2<WMC<=3 and 8<WMC<=9	235.0	0.0006	0.05/78=0.000641	Yes
WMC=0 and 10<WMC<=15	76.0	0.0007	0.05/77=0.000649	No

Table 5.8: Table summarising results of post-hoc multiple analysis on WMC as generated by CCCC

The Kruskal-Wallis test indicates that the lowest value group, *WMC=0*, is not the group least likely to experience change: the interquartile range decreases slightly for low non-zero values. That is, if WMC is between 0 and 2, components seem less likely to be associated with higher numbers of

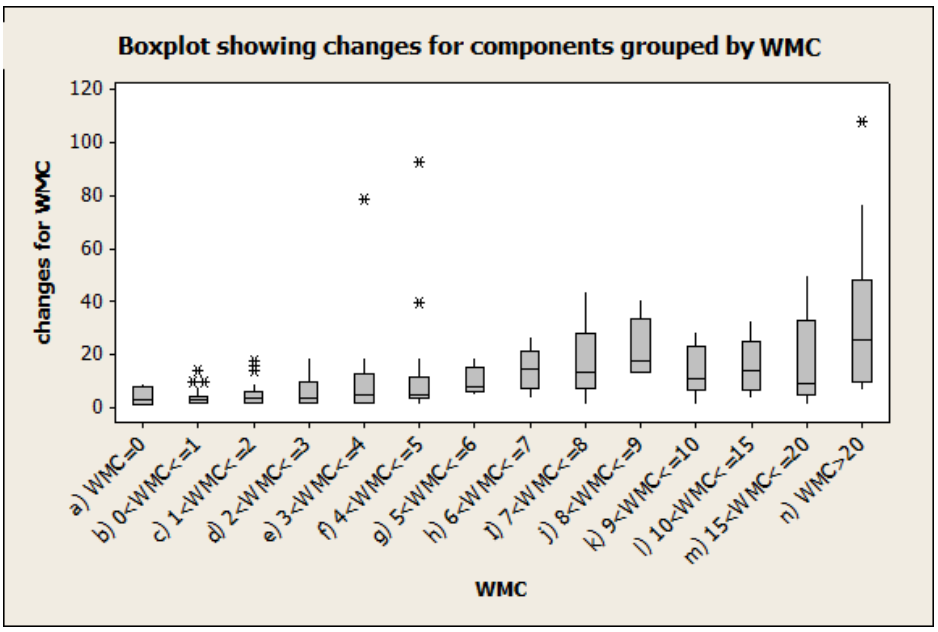

Figure 5.10: Boxplot showing the number of changes experienced by components categorised by values for WMC

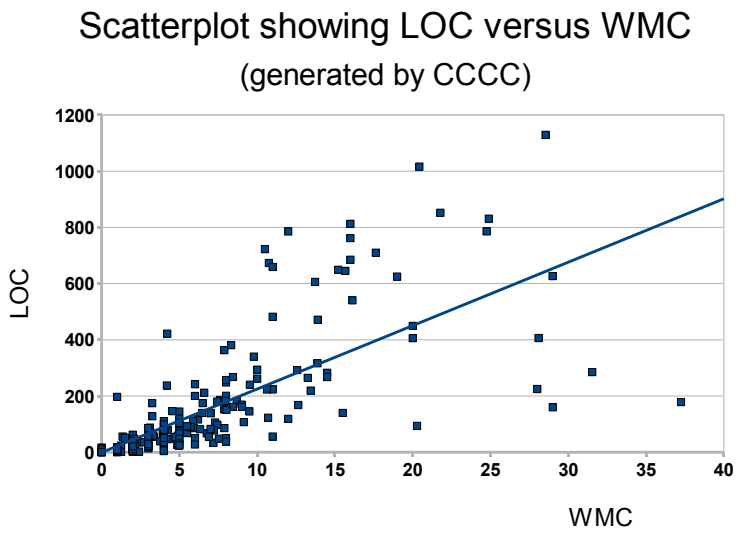

Figure 5.11: Scatterplot showing LOC versus WMC

changes than components for which *WMC=0*. The interquartile range increases slightly as WMC climbs towards 3, and then jumps (as does the median) as WMC reaches 6 or 7. There is a slight dip in the median as WMC passes between 9 and 20.

These observations are supported by the multiple comparison tests (see Table 5.8). The two categories *0<WMC<=1* and *1<WMC<=2* appear in 10 (out of 14) significantly different pairings, whilst the category *WMC=0* only appears to differ significantly to one other category. In these multiple comparisons, there's a general trend of lower values (WMC between 0 and 3) being significantly different to higher values of WMC (of approximately 7 and above). We make the following general observations:

17. Components with WMC between 0 and 3 are less likely to experience higher numbers of

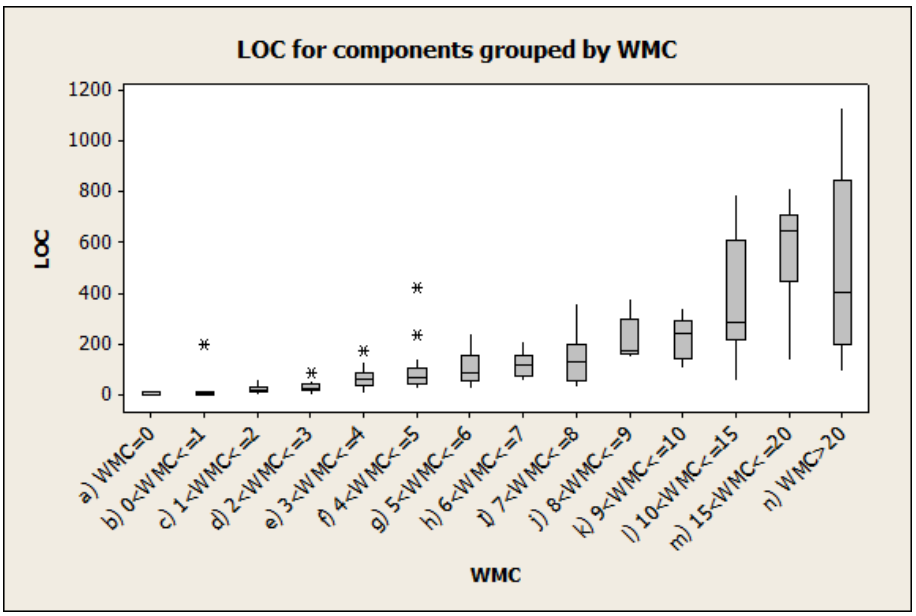

Figure 5.12: Boxplot showing LOC for components grouped by WMC

change.

18. WMC between 6 and 9, or over 20 are most likely to experience higher numbers of changes.

WMC can be thought of as a type of size metric (Chidamber and Kemerer originally allowed for an optional weighting [31], but the data we are using here does not employ a weighting; see Chapter 4). Size has been correlated with change before (e.g., see [94]). Therefore we are not surprised that the most extreme category (here, *WMC>20*) is most change-prone. Components with highest WMC are likely to be key components that implement core functionality, or that instantiate and call upon many other components. Figure 5.11 shows a scatterplot plotting LOC versus WMC. The scatterplot shows that there is a reasonable degree of correlation between the two variables, although this is stronger for lower values, and drifts more as the values increase. Figure 5.10 shows that higher values of WMC tend to experience higher numbers of change; but from Figure 5.11 it is clear that higher values of WMC are not uniformly associated with high LOC. Finally, Figure 5.12 shows values of LOC for components grouped by WMC. The higher categories here also do not show a uniform increase: in fact, the highest value group, *WMC>20*, shows a subtantial drop in median compared to the neighbouring group, *15<WMC<=20*. We conclude that WMC is not simply a surrogate for LOC, but is selecting components with a tendency to change-proneness which may not be identifiable using LOC as well.

Components with very high numbers of methods could play a particular role in the system; perhaps they provide many services (methods) for use by other components. This would result in primarily *export* couplings as defined by Arisholm *et al.*, who found that export coupling is more likely to be associated with change than import coupling. Alternatively, perhaps components with very high numbers of methods - irrespective of their total LOC or CBO values - are key drivers which are responsible for instantiating, storing or marshalling other classes, which are affected by functional changes to requirements.

5.6.6 McCabe's MVG

This metric is normally presented as a figure per method; CCCC generates a total MVG per class. The multiple comparisons in Table 5.9 suggest that significant differences lie between lower values

Pairing	W	p (adj)	α	Reject H_0?
0<=MVG<1 and 15<=MVG<30	4461.0	0.0000	0.05/36=0.001389	Yes
0<=MVG<1 and 50<=MVG<100	4368.5	0.0000	0.05/35=0.001429	Yes
0<=MVG<1 and 5<=MVG<10	4804.5	0.0001	0.05/34=0.001471	Yes
1<=MVG<2 and 50<=MVG<100	153.0	0.0001	0.05/33=0.001515	Yes
0<=MVG<1 and 10<=MVG<15	4588.5	0.0002	0.05/32=0.001563	Yes
1<=MVG<2 and 15<=MVG<30	171.0	0.0003	0.05/31=0.001613	Yes
0<=MVG<1 and 2<=MVG<5	4998.5	0.0004	0.05/30=0.001667	Yes
0<=MVG<1 and 30<=MVG<50	4402.0	0.0005	0.05/29=0.001724	Yes
2<=MVG<5 and 50<=MVG<100	473.0	0.0013	0.05/28=0.001786	Yes
10<=MVG<15 and 50<=MVG<100	175.0	0.0047	0.05/27=0.001852	No

Table 5.9: Table summarising results of post-hoc multiple analysis on MVG as generated by CCCC

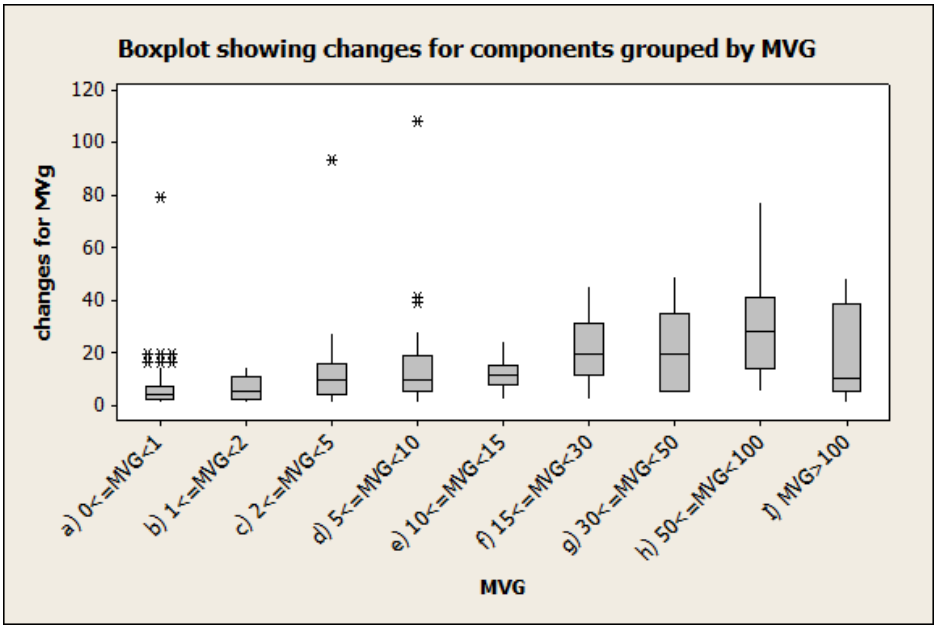

Figure 5.13: Boxplot showing the number of changes experienced by components categorised by values for MVG

for MVG and higher values (almost all the significant differences include a pairing that includes one of the lowest-value categories). There is a significant difference found between the two low-value categories themselves, however (*0<=MVG<1* and *2<=MVG<5*). We can see that the median value for number of changes does creep upwards fairly steadily (although not infalliably) as MVG increases.

The highest value category (*MVG>100*), shows a sudden sharp drop in median number of changes. Changes increase as MVG reaches 15, but between 15 and 100 do not vary substantially (the multiple comparisons do not find any significant differences between these categories). We make the following general observations:

19. Components with MVG below 15 are less likely to experience high numbers of changes than components with MVG above 15.

20. Number of changes generally increases as total MVG for a class increases, until MVG reaches 15, when it plateaus.

21. For values of MVG higher than 100, number of changes is likely to fall.

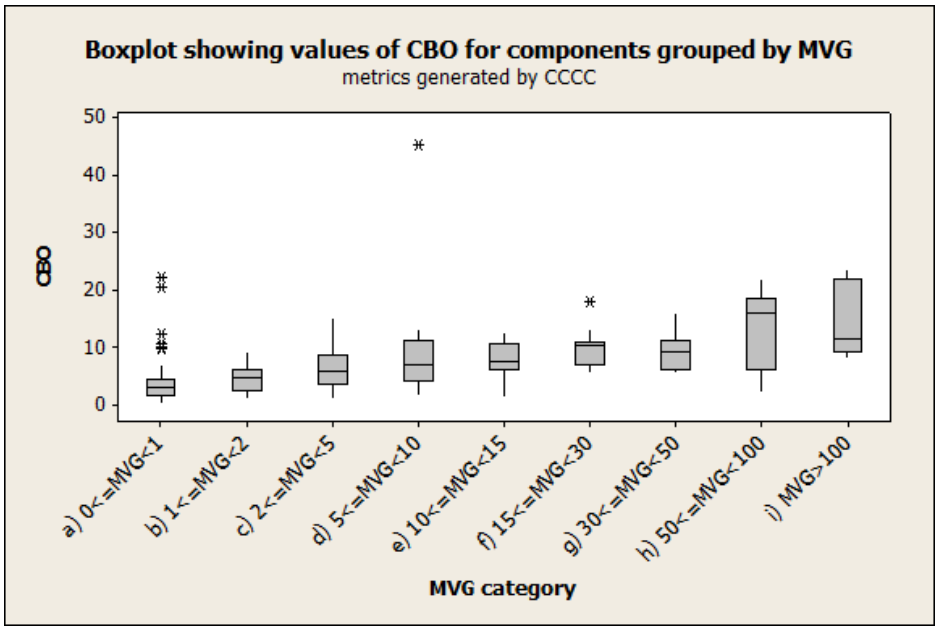

Figure 5.14: Boxplot showing CBO for components categorised by values for MVG

The MVG value can suggest to us which components may be the most complex in terms of the number of paths (a higher number of paths may also imply greater size), and perhaps therefore identify to us the key components which instantiate and marshall other components. Small number of components with very high value for MVG could be classes that exist merely to navigate graphical menus or lists, or other relatively simple features, and are therefore not impacted as heavily either by requirements changes. This could explain changes falls as MVG tops 100.

To investigate this further, Figure 5.14 presents a boxplot showing the value of CBO achieved by components grouped by MVG. The two variables seem to be approximately correlated: as the median number of changes climbs or falls, so does the CBO. Also CBO generally climbs as MVG increases. This is not unexpected, as code with more paths is more likely to be linked to other components. The components with the highest values for MVG seem to have fewer coupling links; this could support the hypothesis that these components probably are relatively simple, standalone classes that navigate switches and options like those found in the graphical user interface.

5.7 Summary

Table 5.10 presents a summary of our observations on our metrics.

Table 5.10: Table summarising findings for our metrics

Metric	No.	Observation
LOC	1.	Low values of LOC (below 40) are less likely to experience higher numbers of changes.
	2.	Components with LOC above 300 are the most likely to high numbers of changes.
	3.	It is still possible to see no (or very few) changes for any

Continued on next page

Table 5.10 – continued from previous page

Metric	No.	Observation
		value of LOC.
Comments	4.	Components with lower numbers of comments (below 50) are less likely to experience higher numbers of changes.
	5.	Components with more than 200 comments are the most likely to see higher numbers of changes
	6.	It is still possible to see no (or very few) changes to components with any number of comments, until the total number of comments is above 500.
CPL	7.	A low value of CPL (below 0.4) or a higher value (above 1) are less likely to experience high numbers of changes
	8.	A mid-range value (between 0.4 and 1) is more likely to experience a high number of changes.
	9.	A component can experience few or no changes irrespective of the CPL.
DIT	10.	Components with DIT greater than 0, but not exceeding 1, are more likely to experience changes than components with many or no children.
	11.	Components can experience few or no changes irrespective of the value of DIT.
NOC	12.	Components with a value of 0 for NOC are more likely to change than others.
	13.	It is possible to achieve little or no change irrespective of NOC.
CBO	14.	Values of CBO below 4 or 5 are less likely to be highly change-prone.
	15.	Values of 13 or over are more likely to be highly change-prone.
	16.	It is possible for a component to experience few or no changes in almost any category.
WMC	17.	Components with WMC between 0 and 3 are less likely to experience higher numbers of change.
	18.	WMC between 6 and 9, or over 20 are most likely to experience higher numbers of changes.
MVG	19.	Components with MVG below 15 are less likely to experience high numbers of changes than components with MVG above 15.
	20.	Number of changes generally increases as total MVG for a class increases, until MVG reaches 15, when it plateaus.
	21.	For values of MVG higher than 100, number of changes is likely to fall.

5.8 Discussion

At the start of this Section we referred to our first hypothesis:

H_1: Existing complexity metrics can be used to predict change-proneness.

As we saw in Section 2.4.4, previous research has looked at existing metrics and whether they can reliably be used to make predictions about change (results from a range of studies are summarised in Table 2.4.5).

5.8.1 Regression results

Looking at our results, we conclude firstly that there is no evidence of significant linear relationship between the existing complexity metrics we have tested, and the number of changes experienced by a component. None of the regression models we constructed produced normally-distributed residuals (and were therefore unreliable).

Some other researchers have also concluded that regression results do not result in accurate prediction models for predicting change. Koten and Gray, for example, [146] used Bayesian networks, a regression tree, and multiple regression models to search for correlations between L&H metrics and number of lines changed, for several case study data sets. Some differences in accuracy between the techniques were found. However, the authors concluded that none of the models they constructed 'satisfy the criteria of an accurate prediction model' [146]. This study differs from ours in a few respects; change was assessed as number of lines changed rather than in number of times a component was checked-in. The case study was implemented in a different language. And the L&H metric set includes LCOM, the number of properties per class ('SIZE2'), and replaces CBO with the L&H metrics DAC, MPC and RFC. Despite the differences, the underlying principle suggests that change and complexity metrics do not participate in an accurate linear model.

Koten and Gray re-use data sets published by Li and Henry [94], who had originally used multiple linear regression to test for correlations between their metrics and number of lines changed. However, whilst Koten and Gray conclude that the models they construct (using multiple linear regression amongst other techniques) are not entirely accurate, Li and Henry conclude that there is a 'strong relationship between the metrics and maintenance effort' in the systems studied [94]. Their models suggest that high proportions (87%) of variance in the number of lines changed can be explained by their input variables (the same variables used by Koten and Gray). This discrepancy is difficult to explain without access to detailed further data. Li and Henry do not mention whether they checked for problematic data effects such as multi-collinearity or non-normally distributed residuals; perhaps effects like these are undetected.

Other studies using regression have concluded that reasonably robust links between metrics and software quality indicators can be demonstrated. Basili *et al.*, for example, found that C&K metrics could be correlated individually with fault-proneness [7] (using data generated from student projects). El Emam *et al.* [53] find that CBO, RFC, WMC and NMA (number of methods added by a subclass) can be linked to fault-proneness using univariate regression, but that this link disappears when size is treated (and controlled) as a confounding factor (see Section 2.4.6 for our discussion on this issue). Fault-proneness is clearly not the same as number of changes, but we expect a certain degree of correlation since a component containing a detected fault is generally altered as a result. Our results do not support the principle that reliable regression models can be constructed using existing complexity metrics, but this may be due to a variety of reasons, such as:

- slightly different data being gathered and analysed (for example, lines changed or faults detected rather than number of times checked-in).
- many components within CARMEN experience no or few changes, regardless of their complexity. However, CARMEN is a relatively young project and not yet established in a long-term maintenance phase - possibly data gathered from a project over a longer time-span would reveal slightly different patterns as more and more files are altered over time.
- results may not be generalisable between different object-oriented languages, or different project domains and cultures. We address limitations of our case study in Section 4.3.7.

5.8.2 Mann-Whitney and Kruskal-Wallis results

The results of our Mann-Whitney and Kruskal-Wallis tests showed that significant differences can be detected using metrics values to distinguish groups of components. Other researchers using similar techniques to analyse the predictive abilities of complexity metrics have also generally found significant differences. Chaumun *et al*, for example, used ANOVA to distinguish between groups of classes with high, low or medium values for WMC [29]. They studied the impacts of some potential changes ('impact' is defined as the average number of impacted classes by a change to each method's signature), concluding that 'there is indeed a relationship between the WMC metric and the change impact of the method signature change' [29].

Wilkie and Kitchenham use the Kruskal-Wallis test to search for significant differences in change ripples between groups sorted by metrics such as CBO, noticing that CBO identifies the most change-prone classes but not the classes most prone to ripple effect changes [152]. Our research does not discriminate between types of change (such as change ripples) but we also discover that grouping components by CBO value can identify the most change-prone classes; in our case study the highest value category for CBO also had one of the highest median number of changes, the highest outlier and the widest interquartile range.

In this chapter, we distil the results of our analysis into some general observations. However, given that this study is exploratory in nature and consists of only one case study, any such observations presented here are intended as a preliminary step and the basis for future investigations with further case studies rather than firm conclusions. Some of our results may not be generalisable to other projects. For example, we believe that individual programmer style/project norms affect lines of comments and CPL such that comparisons with other case studies may be limited.

5.9 Conclusion

We return now to our first hypothesis, H_1:

H_1: Existing complexity metrics can be used to predict change-proneness.

Overall we believe that existing complexity metrics can be shown to share a link with the number of changes experienced by a given component. For most of our metrics, the likelihood of experiencing more changes increases as the metric value increases, although for one (NOC) the opposite it true.

We assume this means that, using data which should be available at design time, some general predictions may be made about the level of changes to which components are likely to be subjected. In general, our findings thus far support previous studies which have found that a connection can be uncovered between complexity metrics and numbers of changes (although the regression techniques which have been used successfully in several other studies fail to reveal a linear correlation for our data set).

We intend to use this information as a benchmark for assessing the effectiveness of our own metrics in predicting change-prone components; we analyse these metrics in the next chapter. We then move on to evaluate the relative effectiveness of different metrics in Chapter 7.

Chapter 6

Linking our metrics to change-proneness

In the last chapter we presented and discussed results from analysis on existing complexity metrics as we addressed our first hypothesis H_1. In this chapter we turn our attention towards our hypothesis H_2, using similar analytical techniques to examine a different set of metrics.

6.1 Introductory summary

In this section we explore our hypotheses H_2 (first introduced in Section 4.4):

> *H_2: Our metrics can be used to predict change-proneness.*

To address this hypothesis we analyse components from CARMEN using our metrics and the number of changes experienced, to determine how far - if at all - our metrics can be associated with change-proneness. We use the same techniques for this work that we used for analysing existing complexity metrics (described in Section 5.3). First of all we employ linear regression techniques (described in Section 5.2). We find no evidence of a linear correlation between any of our metrics and the number of changes, however. We discuss the techniques we employed and the results in Sections 6.2 and 6.3.

Next we move on to compare mean change-proneness for different groups of components (the statistical techniques are described in greater detail in Section 5.2). As in Chapter 5, we use non-parametric alternatives (the Kruskal-Wallis and Mann-Whitney tests) because our datasets are not normally distributed. Briefly, our approach with this type of test is as follows:

- For each metric m, we partition the components into categories based on the value of m.
- We then execute one Kruskal-Wallis test against all of the categories for m. The Kruskal-Wallis test specifically looks at the values of change-proneness associated with the components in each category. The result of the test indicates whether one category tends to see higher/lower values for change-proneness than other categories.
- If the Kruskal-Wallis test indicated that a significant difference did exist then we need to find which categories this applies to. To do this, we match up every category in the test with every other category and execute a Mann-Whitney test on the pairing. As before, we employ a technique to share the error rate across many tests, explained in Section 5.2.2.

- After we have executed the Mann-Whitney tests, we examine the pairings which were substantially diferent and attempt to distil this into a series of heuristics.

Two metrics were excluded from these tests: NSG and NES did not provide us with a sufficiently wide range of values to create more than two categories. For this reason whe subject these two metrics to a two-sample Mann-Whitney test instead.

As with the existing metrics we discussed in Chapter 5, we generally discover significant differences in change-proneness between groups of components. This is the case for almost all of the metrics, although there are some exceptions. NCR (which counts conflicting requirements) was too granular for Mann-Whitney tests so we are unable to perform analysis. And NSG (a count of soft goals) did not reveal any significant differences when we used a Mann-Whitney test to compare two categories.

For metrics associated with requirements and requirements rationale (NR, SRR, NES, RI), we generally find a broadly increasing likelihood of change-proneness as the metric increases. This is only a broadly general trend; there are fluctations amongst metrics that we discuss in this chapter. This conclusion does not hold true for the averaged figure ASRR, however; a boxplot of changes for components grouped by ASRR follows a double-peaked curve, with lower likelihood of change-proneness for components with very high or very low ASRR values. We discuss this particular case further in 6.5.4.

NDP (which counts decision problems) shows a generally-increasing likelihood of change-proneness that drops slightly on the very highest-valued category. This is mirrored somewhat (although not exactly) by the averaged figure ANODP and by the metric ASOR (which counts average items of rationale for decision outcomes). It's not very clear at this stage whether the slight decrease in median changes (which is not a significant difference for any metric) is specific to CARMEN, or part of a wider trend.

For other design-related metrics (NOut, SOR, NOpt, SOS, ASOS, SOO and ASOO) we also find a broadly general increase in likelihood of change-proneness as the value of metric m increases. There are several factors that could explain this. It's possible that higher counts of requirements rationale and/or design decisions could indicate areas which have been problematic or difficult, perhaps meaning that compromises have been forced onto the project, resulting in later changes. Alternatively, there is the possibility that our metrics act as surrogates for size, and that actually the components associated with more requirements/requirements rationale/decision problems/competing options are simply the largest components. We check for correlations between our metrics and size (assessed via LOC) in Section 6.7.1, concluding that RI, SOR, ASOR may be at least partly explained by the size of the components concerned, and that these metrics may be acting as surrogates for component size. Size may be a weak confounding factor in some other design-related metrics, although appears to be less so for requirements-related metrics.

Unlike the code-based metrics analysed in Chapter 5, our metrics rely on having a rigorous understanding of the requirements and design process to populate our models. In the case of CARMEN, our models were populated retrospectively by a researcher and not by a developer or engineer from the case study project. For this reason we validate our models by showing a random sample of components and the requirements, soft goals and decision problems to which they are linked to a CARMEN programmer to check for accuracy. A CARMEN developer suggested some changes, but tests on the newly-corrected, 'perfect' sample suggest that most of our original conclusions are still valid. We discuss validation in Section 6.6.

The rest of this chapter is laid out as follows: we first search for evidence of a linear relationship using regression techniques, described in Sections 6.2 and 6.3. In Section 6.4 we move on to describe the Mann-Whitney and Kruskal-Wallis techniques to compare the changes of different

groups of components Here we present detailed analysis and results for all metrics together with the heuristics we developed for each. We also present boxplots showing the number of changes experienced by compontents in each group, as a visual representation of the data. For each metric we also consider whether any apparent relationships with size can be explained by component size, which has been correlated with change-proneness in the past (for example, see [94]). Results of analyses are examined in Section 6.5. Finally, in Section 6.7 we discuss some implications of our findings.

6.2 Regression testing on individual metrics

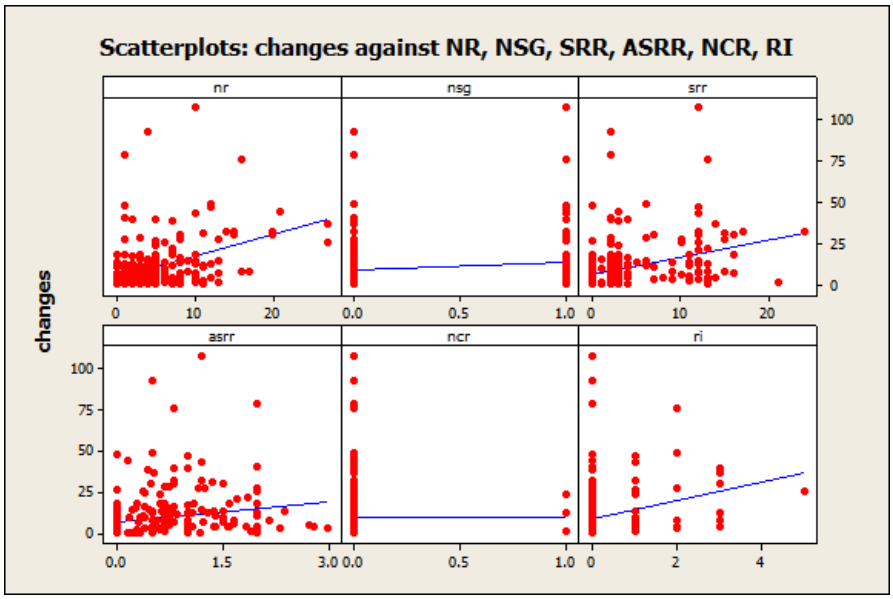

Figure 6.1: Scatter plots showing some of our metrics plotted against the number of changes

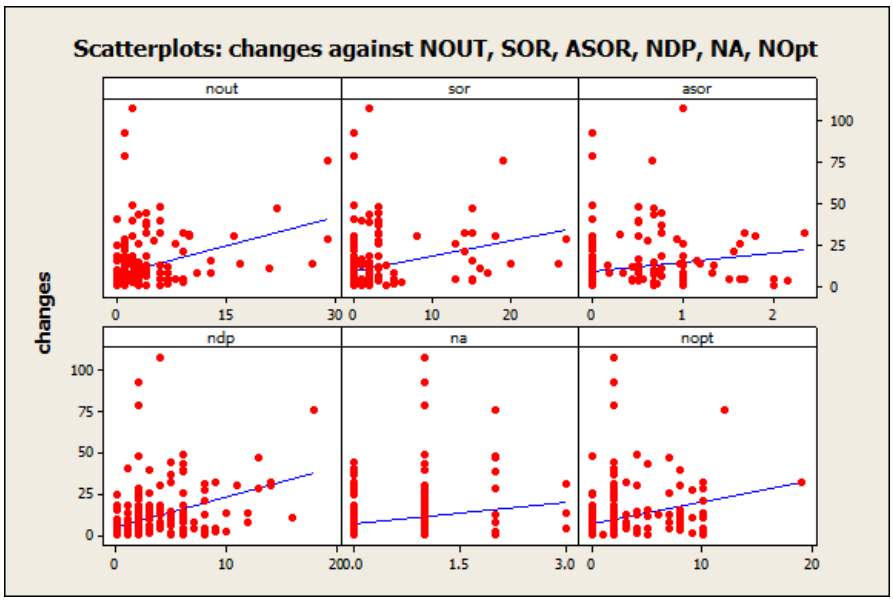

Figure 6.2: Scatter plots showing some of our metrics plotted against the number of changes

Figures 6.1, 6.2 and 6.3 present scatter-plots to illustrate visually the relationship between each of our proposed metrics and number of changes. We regressed each of our individual metrics on

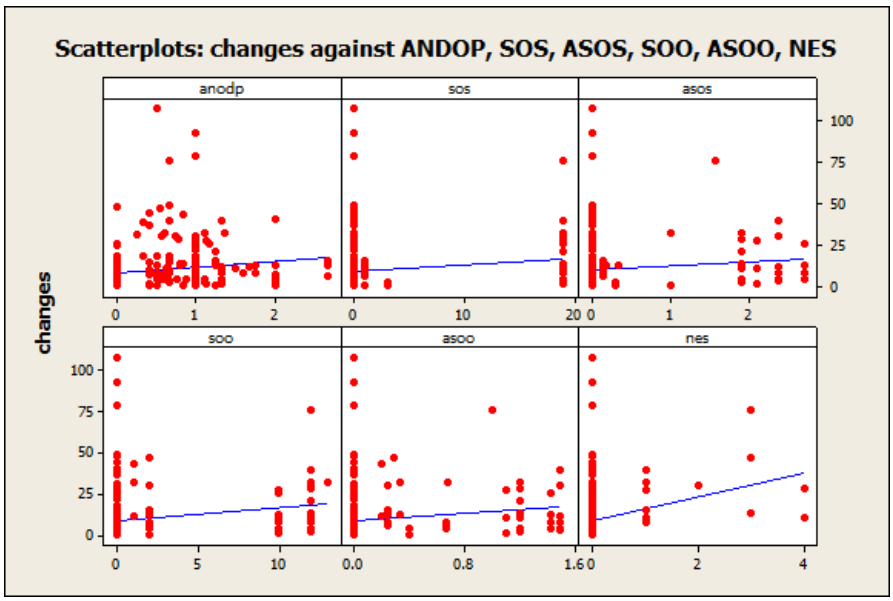

Figure 6.3: Scatter plots showing some of our metrics plotted against the number of changes

the number of changes, and followed this with an Anderson-Darling normality test on the residuals of each regression test. Key results from the regression testing for each metric are shown in Table 6.1.

Metric	***p* value**	**R-Sq (adj)%**	**Residuals normally distributed**
NR	0.00	18.1%	No
NSG	0.011	2.1%	No
SRR	0.00	11.6%	No
ASRR	0.003	3.0%	No
NCR	0.953	0.0%	No
RI	0.00	6.5%	No
NOut	0.00	11.6%	No
SOR	0.00	7.7%	No
ASOR	0.001	3.9%	No
NDP	0.00	15.0%	No
NA	0.003	3.0%	No
NOpt	0.00	7.5%	No
ANODP	0.018	1.8%	No
SOS	0.011	2.2%	No
ASOS	0.062	1.0%	No
SOO	0.004	2.9%	No
ASOO	0.017	1.9%	No
NES	0.00	7.2%	No

Table 6.1: Table summarising results of regression tests on individual metrics

Only two of our metrics (NCR and ASOS) present with a p value higher than 0.05. NCR in particular has a very high value. In fact, CARMEN does not have a high number of requirements identified as conflicting so NCR does not yield much data.

Although other metrics have low values for p the R-Sq% figures are disappointingly low (below 5% in many cases). These are low enough to suggest that a reliable linear relationship does not exist. None of the residuals for these tests are normally distributed, and so we must conclude that none of the metrics can, individually, be claimed to share a linear relationship with change-proneness.

6.3 Multiple regression

As with the existing complexity metrics, we carried out some multiple regression analysis to search for any combinations of input predictors forming a more accurate model than individual metrics. Table 6.2 summarises the results of all multiple regression models using our metrics.

6.3.1 Stepwise regression model

The results of a stepwise regression algorithm performed by Minitab are shown in Appendix G. The stepwise regression involved 6 steps, with the initial inclusion of NES. By the 6th step, however, the p value for NES has fallen below the threshold for inclusion and it is removed from the final model, which consists of NR, ASOS, ASRR and NOut. In testing, however, we do not include ASRR in this model, as the p value for ASRR exceeds our target α of 0.05. The residuals for this model, however, are not normally distributed. Thus we are not able to accept this as a reliable model.

6.3.2 'Best subsets' regression model

Appendix G shows the output generated by Minitab as a result of the 'best subsets' regression algorithm. As we discussed in Section 5.4, selecting the 'best' model from results like these involves a series of trade-offs, as we attempt to construct the most sparing model, with the most appropriate *Mallows Cp* statistic and hopefully with a relatively high R-Sq% value. Looking at the results below, we suggest that the two 9-variable models appear to offer the best compromise between a *Cp* statistic that is as close as possible to the number of input variables n and a higher R-Sq% (adj). The R-Sq% drops very slightly with fewer variables, but doesn't increase significantly by adding more variables. As the number of variables n increases above 9, *Cp* is approximately n-5; this figure doesn't get closer to the value of n as n increases, but does creep further from it with fewer variables.

We created and tested each of these two 9-variable models, which we call A and B. Neither model can make any claims to reliability: the p values for many variables in both models exceeds our target α of 0.05, and neither model produces normally-distributed residuals.

Model	**All *p* values below 0.05**	**R-Sq (adj)%**	**Residuals normally distributed**	**Identified via**
Stepwise model NR, ASOS, NOut	Yes	22.2%	No	stepwise
'Best subsets Model A' NOut, NSG, ASRR, NR, NOpt, ANODP, ASOS, SOO, ASOO	No	22.5%	No	'best subsets'
'Best subsets Model B' NR, NSG, ASRR, NCR, RI, NA, NOUT, SOS, ASOS	No	22.4%	No	'best subsets'

Table 6.2: Table summarising results of multiple regression tests on our metrics

6.4 Analysis with Mann-Whitney and Kruskal-Wallis

To search for any non-linear connections, we use the Mann-Whitney test and the Kruskal-Wallis test (both described in Section 5.2.2). NCR is excluded from all such tests, since all but four components received the same value (0) for NCR. Four is not a large enough group to achieve reliable results.

6.4.1 Kruskal-Wallis tests for comparing three or more groups

For each metric m, we divided the components up into a series of categories with different values of m, and executed a Kruskal-Wallis test for the entire set of categories. When defining categories, we try to achieve as far as possible a series of regularly-spaced groups with at least five components. Results from the Kruskal-Wallis tests are summarised in Table 6.3.

Table 6.3: Table summarising results of Kruskal-Wallis tests on our metrics

Metric	***p* value (adj)**	**H**	**Groups**	***n***	**Median**
NR	0.000	64.17	*NR=0*	57	1.0
			NR=1	28	3.0
			NR=2	35	2.0
			NR=3	29	5.0
			NR=4	20	7.5
			NR=5	19	12.0
			NR=6	9	8.0
			NR=7	10	7.5
			NR>7	47	14.0
SRR	0.000	64.36	*SRR=0*	85	1.0
			SRR=1	9	1.0
			SRR=2	71	4.0
			SRR=3	24	8.0
			SRR>3	65	12.0
ASRR	0.000	66.42	*ASRR=0*	85	1.0
			0<ASRR>=0.5	35	9.0
			0.5<ASRR<1.0	43	9.0
			ASRR=1	36	2.0
			1<ASRR<=1.5	22	13.0
			1.5<ASRR<=2.0	27	4.0
			ASRR>2.0	6	4.5
RI	0.000	17.88	*RI=0*	225	4.0
			RI=1	15	8.0
			RI=2	6	17.5
			RI>2	8	27.5
NOut	0.000	66.63	*NOut=0*	73	1.0
			NOut=1	103	4.0
			NOut=2	19	12.0
			NOut=3	13	9.0

Continued on next page

Table 6.3 – continued from previous page

Metric	p value (adj)	H	Groups	n	Median
			NOut=4	12	11.0
			NOut>4	34	11.5
SOR	0.000	34.57	*SOR=0*	180	3.0
			SOR=1	8	8.0
			SOR=2	27	8.0
			SOR=3	16	21.5
			SOR>3	23	13.0
ASOR	0.000	34.54	*ASOR=0*	180	3.0
			0<ASOR<=0.5	18	11.5
			0.5<ASOR<=1.0	38	11.0
			1.0<ASOR<=1.5	5	12.0
			ASOR>1.5	13	4.0
NDP	0.000	69.40	*NDP=0*	65	1.0
			NDP=1	29	1.0
			NDP=2	79	7.0
			NDP=3	16	10.0
			NDP=4	11	8.0
			NDP=5	13	9.0
			NDP=6	15	12.0
			NDP>6	26	11.0
NA	0.001	13.37	*NA=0*	104	3.0
			NA=1	134	7.0
			NA>1	16	10.0
NOpt	0.000	51.61	*NOpt=0*	91	1.0
			1<=NOpt<3	105	7.0
			3<=NOpt<5	15	8.0
			5<=NOpt<7	6	11.5
			7<=NOpt<9	21	8.0
			NOpt>9	16	12.5
ANODP	0.000	51.93	*ANODP=0*	91	1.0
			0<ANODP<1	46	12.0
			ANODP=1	79	7.0
			1<ANODP<2	26	8.5
			ANODP>=2	12	5.0
SOS	0.001	13.87	*SOS=0*	214	4.0
			0<SOS<=15	13	8.0
			SOS>15	27	11.0
ASOS	0.005	13.04	*ASOS=0*	214	4.0
			0<ASOS<=1	14	8.0
			1<ASOS<=2	10	13.0
			ASOS>2	16	8.0
SOO	0.000	22.32	*SOO=0*	212	4.0
			1<=SOO<6	15	11.0
			6<=SOO<11	9	8.0

Continued on next page

Metric	*N*	median	*p* value (adj)	95% Confidence Interval	W
NSG=0 and NSG>0 (one-tailed test)	183 71	4.0 6.0	0.0824	-2.999,0.001	22605.0
NES=0 and NES>0	240 14	4.0 27.5	0.0000	-26.00,-9.00	29346.5

Table 6.4: Table summarising results of Mann-Whitney tests on our metrics

Table 6.3 – continued from previous page

Metric	*p* value (adj)	H	Groups	*n*	Median
			SOO>=11	18	12.5
ASOO	0.000	22.15	*ASOO=0*	212	4.0
			0<ASOO<=1	17	12.0
			ASOO>1	25	10.0

If the Kruskall-Wallis test indicates that a significant difference exists, we then conduct multiple comparisons to determine which groups actually differ. This is accomplished by matching up each category with every other category, to create all possible pairings within the group. A Mann-Whitney comparison is executed on each individual pairing. As with similar analysis on existing metrics (discussed in Section 5.2.2) we use the Holm sequentially rejective method to control the desired error rate α. We include a summary of the key results for each individual metric in the following section (Section 6.5). The summarised results include the actual p value calculated; α (as calculated following the Holm procedure); and an indication of whether we can reject the null hypothesis H_0.

6.4.2 Mann-Whitney tests for comparing two groups

Two metrics were excluded from the Kruskal-Wallis tests. NSG only provides us with two potential values (one or zero). And whilst a relatively small numbers of components achieved a value of NES more than zero, their actual values were so disparate that the only logical category to put them in is *NES>0*. For these two metrics, then, we created two groups only: *m=0* and *M>0*. We then used a two-sample Mann-Whitney test. Table 6.4 summarises key results.

For NSG, we use a one-tailed Mann-Whitney test; i.e., we specifically test the hypotheses that the first group (*NSG*=0) has a lower median number of changes than the second group (*NSG*>0). This is because we stated that we expected higher values of NSG/NA to result in a higher number of changes, as discussed in Chapter 3.2. We did not venture to predict the direction of any correlations we find for NES, and so we employ a standard two-tailed test.

6.5 Results of our analysis

In this section we examine the results of Kruskal-Wallis and Mann-Whitney analysis for each metric in turn. Our findings are summarised in Table 6.17.

6.5.1 NR

The boxplot in Figure 6.4 shows changes experienced by components grouped by NR. Components are generally capable of achieving no or few changes irrespective of the value for NR; we find a

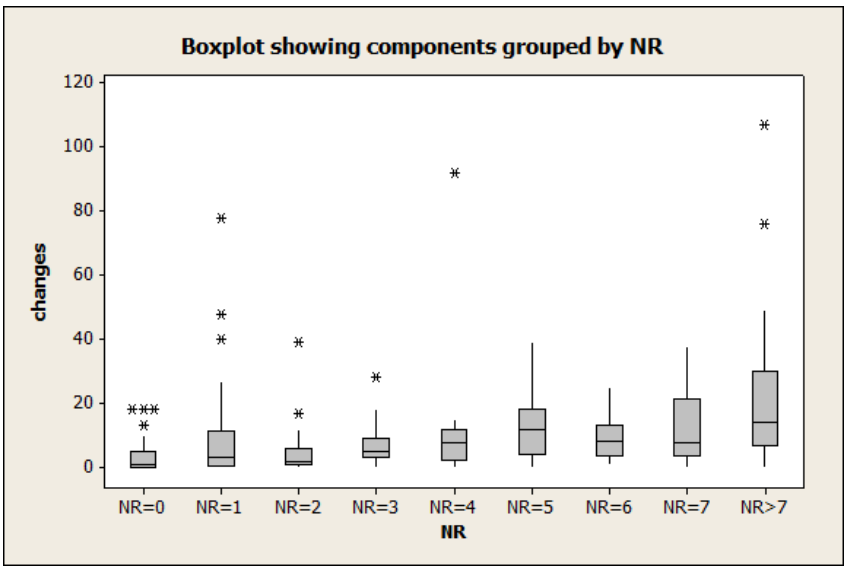

Figure 6.4: Boxplot showing the number of changes experienced by components categorised by values for NR

Pairing	**W**	***p* (adj)**	α	**Reject H_0?**
NR=0 and NR=5	1761.0	0.0000	0.05/36 = 0.001389	Yes
NR=0 and NR>7	1978.0	0.0000	0.05/35 = 0.001429	Yes
NR=2 and NR>7	909.0	0.0000	0.05/34 = 0.001471	Yes
NR=2 and NR=5	744.0	0.0001	0.05/33 = 0.001515	Yes
NR=3 and NR>7	783.0	0.0004	0.05/32 = 0.001563	Yes
NR=0 and NR=3	2034.5	0.0005	0.05/31 = 0.001613	Yes
NR=1 and NR>7	756.5	0.0008	0.05/30 = 0.001667	Yes
NR=0 and NR=6	1691.5	0.0027	0.05/29 = 0.001724	No

Table 6.5: Table showing Holm procedure for adjusting p values on multiple analyses for our metrics

similar situation in results for most of our metrics, which may be why regression fails to detect a strong linear relationship for all of our metrics.

Despite this, there is a noticeable increase in the median number of changes when NR reaches 3. Multiple comparison tests (shown in Table 6.5) support this observation; with one exception, all of the pairings which are found to be significantly different are between categories with lower values for NR (between 0 and 3) and higher values (either *NR=5* or *NR>7*).

The median increases gradually as NR increases, although *NR=6* and *NR=7* buck the trend, with medians, upper limits and interquartile ranges slightly lower than expected when compared to their neighbours. They also contain fewer components, however, than their neighbours (9 and 10 components in *NR=6* and *NR=7*, compared to 19 and 47 respectively in *NR=5* and *NR>7*). This could explain the slightly lower interquartile ranges and/or upper limits.

In general, we observe that:

1. Components with NR of 3 or greater are more likely to experience changes than other components.

2. Components can experience few or no changes irrespective of the value for NR.

In Section 3.4.2 we suggested that implementing more Requirements could result in more changes to a component, as any changes affecting the Requirements will be propagated to the components implementing them. The results here could support this theory. There is another

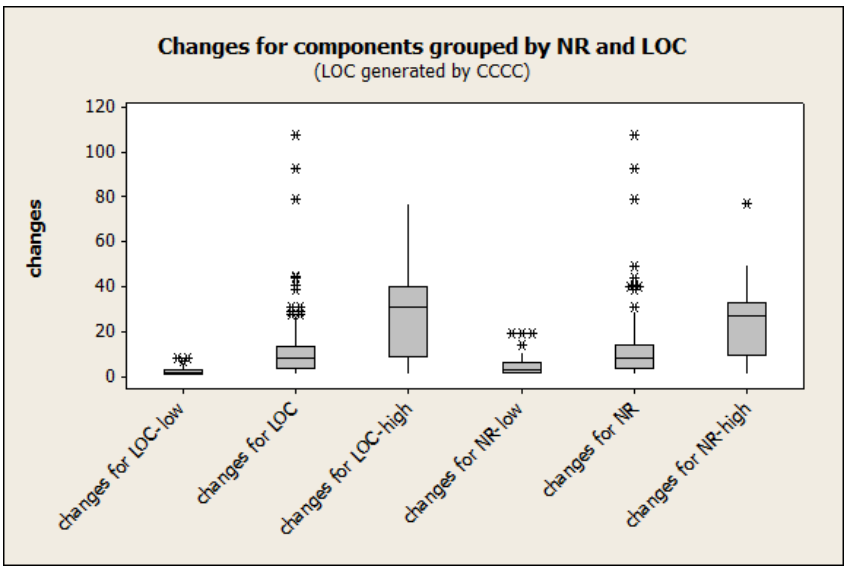

Figure 6.5: Boxplot showing values of LOC (generated by CCCC) for components categorised by NR. Components in the groups not labelled 'high' or 'low' are all other components.

explanation, however: components implementing more requirements may simply be larger (we already know that size is correlated with change-proneness - see 5.6). We discuss this in Section 6.7.1.

6.5.2 NCR

No correlation was found between NCR and number of changes by regression analysis, and the metric was considered too granular for comparison tests such as Mann-Whitney.

6.5.3 NSG

No significant differences between two groups of components (*NSG=0* and *NSG>0*) were detected by a Mann-Whitney test, suggesting NSG does not share a relationship with the number of changes experienced on a project.

6.5.4 NES, SRR and ASRR

These metrics all assess rationale recorded for requirements (although NES also considers design outcomes as well). In the CARMEN data set we identified 152 requirements, 120 of which did *not* have any recorded sources identified via the *supports* or *generates* relationship. Figure 6.6 shows changes experienced by components grouped by NES. For CARMEN, NES is a coarse-grained metric, with most components returning a value of 0. The group *NES=0* contains all but one of the highly volatile 'outliers'. On the other hand, the median, interquartile range and upper limit (excluding outliers) for *NES=0* are much higher in the group *NES>0*. The category *NES>0* is relatively usual in that no components experienced no changes. These differences lead the Mann-Whitney test (see Table 6.4) to conclude that a significant difference exists between the two groups.

Our observations are:

3. Components with a non-zero value for NES will experience changes.

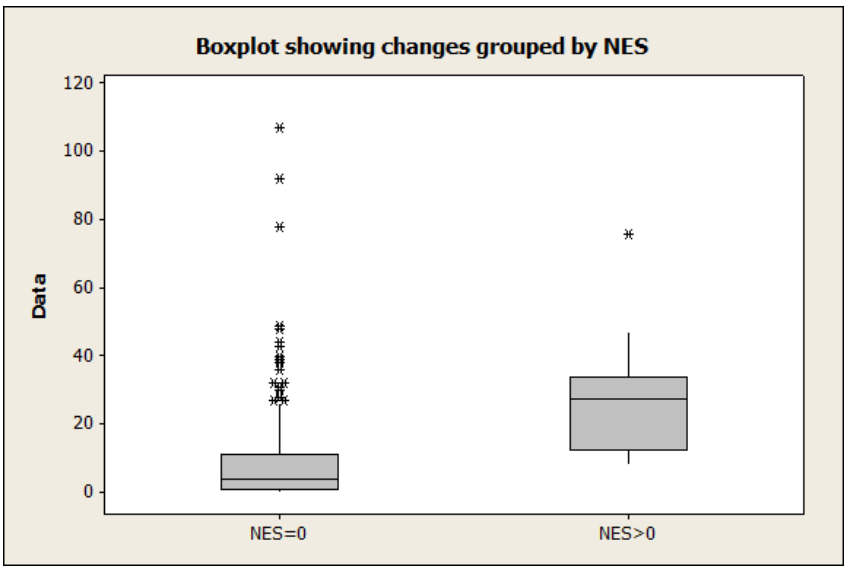

Figure 6.6: Boxplot showing the number of changes experienced by components categorised by values for NES

4. Components with a non-zero value for NES are more likely to experience high numbers of changes than other components.

5. Components can still achieve high numbers of changes even if NES is zero.

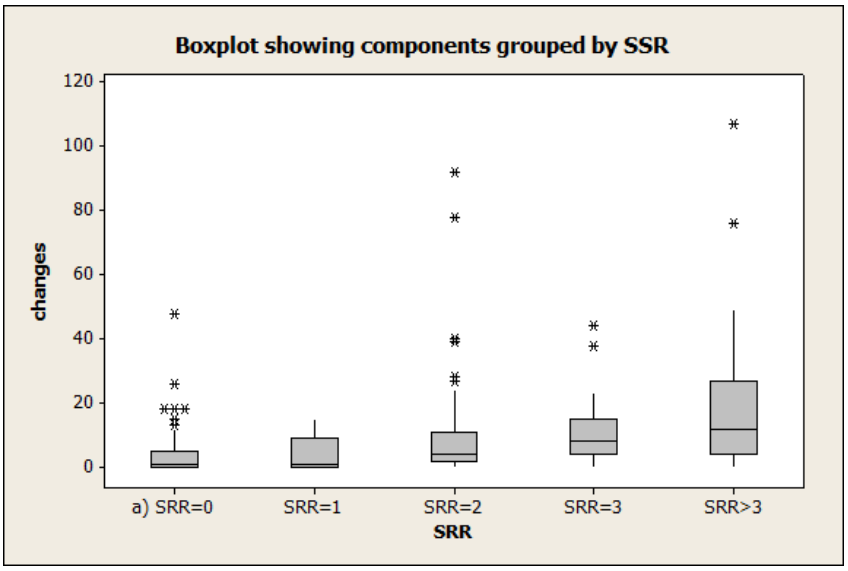

Figure 6.7: Boxplot showing the number of changes experienced by components categorised by values for SRR

Figure 6.7 shows changes for components grouped by SRR. A small but noticeable increase in the number of changes occurs as SRR increases. In particular, the median number of changes increases as SRR reaches 2, and then again as it reaches 3 and higher. The multiple comparisons tests (summarised in Table 6.6) revealed significant differences between low values (SRR below 2) and higher values (SRR greater than 2). There are also significant differences between the lowest value *SRR=0* and *SRR=2*, and between *SRR=2* and the highest value (*SRR>3*). Two seems to be a significant threshold; it's significantly different to lower values, and to higher values (which are in turn different from each other). Based on this, we make the following observations:

Pairing	W	p (adj)	α	Reject H_0?
SSR				
SRR=0 and SRR=3	4065.5	0.0000	0.05/10 = 0.005	Yes
SRR=0 and SRR>3	4525.5	0.0000	0.05/9 = 0.005556	Yes
SRR=2 and SRR>3	3901.5	0.0000	0.05/8 = 0.00625	Yes
SRR=0 and SRR=2	5593.0	0.0001	0.05/7 = 0.007143	Yes
SRR=1 and SRR>3	148.0	0.0017	0.05/6 = 0.008333	Yes
SRR=1 and SRR=3	97.0	0.0242	0.05/5 = 0.01	No
ASSR				
0.5<ASRR<1.0 & and ASRR=1	2176.5	0.0000	0.05/21 = 0.002381	Yes
ASRR=0 & 1<ASSR<=1.5	3893.5	0.0000	0.05/20 = 0.0025	Yes
ASRR=0 & 0.5<ASRR<1.0	4175.0	0.0000	0.05/19 = 0.002632	Yes
ASSR=1 & 1<ASSR<=1.5	803.5	0.0000	0.05/18 = 0.002778	Yes
ASRR=0 & 1.5<ASSR<=2.0	4261.5	0.0002	0.05/17 = 0.002941	Yes
ASRR=0 & 0<ASRR<=0.5	4540.0	0.0004	0.05/16 = 0.003125	Yes
ASSR=1 & 1.5<ASSR<=2.0	980.0	0.0167	0.05/15 = 0.003333	No

Table 6.6: Table showing Holm procedure for adjusting p values on multiple analyses for SRR and ASSR

6. Components with an SRR of 2 are more likely than those with 0 or 1 to experience higher numbers of changes.

7. Components with an SRR more than 2 are still more likely to see higher numbers of changes.

8. Components are capable of experiencing few or no changes irrespective of SRR.

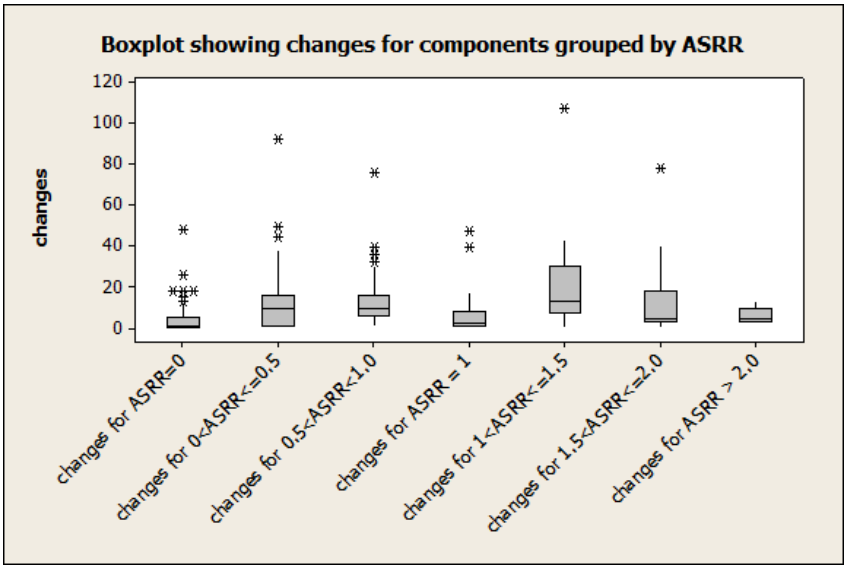

Figure 6.8: Boxplot showing the number of changes experienced by components categorised by values for ASRR

SRR is a metric liable to become skewed, since components with a higher number of requirements are naturally more likely to be connected with more requirements rationale. The metric ASRR combats this by normalising SRR to the number of requirements, producing an averaged figure. The boxplot in Figure 6.8 suggests that the median number of changes (and the interquartile range of likely numbers of changes) begin and end low (when ASRR is either at 0 or over 2.0). Medians and ranges increase between those points, although they do shrink again for the mid-range, integer category *ASSR=1*. These observations are supported by multiple comparison

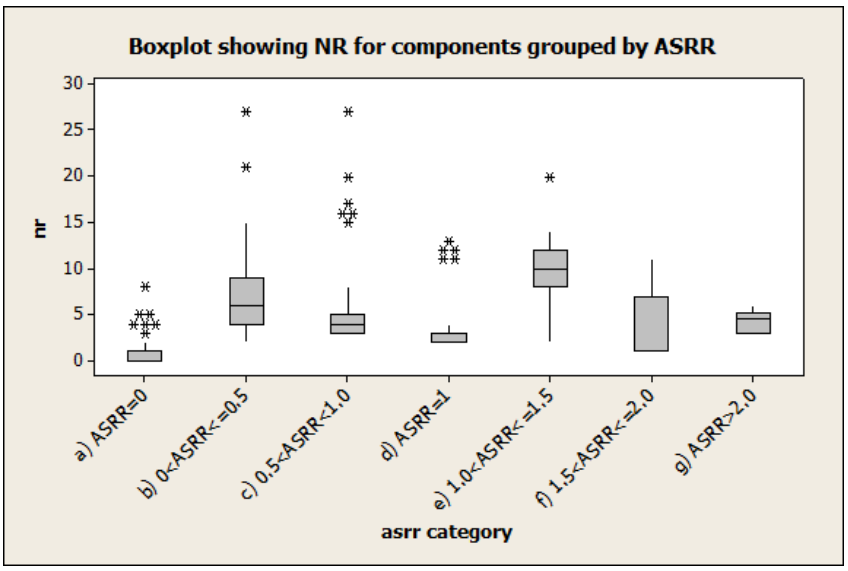

Figure 6.9: Boxplot showing NR for components categorised by values for ASRR

tests (Table 6.6), which reveal significant differences between the low-value category *ASRR=0* and three of the four categories containing non-integer values. Significant differences are also found between the category *ASSR=1* and its immediate neighbours.

ASSR>2 is not significantly different to any other category. Unlike other categories with low medians, it contains mainly non-integer values, but it also contains a small number of entities compared to other categories, which may explain the smaller range of values.

We make the following general observations:

9. Components with an ASRR of 0 or 1 are less likely to be prone to change than other components.

10. Components with ASRR between 0 and 1, or between 1 and 2, are more likely to be prone to change than other components.

11. Components can achieve few or no changes irrespective of ASRR value.

Results for both ASRR and SRR suggest that components with Requirements which have some recorded sources are more likely to see higher numbers of changes than components implementing Requirements that have *none*. SRR suggests an increasing trend, but when normalised to account for NR some exceptions appear. *ASRR=1* and *ASRR>2* which see low medians and interquartile ranges compared to their neighbours. We believe that placing components into groups with *either* an integer value or a non-integer value for ASRR serves to act as a surrogate for selecting components by number of requirements. We can deduce that components in non-integer groups must be linked to a mixture of Requirements with sources and Requirements without sources, in order to reach an average non-integer figure. The minimum number of requirements for components in non-integer groups is therefore 2, rather than 1, as it is for all other groups.

The boxplot in Figure 6.9 shows values of NR for components grouped by ASRR. The median NR is generally much lower in the non-integer categories than for the integer categories. This observation suggests that NR values may explain much of the variance in number of changes experienced by components when they are grouped by ASRR.

In Section 3.4.2 we hypothesised that links to rationale generally (and specifically, External Standards) may render components more vulnerable to changes, since the presence of external

Pairing	W	p (adj)	α	Reject H_0?
RI=0 and RI=1	26585.5	0.0422	0.05/6 = 0.008333	No

Table 6.7: Table showing Holm procedure for adjusting p values on multiple analyses for RI

sources of rationale may propagate change 'ripples' outside the control of the project team. Alterations to standards are not made lightly and are usually signalled well in advance so components affected by External Standards may be *more* stable than other components. Our results for all three metrics, however, appear to support the former hypothesis. *NES>0*, for example, does not contain a large number of components, but those in this group seem to have experienced more changes than 'normal'.

We might expect that the actual effects (stabilising or otherwise) of dependencies on External Standards are only detectable over a long period of time. To establish this we would need data from a project much further into its maintenance period (ideally a number of years) than our current case study. An alternative explanation is that developers are initially less familiar with a standard's fine details and increased changes during development result from iterative work to ensure that the system fully meets the standard's expectations.

6.5.5 RI

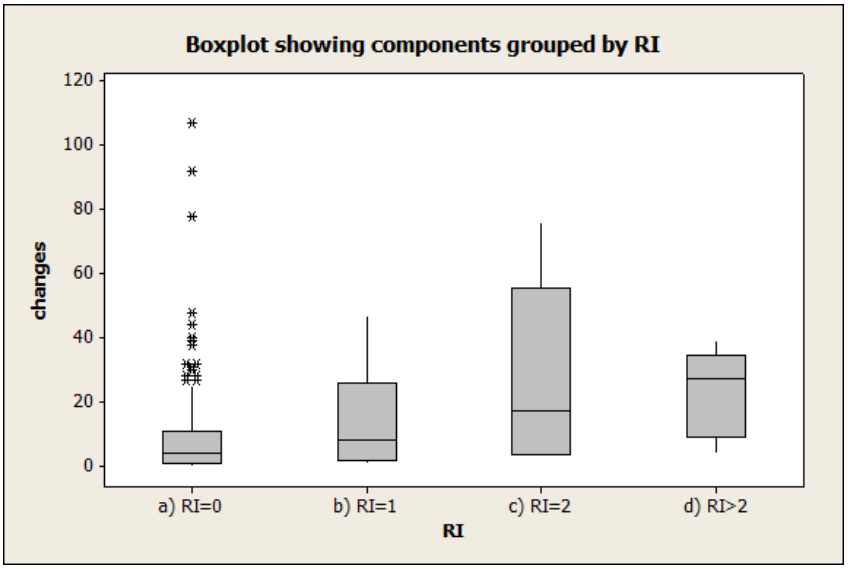

Figure 6.10: Boxplot showing the number of changes experienced by components categorised by values for RI

CARMEN does not employ large quantities of heavily nested Requirements and sub-Requirements. As a result the number of components with RI values greater than 0 is relatively small (225 out of a total 254 components have an RI value of 0). Despite the small number of components, categories with RI above zero have greater medians, upper limits and interquartile ranges (see Figure 6.10). There is a steady progression of increasing medians as RI increases (although the same pattern is not quite sustained for interquartile range and upper limit). However, there is a very large number of outliers in the category *RI=0*. Perhaps this is the reason that the multiple comparison tests do not uncover any significant differences between RI groups (see Table 6.7 for a summary).

Whilst the Kruskal-Wallis tests did not discover any significantly different categories, a simpler Mann-Whitney test on two categories (*RI=0* and *RI>0*) does find a significant difference, shown

Metric	*N*	median	*p* value (adj)	95% Confidence Interval	W
RI=0 and RI>0	225 29	4.0 12.0	0.0001	-15.00,-3.00	27216.0

Table 6.8: Results of Mann-Whitney tests on RI

Pairing	W	*p* (adj)	α	Reject H_0?
NOut=0 and NOut=1	5050.0	0.0000	0.05/15 = 0.003333	Yes
NOut=0 and NOut=2	2844.0	0.0000	0.05/14 = 0.003571	Yes
NOut=0 and NOut=4	2801.5	0.0000	0.05/13 = 0.003846	Yes
NOut=0 and NOut>4	3071.0	0.0000	0.05/12 = 0.004167	Yes
NOut=0 and NOut=3	2866.5	0.0002	0.05/11 = 0.004545	Yes
NOut=1 and NOut>4	6376.0	0.0003	0.05/10 = 0.005	Yes
NOut=1 and NOut=2	5847.5	0.0006	0.05/9 = 0.005556	Yes
NOut=1 and NOut=4	5663.0	0.0044	0.05/8 = 0.00625	Yes
NOut=1 and NOut=3	5834.0	0.0934	0.05/7 = 0.007143	No

Table 6.9: Table showing Holm procedure for adjusting p values on multiple analyses for NOut

in Table 6.8. Based on this we suggest the following guideline:

12. Components with a non-zero value of RI are more likely to experience a high number of changes than those with a zero value of RI.

In Section 3.4.2, we suggested that a positive correlation between RI and changes could imply that a Requirement with many sub-Requirements has greater ability to propagate change ripples. The Mann-Whitney test does supply some evidence that a Requirement with any number of sub-Requirements is associated with more changes than a Requirement no *no* sub-Requirements.

6.5.6 NOut

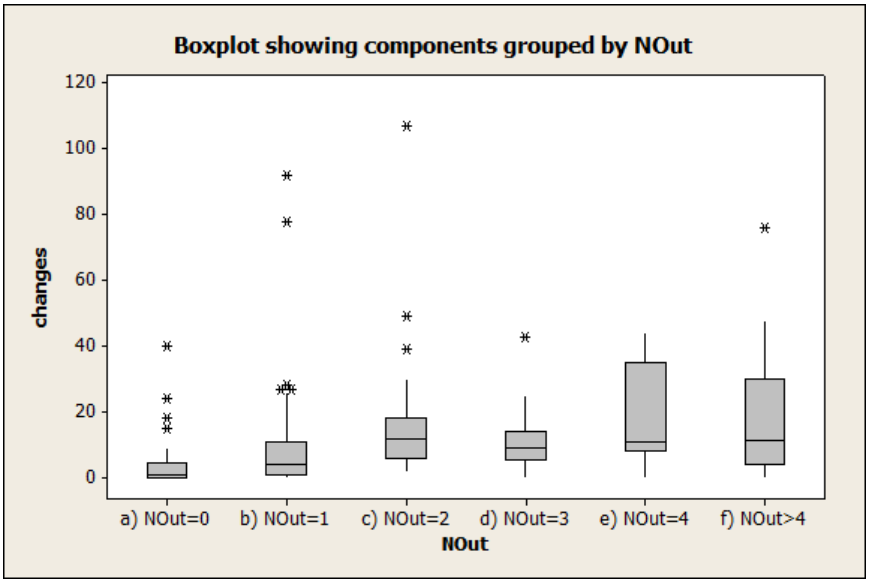

Figure 6.11: Boxplot showing the number of changes experienced by components categorised by values for NOut

In general, the boxplot showing the number of changes for categories in the Kruskal-Wallis test (Figure 6.11) suggests that a higher median number of changes is experienced by components

with a higher NOut value. In particular, the medians for the categories *NOut=0* and *NOut=1* are markedly lower than for other categories.

These observations are supported by the post- Kruskal-Wallis multiple comparisons tests, which finds significant differences between the category *NOut=0* and all other categories where NOut>1; and between *NOut=1* and three of the four categories with a higher value (see Table 6.9 for a summary). There thus seems to be a threshold of 2, beyond which the chances of experiencing a high number of changes increases:

13. Components with a NOut value of 2 or more are more likely to experience change than components with a NOut below 2.

14. Components may experience few changes irrespective of the value of NOut.

NOut is the most basic of our design-related metrics, representing represents the number of Decision Outcomes that can be linked to a component. Potentially it is a marker for indicating the quantity of design effort. In Section 3.4.3 we suggested that a greater focus on the design process (resulting in a higher value of NOut as well as other design metrics) helps to reduce the future number of changes due to the discovery of previously unforeseen design problems (in which case we would see a negative correlation between changes and NOut). Alternatively, a positive correlation between NOut and number of changes could indicate that the metric flags up areas where design was more 'difficult'. This may mean that a greater number of design constraints placed on the component leads to compromises and trade-offs that result in *increased* design-related changes.

There is a slight positive correlation visible between NOut and change-proneness, which suggests that the second explanation could be true. The same conclusion can be applied to many other of our design metrics (which are all discussed below).

6.5.7 SOR and ASOR

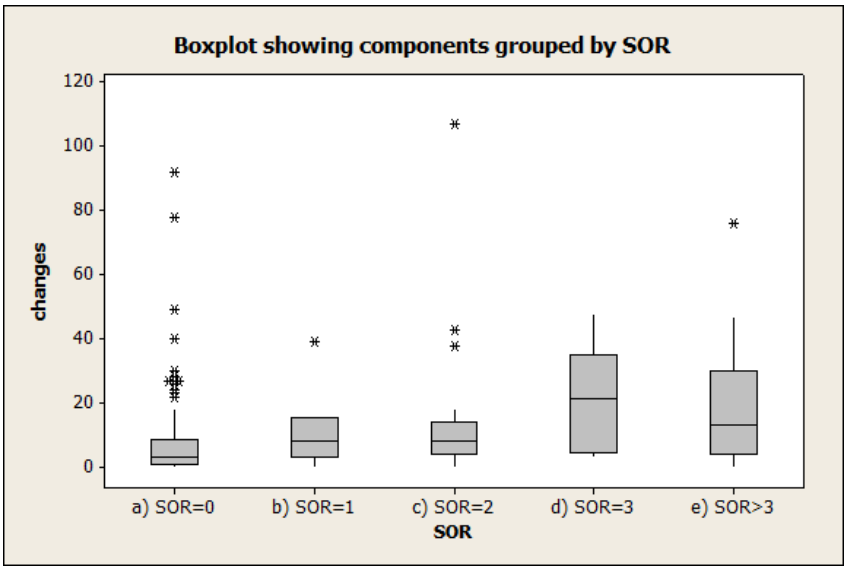

Figure 6.12: Boxplot showing the number of changes experienced by components categorised by values for SOR

Figure 6.12 appears to show a sudden increase in the number of changes as SOR climbs above 3. Both the median and potential range of changes increase substantially at this point. Multiple comparison tests (see Table 6.10 for a summary) support this observation, detecting significant differences between the lowest category (*SOR=0* and the two highest (*SOR=3* and *SOR>3*).

Pairing	W	p (adj)	α	Reject H_0?
SOR				
SOR=0 and SOR=3	16811.0	0.0000	0.05/10 = 0.005	Yes
SOR=0 and SOR>3	17311.5	0.0001	0.05/9 = 0.005556	Yes
SOR=0 and SOR=2	17990.5	0.0115	0.05/8 = 0.00625	No
ASOR				
ASOR=0 and 0<ASOR<=0.5	16953.0	0.0000	0.05/10 = 0.005	Yes
ASOR=0 and 0.5<ASOR<=1.0	18149.5	0.0000	0.05/9 = 0.005556	Yes
ASOR=0 and 1<ASOR<=1.5	16496.0	0.0380	0.05/8 = 0.00625	No

Table 6.10: Table showing Holm procedure for adjusting p values on multiple analyses for SOR and ASOR

Our observations are:

15. Components with values of SOR of 3 and above are more likely to experience a high number of changes than components with SOR of 0.

16. Components may experience few or no changes irrespective of the value of SOR.

SOR is a metric which is somewhat liable to become skewed: components which are linked to a higher number of Outcomes are likely to be linked, in turn, to more items of rationale for those outcomes. ASOR is designed to present a normalised value by calculating the average amount of rationale per Outcome.

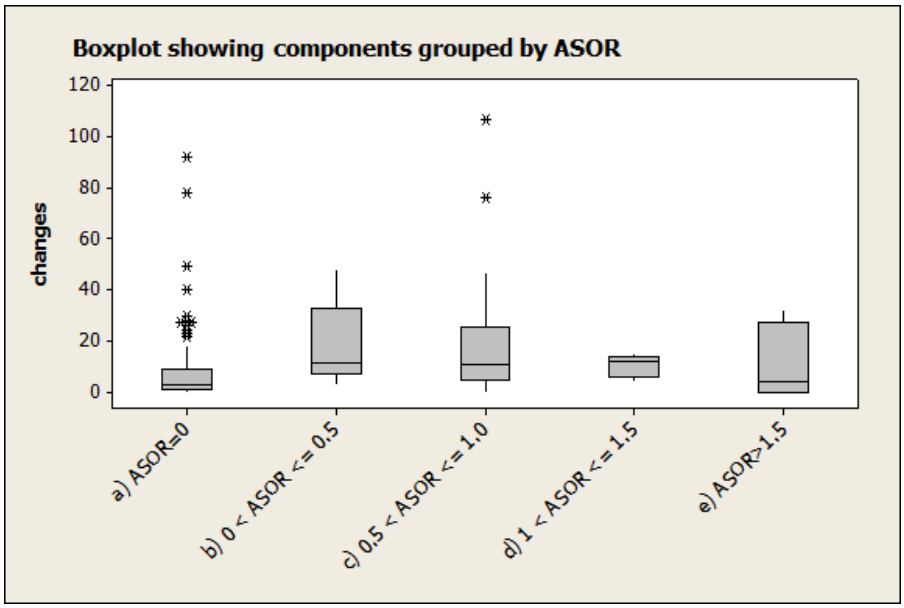

Figure 6.13: Boxplot showing the number of changes experienced by components categorised by values for ASOR

Figure 6.13, showing the Kruskal-Wallis categories for ASOR, reveals a slightly different picture than for SOR. Like SOR, the lowest value category (*ASOR=0*) shows a low median and short range, but many outliers. However, rather than an increase in the number of changes for larger values of ASOR, we see an immediate increase in the median (in *0<ASOR<=0.5*) and then a reduction in the median value as we move towards the largest-value category *ASOR>1.5*. The median number of changes for ASOR categories doesn't vary much, however, and multiple comparison tests (summarised in Table 6.10) only reveal two significantly different groups: *ASOR=0* differs significantly to the next two lowest-value categories (*0<ASOR<=0.5* and *0.5<ASOR<=1.0*).

With the exception of the first category ($ASOR{=}0$), the overall relationship between ASOR and change-proneness appears to follow a slight negative trend (looking at the median no of changes). The largest-value category for ASOR bucks the trend slightly, producing a wide interquartile range, although the median is still very low and follows the generally negative trend. This unexpectedly wide range, however could suggest that our observations do not necessarily scale upwards towards very high values:

17. Components with ASOR of 0 are less likely to experience high changes than components with ASOR between 0 and 1.

18. Components can experience few or no changes irrespective of ASOR.

In Section 3.4.3 we suggested that a positive correlation between the number of items of rationale per Outcome and change-proneness could mean that the presence of recorded sources for Outcomes could make the related components more susceptible to propagated changes from those sources. We also suggested that a negative correlation with change-proneness could imply that the presence of these sources actually has a stabilising effect. Our results suggest a complex situation. Components with a low, non-zero value for ASOR (i.e., a very small number of the Outcomes related to a component have recorded sources) has an increased likelihood of higher changes. However, a higher number of recorded sources appears to have more of a stabilising role. This could be because effecting changes in components with many design sources recorded requires the balancing of those sources, and this deters users and programmers from embarking on unnecessary changes. This deterrent is less likely to exist where only a few sources have been identified.

6.5.8 NDP, NOpt and ANODP

We consider these metrics together as they all address the related measures of Decision Problems and/or Options. The boxplot illustrating the categories used for the Kruskal-Wallis tests (see

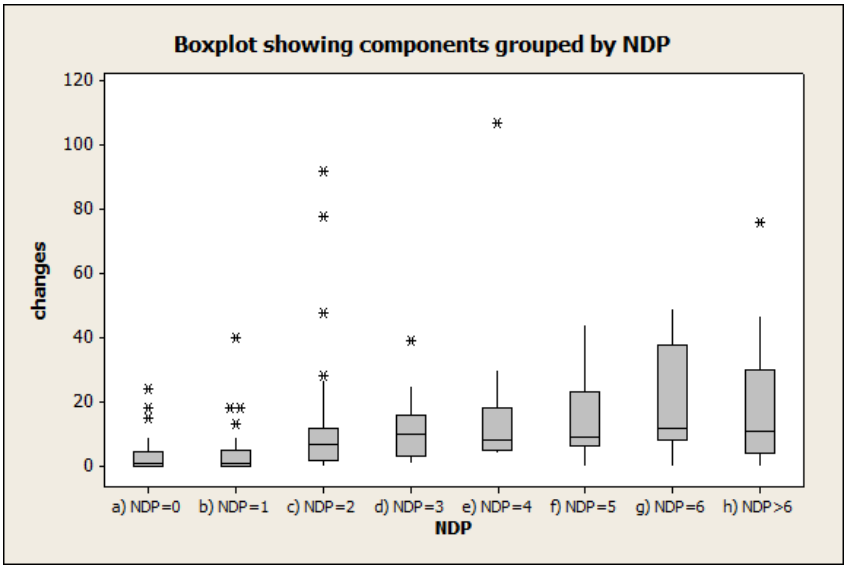

Figure 6.14: Boxplot showing the number of changes experienced by components categorised by values for NDP

Figure 6.14) shows a long-tailed curve in the median number of changes. Lower values of NDP (where NDP is below 2) have a low median number of changes and a very short range of potential number of changes. However, once NDP reaches 2 there is a noticeable increase in both median

number of changes and the range of potential values (including outliers). The medians creep upwards gradually until we reach *NDP=6*, where we see a small peak which drops slightly for the highest values of NDP. Medians, upper limits (excluding outliers) and interquartile ranges are highest at *NDP=6*.

Pairing	**W**	***p* (adj)**	α	**Reject H_0?**
NDP				
NDP=0 & NDP=2	3437.0	0.0000	0.05/28 = 0.001786	Yes
NDP=0 & NDP=3	2315.5	0.0000	0.05/27 = 0.001852	Yes
NDP=0 & NDP=4	2216.0	0.0000	0.05/26 = 0.001923	Yes
NDP=0 & NDP=6	2260.5	0.0000	0.05/25 = 0.002	Yes
NDP=0 & NDP>6	2407.5	0.0000	0.05/24 = 0.002083	Yes
NDP=0 & NDP=5	2273.5	0.0001	0.05/23 = 0.002174	Yes
NDP=1 & NDP>6	584.5	0.0001	0.05/22 = 0.002273	Yes
NDP=1 & NDP=6	509.0	0.0004	0.05/21 = 0.002381	Yes
NDP=1 & NDP=4	485.5	0.0009	0.05/20 = 0.0025	Yes
NDP=1 & NDP=2	1112.5	0.0011	0.05/19 = 0.002632	Yes
NDP=1 & NDP=3	535.5	0.0017	0.05/18 = 0.002778	Yes
NDP=1 & NDP=5	514.0	0.0027	0.05/17 = 0.002941	Yes
NDP=2 & NDP=6	3512.5	0.0132	0.05/16 = 0.003125	No
NOpt				
NOpt=0 & 7<=NOpt<9	4442.0	0.0000	0.05/15 = 0.003333	Yes
NOpt=0 & 1<=NOpt<3	6891.5	0.0000	0.05/14 = 0.003571	Yes
NOpt=0 & 3<=NOpt<5	4451.0	0.0001	0.05/13 = 0.003846	Yes
NOpt=0 & NOpt>=9	4471.5	0.0001	0.05/12 = 0.004167	Yes
NOpt=0 & 5<=NOpt<7	4316.5	0.0303	0.05/11 = 0.004545	No
ANODP				
ANODP=0 & 0<ANODP<1	5039.0	0.0000	0.05/10 = 0.005	Yes
ANODP=0 & ANODP=1	6063.5	0.0000	0.05/9 = 0.005556	Yes
ANODP=0 & 1<ANODP<2	4692.5	0.0000	0.05/8 = 0.00625	Yes
0<ANODP<1 & ANODP=1	3403.5	0.0096	0.05/7 = 0.007143	No

Table 6.11: Table showing Holm procedure for adjusting p values on multiple analyses for NDP, ANODP and NOpt

The multiple comparison tests (see Table 6.11 for a summary) support only the observation that there is a change between *NDP=1* and *NDP=2*, however. This change is a significant threshold: the categories *NDP=0* and *NDP=1* differ significantly from all categories where NDP is 2 or above. Based on this, we make the following observations:

19. Components where NDP is 2 or higher are more likely to experience higher numbers of changes than components with NDP below 2.

20. Components may experience no or few changes irrespective of the value of NDP.

A boxplot showing the categories used in the Kruskal-Wallis test for NOpt (Figure 6.15) shows a climb in the upper limit, median number of changes, and the interquartile range, between *NOpt=0* and the next category, *1<=NOpt<3*. After this climb, the median remains relatively stable, with just very small increments as the value of NOpt climbs. The interquartile range also increases a little (although not without decreases too).

The findings of the multiple comparison tests are summarised in Table 6.11. *NOpt=0* differs significantly from every other category, with the exception of *5<=NOpt<7*. In this case, the Kruskal-Wallis does not really offer us much advantage over the simpler Mann-Whitney test, which compared a zero and non-zero group (see Table 6.12), and so we use results from this instead.

Our observations:

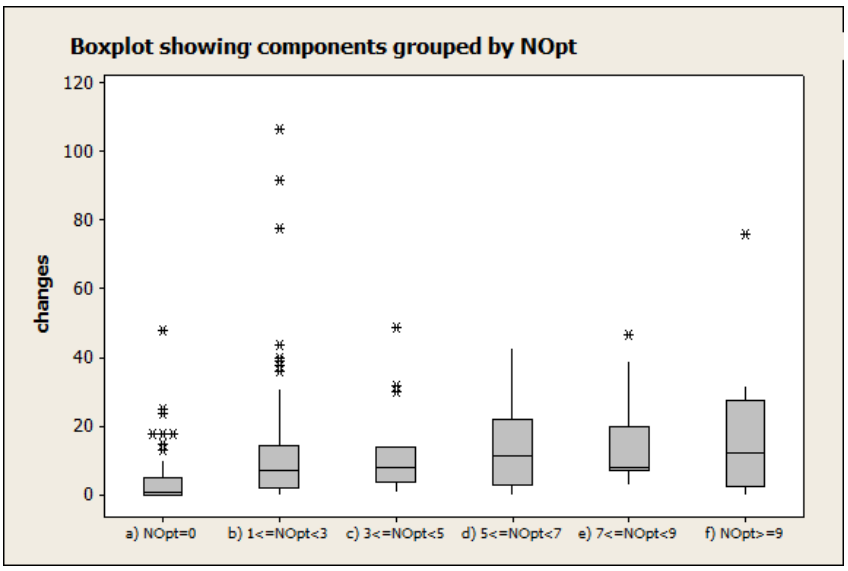

Figure 6.15: Boxplot showing the number of changes experienced by components categorised by values for NOpt

Metric	***N***	**median**	***p* value (adj)**	**95% Confidence Interval**	**W**
NOpt=0 and NOpt>0	91 163	1.0 8.0	0.0000	-6.999,-3.001	7828.5

Table 6.12: Results of Mann-Whitney tests on NOpt

21. Components with a NOpt above zero are more likely to experience higher numbers of changes than other groups.

22. Components may experience few or no changes irrespective of NOpt value.

NOpt is a metric that may be skewed by the number of Decision Problems associated with a component, since components with higher numbers of Decision Problems are naturally more likely to be associated with a higher number of Options. ANODP is designed to take this into account by normalising the figure and producing an average per Decision Problem.

The situation for ANODP (a boxplot is shown in Figure 6.16) is almost the inverse of the situation for NOpt. Like NOpt, the zero category *ANODP=0* has a low median, upper limit and interquartile range. However, the next category shows a large increase in all three. This increase is not sustained and falls slightly for larger values of ANODP. Multiple comparison testing (see Table 6.11 for a summary) finds that *ANODP=0* differs significantly from all other categories except the highest-value (*ANODP>=2*).

23. Components with a value between 0 and 2 for ANODP are more likely to see higher numbers of changes than components with ANODP of 0 or with ANODP over 2.

24. Components can experience few or no changes regardless of ANODP.

We hypothesised in Section 3.4.3 that the number of Options linked to Decision Problems relevant to a component c provides an indication of effort spent on researching and weighing up design decisions relevant to c. Therefore components with higher values for NOpt would be linked to the most 'difficult' decisions; this could be (we had suggested) because the area is risky, poorly-understood, particularly important, controversial, incapable of rendering a perfect solution or simply a field with a very wide choice of solutions available.

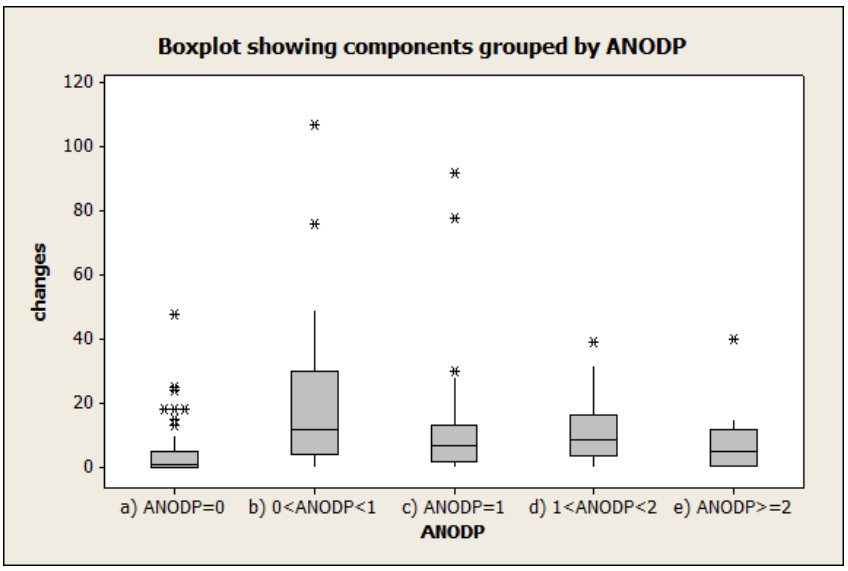

Figure 6.16: Boxplot showing the number of changes experienced by components categorised by values for ANODP

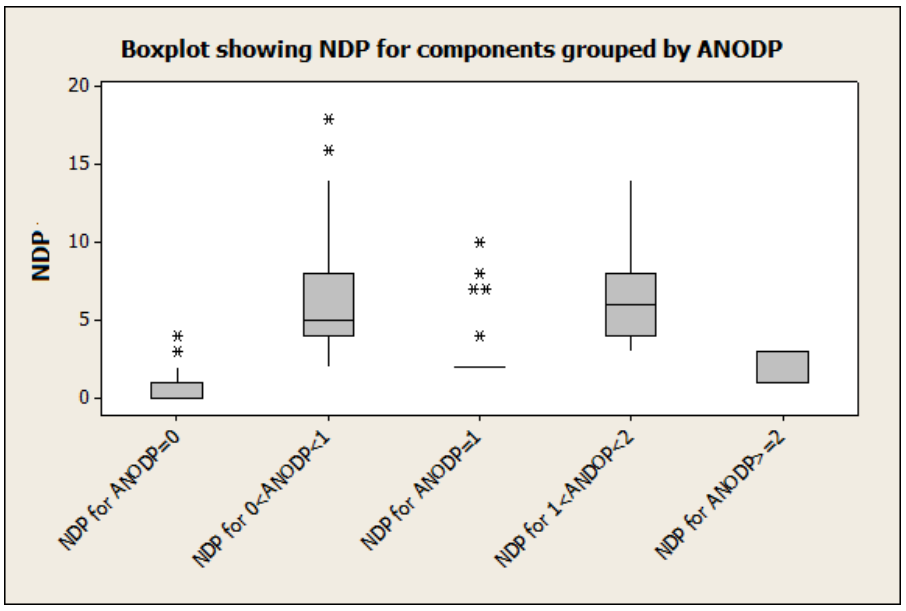

Figure 6.17: Boxplot showing NDP for components grouped by ANODP

The actual results for NOpt and for NDP suggest that higher numbers of Decision Problems or Options are associated with higher levels of change. This result supports our hypothesis that a higher value for NOpt or NDP flags up areas which are 'difficult' to design or result in compromises (we also came to this conclusion for NOut). Normalising NOpt for the number of Decision Problems reduces this connection, however: components with the most Options per Decision Problem seem to experience *fewer* changes as ANODP increases. Components with a very low but non-zero average number of Options are most likely to be prone to change.

Components falling into the group *0<ANODP<1* have non-integer value for ANODP. This could be because they are naturally a group with a higher number of Decision Problems (in order to achieve a non-integer result). If this is true, it would act as a confounding factor that helps to explain our results. The same argument (that non-integer groups contain components with higher NDP) should also apply for components in the group *1<ANODP<2*, but this group is

smaller, (with 26 components as opposed to 46 in *0<ANODP<1*) and this may account for a shorter interquartile range. We compare ANODP to NDP, generating a boxplot (Figure 6.17) to examine the values of NDP for components in their ANODP groups. The correspondence between the NDP for categories of ANODP here and the number of changes (in Fig. 6.16) is not evident, however. The category *ANODP=0* has a lower median and shorter interquartile range than other categories in both plots. And the two mid-range non-integer categories *0<ANODP<1* and *1<ANODP<2* do have noticeably higher values for NDP. The non-integer categories are both associated with higher higher NDP, so why is *1<ANODP<2* not displaying the same increase in changes as *0<ANODP<1*? We suggest that, as ANODP rises, irrespective of NDP, a stabilising effect is visible. Where there are a few Options associated with a Decision Problem are associated with change-proneness. However, once ANODP rises above an average of 1 option per Decision Problem, the propensity to change declines. This implies the opposite of the conclusion reached for NOpt. This tends to confirm our alternate hypothesis: Decision Problems with more Options have been researched and discussed more thoroughly, and perhaps are more likely to result in a better match with project and user requirements.

6.5.9 SOS and ASOS, SOO and ASOO

We consider these four metrics together, since they address similar attributes. SOS and SOO are metrics which may be skewed by the number of options associated with a component, since components with higher numbers of options are naturally more likely to be associated with a higher number of supporters/opposers to those options. ASOS and ASOO are designed to take this into account by producing an average figure of SOS or SOO per option.

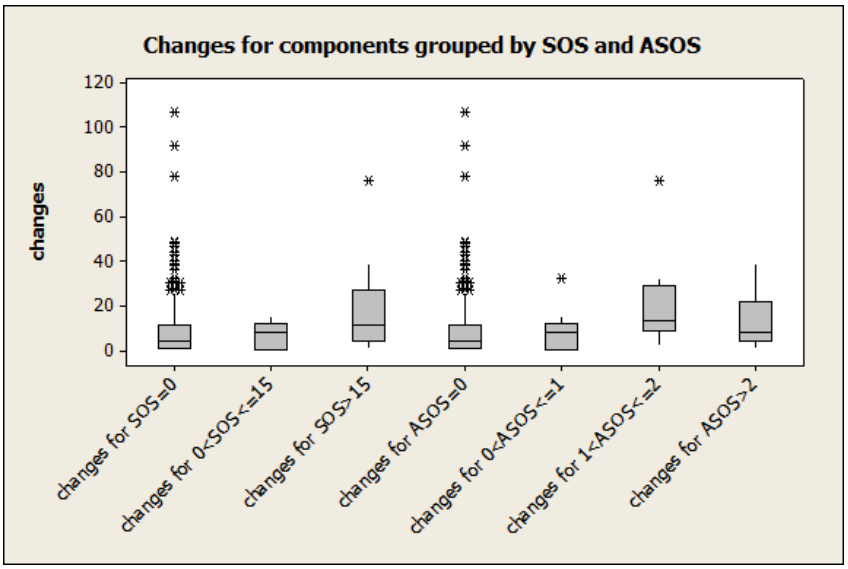

Figure 6.18: Boxplot showing the number of changes experienced by components categorised by values for SOS and ASOS

Figure 6.18 shows a gradual increase in median number of changes, and in the interquartile ranges, as SOS increases. The lowest-value category, however (*SOS=0*) contains almost all most volatile components (as outliers), which dilutes the effect of the gradual increase we otherwise see. As a result the multiple comparison tests (see Table 6.13 for a summary) only detect one significant difference - between the lowest-value category *SOS=0* and the the highest (*SOS>15*).

25. Components with a high value of SOS (above 15) are more likely to experience a higher number of changes than components with SOS of 0.

Pairing	W	p (adj)	α	Reject H_0?
SOS				
SOS=0 and SOS>15	24630.0	0.0002	0.05/3 = 0.016667	Yes
0<SOS<=15 and SOS>15	191.5	0.0309	0.05/2 = 0.025	No
ASOS				
ASOS=0 and 1<ASOS<=2	23502.0	0.0041	0.05/6 = 0.008333	Yes
ASOS=0 and ASOS>2	24120.5	0.0197	0.05/5 = 0.01	No
SOO				
SOO=0 and SOO>11	23517.5	0.0003	0.05/6 = 0.008333	Yes
SOO=0 and 1<=SOO<6	23477.5	0.0048	0.05/5 = 0.01	Yes
SOO=0 and 6<=SOO<11	23161.0	0.0474	0.05/4 = 0.0125	No
ASOO				
ASOO=0 and ASOO>1	24087.5	0.0004	0.05/3 = 0.016667	Yes
ASOO=0 and 0<ASOO<=1	23490.5	0.0007	0.05/2 = 0.025	Yes
0<ASOO<=1 and ASOO>1	402.0	0.3548	0.05/1 = 0.05	No

Table 6.13: Table showing Holm procedure for adjusting p values on multiple analyses for SOS, ASOS, SOO and ASOO

26. Components can experience few or no changes irrespective of the value of SOS.

Normalising the figure for the number of options removes much of this correlation, however. The multiple comparison tests for ASOS also do not find large numbers of significant differences. The boxplot in Figure 6.18 shows that, as with the SOS metric, the lowest value category *ASOS=0* has a low median number of changes, but all of the volatile outliers. There is a gradual increase in the median number of changes and interquartile range until the final, highest-value category, *ASOS>2*, when the median and the range drop slightly. The second highest category, *1<ASOS<=2*, is unusual in that no components experienced no changes.

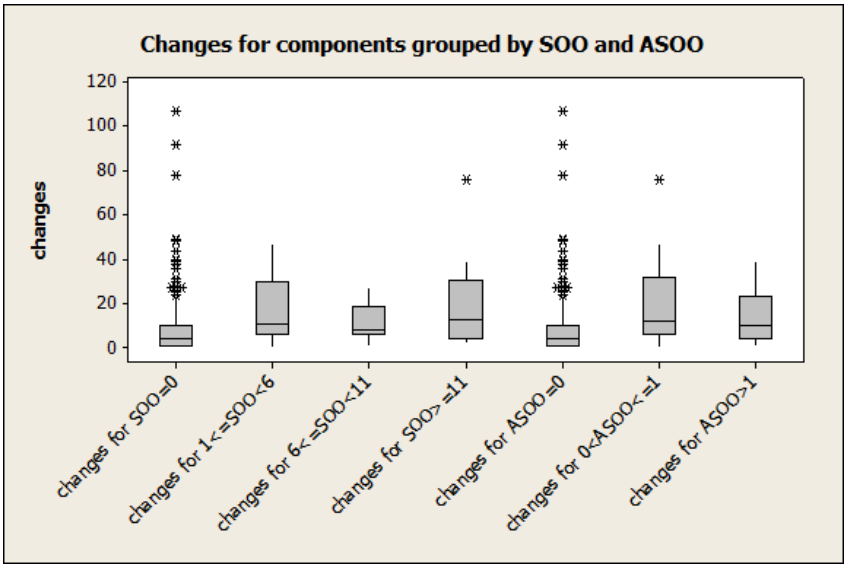

Figure 6.19: Boxplot showing the number of changes experienced by components categorised by values for SOO and ASOO

The multiple comparison test results in Table 6.13 detect only one significant difference between these groups, between the lowest-value category (*ASOS=0*) and the second highest (*1<ASOS<=2*). It's much simpler to get a general idea of which components will have a zero or non-zero figure for a metric, so we also executed a Mann-Whitney test on zero and non-zero groups for this metric (results in 6.14). The Mann-Whitney test, as well as being simpler, also has a slightly better

Metric	*N*	median	*p* value (adj)	95% Confidence Interval	W
ASOS=0 and ASOS>0	214 40	4.0 9.5	0.0000	-7.000,-1.003	26023.5
SOO=0 and SOO>0	212 42	4.0 9.5	0.0000	-8.999,-3.002	25000.0
ASOO=0 and ASOO>0	212 42	4.0 11.0	0.0000	-8.999,-3.002	25000.0

Table 6.14: Results of Mann-Whitney tests on ASOS, SOO and ASOO

significance level (0.003) than the Kruskal-Wallis test revealed, so we base our observations on this.

27. Components with an ASOS more than zero have a greater likelihood of experiencing a higher number of changes than components with zero.

28. Components may experience few or no changes regardless of values for ASOS.

29. Components with ASOS of 0 may experience high numbers of changes.

A boxplot showing numbers of changes for the SOO categories in the Kruskal-Wallis test is shown in Figure 6.19. There is no clear pattern in the graph, except that the median number of changes, and the interquartile range, shift upwards between *SOO=0* and *1<=SOO<6*. However, although the median and interquartile range are lower for *SOO=0*, all the outliers with higher numbers of changes except one fall into this category. In fact, multiple comparison testing (see Table 6.13) only finds two significant differences: between *SOO=0* and the two categories with slightly higher medians.

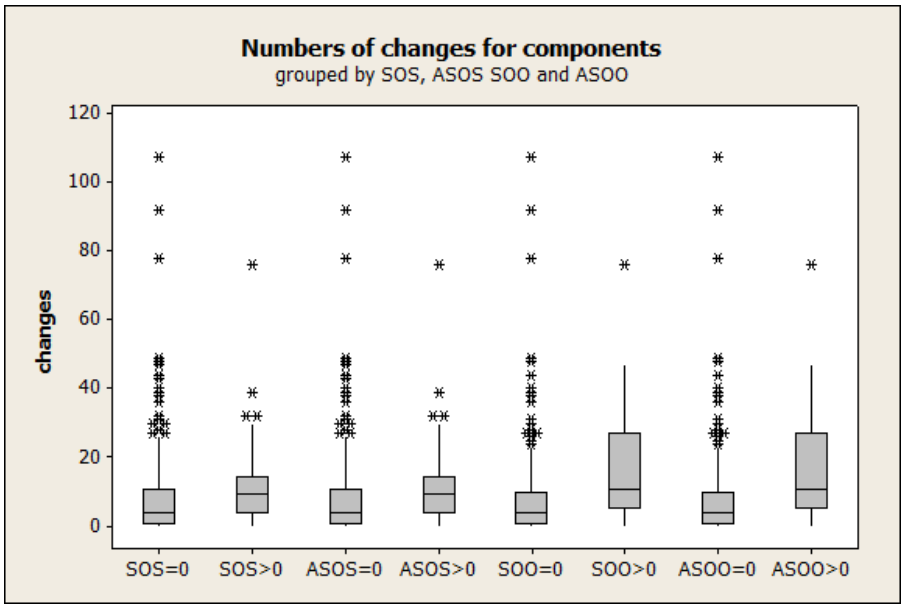

Figure 6.20: Boxplot showing the number of changes experienced by components categorised by values for SOS, ASOS, SOO and ASOO

The Mann-Whitney test detected a difference between zero and non-zero groups of components (see Table 6.14). This is a more significant difference than those detected during the Kruskal-Wallis tests, as well as a simpler test to use, so we use the Mann-Whitney categories instead:

30. Components with a non-zero SOO are more likely to experience large numbers of changes than components with SOO of zero.

Metric	*N*	median	*p* value (adj)	95% Confidence Interval	W
NA=0 and NA>0	240 14	4.0 27.5	0.0000	-26.00,-9.00	29346.5

Table 6.15: Results of Mann-Whitney tests on NA

31. Components may experience few or no changes regardless of the value of SOO.

32. Components with SOO of zero may experience very high numbers of changes.

Figure 6.19 shows us that the lowest-value group (in this case, *ASOO=0*) has a lower median, upper limit and interquartile range than other groups. However, it also contains all highly volatile outliers bar one (as is the case with SOO). The multiple comparison tests confirm that *ASOO=0* is found to differ significantly to both other groups, but no other differences are found. A Mann-Whitney test found that zero and non-zero groups for ASOO differed with a significance of 0.000 (see Table 6.14 and Figure 6.20 for a Mann-Whitney boxplot), so once again we suggest using these groups, meaning that our conclusion does not change when we normalise SOO for the number of options.

33. Components with an ASOO of zero are less likely to experience large numbers of changes than components with a higher value of ASOO.

34. Components with an ASOO of zero are capable of achieving high numbers of changes.

35. Components may experience few or no changes regardless of ASOO.

Also in Section 3.4.3, we suggested that a generally positive correlation between ASOS/SOS/SOO/ASOO and the number of changes, could mean that the greater concentration of design effort represented by these metrics is indicative of design difficulties or disagreements, and that perhaps ideal solutions could not be reached. As with the result for NDP, NOpt and NOut, we find a generally positive correlation between the number of changes and SOS, ASOS, SOO or ASOO. Despite this we note that the *most* volatile components all sit outside the very highest value groups of SOS and ASOS (they are visible as outliers on boxplots). These outliers imply that the most volatile components of all have *low* amounts of argumentation recorded for their decisions. Two general principles may be at work: for the most part, a higher amount of design activity (which we can assess using NDP, NOpt, NOut, SOO, ASOO, SOS and ASOS) seems to flag up difficult areas and result in more changes. However, some components with less design argumentation have the *most* changes. Less design argumentation implies less time spent on exploring the design space. Perhaps the changes on CARMEN which result from design difficulties (such as, for example, unforeseen incompatibilities) are less in quantity than the types of changes which result from less design work in total. The small number of *most* volatile outliers (visible as asterisks on boxplots) can - with a few exceptions - also be found outside the highest categories for ANODP, NOpt, NDP, ASOR, SOR and NOut.

6.5.10 NA

Figure 6.21 indicates a substantial increase in the interquartile range and the upper limit for the highest-value category, *NA>1*. However, all of the highly volatile outliers lie within the lower-value categories (in the form of outliers). As a result, the only significant difference detected by multiple comparison testing is between *NA=0* and *NA=1*. A Mann-Whitney test detected a difference

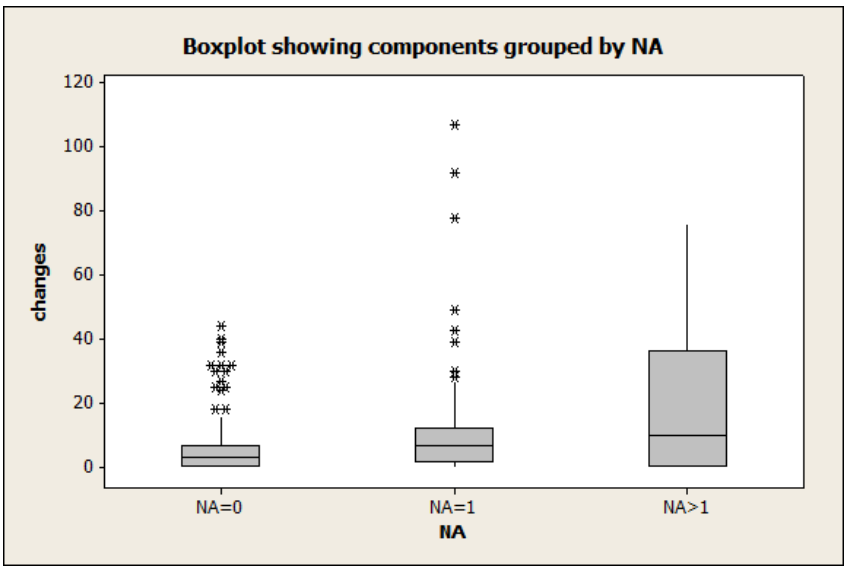

Figure 6.21: Boxplot showing the number of changes experienced by components categorised by values for NA

Pairing	**W**	***p* (adj)**	α	**Reject H_0?**
NA=0 and NA=1	10592.0	0.0005	0.05/3 = 0.016667	Yes
NA=0 and NA>1	6053.5	0.0635	0.05/2 = 0.025	No

Table 6.16: Table showing Holm procedure for adjusting p values on multiple analyses for NA

between groups with zero and non-zero values for NA with a significance of 0.000 (see 6.15), and so we suggest that this is a more helpful and simple guideline to follow:

36. Components with a non-zero value of NA are more likely to experience higher numbers of changes.

37. Components may experience no or few changes irrespective of NA value.

In Section 3.4.3 we suggested that we expected a positive correlation between NA and change-proneness. We hypothesised that the reliance on Assumptions would leave components vulnerable to changes if the Assumptions turned out to be inaccurate or unreliable. This appears to be true, based on the Mann-Whitney test.

6.5.11 Summary of observations

Table 6.17 presents a summary of our initial findings on our metrics. Discussions of correlation with LOC are presented in Section 6.7.

Table 6.17: Table summarising findings for our metrics

Metric	**No.**	**Observation**	**Correlated with LOC?**
NCR		Insufficient data.	
NSG		No evidence of a relationship between NSG and changes.	
NES	1.	Components with a non-zero value for NES will experience changes.	We do not have enough data to draw firm

Continued on next page

Table 6.17 – continued from previous page

Metric	No.	Observation	Correlated with LOC?
	2.	Components with a non-zero value for NES are more likely to experience high numbers of changes than other components.	conclusions.
	3.	Components can still achieve high numbers of changes even if NES is 0.	
NR	4.	Components with a value of NR of 3 or greater are more likely to experience changes than other components.	Yes - components with high value for NR have very similar
	5.	Components can experience few or no changes irrespective of the value for NR.	changes to those with high LOC.
SRR	6.	Components with an SRR of 2 are more likely those with 0 or 1 to experience higher numbers of changes.	Yes - components with the highest SRR and those
	7.	Components with an SRR more than 2 are still more likely to see higher numbers of changes.	with highest LOC experience similar changes.
	8.	Components are capable of experiencing few or no changes irrespective of SRR.	
ASRR	9.	Components with ASRR or 0 or 1 are less likely to experience more changes than other components.	Yes. Although ASRR-high & LOC-high are different,
	10.	Components with ASRR between 0 and 1, or between 1 and 2, are more likely to experience more changes than other components.	following our ASRR leads to selection of the largest components.
	11.	Components can achieve few or no changes irrespective of ASRR value.	
RI	12.	Components with a non-zero value of RI are more likely to experience a high number of changes than those with a 0 value of RI.	Yes - components with high RI and those with high LOC shown to be similar.
NOut	13.	Components with a NOut value of 2 or more are more likely to experience change than components with a NOut below 2.	Weakly - components with high NOut and those with high LOC are
	14.	Components may experience few changes irrespective of the value of NOut.	not very similar, but NOut categories with high changes also have high LOC.
SOR	15.	Components with values of SOR of 3 and above are more likely to experience a high number of changes than components with SOR of 0.	Yes.
	16.	Components may experience few or no changes	

Continued on next page

Table 6.17 – continued from previous page

Metric	No.	Observation	Correlated with LOC?
		irrespective of the value of SOR.	
ASOR	17.	Components with ASOR of 0 are less likely to experience high changes than components with ASOR between zero and 1.	No.
	18.	Components can experience few or no changes irrespective of ASOR.	
NDP	19.	Components where NDP is 2 or higher are more likely to experience higher numbers of changes than components with NDP below 2.	Weakly.
	20.	Components may experience no or few numbers of changes irrespective of the value of NDP.	
NOpt	21.	Components with a NOpt above 0 are more likely to experience higher numbers of changes than other groups.	Weakly.
	22.	Components may experience few or no changes irrespective of NOpt value.	
ANODP	23.	Components with a value between 0 and 2 for ANODP are more likely to see higher numbers of changes than components with ANODP of 0 or with ANODP of 2 and more.	Weakly.
	24.	Components can experience few or no changes regardless of ANODP.	
SOS	25.	Components with a high value of SOS (above 15) are more likely to experience a higher number of changes than components with SOS of 0.	Weakly.
	26.	Components can experience few or no changes irrespective of the value of SOS.	
ASOS	27.	Components with an ASOS more than 0 have a greater likelihood of experiencing a higher number of changes than components with ASOS 0.	No - components with high ASOS and those with high LOC shown to be different.
	28.	Components may experience few or no changes regardless of values for ASOS.	
	29.	Components with ASOS of 0 may experience high numbers of changes.	
SOO	30.	Components with a non-zero SOO are more likely to experience large numbers of changes than components with SOO of 0.	Weakly.
	31.	Components may experience few or no changes regardless of the value of SOO.	
	32.	Components with SOO of 0 may experience	

Continued on next page

Table 6.17 – continued from previous page

Metric	No.	Observation	Correlated with LOC?
		very high numbers of changes.	
ASOO	33.	Components with an ASOO of 0 are less likely to experience large numbers of changes than components with a higher value of ASOO.	No - components with high ASOO and those with high LOC shown to be different.
	34.	Components with an ASOO of 0 are capable of achieving high numbers of changes.	
	35.	Components may experience few or no changes regardless of ASOO.	
NA	36.	Components with a non-zero value of NA are more likely to experience higher numbers of changes.	High LOC may explain some variation in changes.
	37.	Components may experience no or few changes irrespective of NA value.	

6.6 Validation

6.7 Discussion

Some of our metrics have proven too coarse-grained or sparsely-populated to be useful. Analysis of NSG, for example, fails to provide any evidence of a link between changes and soft goals. This may be because one does not exist, or because the data from our case study project results in a small number of potential values for NSG.

NES and NCR also provide us with small quantities of data. In the case of NCR the resultant groups are too small to analyse.

We discuss factors relating to the results of our other metrics below.

6.7.1 Are our metrics simply surrogates for size?

We consider here whether the variation in change-proneness associated with our metrics can be explained - at least partially - by size. Size metrics (such as LOC) have been linked to increased changes in previous work (for example, [94]). Purely as a function of size, a larger component may be more likely to:

- implement a higher number of Requirements (assessed through NR)
- implement Requirements which are more important or more thoroughly justified in documentation (RI, SRR and ASRR)
- be affected by a greater number of Decision Outcomes (NOut), or decisions that are more thoroughly documented (SOR and ASOR)
- be linked to a higher number of Decision Problems (NDP)
- be linked to more complex or controversial decisions, or decisions which have a wider range of Options (NOpt and ANODP) or are more thoroughly documented (SOS, ASOS, SOO and ASOO)

Metric	p value	R-Sq (adj)%	Residuals normally distributed
NR & LOC	0.000	44.8%	No
NR & CBO	0.000	14.9%	No
NOut & LOC	0.000	33.0%	No
NOpt & LOC	0.000	23.1%	No
ANODP & LOC	0.009	2.6%	No
NDP & LOC	0.000	37.9%	No
SOR & LOC	0.000	27.0%	No
ASOR & LOC	0.000	13.2%	No
SOS & LOC	0.000	16.4%	No
ASOS & LOC	0.000	13.9%	No
SOO & LOC	0.000	20.1%	No
ASOO & LOC	0.000	19.7%	No
RI & LOC	0.000	19.6%	No
SRR & LOC	0.000	28.1%	No
ASRR & LOC	0.008	2.7%	No

Table 6.18: Table summarising results of regression tests of NR with CBO or LOC

We use LOC (calculated by CCCC) as to assess component size. We have LOC data (averaged over a series of snapshots) for 223 of the total number of 254 components (some components did not appear in any of our snapshots; see Chapter 4 for details). For this reason, comparisons between our metrics and LOC cover only those 223 components.

To answer the question: 'are our metrics simply surrogates for size?' we first search briefly for a linear relationship using linear regression; a summary of regression results is shown in Table 6.18. None of our metrics show a robust linear correlation with LOC, however, since none of the tests produce normally-distributed residuals. This result means that none of our metrics are a *direct* substitute for LOC. Scatter-plots showing the relationships between each of the metrics listed above and LOC are shown Appendix H.

We also executed another set of Mann-Whitney tests. For each metric m, we used the observational rules we developed for the metric (listed in Section 6.5) to 'predict' which components were more likely to be volatile. So, for example, for the metric NR, we selected all components with NR of 3 or more because we had observed earlier that these components were more likely to experience higher numbers of changes. This group is named the m-high group. We did this also for LOC (selecting all components with LOC of 300 or more, as observed in Section 5.6). The group of components predicted by LOC is relatively small (29 components) whilst some other metrics make predictions for much larger groups (the average size of prediction for for all the metrics in these Mann-Whitney tests is 78 components). However, groups do not have to be equally-sized for the Mann-Whitney test to be effective.

We then executed Mann-Whitney tests to compare the two groups m-high and LOC-high. This allows us to see whether there is a significant difference in the number of changes between the components 'predicted' as volatile by LOC, and those predicted by m. If there is very little difference, this could suggest that our metric is (at least partially) a surrogate for size, in that it selects the same components (or ones with similar values) as LOC. Results for these tests are shown in Table 6.19. We do not consider in this chapter whether our metrics or LOC perform *better* as predictors; we address this issue in Chapter 7.

Metric	n	median	p value (adj)	95% Confidence Interval	W
LOC-high and	29	31.0	0.0010	(4.00,21.00)	2957.0
NR-high	125	10.0			
SRR-high and	138	10.0	0.0014	(4.00,21.00)	3192.5
LOC-high	29	31.0			
ASRR-high and	103	11.0	0.0035	(3.00,21.00)	2459.5
LOC-high	29	31.0			
RI-high and	26	19.0	0.3761	(-4.00,17.00)	865.0
LOC-high	29	31.0			
LOC-high and	29	31.0	0.0587	(0.00,18.00)	1731.0
NOut-high	72	13.0			
LOC-high and	29	31.0	0.1843	(-2.01,17.00)	1074.5
SOR-high	37	16.0			
LOC-high and	29	31.0	0.2611	(-3.00,17.99)	1091.5
ASOR-high	39	13.0			
LOC-high and	29	31.0	0.0016	(4.00,21.00)	3222.0
NDP-high	104	11.0			
LOC-high and	29	31.0	0.0010	(4.00,22.0)	3271.5
NOpt-high	141	10.0			
LOC-high and	29	31.0	0.0014	(4.00,21.00)	3053.5
ANODP-high	231	10.0			
LOC-high and	29	31.0	0.0747	(-0.00,20.00)	900.5
SOS-high	25	13.0			
LOC-high and	29	31.0	0.0058	(2.00,22.01)	1185.0
ASOS-high	37	11.0			
LOC-high and	29	31.0	0.0525	(-0.00,20.00)	1157.0
SOO-high	39	13.0			
LOC-high and	29	31.0	0.0525	(0.00,20.00)	1157.0
ASOO-high	39	13.0			
LOC-high and	29	31.0	0.9896	(-10.01,14.99)	637.0
NES-high	14	28.5			

Table 6.19: Table summarising results of Mann-Whitney tests comparing samples of components selecting using different metrics

Requirements-related metrics

We consider NR, SRR, ASRR, NCR, NSG and RI to be primarily requirements-related and consider them together here. NSG and NCR failed to provide any evidence of a link with change-proneness. The Mann-Whitney tests show that components in the RI-high group experienced similar changes to those in LOC-high, because the test produces a p value of 0.3761. We can assume that there is little difference between the two groups in terms of number of changes, and that therefore size probably plays a major role in explaining the variance in change-proneness described by RI (i.e., components implementing requirements which have sub-requirements are very likely to be large requirements).

We achieve different results for NR, SRR, and ASRR. Comparing changes for m-high to LOC-high produces p values of 0.001, 0.0014 and 0.0035 respectively, low enough to conclude that components predict to be volatile by these metrics are not the same as those predicted by those with high LOC. We thus assume that these metrics are not simply selecting the larger components.

Design-related metrics

SOR and ASOR assess the rationale associated with Decision Outcomes. The p values for Mann-Whitney tests with LOC-high are 0.1843 and 0.2611 respectively - high enough to suggest that

there are no significant differences between predictions made by the two metrics. This implies that changes experienced by components with high SOR/ASOR are at least partially explained by size.

NDP, NOpt and ANODP are concerned with assessing the decision-making process. p values are all well below our target α of 0.05, suggesting a lack of similarity. Like NR, SRR and ASRR, these metrics are unlikely to be simply selecting larger components.

Our most basic design metric is the number of Decision Outcomes affecting a component (NOut). Comparing NOut-high to LOC-high produces a p value of 0.0587 - not quite low enough to reject the null hypothesis and assume that a significant difference exists between the two groups. However, it's very close to our target α of 0.05, indicating we should be cautious.

Values for ASOS and for SOS (and to some extent for SOO and ASOO) are quite polarized: many values lie at zero, whilst others cluster around 20. This creates two major groups of components: those with several supporting arguments made per option, and those with none. The lack of components with an intervening value for any of these metrics suggests there are many components in the system affected by a single option that has 20 supporters/opposers (hence the very similar results for tests on SOS and SOO, and also on ASOS and ASOO). This is specific to our case study project, and may skew our results such that they are not generalisable to other projects. The p values for comparisons with LOC-high are very close to our target α of 0.05, meaning that we need to exercise caution (as with NOut).

Accepting the null hypothesis too readily for, ASOS, SOS, ASOO, SOO or NOut could result in a Type II error (erroneously accepting a false null hypothesis). The confidence intervals for these metrics are quite large (as they are for many metrics in this test, incidentally), which indicates that it is more difficult to find sufficient data to reject the null hypothesis, even if it is false [46]. For this group of metrics, we suggest that the predictions made may be partially explained by size (meaning that a components with, for example, a higher value for SOS is more likely to be a larger component). However, evidence for this is not strong.

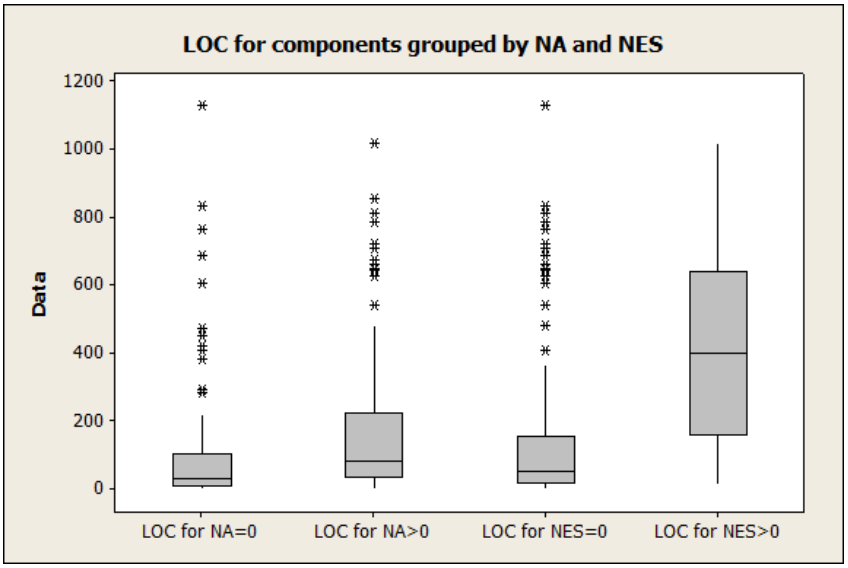

Figure 6.22: Boxplot showing the LOC of components grouped by NES and by NA

We do not compare components with high values for NES or NA and high values for LOC, since both are very coarse-grained metrics that do not make it possible to select a small group of highly scoring components. However, Figure 6.22 presents a boxplot showing the values of LOC for components grouped by NES. The boxplot shows that total range in size is roughly equal between the two groups, with the interquartile range of the group *NES>0* matched by the full range of

'outliers' in the group *NES=0*. Relatively few components achieve a non-zero value for NES, which may add a bias to the diagram.

The plot also suggests that LOC for components in the group *NA>0* is generally higher than for those in *NA=0*, although there are a number of outliers in both groups and the difference in medians between the two groups is not large. This suggests that LOC may explain *some* of the variation in numbers of changes experienced components grouped by NA, but the evidence for this is not large.

In many cases, we conclude that size is only a weak confounding factor for design-related metrics. We hypothesised in Section 3.4.3 that one of two major effects could arise: either high values for design metrics would serve to indicate areas where where design difficulties were encountered which could result in changes later; or low values for design metrics would flag up areas which received relatively little design attention, also resulting in problems later. As we explained in Section 6.5.9, in many cases a substantial portion of the *most* volatile metrics lie outside the highest-valued categories for our metrics. We suggest that both of these effects may be occurring on CARMEN. Many of the *most* volatile components have relatively low values for design metrics, suggesting that they were not a particular focus of design attention. On the other hand an increasing value for most design metrics is associated with an increasing number of changes, suggesting that more design attention in general tends to flag up more 'difficult' components.

We compare the performance of existing metric such as LOC and our new metrics in the next chapter.

6.8 Conclusions

In this chapter we studied our proposed new metrics and analysed their relationship with the number of changes experienced by components from the CARMEN project. In general, we found that most metrics can be correlated with numbers of changes to a greater or lesser extent, with only NCR and NSG failing to provide evidence of this. We made some general observations suggesting threshold values for each metric which does share a relationship with change-proneness. A validation exercise undertaken to check the accuracy of models based on CARMEN's data was described. Although there were some inaccuracies, the validation suggested that most of our observations were reasonably reliable, with the exception of observations on NA. We also checked whether our metrics were simply alternative methods of selecting the larger components in the system, by comparing components predicted as volatile by our metrics to components predicted as volatile by LOC. Some of our metrics were found to act as a surrogate for size, but in many cases evidence that similarities exist between these groups is weak. We accept that size is a likely confounding factor for RI, SOR and ASOR. There is weak evidence that size *may* be a confounding factor for ASOO, SOO, ASOS, SOS and NOut.

Whilst we have found that many metrics produce statistically significant results, we accept that this does not mean that all metrics can be used equally as predictors. In the next chapter we compare the performances of existing metrics, such as LOC, and our proposed new metrics.

Chapter 7

Using our metrics to improve change-proneness predictions

In the previous two chapters we examined the data from our case study project for evidence that change-proneness can be linked to existing complexity metrics and to our metrics. We produced a set of draft guidelines based on these metrics. In this chapter we turn to our final hypothesis:

> H_3: A predictive model using our metrics combined with other types of existing complexity metrics to predict change-proneness outperforms a similar model using the existing complexity metrics alone.

7.1 Introductory summary

We have already shown that a subset of our metrics can be linked to change-proneness. However, finding statistically significant differences does not necessarily mean that all metrics have equal predictive ability. Here we consider whether examining data from the early stages of development (encapsulated in our metrics) *improves* the accuracy of predictions regarding change-proneness.

To do this, we first of all identify the volatile components which we believe that a project manager would want to be able to predict. Our research is most likely to be useful, we believe, to a project manager who might wish to prioritise components in terms of likely volatility so that techniques such as more fine-grained traceability can be applied to these areas, and coarser or even no traceability may be applied in non-volatile areas. This is one way to maximise return on investment for practices such as traceability, by trying to ensure that it is selectively applied to areas where it will be most useful. Projects likely to find this helpful will probably be those with relatively tight margins and deadlines. We are making some assumptions about the likely priorities and preferences of such a project, but throughout this chapter we assume that a small prediction (i.e., predicting volatility for a small number of components) is more useful, even if some volatile areas are missed, than a very large but comprehensive prediction. This is because we envisage that the main purpose of making predictions is to reduce effort where possible; returning a very large prediction, which will include many components which will not be volatile, increases project effort instead without guaranteeing the benefit of usefulness during future volatility.

We do, however, assume that different projects will have different standards for what they might consider to be a sufficiently small prediction. For this reason we identify three different 'tiers' of volatile components. The first and second tiers are subsets of the third tier, and the first tier

is a subset of the second: to reach these tiers we have simply selected the n most change-prone components where n increases for each tier.

Next we determine how to measure 'accuracy'. There are two fairly standard meaures: recall and precision. Recall is concerned with ensuring that there are no omission - that all the components in our volatile set are predicted. Precision is concerned with not predicting change-proneness for components which are not actually volatile. We produce definitions for these measures in Section 7.2.1.

We obtain a 'prediction' from each individual metric (both existing metrics and our own new metrics) by following the heuristics we obtained in Chapters 5 and 6. So, for example, we suggested that components with CBO above 4 are more likely to become change-prone than other components. We therefore select all components with CBO above 4, and use that as a prediction. We consider predictions from all individual metrics, noting those which perform well for recall and those which perform well for precision. We discount metrics which select more than 50 (approximately 1 in 5) entities, assuming that this is this size of prediction exceeds what would be considered practical by a project manager. We discuss performance of individual metrics in Section 7.3.

We then combine metrics to form two-metric models, taking the intersect of both predictions as a joint prediction. When creating combinations we disregard metrics that achieve high precision, since combining these with other metrics tends to reduce the precision without improving recall. We therefore use all those metrics which have achieved good recall, compiling them into two-metric models. We discuss this further in Section 7.4.

There are many ways to assess which model performs 'best'. We present a selection of models that perform well in Section 7; these models have different sizes of predictions and perform slightly differently in different tiers, so they could suit project managers with different expectations. The models which performed best are a mixture of models with only existing metrics (WMC and comments; LOC and WMC), and one existing metric mixed with one of our new metrics (NOut and WMC; WMC and SRR; NDP and LOC). Despite this, no model produced a good level of recall for the top tier (i.e., the small set of the most volatile components). This could suggest that, for the most volatile components, there are factors at work not linked to code structure, design activity or requirements elicitation, and that these volatile components would be difficult to identify by any metric based on this information. We discuss performance of two-metric models in Section 7.5. However, we believe that, in general, our results demonstrate that metrics which attempt to assess requirements and design involvement can be employed for predictive purposes and deserve further research. The rest of this chapter is laid out as follows: Section 7.2 introduces our evaluation criteria and the techniques we use. In Section 7.3 we present the results of evaluation on individual metrics, and in Section 7.4 we examine results from combinations of metrics. We present evaluative discussions in Section 7.5.

7.2 Evaluation techniques

We will use all metrics - both existing metrics and our newly proposed ones - to predict which components are likely to be the most volatile, and then compare the results to determine which are more accurate. In order to do this, we must first consider what we mean by 'accurate'. There are two types of inaccuracies which may arise: false negatives (i.e., not identifying components which are actually volatile); and false positives (i.e., identifying as volatile components which are not especially volatile). Traditional information retrieval techniques permit measuring these inaccuracies by means of:

- *precision*: the ratio of predicted items that are correct
- *recall*: the ratio of correct items that were successfully predicted/retrieved (including those omitted from prediction)
- *F measure*: a single, dimensionless measure combining precision and recall [102]

These measures have been used in many previous studies - to evaluate predictive models such as those proposed in [68, 127, 158], and also for evaluating information retrieval (for example, see [38]). Because they are relatively simple, widely-used measures, we use them again here. We can conceptualise our predictive system as an information retrieval system, because the predictive effort returns some information (lists of components) which we expect should conform to some ideal of relevance. In this case, 'relevance' refers to change-proneness.

An ideal prediction would achieve 100% for both recall and precision. In practice achieving such results is very unlikely. Typically, improving the rates for recall has a negative impact on precision, and vice versa. For example, we can improve the recall figure by simply increasing the number of predictions made (if all items were selected we would achieve a recall of 100%) but this usually results in poorer precision.

Defining success criteria

First we define a region of volatile components in which we are interested. Figure 7.1 is a histogram showing frequency of numbers of changes experienced by components in CARMEN. There are a number of points where we could set a threshold to delimit 'volatile' components of interest:

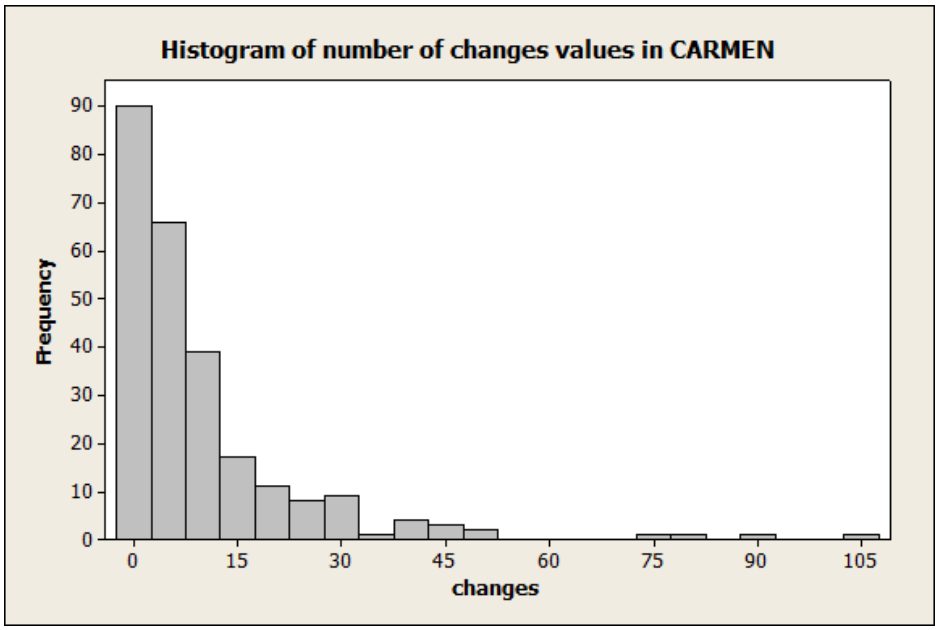

Figure 7.1: Histogram of number of changes

- We should certainly like to identify the 4 outliers with numbers of changes higher than 60. This represents the top 1.5% most volatile components.
- We could place a threshold at 30. This represents the top 7% of most volatile entities - 18 components. Figure 7.1 shows that the number of components drops sharply past 30 changes.
- We could place a threshold at 20. This represents the top 13% of volatile entities (33 components). The frequency of number of changes also drops as it falls between 20.

For our evaluation we use all three thresholds and evaluate metrics' ability to predict he contents of all three tiers. We choose these points because:

- Most components experience fewer changes than this - these are the extremes
- All three of these thresholds represent values where the frequency falls sharply, meaning that there is a clear difference between numbers of components with changes higher than this and lower than this.
- Searching for the top 1.5%, top 7% and top 13% components are reasonably spaced intervals.
- Different types of projects may have different objectives for the total number of volatile components they wish to predict, or where they would set their threshold.
- We might wish to apply different success criteria to the different tiers. For example, it might be more important to recall as many as possible of the top 1.5% volatile components, but more important to be precise in predictions for other tiers.

For many predictive contexts, precision and recall are not equally important [102]. For example, in some situations it is more important to ensure that every possible matching result is returned, even if this also increases the irrelevant results in the result set. On the other hand, for some uses we want to see just a small number of matches, as many as possible of which are highly relevant.

We explained in Chapter 1 that an ability to predict which components will be the most volatile could be useful because this information can be used to distribute resources (such as time-consuming traceability efforts). Unlike some data retrieval systems (like search engines or file searches) the user will not be able to scan lists of data quickly and determine immediately which areas are relevant, because it will not be known until the end of the project which areas were actually volatile. Likewise, if many results are returned the usefulness of the prediction declines, as managers need a short, focussed list in order to concentrate useful change management techniques.

Finally, the *most* volatile is the most important tier. There are a very small number of components in this tier and they are much more volatile than other components.

We rank these success criteria below.

1. We want a parsimonious metric. The number of components predicted must be small, so that project managers can realistically act on those predictions.
2. The *1.5%* tier is more important than lower tiers. Therefore we should like a high recall value for this group, meaning that we'd like to be able to predict as many of the 4 components as possible.
3. In general, we want to ensure that results returned are highly relevant, meaning that precision is more important to us than recall. We prefer to miss a few volatile areas, as long as the ones we do predict are highly relevant.

As mentioned before, when considering precision and recall there are often trade-offs that need to be made because none of these criteria are independent of the others. For example, the requirement to ensure that small numbers of results are returned is related to precision, which tends to be low if large numbers of components are returned. It is highly unlikely that any one metric will be able to satisfy all criteria at once, hence the importance of ranking our criteria as above. It will be important to consider what the information will be used for before selecting a model. For example,

- A project planner looking to to minimise time setbacks and knowledge loss through staff turnover on a large project may wish to ensure that at least two developers are familiar with components likely to be highly volatile. For this purpose, as many of the volatile components should be returned as possible (within a manageably-sized prediction). The planner needs to decide whether it is preferable that developers are familiar with more components than they need to be, than risk losing some knowledge if developers move on in the future. This may depend on project budget. Some may prefer that developers are as familiar with as many components as possible (recall in our models would be needed here), whilst others may decide that developers are busy and only need to concentrate on a few components predicted to be volatile. Any that are missed can then be covered in hand-overs by leaving personnel (a small but precise model would be needed here).

- Impact analyses are an important feature of change management and are heavily underpinned by traceability data. A project planner looking to optimise these techniques could concentrate on components most likely to be volatile, by, for example, using automated techniques for all components to gather the requisite traceability data, but asking humans to refine data for components expected to be volatile. For this purpose, a small, precise prediction may be more useful, since automated data capture may be adequate for many cases and the refinement is only intended for quality control purposes anyway.

- A project planner may wish to use traceability to record and connect design and/or requirements decisions to components. This can help to reduce re-work at a later date if changes are needed and the reasons behind some of the original decisions is not clear. Ensuring that all decisions and rationale are fully traced is time-consuming but a planner could concentrate instead on the most volatile components. Depending on project resources, and the expected product lifespan, it may be more useful to have high recall and document in detail components that may not be volatile, or to concentrate on documenting a small set of the most volatile and accept that some detail for some volatile components may be missed.

7.2.1 Defining our measures

In defining precision and recall we define CARMEN's components using sets:

- V is the total set of volatile components that we wish to identify.

- P is the total set of components predicted using our metrics.

We define *precision* thus: Precision $= \frac{|V \cap P|}{|P|}$
We define *recall* thus: Recall $= \frac{|V \cap P|}{|V|}$
The same definitions are widely used, for example, by [102, 68] and[158].
F measure is a type of mean of precision and recall, calculated thus [72, 102]:
F measure $= \frac{(\beta^2+1)pr}{\beta^2 p+r}$
where p is the value for precision and r is the value for recall. β is normally set to a value of 1, which results in an equal weighting for both precision and recall. In this case, the equation simplifies to [102, 129]:
F measure $= \frac{2rp}{p+r}$
We present standard F-measures in our results, so that our results may be compared to other studies. However we don't wish to apply equal weighting to precision and recall in our results. We have already determined that precision is likely to be more important than recall in general.

Therefore we set β to 0.5, which gives precision twice the weighting of recall (values of β below 1 swing weightings towards precision, and a β above 1 favours recall above precision [102]).

On the other hand, we wish to try and achieve high recall for the *1.5%* tier in particular. For this reason we also calculate, for this tier only, a third F-measure that favours recall; for this F-measure we set β to 2, which results in a figure that gives recall twice as much weight as precision. We use these two F-measures in our evaluations.

7.3 Individual metrics

For each metric m we follow the observations we established in Chapters 5 and 6 to obtain sets of components predicted to be volatile. So, for example, in Chapter 6 we observed that components with NR above 3 were more likely to experience high numbers of changes. Following this observation, we predict volatile elements using NR by selecting all components with NR greater than 3. We do the same for all metrics, both existing complexity metrics generated by CCCC (for which observations were presented in Chapter 5) and our own metrics (observations presented in Chapter 6). Exceptions are NCR and NSG, as our initial analysis failed to uncover evidence that a relationship exists between these metrics and change-proneness. We also exclude NA, since validation showed that our observations on the use of this metric were unsound (see Chapter 6). The average number of components 'predicted' to experience change-proneness was 74.04.

We then calculated values for precision and recall for each of the three change-proneness tiers (which we call *1.5%*, *7%* and *13%*), following the definitions of precision and recall outlined in Section 7.2.1. The total number of volatile entities across all three volatile tiers is 33 (out of the total of 223 components, for which we have data on both CCCC metrics and our own metrics). Full results are presented in Appendix I, and visualised in Figures[1] 7.2 to 7.7.

In Figures 7.2 to 7.7 the trade-off between recall and precision is demonstrated by several metrics. For example, NOC achieves perfect recall values for all three groups (the only metric to do so - visible in Figure 7.2) but low precision (7.5). This is because our observational rules for NOC led to the selection of most of the components as potentially volatile.

7.3.1 Parsimony in individual metrics

The numbers of components predicted to become volatile by each metric vary quite substantially, with the most prolific metric (NOC) including more than 200 components (out of a total 223) in its prediction, whilst NES predicts just 14 components. In general, we feel that any metric including more than 50 components in its prediction is unlikely to be useful for our purpose, although ideally the actual prediction rate will be lower than this still. This means that DIT, NOC, CPL, NR, SRR, ASRR, NOut, NDP, NOpt and ANODP are unacceptable.

7.3.2 Achieving high recall for the *1.5%* tier

Our next criteria stipulated that recall is particularly important for the *1.5%* tier. LOC and MVG, which achieved good scores for recall in other tiers do not perform quite as well here. Neither does NES (LOC, MVG and NES all predict just 1 component of the 4 in this tier). Excellent results (predicting 3 or 4 components) are generally obtained by highly prolific metrics NOC, DIT, CPL, NDP, NOpt, ANODP, NR, CPL, ASRR and DIT, none of which meet our requirements for parsimony.

[1]Figures in this chapter are generated using Open Office Calc unless otherwise stated.

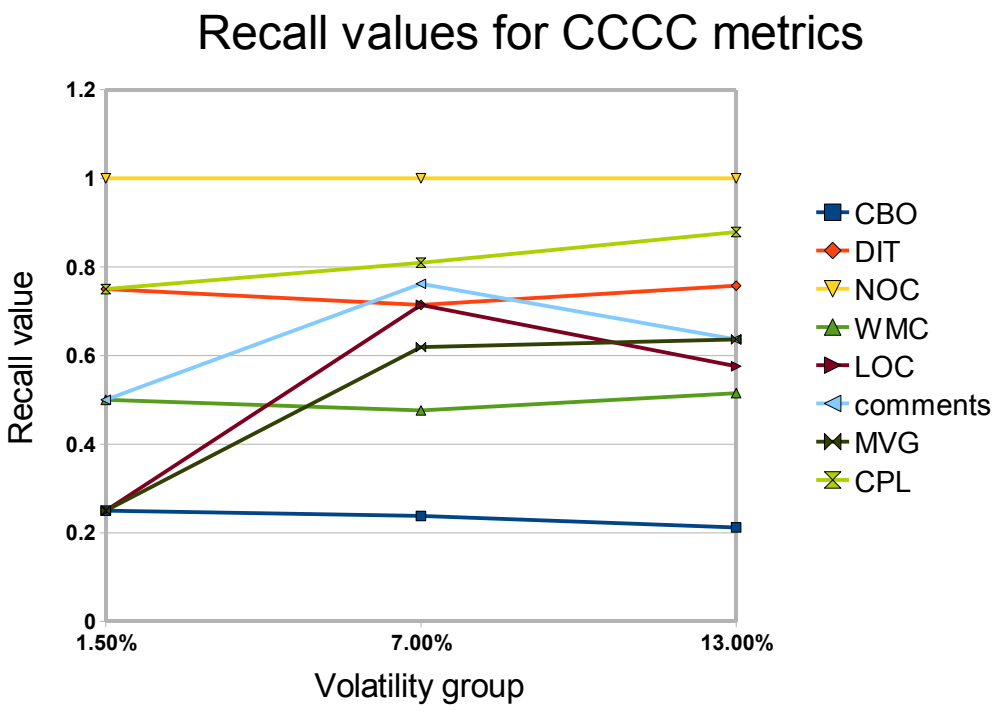

Figure 7.2: Graph showing recall values for CCCC-generated metrics

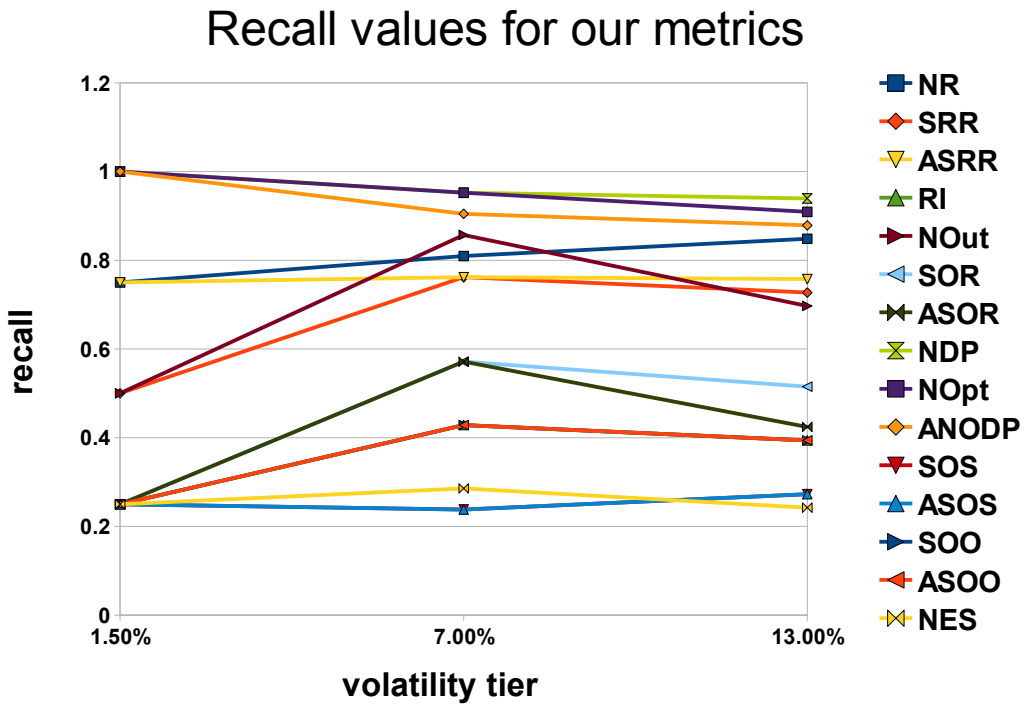

Figure 7.3: Graph showing recall values for our metrics

Slightly different recommendations are produced by the F-measure weighted towards recall (shown in 7.4), which takes both precision and recall into account. This weighted F-measure

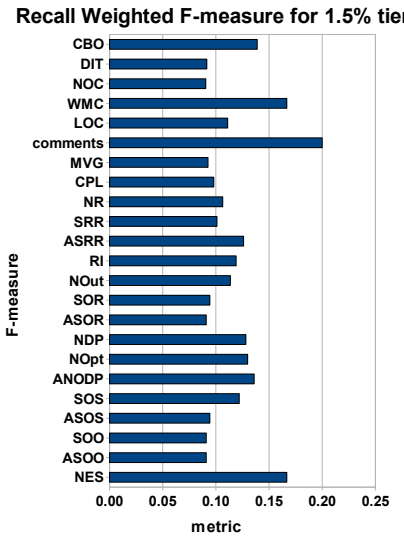

Figure 7.4: Graph showing F-measure values weighted to favour recall (for *1.5%* tier only)

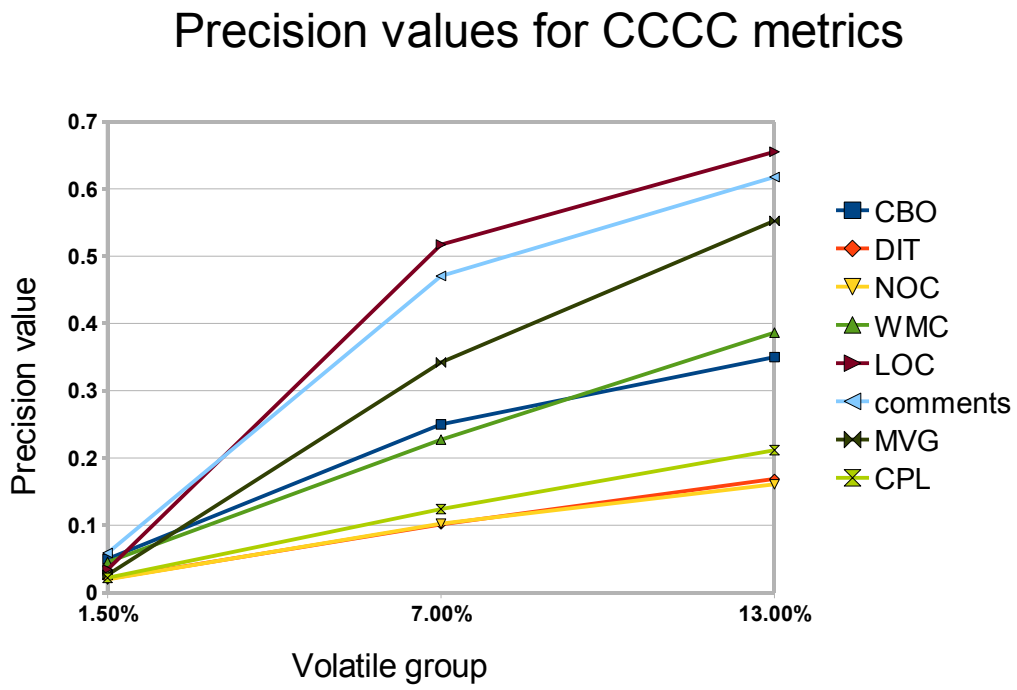

Figure 7.5: Graph showing precision values for CCCC-generated metrics

suggests that best results are achieved by comments, closely followed by WMC and NES. These metrics all make relatively small predictions (below 50), although in absolute terms these are suboptimal metrics for recalling the top 1.5% of components, as each predicts only 1 or 2 of the volatile components in this tier. In fact, WMC and comments are the only two metrics to predict more than 1 of the more volatile components and still meet our parsimony criteria.

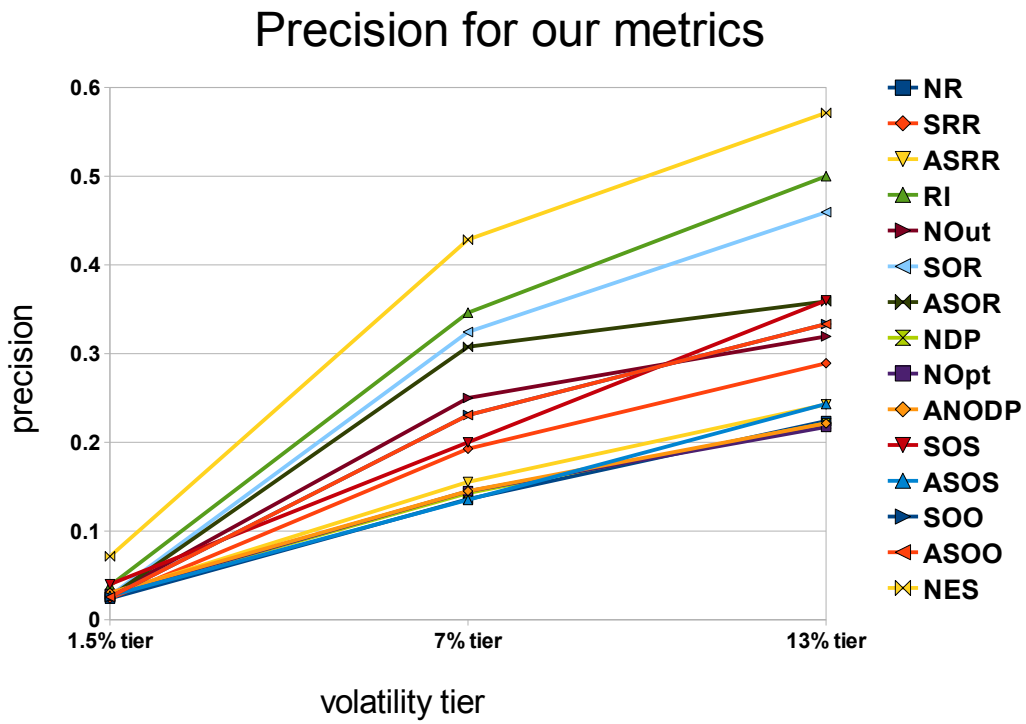

Figure 7.6: Graph showing precision values for our metrics

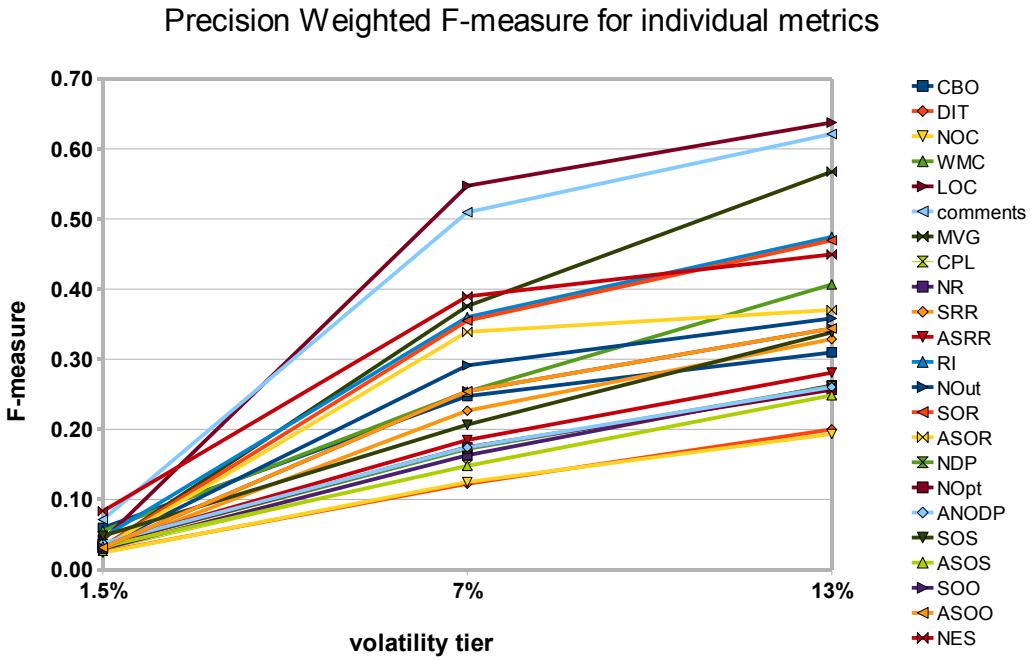

Figure 7.7: Graph showing F-measure values weighted to favour precision

7.3.3 Achieving high precision

As with recall values, metric performance tends to vary for different tiers.

7% and *13%* tiers

LOC achieves the highest absolute precision or precision-weighted F-measure of any metric for the *7%* and *13%* tiers, and comments is not far behind. NES also performs well in the *7%* and *13%* tiers, as do MVG and RI.

NES's strong performance here is derived from its parsimony; it only predicts 14 components, a relatively high proportion of which fall within our three tiers (in fact, as we noted in Section 6.5.4, none of them experience *no* changes at all). LOC, comments and MVG are all well-known metrics which are easily calculated by existing tools, and so their good performance here means they may easily be put to practical use. The disadvantage of using these metrics is that values are generally only available once the code has been produced, and not earlier. NES does not have this disadvantage; information on which components are influenced by external standards would normally be available once a reasonably complete design has been developed.

1.5% tier

Precision results for this tier are generally low. This is partly because the tier only contains four components, and all metrics predict more than four components. The most powerful metric, however, is NES, which outperforms all others (although only by a relatively small margin). This can be seen in absolute terms (see Figure 7.6) as well as in the standard and precision-weighted F-measures. This mirrors precision results for NES in other tiers, too.

Whilst LOC achieves comparatively high precision in other tiers, precision is much lower for the *1.5%* tier, where LOC is outperformed by six other metrics (reflected in the standard and precision-weighted F-measures). Comments, on the other hand, performs well in all three tiers, achieving relatively high precision for the *1.5%* tier in particular (again, reflected in the F-measures). Other metrics that achieve relatively good performance in this tier include CBO and WMC (yet again, reflected in high scores for the standard and precision-weighted F-measures).

Although arguably measuring a similar attribute, CPL does not perform as well as comments. However, it is more generous than comments, predicting change-proneness for a much larger number of components, and this results in a much lower precision.

7.4 Combining metrics

Combinations of metrics may have a stronger predictive power than they do individually. We can group metrics together to make a prediction by requiring that a component meets the observational rules for both metrics, such that the set of predicted components for a combination of metrics m_1 and m_2 is the intersection of predicted components for metric m_1 and predicted components for metric m_2.

An initially simple solution would seem to be a two-metric model, employing one metric with good recall and one with good precision. However, such a model tends produce a reduction in precision for the precise metric (since the final intersection of volatile components produced by the combined metrics will be a subset of the selection made by the more precise metric) and a similar reduction in recall for the metric with good recall. Our two metric models thus consist of two metrics with high recall values - one specifically noted for success the *1.5%* tier, and one for success in other tiers. Combining together two metrics which make large predictions has the potential to create a smaller but more focussed prediction.

So far, the 'best' metrics for recall include:

- Metrics with good recall: CPL, NR, NDP, NOpt, ANODP, NOut, LOC, comments, DIT, SRR and ASRR.
- For the *1.5%* tier, metrics include NDP, ANODP, NOpt and NOC (all of which predict all of the components in this tier).
- Also with the *1.5%* tier, the recall-weighted F-measure favours WMC, NES and comments over the design-related metrics, so we include these also.

Metrics	**% of predictions the same**
NDP and ANODP	95.96%
NDP and NOpt	93.72%

Table 7.1: Metrics with similar predictions

NDP, ANODP and NOpt are metrics with very similar performance, such that they select almost identical sets of components in their predictions. In fact, almost 96% of the predictions made by NDP and ANODP are the same, as are nearly 94% of predictions made by NDP and NOpt (figures provided in Table 7.1). Because they are so similar, we only use only NDP and assume that a similar model employing either NOpt or ANODP in place of NDP would achieve very similar results. We create models using combinations of the other metrics as described above, although we exclude the combination of LOC and comments, since we have already seen that a strong linear relationship exists between these two metrics (see Section 5.6).

A total of 41 two-metric models were created and tested. Appendix I presents precision, recall and F-measures for various combinations of the best-performing metrics. Precision, recall and the two weighted F-measures are also presented in 7.8 to 7.11. The plots each show only the best performing combinations of metrics; for each tier we selected the ten strongest-performing combinations and the plot shows the resulting subset consisting of best performers for all three tiers. There is considerable overlap, however, with many combinations falling into the top ten for more than one tier.

Deciding which combination of metrics satisfies our criteria best is, as expected, a matter of judging between trade-offs. Table 7.2 shows the average number of predictions and precision/recall values for individual metrics, and again for combined metrics. The result of combining metrics together generally reduces the number of predictions, with an accompanying improvement in precision, and a corresponding decrease in recall. We address our criteria one by one below.

		Precision			**Recall**		
Type of model	**No. of predictions**	**1.5%**	**7%**	**13%**	**1.5%**	**7%**	**13%**
Individual metrics	74.04	0.03	0.24	0.35	0.51	0.63	0.6
Two-variable models	48.71	0.06	0.36	0.49	0.48	0.56	0.51

Table 7.2: Some averages for individual and combined metrics

7.4.1 Parsimony for combined metrics

Models with two metrics tend to more parsimonious than individual metrics. As before, we deem any model that makes predictions for more than 50 components to be unacceptable, although a

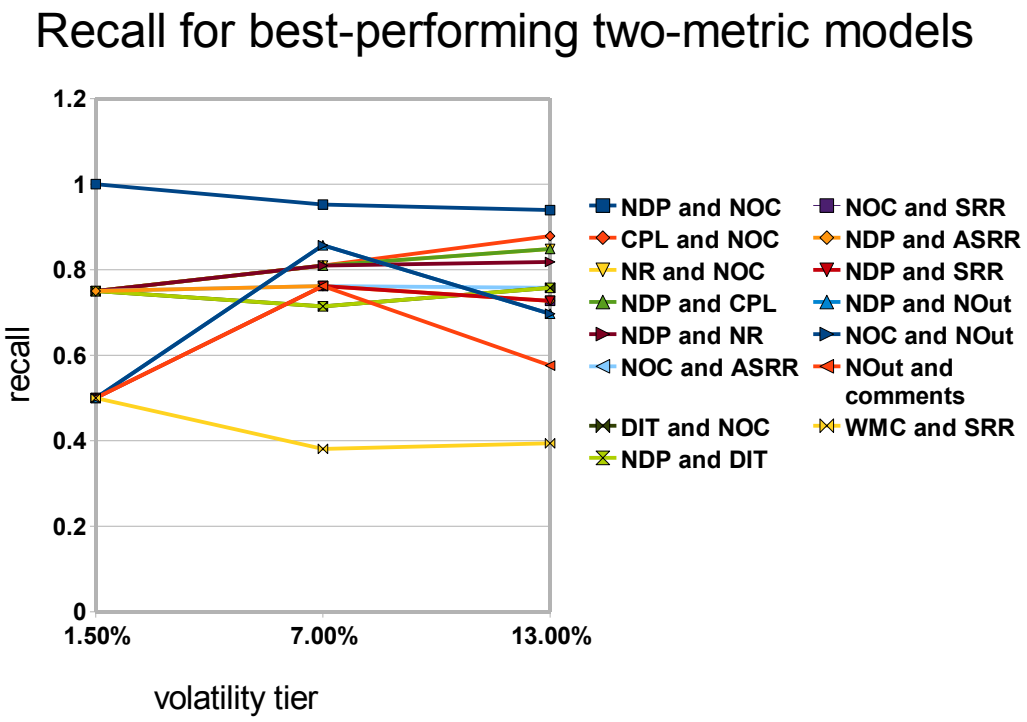

Figure 7.8: Graph showing recall values for combined metrics

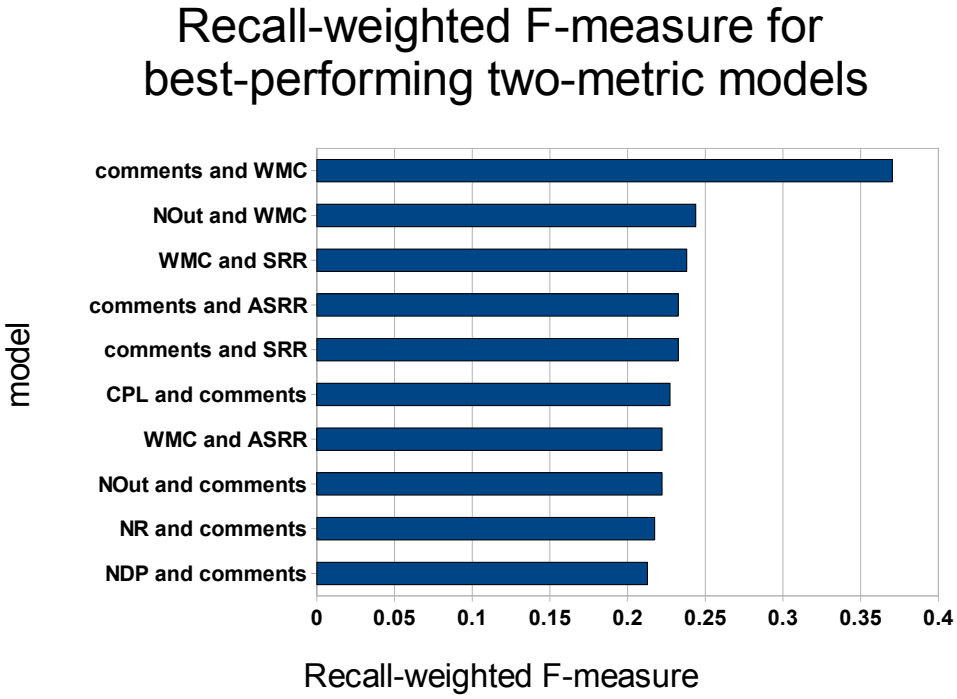

Figure 7.9: Graph showing F-measure values weighted to favour recall

figure considerably lower than 50 is desirable.

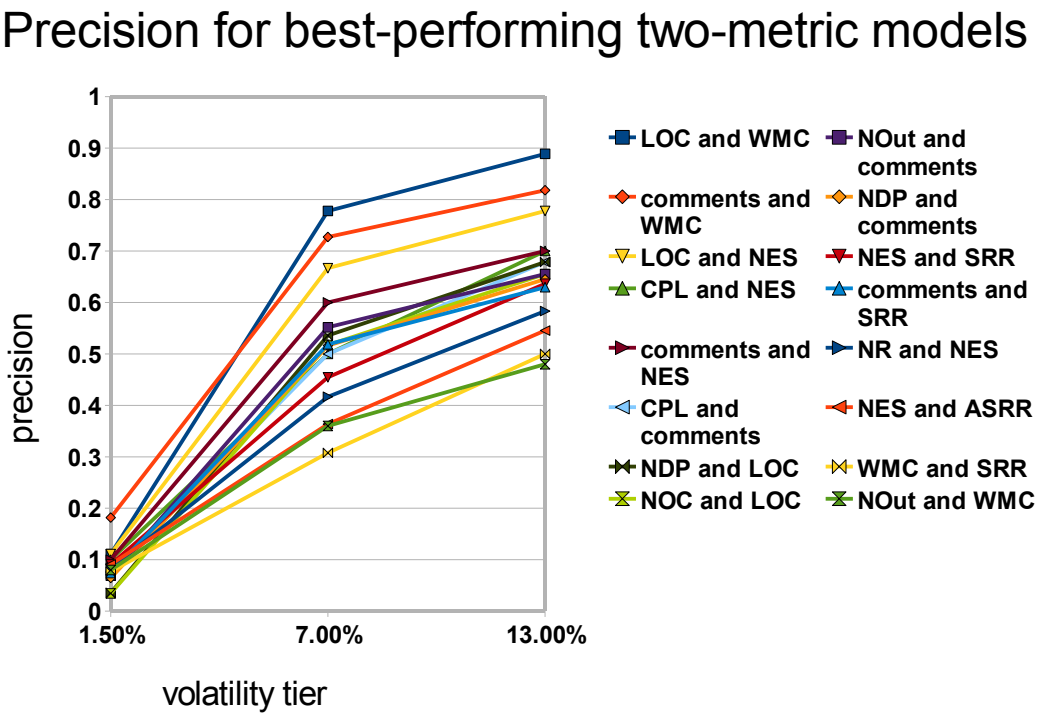

Figure 7.10: Graph showing precision values for combined metrics

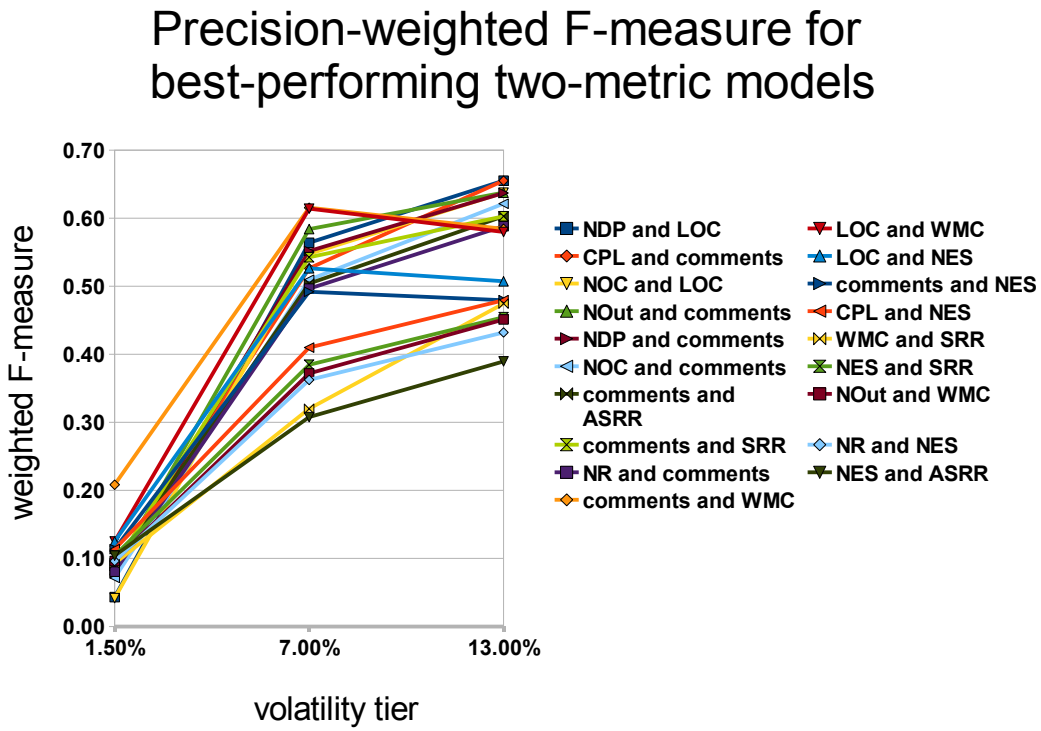

Figure 7.11: Graph showing F-measure values weighted to favour precision

7.4.2 Achieving high recall

Recall values for the best-performing combinations can be seen in Figure 7.8 and a recall-weighted F-measure in Figure 7.9.

1.5% tier

Figure 7.8 shows that the highest result is achieved by *NDP and NOC*, which predicts all four of the volatile entities. This model predicts high change-proneness for 138 components, however, which probably accounts for the low recall-weighted F-measure it achieves. In fact, all of the models predicting more than half of the components in this tier make very large predictions: for example, *NDP and CPL* (99 predictions); *NDP and NR* (113); *NR and NOC* (124); *NOC and NOut* (71); *NDP and DIT* (118); and *CPL and NOC* (134).

The recall-weighted F-measure ranks *comments and WMC* very highly. This model predicts only 2 of the 4 highly volatile components, but also only makes 11 predictions, while 82% of the components predicted fit into one of the volatile tiers and one quarter of components in the tiers are recalled. The combined model makes fewer predictions and achieves higher precision values than either of its constituent metrics do individually.

Other combinations achieve the same result as *comments and WMC* for recall in the *1.5%* tier. WMC is key: as well as *comments and WMC* mentioned above, *NOut and WMC* and *WMC and SRR* are also ranked highly by the recall-weighted precision. These combinations make slightly more predictions than *comments and WMC* (25 and 26 respectively), and achieve lower precision values, but generally have better recall (except for the *1.5%* tier where is it still 0.5).

Many other combinations achieving 0.5 for recall in this tier employ comments, but without achieving quite the same high precision results as *comments and WMC*. We are cautious about accepting a single model which employs comments, however, because (as we mentioned in Section 5.6.2), *comments* is a highly subjective metric likely to be strongly skewed by individual programmer habits.

7.4.3 Achieving high precision

Models employing combinations of metrics tend to achieve better precision scores overall than do the individual metrics.

7% and *13%* tiers

Best results here in absolute terms (shown in Figure 7.10) are achieved by *LOC and WMC*, *comments and WMC* and *LOC and NES*. These combinations also rank highly on the precision-weighted F-measure (Figure 7.11), although the F-measure suggests that effectiveness for all three models declines as we move through the tiers. All three models produce very sparing predictions.

For the final tier (*13%*) the precision-weight F-measure favours *NDP and LOC* and *CPL and comments*. *CPL and comments* selects components which have a large number of comments and a mid-range proportion of comments to code. There's a linear relationship between comments and LOC - see Figure 5.6 - so this could also be interpreted as simply larger components with mid-range proportion of comments to code.

In general, comments appear to be quite potent in achieving high precision. If we rank combined models by their precision in either the *7%* tier or the *13%* tier, then in both cases 11 of the top 16 combinations employ either/both comments or CPL. However, having already identified one high-precision model employing comments (*comments and WMC*), we exclude other comments-based models until further research can show that this is generalisable.

1.5% tier

Best results here (confirmed by absolute values in 7.10 and in the precision-weighted F-measure) are achieved by three combinations already noted for their performance on other tiers: *comments*

and WMC, LOC and WMC and *LOC and NES*.

7.5 Discussion

Size has been connected to change-proneness in the past (for example, see [94]). The results of our study do not contradict this. As a predictor, for example, LOC is reasonably sparing in our case study, predicting high change-proneness for just 29 components. Furthermore, it achieves good recall and precision rates for the upper two tiers (*7%* and *13%*). It does not perform well for the *1.5%* tier, however, either for recall or precision.

Comments achieves good precision in all tiers. As we noted in Section 5.3, comments is a metric that straddles both size and complexity. Similarity to LOC may explain the performance of comments in the *7%* and *13%* tiers. However, the divergence of LOC and comments in the *1.5%* tier (where comments performs reasonably strongly and LOC much less so) suggests that other factors are also at play. Comments may be reflecting complexity - rather than size - for components in the *1.5%* tier. Complexity metrics CBO and WMC also achieve good precision in the *1.5%* tier, which also imply that complexity is more important for the *1.5%* tier.

As we mentioned in Section 7.4, we are reluctant to accept a model that relies heavily on comments until further research demonstrates that comments is generalisable. Arguably the best-performing model we find employs comments (*comments and WMC*, but we choose the almost-as-effective model *LOC and WMC*, which we believe is likely to be more generalisable. LOC and comments are probably measuring very similar attributes here (size). WMC can be characterised partly as a size-related metric, as well as measuring complexity, because a component with many methods is likely to be larger, and because adding methods WMC for an existing component without making other changes also increases LOC. Combining metrics which have some relationship with size (LOC, comments, WMC) generally improves predictive ability, which suggests that size is still the most effective overall predictor we have.

WMC is quite a powerful metric, appearing in several high performing models (*comments and WMC, NOut and WMC* and *WMC and SRR*). This does support some previous work which has found that WMC is linked to changes. Chaumun *et al.* [29], for example, have published a study suggesting that WMC can be used to identify classes which are likely to propagate changes. Our work, in contrast, shows that a higher WMC can be used to identify change-prone classes.

CBO in general is rather patchy in its performance, achieving relatively good precision for the most volatile group, but it is less precise for larger groups of volatile metrics and has poor recall. Previous studies suggest that CBO has relationship with change-proneness, but not necessarily a strongly predictive one. For example, Briand *et al.* [26] found that there were many changes propagated through the system which could not be explained by CBO values.

7.5.1 Predictability of the most volatile components

For the *1.5%* tier, recall is very low except for metrics which make large numbers of predictions. This could imply that there is no one single factor which can be linked to the small numbers of components experiencing extreme change-proneness, and unique individual contexts and/or factors as yet undetected play a role.

The strongest predictive relationships for these components is between change-proneness and NDP, NOpt and ANODP; all three of these metrics recall all the components in this group. It is tempting to suggest that there may be a relationship between very volatile components and design factors. However, we can't ignore the generosity of these metrics, each 'predicting' at least

130 components each (i.e., more than half of all components). Metrics which make acceptably parsimonious predictions generally achieve poor recall for this tier. This applies to both individual metrics and combined models. In fact, the only combinations that predict more than half of the volatile components are ones that return at least 95 components.

7.5.2 Evaluating individual metrics and combined models

Here we evaluate how well our proposed new metrics compare to existing complexity metrics. A major stumbling block for many of our metrics is parsimony; existing metrics generally produce more sparing predictions, which makes them more practicable. We take parsimony into account below. Table 7.3 summarises the models we select as 'best'.

Metric	**Precision**			**Recall**			**Total**
	1.5%	**7%**	**13%**	**1.5%**	**7%**	**13%**	**Predicted**
comments and WMC High precision for all tiers, good recall for *1.5% & 7%* tiers & very sparing.	18%	73%	82%	50%	38%	27%	11
NOut and WMC Reasonable recall-weighted F-measure for *1.5%* tier.	8%	36%	48%	50%	43%	36%	25
WMC and SRR Reasonable recall-weighted F-measure for *1.5%*	8%	31%	50%	50%	38%	39%	26
LOC and WMC Good precision & precision-weighted F-measure for all tiers & is very sparing.	11%	78%	89%	25%	33%	24%	9
NDP and LOC Good precision-weighted F-measure for *13%* tier.	4%	54%	68%	25%	71%	58%	28

Table 7.3: Detailed results for our recommended models

AS mentioned previously (in Section 7.4), NOpt, NDP and ANODP all produced very similar predictions. We used NDP alone from this group but our assumption is that replacing NDP with either NOpt or ANODP would probably produce very similar results. We suggest that NDP is a better option, however, as it is easier to calculate the NDP figure.

Recalling the most volatile set

We can see that the best results for recall of the *1.5%* is achieved individually by WMC or comments, both of which are existing metrics. These results are equalled by some combined metric models: *comments and WMC; NOut and WMC; WMC and SRR; WMC and ASRR; NR and WMC; NDP and WMC* and *CPL and comments*. In the absence of a strong leader in this pack, we look to the number of components predicted to select the best performers here. The most parsimonious models are *comments and WMC, NOut and WMC* and *WMC and SRR*, which predict change-proneness for 11, 25 and 26 components only.

Although the model *comments and WMC* achieves good results, we prefer not to rely upon it until it can be shown to be a generalisable model. Thus the best performing models are mixtures of existing metrics with proposed new metrics: *NOut and WMC* and *WMC and SRR*. The inclusion of either NOut or SRR suggests that a larger and/or more complex code module (flagged by the WMC value) combined with either more design activity (flagged by NOpt) or more requirements rationale activity (flagged by SRR) is a better predictor than WMC alone. Combining WMC with either of these metrics does not reduce recall for the volatile few components, and the number of predictions is much smaller.

For the project manager wishing to ensure that as many as possible of the *most* volatile components are predicted, we believe that *NOut and WMC* or *SRR and WMC* would be an appropriate choice. These two models predict half of the extremely volatile tier and between a third and a half of the next most volatile set. Approximately half of all predictions made by these models fall into one of our three volatile tiers.

Good recall in general - rather than only in the *1.5%* tier is achieved by *NDP and LOC*, which might be useful for a project manager wishing to predict a higher number of volatile components. In CARMEN this model predicts more than half of all 33 highly volatile components, and over three quarters of the top 7% most volatile. The same recall results are also achieved by *NOC and LOC*, but the latter achieves very slightly lower precision than *NDP and LOC*.

Achieving high precision

Best results for individual metrics are achieved by NES (for the *1.5%* and *7%* tiers, and comments, LOC and MVG for the *7%* and *13%* tiers. For a project manager wanting an extremely simple rule of thumb with very little effort attention should be paid to components which are linked to an external standard. Based on results from CARMEN, these will all experience some change-proneness and are more likely than not to fall within the top 13% of volatile components (57% of components predicted fell into one of our tiers).

Despite this, NES's performance in the *1.5%* tier is equalled by models *NOut and WMC* and *WMC and SRR*, although neither exceeds NES's precision or recall for the other tiers. Even better results for precision are achieved by *comments and WMC*, as well as *LOC and NES; LOC and WMC; NDP and LOC;* and *CPL and comments*. We exclude models reliant on comments, leaving *LOC and NES* and *LOC and WMC*. These models achieve better precision in all tiers, better recall in the *1.5%* tier and are more sparing than the next best, *NDP and LOC*. The best performer for precision out of these is *LOC and WMC*.

Thus, unless recall for the most volatile metrics is absolutely crucial *LOC and WMC* is a good choice as and almost all predictions (89%) fall into one of the volatile tiers. This is a very sparing metric, however (with only 9 predictions made); if a higher number are required and precision is still important, then *NDP and LOC* could be appropriate. *NDP and LOC* is a little less sparing, but 68% of predictions fall into the top 13% most volatile components, and whilst precision is a

little lower than *LOC and WMC*, recall is better.

7.5.3 Comparing our metrics with existing metrics

For recall, we identified *NOut and WMC* or *SRR and WMC* for performance in the *1.5%* tier, and *NDP and LOC* for recall in other tiers as the best performers. These models all consist of an existing metric and a new one; the addition of our metrics improves the performance of existing metrics. As we had suggested in Chapter 1, we believe that this improvement is the result of adding new information (from the requirements and/or design stage) to existing information on the code's size and complexity.

Despite this, it is worth noting that no optimal metric or combination has been found that can adequately predict all of the most volatile components.

Best performers for precision include *comments and WMC* and *LOC and WMC*. This suggests to us that, in general, the best precision is achieved when existing metrics are combined together; adding our metrics do not seem to improve this hit rate. Individually, however, NES outperforms existing complexity metrics for the *most* volatile group of components, and is data that would be available at a relatively early stage of the project.

7.6 Conclusions

In conclusion, we return to our final hypothesis, H_3:

> H_3: A predictive model using our metrics combined with other types of existing complexity metrics to predict change proneness outperforms a similar model using the existing complexity metrics alone.

It is important to take a number of factors into account when evaluating the 'best' models, but some of key observations are as follows:

- For the project which wished to ensure that as many of the *most* volatile components were predicted, *NOut and WMC* or *SRR and WMC* would be an appropriate choice.
- Good general recall is achieved by *NDP and LOC*, useful for a manager wishing to predict a high number of volatile components
- If good precision within a sparing metric is needed, *LOC and WMC* is a good choice, although it only predicts a quarter of the most volatile components
- If a larger prediction is required and precision is important, then *NDP and LOC* could be appropriate. This model makes a larger prediction than *LOC and WMC*, but 68% of predictions still fall into the top 13% most volatile components.

In addition, we note that *comments and WMC* was a particularly strong performer in many areas, although further research from other projects is needed to determine the extent to which the performance of comments is dependent on individual programmer habits or project norms.

Taking these results into account, we believe we can argue that, for some uses, adding our metrics can improve the ability of models to recall the most volatile components, when compared to existing metrics alone. In particular, the best recall was produced by models that combined our metrics with existing metrics. This suggests that adding our metrics (which characterise design and/or requirements efforts) to existing complexity metrics adds an extra dimension to the prediction.

On the other hand, adding our metrics to existing complexity metrics does not particularly result in an improvement in precision. The best results here are achieved by a model that combines existing complexity metrics only (*LOC and WMC*). This model is sparing to the point where most volatile metrics will not be returned, however, because the prediction is substantially smaller than the number of volatile metrics in, for example, the top 7% most volatile. A good compromise for a wider prediction, with better recall and slightly impaired precision is the model *NDP and LOC*, which does mix together existing and new metrics.

In general, we believe we have shown that our new metrics can be added to existing metrics to achieve some improved results, in some contexts. In the next chapter we present a general evaluation of the study.

Chapter 8

Study evaluation

In this chapter we evaluate the success of our case study. We return to the success criteria we established in Chapter 1 and consider how well they have been met. We also discuss some other possible threats to validity, and describe our efforts to validate the data gathered in our case study. The rest of this chapter is laid out as follows: Section 8.1 considers our original success criteria. In Section 8.2 we revisit the question (raised in Chapter 2 of whether our metrics are confounded by component size. We discuss in Section 8.3 possible threats to validity and in Section 8.4 we consider the limitations of our study. finally in Sections 8.5 and 8.6 we discuss validation of our metrics and the data used to generate them, respectively.

8.1 Success criteria

In Section 1.1.3 we laid out some criteria which our models must meet in order to be useful for the purposes we have described in Chapter 1. Here we re-visit these criteria and evaluate our results, summarised in Table 8.1.

8.1.1 Metrics generalisability

Some of our metrics are more generalisable than others. We have argued, for example, that the metric *comments* is likely to be skewed by individual programmer habits, and as such should not be generalised to other projects before research has been conducted to confirm our results. We list below other metrics which have issues relating to generalisability:

- RI. We translated RI into a count of the sub-requirements a requirement has. CARMEN does not make very heavy use of sub-requirements, but other projects may use this more.
- Rationale metrics, including NES, NA, SRR, ASRR, SOS, ASOS, SOO, ASOO, SOR and ASOR; and decision-making metrics, including ANODP, NOpt, NOut and NDP. In Section 8.3, we explain that we expect most projects to experience a publication bias when deciding what to record. There are almost certainly more items of rationale playing a role in CARMEN that we have not managed to capture. It is not known at this stage whether the documentation processes and publication bias for different projects will be similar enough to generalise our metrics between projects; more study is needed to ensure that design data represented by our metrics is fully generalisable, although our assumption at this stage is that projects with a culture like CARMEN's probably record a similar level of detail.

The metrics in our recommended models include NDP, SRR and NOut. We believe these are all generalisable to projects that meet the limitations set in Section 4.3.7 and that also have recorded

Criteria	Evaluation
Model is Accurate	We discuss 'accuracy' in Section 7.5. We fail to find any combination of metrics (new or existing) to make acceptably 'accurate' predictions of the *most* volatile components whilst being sparing, but our models are capable of recalling 71% of the top 7% of volatile metrics (*NDP and LOC*), or of returning a prediction of which 89% are components in the top 13% of volatile components.
Model is sparing.	We discuss parsimony in Section 7.5. Many metrics/ combinations of metrics are acceptably sparing; size of prediction for models recommended in Table 7.3 varies from 9 to 28.
Little effort is required to gather data/calculate values	We discuss this in Section 8.1.2. We argue that data in our recommended models can be captured with some small effort.
The model is generalisable	We discuss issues of generalisability in Sections 4.3.7 and 8.1.1.
Necessary data is preferably available by the end of the design stage.	We have described the stages at which we gathered our data in Section 4.5.8, but discuss the implications of this criteria for our metrics in Section 8.1.3.

Table 8.1: Evaluating our original success criteria

their most important requirements rationale and a high level design document that documents major decisions taken.

8.1.2 Ease of generation

We touched on this topic briefly in Section 3.5. Some of our metrics are easier to generate (capturing the necessary data, calculating a value) than others. Our assumption is that existing metrics can generally be generated by case tools and therefore are easy to generate with very little effort. We group our metrics into three groups based on the ease of generating them:

- **Minimal effort** metrics are those which rely on data which are generally already captured by personnel on the project, or by existing case tools. This includes NR and NSG.
- **Small effort** metrics are those which are not necessarily captured already, but which are not difficult to gather given a short amount of time. In many cases, designers and developers on the project may be able to identify intuitively components which have been associated with, for example, more Decision Problems or Outcomes without needing recourse to documentation to check. This includes: SRR, RI, NCR, NOut, NA and NDP.
- **Large effort** metrics are those which require some thought and calculation in order to derive . This includes: ASRR, SOS, ASOS, SOO, ANODP, NOpt and ASOR. Some of these are more difficult to calculate because they represent average figures (meaning that two figures are needed before the final value can be calculated), and some because they are quite distant from the concept of the component on our models. This means some effort is needed to identify intermediary connecting entities as well.

The metrics in our recommended models include NDP, SRR and NOut, all of which fall into the group *small effort*.

8.1.3 Stages at which data for metrics are available

The metrics in our recommended models include NDP, SRR and NOut. All of these should be readily available by the end of the design phase.

8.1.4 Further remarks

Unlike metrics which characterise code size and coupling, our metrics can be generated immediately whenever it is decided that a new component is to be added to the system. Not all components in CARMEN's final solution were planned at the outset - some were added in response to changing conditions. Metrics to describe the size or coupling characteristics of a new, unexpected component cannot be calculated until the new component is relatively complete and added to the system. However, as soon as a decision is made to add a new component, it should be relatively easy to calculate the requirements it implements (and therefore the requirements rationale attached to it), and the decision problems it addresses (and therefore the Decision Outcomes and the number of Decision Problems affecting it). Predictions can thus be made for all components, even unplanned ones, as soon as they are conceived. This does not necessarily apply to code-based metrics, particularly those which measure coupling or size; for these, the component must first be completed and added to the system before values can be generated.

Despite this, our results suggest that code-based or complexity metrics generally perform strongly in making predictions. Existing metrics feature more frequently our recommended models (shown in Table 7.3). As we suggested in Chapter 1, the design tends to be refined iteratively over time, and after implementation begins it may drift to a varying degree from the originally-conceived structure. This is likely to mean that information from the earlier development stages is less likely to be up-to-date with current system structure as the system is developed, and this is likely to have an effect on prediction accuracy, particularly for design-related metrics. This may explain why predictive models that feature our metrics alone do not outperform models of mixed new and existing metrics, or just existing metrics.

8.2 Impact of size as a confounding factor

In Section 6.7.1 we examined the possibility that size is a confounding factor for our metrics. We used LOC to measure 'size'. In general, we argued that:

- size is a likely confounding factor for RI, SOR and ASOR
- size *may* be a confounding factor for ASOO, SOO, ASOS, SOS and NOut
- there's no evidence that size is a confounding factor for NA, NES, ANODP, NDP, NOpt, ASRR, SRR and NR.

In general those metrics which are more strongly linked with LOC do not make strong appearances in lists of the best performing metrics (see Table 7.3). A few exceptions include: RI, which we have observed produces reasonable precision; and NOut, which has reasonable recall for the *1.5%* tier when coupled with WMC. Good precision can be achieved by mixing metrics which are not associated with size with size-related metrics (for example, *LOC and NES, NDP and LOC*). We have already established that neither NES or NDP predicts similar components to LOC (in Section

6.7.1), so combining the two effectively adds extra, non-size-related information to the model and appears to improve its performance. However, these do not ultimately outperform combinations of existing metrics - *LOC and WMC* is arguably the best combined model in terms of precision.

Recalling the most volatile set has been shown to be a difficult task, with no optimal solution. The best models here are *NOut and WMC* and *WMC and SRR*. Although there is no evidence to suggest that SRR and LOC make similar predictions, there is some weak evidence to suggest similarities between predictions made be NOut and LOC. It is less the case here that a traditional metric (LOC) can be combined with entirely new information to produce a better result, but perhaps this is to be expected since LOC itself does not perform particularly well in this task.

8.3 Threats to validity within our case study

In this section we attempt to identify potential pitfalls we recognise in our approach and describe what steps we have taken (where appropriate) to contain them.

8.3.1 Publication bias

One obvious potential flaw is that our design structure - and rationale - is limited to data that was explicitly recorded as part of the overall design document. Decisions that were made without being recorded in this way would escape our notice completely. We have attempted to minimise the risk that this has happened by showing a random selection of our entities to developers from the project and eliciting their thoughts.

Certain situations increase the likelihood that this data will be gathered and recorded. This could include:

- design decisions where the decisions of one development team will impact on other developers' components (e.g., security-related decisions may impact on another team's work).
- decisions - including requirements decisions - which will directly govern or restrict a user's ability to interact with the system. For example, the choice of file types to be supported by CARMEN was a decision on which user input was actively sought, since the outcome(s) of the decision was likely to have a direct impact on user commitment to the system. This results in a thoroughly documented decision process.
- design or requirements decisions where disagreements exist.
- situations where control over some part of the the project is being allocated elsewhere. For example, committing to adhere to an externally controlled standard leaves the team vulnerable to future changes in that standard.
- where developers are aware that a requirement is likely to change in the foreseeable future.

All these situations tend to result in a decision that is documented thoroughly. Our metrics are thus likely to be skewed such that conflicting requirements, disagreements and external standards are overrepresented in the finished models. Simpler, lower-level decisions made by developers during implementation are less likely to be documented and to feature in our metrics models. It is important to take this into account when we are using our metrics to reason about the underlying processes affecting change-proneness on the project. However, we don't believe that it necessarily restricts the generalisability of our results, for the following reasons:

- If we wish to generalise results from CARMEN, any other project we consider will most likely see the same type of bias occurring

- We are specifically interested in improving our ability to predict changes at the design stage. Decisions made during implementation (which are less likely to be documented) would not be available at this stage.

Bearing these points in mind, it is still true that each project will have different expectations regarding the documenting of design and requirements processes. We would need to gather data from other projects to determine whether this would result in different conclusions.

8.3.2 Missing data

It is not unusual for one component to be superseded/replaced by another. For example, a developer could:

- develop solution A and check it in
- at a later date, decide that Solution B would be better; implement solution B and check it in, and perhaps make some changes to it as well
- cease making changes to A
- test B and decide that B is better, but to keep A for the time being as a back-up
- at some point in the future, delete A

Since there's no trail left for this in the subversion repository, we won't identify B as a replacement for A. This would have an effect on our results, since our figures appear to show a large number of components introduced mid-lifecycle; ideally at least some of these should be identified as newer versions of existing files, with a continuous change history.

However, we don't attempt to track replacements extensively (except where the same file has been moved or renamed), since it can be very difficult to decide what component is a continuation of which other component if, for example, solutions A and B consist of multiple files that are structured quite differently. We suspect that our results will appear to show a shorter history - and fewer changes - for many components than is truly the case if we could lace together all replacement components with their originals. This would result in fewer significant relationships detectable between metrics and changes, but we believe that general trends will still be visible.

8.3.3 Structuring project data retrospectively

Relationships have generally been identified and added by a researcher who lacks detailed technical knowledge of the system. It is possible that the main point of the file has been missed and the appropriate links omitted. We try to minimise risk by

- randomly selecting a range of files and asking developers to check the relationships between requirements and files
- randomly selecting a range of structures (including design work) and asking developers to check them over

8.3.4 Developer preferences

Developers have different working habits. For example, some people (especially if collaborating) may elect to check work in early so that changes can be made in tandem with others, or to make use of the back-up functionality of Subversion. Others may complete work outside Subversion

before finally checking in the finished component. However, whilst a number of technical staff have contributed to various aspects of CARMEN's development, committals specifically of the Java components that we are interested in are overwhelmingly dominated by a single developer, meaning that differences between developer habits are less likely to be an issue.

8.3.5 The Hawthorne Effect

The Hawthorne Effect[1] is major issue for many case studies. This term is used to describe a variety of effects, all resulting in the same outcome: a bias introduced into the results, caused by the fact that the subjects of observation are aware they are being observed[2]. This is a recognised problem in fields where researchers are interested in studying human behaviour, such as management, manufacturing, education, psychology and social sciences. Being observed can change a subject's behaviour in itself, but other factors may also come into play, such as positive or negative consequences for the subject; what they perceive the observer is trying to achieve and whether they want to please the observer or not. 'The important effect here was the feeling of being studied' [142]. This has implications for software engineering case studies. For example, team members being observed may they may take extra time to complete all tasks as 'correctly' as possible, or alternatively try to work as quickly as possible. Such a study could not be generalised to other projects accurately.

We do not believe that our presence has had a significant effect on The WP0 team's activity, however, for the following reasons:

- Requirements and design work (our principal focus) was partially completed already when the WP0 team agreed to collaborate in their research.

- There are regular 'all-hands' meetings at which all teams and end-users meet; teams present their current status and progress since the last meeting and other teams and/or end-users can raise questions, place criticism or make requests. Critical input from end-users (most of whom are researchers themselves) throughout the project as well as from other academic specialists (such as usability experts) has been actively sought and acted upon, so the team is accustomed to external scrutiny.

- CARMEN is an academic project. Many personnel involved in the project have PhDs or are working towards a PhD, and progress with aspects of the system forms material for academic papers produced by various members of the team. The research outlined in this thesis represents the observations of one researcher amongst many, on a project which is already the focus of much academic observation (including from from team members themselves).

This particular working culture in itself may limit CARMEN's generalisability. However, we believe that the strong culture of open inspection, examination of each other's work and reflection on one's own practises is not limited to academic projects alone. Many projects - working in a variety of scenarios - hold regular code inspections where constructive criticism may be raised.

[1] A description and discussion by Steve Draper from the Department of Psychology at the University of Glasgow can be found at http://www.psy.gla.ac.uk/ steve/hawth.html

[2] The effect is named after the Hawthorne factory where well-known experiments conducted by Elton Mayo in the 1920s - making a series of sequential alterations to lighting levels, piecework pay rates and team structures - repeatedly resulted in increased productivity, even when the original settings were restored. It was concluded that workers' productivity increased because they knew they were being observed and experimented on - although there has been disagreement as to the reason [142]. Some have also argued that the original data does not support this conclusion[75].

8.4 Limitations in our study

Our study does present some issues to consider.

8.4.1 Defining and measuring 'change'

We are specifically interested in examining change-proneness. In an ideal study, the 'change-proneness' variable could include some awareness of the size of a change made, or the time a developer spent on the change, or the purpose of the change (e.g., planned development versus unexpected change, or refactoring versus bug-fixing). However, as we are working with data drawn from a real project, we are working within constraints determined by the availability of data as well as the time available to us.

CARMEN's team do not maintain logs detailing the reasons for changes (although there are notes added to many components and/or Subversion logs to explain changes made) and also do not maintain time-sheets showing time spent on amendments. Many of the check-in events that we have counted as 'changes' will contain a single, completed change. For example, a relatively common working pattern for developers in an established project is to check out files requiring amendment for a specific purpose, achieve the amendment and then check files back in. There is an incentive to work in this way, since checking files in straightaway ensures that the change is both backed up and available immediately to other developers. However, we do accept that this will not always be the case. Another common habit is to check-in changes only at the end of the day (or even every few days), in which case a large number of small, unrelated changes may have been achieved for a single check-in event, or a partial, large change[3]. Additionally, some team members, particularly early in the project, may not check in work regularly if their work does not yet need to be integrated into the project or shared with others developers. The risk of this is minimised once coding begins in earnest, however, because the benefits of regular backing up, sharing of work with other developers, the need to make fixes available to others quickly and the ability to roll back if necessary all ensure that developers tend to check in work relatively regularly.

We believe that our compromise of counting Subversion check-ins is a sub-optimal surrogate measure for change-proneness, but still acceptable. Files which have not changed are not counted under our scheme. And although some check-in events may cover many smaller 'changes' in one go, others will actually only represent part of a larger amendment. We believe that this does provide an approximation of the amount of effort spent on individual files over the course of development, in the absence of other methods for counting 'changes'.

On a similar subject, we have included in this study all changes made on a project under active, pre-maintenance development. Whilst some changes enacted on code are driven by the need to resolve bugs or respond to user feedback, many more are planned changes that occur because the system is still being expanded to its finished state. Whilst we should like to be able to separate out changes and categorise them, time constraints have not permitted this. However, we suggest that predictions which take into account planned expansion as well as unplanned changes (such as bug fixes, refactoring which becomes necessary or refinements to system performance) are still useful data for project planners.

8.4.2 Handling components added at a later stage

Many of the components present in the later stages of development may not have been planned at design time, for a variety of reasons. For example:

[3]Subversion only 'checks in' files for which a local change exists, so files are not checked in unless they have been altered

- Users may request changes to the system that the developers and designers had not anticipated
- Iterative refinement at a relatively late stage may be necessary to achieve acceptable satisfaction of NFRs
- Decisions on the use of third party components may alter during the course of development, because of unforeseen compatibility or performance issues, because better solutions become available, or because constraints they impose (such as choice of platform) are no longer acceptable
- Refactoring may become necessary

As explained in Section 4.5.8, the data used to calculate values for our metrics was completed concurrently with early implementation. There are some obvious logical problems with attempting to make predictions at this stage about components which are not yet expected to appear. However, we believe our results are still valid for components which are added at a later stage in the project. We are able to state that the components are linked to a certain number of requirements (and their rationale), a certain number of design entities and that they experienced a certain number of changes. As we discussed previously (see Section 8.1.3) we can make predictions about components using our metrics as soon as the new components are conceived, because information on the relevant requirements and design decisions will be known at that stage. Code-related metrics (such as size or coupling measures) will not be available quite so quickly, because the component to be written and added to the system first before size and coupling may be measured.

Adding new components to the system does pose some problems for metrics which assess code coupling, such as CBO, NOC and DIT, because they also have the capacity to change the values of components around them. This means that the values calculated at the start of the development will not necessarily remain stable, and that therefore some predictions will be erroneous: for example, the value of metric m for component c may initially suggest change-proneness, but later in the project another component is added, the values change and there is now no predicted propensity towards change-proneness. We described in Chapter 4 how we obtained a series of snapshots of the components in CARMEN and averaged the values they held for existing, code-based metrics over their lifetime. We believe that using this mean value helps to smooth out large differences occasioned by the addition of nearby components and achieve a more usable figure. However, a useful further exercise could involve studying changes in code-based metrics over time, and determining whether the values calculated at a certain point can be used for making predictions for a limited future window, or indefinitely[4].

8.4.3 Collecting accurate change lifetime counts

We have attempted to trace the life of each component as CARMEN develops. However, many components which are present in the early stages of the system have been removed in the later stages. In many cases they are replaced by new components. We have not traced between new and old versions of components, which means that we do not have as full a history of the changes applied to each area as we would like. Tracing between different versions of components is not simple, because:

- it relies on considerable system knowledge, which we lack without substantial advice and time input from project developers.

[4]In a previous study Ratzinger *et al.* [127] found that data from the previous three months was more useful in predicting refactoring events in the following two months than data from the previous six months [127].

- re-implementing an area of functionality *might* involve a fairly straightforward substituting one component for another. In our system we could represent this as a development in the life of a single component. However, re-implementation may also commonly result in removing one set of components and replacing it with a new set of components, which are organised and structured differently. For example, for CARMEN, changes to the user interface have several times resulted in a restructuring of components which implement it. Thus is no simple mapping from one component to a new version of the same component.

This limitation has the potential to skew the results of our study, because in actual fact the total number of 'components' in the system is probably lower than our system shows. Many components listed separately in our system may be different implementations of the same component, just reworked and replaced, but not linked together. In such a situation, the single combined component would have experienced changes equal to the total number of changes experienced by the two separate components.

In general, we might expect that significant patterns should be easier to detect in system which had fewer components, each with more changes. In our system currently there are many components which have not experienced any changes. This could be one reason why our study fails to detect linear relationships which have been successfully detected by other, previous studies (such as [94]). Despite this, we believe that our study still provides a usable approximation of the changes experienced.

8.5 Metric validation

In Chapter 3 we compared our metrics to some existing standards proposed for the purpose of validating software metrics, concluding that all of our metrics except for averages (ASOR, ASOS, ASOO, ASRR and ANODP) could be claimed to behave as a 'size' metric according to Briand *et al.*'s standards [24]. These particular metrics, however, do not perform particularly well as predictors on the CARMEN study, so we do not feel that our failure to demonstrate that they behave in accordance with some objective definition of a software metric threatens the validity of our results. Metrics which we recommend in Chapter 7 can all be validated as some measure of 'size'.

8.6 Data validation

In Chapter 4 we described how we went about gathering requirements and design data from CARMEN in order to populate our models (and thereafter generate values for our metrics). Extracting data from text and using it to populate models is a highly subjective activity and input from project personnel is necessary to validate accuracy [8]. There are two major areas where inaccuracies may enter our system:

- Relationships between requirements/soft goals and components were added retrospectively by a researcher lacking detailed project and problem domain knowledge.

- The creation of the design entities (Decision Problem, Claim, Outcomes, Options) and entities used as rationale for requirements and decisions (Generators and Claims) was also carried out retrospectively by a researcher without detailed system knowledge.

Several validation exercises were undertaken, detailed in the following sections.

8.6.1 Validating design models

A validation exercise to examine the accuracy of our design models was undertaken at an early stage in the research. For this validation exercise we populated data structures with information drawn from CARMEN's documentation. Then we randomly selected entities from our database, extracted the full structure that surrounded each entity (as can be seen from the example diagrams in Section 4.5.5, there is normally a logical boundary grouping entities from a single model together) until a total of 10% of all entities had been extracted, counting all unique entities in any diagram. These entities were then shown to and discussed with developers from CARMEN.

The specific aim of discussions was to ascertain whether the developers thought that the arrangement of entities was one that represented CARMEN's design process with reasonable accuracy, and whether they thought any entities (e.g., reasons for accepting/rejecting an option, lists of options considered) were missing. The diagrams that were actually shown to developers are presented in Appendix D.

The diagrams included show entities in an earlier version of our design models. This early version of the structure was based very heavily on DRCS (see Section 2.2.3 and [85]). The structures have since changed; see Section 3.2.3 for explanation of the changes. Although the structures shown to developers differ from the final models we have populated, the same data was re-used from the old DRCS structures for the new models. Decision problems, options and claims which support/oppose options have all remained the same. The best option for a relationship was already modelled in DRCS, although in a different way. As a result we are confident that the validity of our *design* entities as a whole still stands. The representation of requirements and the data used to populate them had changed substantially between the earlier version of the model and the later version we carried out further validation of requirements-related material in our more recent models, as described in Section 8.6.2.

A total of 215 entities were shown to CARMEN's developers, and 223 relationships linking those entities. CARMEN's developers suggested some changes to the structures, which included some relatively minor changes (such as altering the wording of entities or relationships) and some more major (such adding missing entities to decision structures). In total 20 relationships were changed (i.e., approximately 9 % of relationships in the random sample), and 9 entities (i.e., approximately 4% of entities in the random sample). We felt this is a reasonable error margin. Assuming that the randomly-selected sample was representative, 96% of entities and 91% of relationships should be accurate.

8.6.2 Validating links between components and requirements

We also validated the links we had retrospectively added between requirements and components by showing a random sample of components to a CARMEN developer. This validation took place approximately one year after the first exercise, which validated design entities. Results of this requirements-oriented validation were only available after the initial data analysis (presented in Chapter 6) had taken place.

We randomly selected 10% of the components in the system (i.e., 25 entities), listed all the requirements, soft goals and design issues which had been allocated to each, and asked CARMEN developers to check that they agreed with the assignments; feedback was provided by the team's most prolific programmer (at time of data-gathering). The data presented to CARMEN's developers are shown in Appendix B. We also asked the same developer to comment on the general guidelines we developed for linking general requirements to large groups of entities (these were listed in Section 4.5.4). The results of the validation became available only after initial analysis

had been carried out.

Area	No. of components changed	Total links before	Total links after	Mean links before	Mean links after	% components correct
Requirements	12	81	92	3.52	4.0	47.83%
Soft goals	8	10	15	0.22	0.65	65.22%
Design	5	54	59	2.35	2.57	78.26%

Table 8.2: Table summarising results of validation on our requirements, design and soft goal links (all results on 25 randomly-selected components)

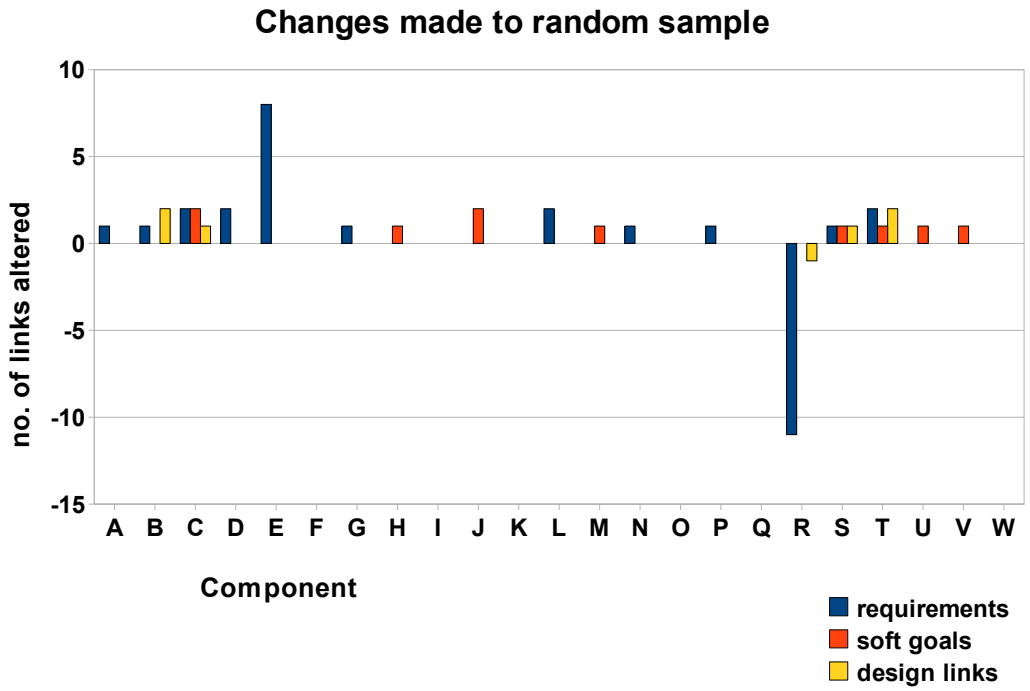

Figure 8.1: Bar chart showing the numbers of links changed as a result of validation exercises. Components are represented by a letter A-T. A positive value indicates number of additions and a negative value indicates number of deletions recommended.

CARMEN staff commented that they did not believe that two of the components in the random sample had been included in the CARMEN system. These were files imported at an early stage by a programmer who had left the project before these comments were made. However, the last-modified dates for these files are after CARMEN's inception and variable names in the files clearly reference the CARMEN project itself. This suggests to us that, like many other files in CARMEN, they are initially used to research ideas but are later superseded by new implementations (current developers confirmed that CARMEN's current implementations built on designs established early on by the departed programmer).

A large proportion of files in CARMEN's system (including some components added at a much later stage) have been employed in a similar way for early prototypes and research. We are interested in gathering data on all the files used throughout CARMEN's life, and so we leave

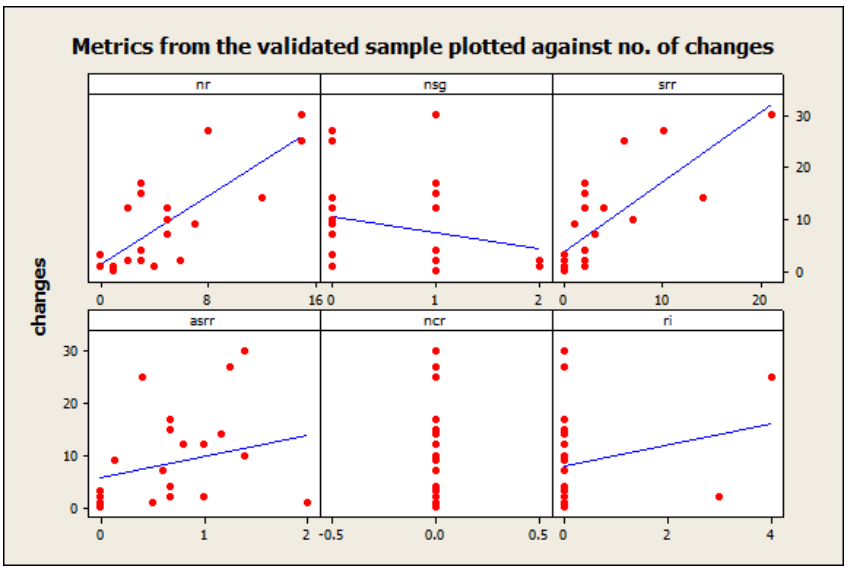

Figure 8.2: Scatter-plot of relationship between metrics and number of changes for corrected random sample

these files in our study. However, because the current developers are not able to comment on the correctness of links for these files we exclude them from our random, validated sample.

Thus, after comments from developers have been implemented, the random sample consists of 23 components with a 'perfect' set of relationships. The amendments that were made varied in scope:

- Design issues identified were largely accurate. We had identified 54 possible relationships linking the 23 randomly selected components with various design issues (these are Design Problems or Design Outcomes *affecting* Components). Developers suggested making changes to 5 components. Assuming that the random sample is representative of the rest of the system, this suggests that approximately 78% of the components are correctly matched up to design issues. Changes were small, with either 1 or 2 relationships added (or removed) in each case.

- Out of the 23 components CARMEN staff suggested soft goal changes to 8, adding either 1 or 2 links in each case. Extrapolating from this to the remaining components, we assume that only around 65% of components are correctly matched to up to soft goals.

- There were 81 individual relationships linking the 23 components to various requirements. CARMEN staff suggested changes to 12 of these components in terms of requirements links, suggesting that only around 48% of our components are linked correctly to requirements. Changes suggested by the team were mostly small, with either 1 or 2 requirements added (or removed) in each case, although two components were changed more radically with larger numbers of adjustments.

These changes are summarised in Figure 8.1.

We wish to test that our conclusions still stand using the randomly-selected, corrected sample. After making the suggested alterations to the random sample, metrics values were regenerated again for these 23 components and analysis performed on them.

Many metrics values remained unchanged. Requirements-related metrics NR, NSG, SRR and ASRR were affected most strongly. For the remaining 14 metrics, only 21 out of a total 322 values[5]

[5]322 is the total number of values, for 14 metrics, each for 23 components.

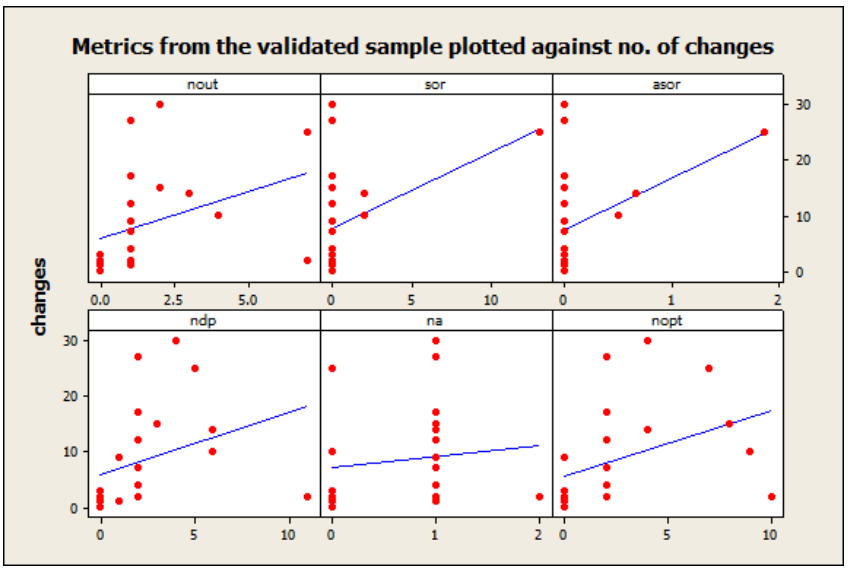

Figure 8.3: Scatter-plot of relationship between metrics and number of changes for corrected random sample

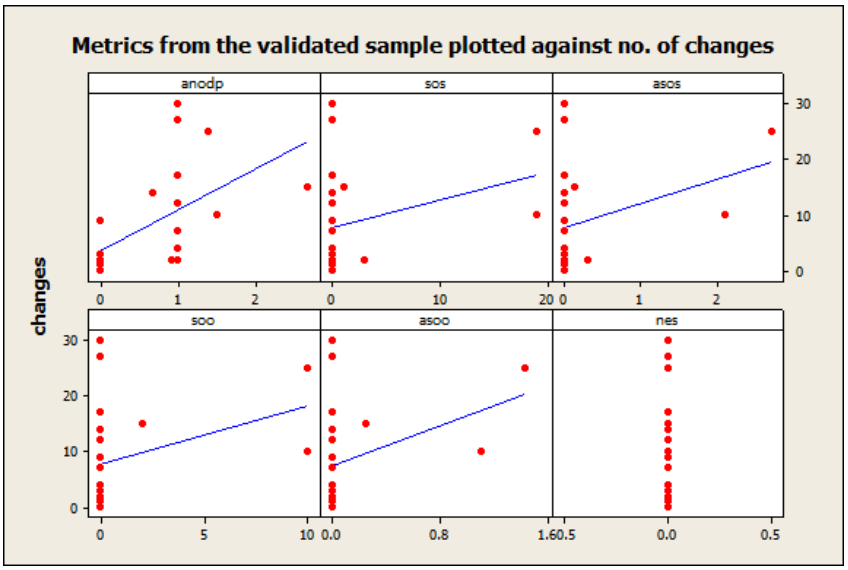

Figure 8.4: Scatter-plot of relationship between metrics and number of changes for corrected random sample

had been altered. These changes were generally small.

The amended values for the random sample were regressed against the number of changes. Scatter-plots of the results are shown in Figures 8.2 to 8.4 and results of regression summarised in Table 8.3. As with earlier analysis, NCR and NES were excluded; all change values in the corrected sample for these metrics were the same. All metrics fail to produce a reliable regression model (indicated by high values for p and/or failure to produce normally-distributed residuals.

We also checked for non-linear relationships, by dividing each sample of 23 into two groups, labelled 'low' and 'high'. We use the same observational rules developed in Section 6.5 to separate components into groups. For example, we had observed in Section 6.5 that components with NOut of 2 or more are more likely to experience higher numbers of changes. We thus placed all components from our corrected sample with NOut of 2 or more into a 'high' group, and all other components into a 'low' group. This will allow us to determine if the significant difference we

Metric	p value	R-Sq (adj)%	Residuals normally distributed	Anderson-Darling result
NR	0.000	60.5%	No	$p = 0.021$
NSG	0.299	0.6%	No	$p = 0.012$
SRR	0.000	56.0%	No	$p = <0.005$
ASRR	0.192	3.6%	No	$p = 0.047$
RI	0.303	0.5%	No	$p = 0.011$
NOut	0.097	8.4%	No	$p = 0.013$
SOR	0.050	13.1%	No	$p = <0.005$
ASOR	0.046	13.7%	No	$p = <0.005$
NDP	0.146	5.5%	No	$p = <0.005$
NA	0.605	0.0%	No	$p = <0.005$
NOpt	0.058	12.1%	No	$p = <0.005$
ANODP	0.006	27.2%	No	$p = 0.013$
SOS	0.169	4.5%	No	$p = <0.005$
ASOS	0.123	6.7%	No	$p = <0.005$
SOO	0.124	6.7%	No	$p = <0.005$
ASOO	0.092	8.8%	No	$p = <0.005$

Table 8.3: Table summarising results of regression tests on individual metrics in a random sample validated by CARMEN's developers

Metric	N	median	p value (adj)	95% Confidence Interval	W
NR low and	9	1.0	0.0032	(-15.00,-2.00)	61.0
NR high	14	11.0			
NSG low and	11	9.0	0.4767	(-3.00,10.00)	144.0
NSG high	12	2.0			
SRR low and	7	2.0	0.0224	(-15.00,-1.00)	49.5
SRR high	16	11.0			
ASRR low and	10	1.5	0.0018	(-16.00,-3.00)	69.5
ASRR high	13	12.0			
NOut low and	17	2.0	0.0243	(-21.004,-1.00)	171.5
NOut high	6	14.5			
NDP low and	9	1.0	0.0008	(-16.00,-2.00)	54.5
NDP high	14	12.0			
NA low and	7	2.0	0.1684	(-12.00,1.00)	63.0
NA high	16	8.0			
NOpt low and	9	1.0	0.0008	(-16.00,-2.00)	54.5
NOpt high	14	12.0			
ANODP low and	10	1.0	0.0041	(-16.00,-1.00)	73.5
ANODP high	13	12.0			

Table 8.4: Table summarising results of Mann-Whitney tests on the validated sample

detected earlier is still valid for components with a corrected set of relationships. SOS, ASOS, SOO, ASOO, NCR, RI, NES, SOR and ASOR are excluded as there are insufficient components

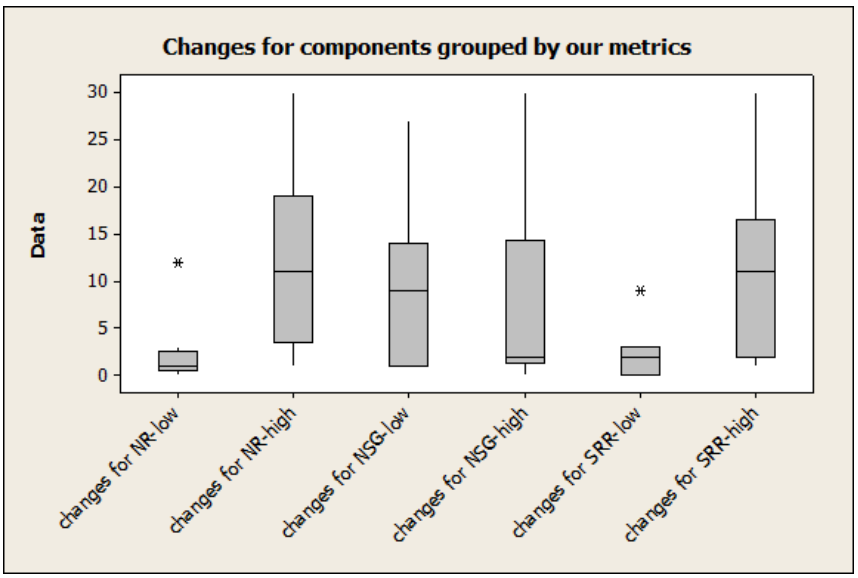

Figure 8.5: Boxplots showing changes experienced by components grouped by metric

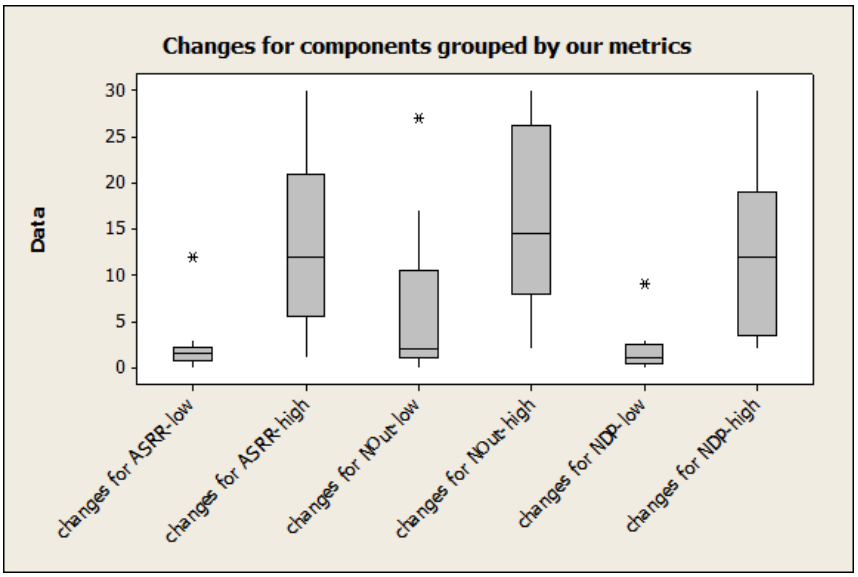

Figure 8.6: Boxplots showing changes experienced by components grouped by metric

with values above zero to create two groups. This does not pose a major problem for SOS, SOO, ASOO, NES, NCR or SOR, because the values of these metrics did not change at all for the validated components after the developers' suggested amendments had been carried out. One value each changed for ASOS and ASOR, and two for RI, as a result of the same amendments.

We then executed two-sample Mann-Whitney tests against the changes experienced by components in the 'low' and 'high' groups. Table 8.4 shows summary results for these tests and Figures 8.5 to 8.7 show the corresponding boxplots.

Impact of validation on earlier results

Some of our earlier observations (summarised in Table 6.17) are supported by the results of analysis on the corrected sample:

- There is no evidence that NSG or NCR can be linked to quantity of changes.

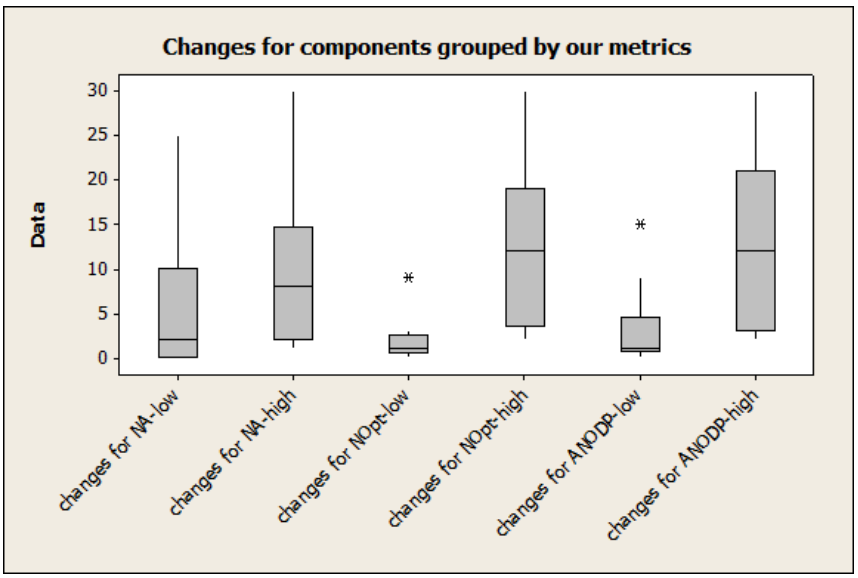

Figure 8.7: Boxplots showing changes experienced by components grouped by metric

- There is evidence that there are significant differences in the quantity of changes experienced by components with lower and higher values of NR, SRR, ASRR, NOut, NDP, NOpt and ANODP which matches earlier conclusions.

On the other hand, analysis on the corrected sample also suggests that our earlier observations may be inaccurate:

- In earlier analysis, we observed that components with a non-zero value for NA were more likely to experience higher change. Mann-Whitney tests on the lower- and higher- value groups in the corrected sample (in practice, this equates to a zero and non-zero group) suggested that there was no evidence of a significant difference, however. NA is one metric that had experienced several changes in values as a result of the corrections.

- Results for ASRR and ANODP seem to be clearer in the corrected sample than in earlier analysis. Both of these metrics earlier were associated with a higher likelihood of increased change for mid-range values, but not necessarily higher values. Analysis with the corrected sample suggests that a higher set of values is associated with a higher risk of changes. However, in practice, these two groups (higher and lower values) actually equate to a zero and non-zero group for ANODP, and a group almost entirely of zero values versus higher values for ASRR.

- We did not have sufficient data available within our sample to carry out similar tests for some metrics. Since SOO, SOS, ASOO, NES, NCR and SOR did not see any changes in values for the corrected sample after we had carried out corrections we assume that observations made earlier still stand.

- We have no way of testing whether observations made earlier still stand for ASOS, ASOR and RI. The values of these metrics in the corrected sample were not very different to the uncorrected values; there was one alteration in values each for ASOR and ASOS, and two for RI. Because the changes were small, we continue with these metrics, but assume that the observations we've made so far are not conclusively demonstrated.

Overall, observations made earlier for most of our metrics appear to be unchanged, with the major exception of NA. We are particularly keen to validate the observations made in Section 6.5 for

requirements-related metrics (NR, NSG, SRR, ASRR), because only around half of the components in the validated sample were correctly matched to requirements. RI is also a requirements-related metric, but values for RI within the corrected sample have altered much less than the former four metrics.

The amendments made do not seem to have rendered the original observations invalid, however:

- Testing groups divided by metric value for the requirements-oriented metrics generally shows that the original observations on requirements-related metrics NR, NSG, SRR and ASRR still hold true on a sample with a 'perfect' set of relationships.
- There are a reasonably high number of relationships existent already (the average component in the random sample was linked to around 3.5 requirements at the start of the validation exercise, and around 4 afterwards), so adding/removing them does not necessarily have a large effect.
- CARMEN does not maintain a system to link requirements to components, and deriving such links afterwards can be a subjective process. Requirements and components are expressed at different levels of abstraction and granularity; requirements, as very early-stage artefacts, can be very difficult to link to later-stage artefacts such as code modules. In addition, some requirements may be implemented by a large collection of components, some of which only play a very minor and/or tangential role. And finally, understanding of the requirements may alter during development, such that implementation differs substantially to the solution envisaged during requirements elicitation. Developing a definitive mapping afterwards is thus a laborious and subjective process. As an example, the requirements engineers working on CARMEN had envisaged entirely separate modules for major areas of functionality, and the wording of requirements reflects this. Later in the project, however, it became more convenient to compile all areas of functionality into 'portal' module.

In total, although it is an area of concern that around half of our randomly-selected sample had errors in their requirements links, the changes made are generally relatively small and our observations still hold true for requirements-related metrics when we test the fully corrected sample. As such, we are content to assume that the same observations still hold true throughout the population of components.

8.7 Conclusions

We conclude that, although our study is not without possible threats to validity, we are satisified that we have met the success criteria we laid out in Chapter 1. Our models meet reasonable expectations of accuracy and sparingness, as well as meeting requirements for ease of generation and utility.

We believe that the study and its findings are sufficiently robust to conclude that there does appear to be a link between requirements and/or design data and change-proneness, and that some models that mix this information with existing code-based metrics have the potential to be useful.

In the next chapter we present our overall conclusions.

Chapter 9

Conclusions and Further Work

This thesis aims to determine whether information describing the requirements and/or design phases of a project can be used to improve our ability to make predictions about where change-proneness will occur in a software project. Our original hypothesis postulated:

> *Assessing the extent of requirements and design activities can lead to improvements in our ability to predict change-proneness in a software system.*

We divided our hypothesis into three sub-hypotheses (in Section 1.1):

- H_1: Existing complexity metrics can be used to predict change-proneness.
- H_2: Our metrics can be used to predict change-proneness.
- H_3: A predictive model using our metrics (combined with other types of existing complexity metrics if necessary) to predict development change-proneness outperforms a similar model using the existing complexity metrics alone.

We have built on previous studies which use metrics to measure code complexity and structure to predict change-prone components. Predictions of change-proneness could be very useful to a project planner:

- if a small number of at-risk components can be predicted relatively early, then efforts such as traceability capture may be concentrated in these areas. This would minimise outlay and maximise the cost-benefit.
- project planners could make sure that areas at risk of high change-proneness are not confined to one member of staff, minimising the risk of knowledge loss.
- documentation (e.g., capturing design decisions and requirements rationale) could be prioritised for the components predicted to be volatile.

We specifically aimed to develop some simple heuristics that could be employed by time-pressed projects to develop an idea of where change-proneness might occur, so that project resources could be concentrated in areas with the maximum impact for the minimum cost. Our results thus need to rely on simple, easy to gather metrics.

To approach this problem, we surveyed some existing methods for capturing and organising design and requirements data in Chapter 2. We followed the Model-Order-Mapping methodology [64] to develop our metrics; this methodology involves creating a model to illustrate how entities relate to each other, and then applying partial ordering rules to develop metrics from the models.

Thus, useful features from existing design and requirements capture systems surveyed in Chapter 2 were distilled into models that structure key pieces of requirements and design data. The models were presented in Chapter 3. In that same chapter we presented definitions of our metrics, and then validated our metrics using existing validation schemes directed specifically at software metrics.

In Chapter 4 we characterised our case study project, CARMEN, explained how we had derived data from CARMEN to populate our models, and documented tool use to generate values for existing metrics. In Chapters 5 and 6 we tested H_1 and H_2, searching for evidence of a relationship between change-proneness and either existing software metrics, or our own software metrics, respectively. This included analysis on existing metrics, which we test to use as a benchmark for comparing to our own new metrics. We searched firstly for linear relationships using linear regression techniques (we found no evidence of such relationships for any metric). We also searched for non-linear relationships using non-parametric Mann-Whitney and Kruskall-Wallis tests to compare components which were grouped by metric values. We discovered that statistically significant differences could be detected for all metrics except NCR and NSG (the numbers of conflicting requirements and the number of soft goals). We used a sequentially-rejective technique to control the error rate across the many comparisons that we conducted, so ensure that our hypothesis testing is sufficiently stringent. The results suggested that our hypothesis is feasible, and that, as a first step, our metrics can be used to make some predictions about the future change-proneness of components in a software system. The results also supported previous research that found existing complexity metrics (such as coupling metrics or size) could be used to predict change-proneness.

Also in Chapter 6, we described a validation exercise which suggested that our attempts to retrospectively add links between requirements and components in CARMEN contained a larger than acceptable number of errors. Despite this, re-running the same tests on a random sample of components from CARMEN which had been corrected by CARMEN staff suggested that the results of our analysis still held firm. The exception to this was the result for NA; no significant results were found for NA in the corrected random sample, and so we considered these results unsound and disregarded NA for subsequent analysis.

Finally, in Chapter 7 we turned to H_3, evaluating whether we can truly argue that our metrics 'improve' the performance of existing metrics in predicting change-proneness. There are a number of criteria to consider in evaluating success, some of which we discussed in Chapter 7. These include:

- Traditional measures of 'success' in information retrieval: recall and precision
- Parsimony in prediction (since models that make very large predictions are not useful to the types of projects and purposes we identified in Chapter 1)
- The development stage at which the metric is available
- How easy metrics are to gather/generate

Our final results from Chapter 7 suggested that we achieved better results when combining different metrics to create a two-metric model. For some purposes, the 'best' models were those that combined one of our metrics containing requirements or design information with an existing metric. For other purposes, however, combinations using existing metrics alone were 'best'.

This result provides us with enough evidence to uphold our hypothesis within some contexts. However, we acknowledge that ours is an exploratory study, and only weak conclusions may be drawn from it until confirmatory studies have also been completed.

The reasoning underpinning our hypothesis relied on the notion that adding contextual information about requirements and design activity would add extra, useful information to the code-based

metrics already in use:

- Adding information on requirements and/or design metrics reveals previously hidden dependencies that could not be discerned from examining the code structure alone.
- Taking account of external rationale for design decisions or requirements inclusion (e.g., acknowledging assumptions or external standards) identifies components more likely to be affected by changes from external sources (a major source of changes for a project, as discussed in Chapter 1).
- Taking account of soft goals allows the model to identify areas likely to be subjected to repeated changes due to difficulty in encoding and designing aspects of performance.
- More time spent on some areas during requirements and/or design acts as a complexity metric in and of itself, by suggesting where particularly difficult components are located.

9.1 Our contributions

This thesis makes the following original contributions:

- Results from this exploratory case study study demonstrate the feasibility of employing requirements and/or design data in addition to code-based complexity metrics to make predictions about change-proneness. Our results suggest that some improvements can be made to existing metrics-based change prediction, depending on whether recall or precision is considered more important. For this reason, we have tried to make clear in Chapter 7 the importance of thinking about the purpose of the prediction before choosing a predictive model.
- We have shown that most of our metrics can be used to detect significant differences in the numbers of changes experienced by components in our case study project. This demonstrates that there are relationships between early stage data and subsequent numbers of changes.
- Previous, similar studies using existing complexity metrics to make predictions about change-proneness have achieved mixed results. This study thus contributes a new case study to a small but expanding body of data. Our results support the notion that metrics can be used to predict changes, although it cannot supply any evidence to support the hypothesis that linear relationships exist (some previous researchers have found this to be the case [94]).

9.2 Further work

We have shown that this a feasible area for future study, but further work needs to be completed to confirm our results. The study is exploratory; we have generated many datasets and searched for those with useful relationships. Whilst this is a valid technique for exploring a new area, only weak conclusions should be drawn from such a study [84]. An appropriate next step is to conduct a confirmational study, from which stronger conclusions can be drawn [84]. Such a case study would be designed explicitly to test whether the findings from CARMEN hold true on another project, and test the specific recommended models we identified in Table 7.3, rather than the wider, more exploratory approach we have adopted here. Conclusions from the second study would:

- demonstrate the robustness of our findings from the CARMEN case study

- provide some initial, useful data on whether comments is a feasible metric, or whether it is influenced too heavily by individual preferences. One study is unlikely to demonstrate this conclusively, however. A sensible method to verify this would be study a variety of projects, examining only the relationship between comments and change-proneness

- demonstrate whether findings related to design metrics can be generalised between different projects, or whether the types of decisions made vary too greatly to become useful metrics

A confirmatory study should also re-examine the thresholds established for each metric in Chapter 5 and 6, to determine whether they are fully generalisable between projects. Some other areas for expanding this work include:

- We have discussed CARMEN's development model and the limits of generalisability that this imposes on our results (Section 4.3.7). We believe that it is worth conducting studies on projects with radically different lifecycle methodologies (such as a more strictly linear, 'waterfall' development or an 'agile' development), to determine whether design and/or requirements data exhibit the same relationship with subsequent changes that are seen on an iterative development such as CARMEN's. Substantially different development methodologies could fundamentally change the relationship between requirements/design and later implementation stages. This would not be easy to detect, given that documentation norms are likely to be different, which would result in differently weighted values for design decisions and requirements rationale metrics. However, this could also enhance our understanding of how early stages can impact on later implementation for different project types.

- NR (the number of requirements) did not achieve particularly outstanding results in our evaluations in Chapter 7 and does not feature in any our recommended models (listed in Table 7.3)[1]. Requirements can differ quite drastically in their relationships with subsequent project artefacts, however. Some requirements (for example, a requirement that the user interface is consistent) are very general and can be applied to a very large number of components, whilst others are far more specific and only apply to a small number of components. An interesting further study could examine whether requirements which are attached to very large numbers of components are associated with less (or more) predictive power than requirements which dictate specific functionality and are implemented by a small set of requirements.

- A potentially useful further study could compare our results from CARMEN to a results from a similar study on a project already in a maintenance phase. The relationship between metrics and subsequent changes may change substantially over the course of a longer project.

- We have discussed previously (in Section 4.3.7) CARMEN's funding model and the limitations this places on generalisability to other projects. Another useful confirmatory study could compare results from CARMEN to results from a project that has a different funding model, such as a team developing boxed products or developing bespoke software for a privately paying client. The requirements elicitation and design phases are likely to receive different treatments in different types of projects (for example, we suggested in Section 4.3.7 that a firm contracted to develop bespoke software has an incentive to ensure that requirements are tightly specified and cannot be changed without incurring extra cost to cover the extra development time) and this could affect the relationship between our metrics and subsequent change-proneness.

[1] This is perhaps because of a high error rate in fitting links between requirements and components; see Section 6.6.

- We have mentioned previously that a useful subject for further study would be to determine the window of time within which our models can be effective, and to consider whether the models become less effective as time passes and the project progresses.

9.3 Final remarks

In this thesis we have outlined the process we have followed to develop some provisional, new metrics to assess the extent of requirements and/or requirements rationale associated with individual components in a software project, and demonstrated the feasibility of using some of these metrics to identify likely change-prone components. A prediction as to the components most likely to become change-prone could be very useful to a project manager at an early stage in the project, who could then take steps (such as employing a less granular traceability) to ensure that changes affecting these areas can be managed cost-effectively. The major original contribution from this body of work is the proposal of the new metrics and attempting to use data from the early stages of project development to make predictions about likely volatility.

Glossary

ANODP Average Number of Options per Decision Problem. Based on the NOpt but a mean figure of Options per Decision problem is calculated. Part of a series of metrics used to assess a system's requirements and design activities, outlined in Chapter 3.

ASOO Average Strength of Option Opposition. Based on the SOO metric but a mean figure of opposing entities per Option is calculated. Part of a series of metrics used to assess a system's requirements and design activities, outlined in Chapter 3.

ASOR Average Strength of Outcome Rationale. Based on the SOR but a mean figure of rationale per outcome is calculated. Part of a series of metrics used to assess a system's requirements and design activities, outlined in Chapter 3.

ASOS Average Strength of Option Support. Based on SOS but a mean figure of supporters per Option is calculated. Part of a series of metrics used to assess a system's requirements and design activities, outlined in Chapter 3.

ASRR Average Strength of Requirements Rationale. Based on SRR, but with a mean figure of rationale per Requirement recorded. Part of a series of metrics used to assess a system's requirements and design activities, outlined in Chapter 3.

Assumption An entity used to represent a assumption unpon which some decision outcome depends. Part of a series of entities used to model a system's requirements and design activities, proposed in Chapter 3.

Attribute hiding factor The percentage of attributes which are private or protected, out of all attributes in a class. One of a series of complexity metrics (the MOOD set of metrics) suggested by Abreu *et al* [48].

Attribute inheritance The percentage of attributes inherited, out of all attributes in a class. One of a series of complexity metrics (the MOOD set of metrics) suggested by Abreu *et al* [48].

CBO Coupling Between Objects. A metric which measures the relationships created when one class acts on another, for example, accessing a member variable or method. One of a series of complexity metrics suggested by Chidamber and Kemerer [31].

Claim An entity used to represent a argument that supports or opposes a requirement or an option, and used to represent some simple requirements or design rationale in our system. Part of a series of entities used to model a system's requirements and design activities, proposed in Chapter 3.

Component An entity used to represent a component in a software system. For our case study we defined Java classes and interfaces as Components. Part of a series of entities used to model a system's requirements and design activities, proposed in Chapter 3.

Constraint An entity used to represent the outcome of a decision. A Decision Outcome entity presents a selection between options whilst a Constraint is a result that narrows the potential range of system behaviour. Part of a series of entities used to model a system's requirements and design activities, proposed in Chapter 3.

Coupling factor The number of links which are not internal and not inherited, out of all possible inter-class couplings minus the maximum number of couplings due to inheritance. One of a series of complexity metrics (the MOOD set of metrics) suggested by Abreu *et al* [48].

DAC Data Abstraction Coupling. The number of abstract data types (ADTs) defined in a class. 'A class can be viewed as an implementation of an ADT'[94]. One of a series of complexity metrics suggested by Li and Henry [94].

Decision Outcome An entity used to represent the actual result of a decision (e.g., a selection between various options). Part of a series of entities used to model a system's requirements and design activities, proposed in Chapter 3.

Decision problem An entity used to represent a situation where a decision is made and recorded. The actual outcome is stored separately. Part of a series of entities used to model a system's requirements and design activities, proposed in Chapter 3.

DIT Depth of Inheritance Tree. Metric representing the number of ancestor classes. One of a series of complexity metrics suggested by Chidamber and Kemerer [31].

External standard An entity used to represent any standard published by a third party (e.g., regulations or data formats) which acts as a constraint or influence on a system. Part of a series of entities used to model a system's requirements and design activities, proposed in Chapter 3.

Generator The set of entities which are Scenarios, External standards or Assumptions. These entities play a role in 'generating' Requirements. Part of a series of entities used to model a system's requirements and design activities, proposed in Chapter 3.

Internal Standard An entity used to represent a standard drawn up by and within the control of the development team themselves (e.g., the format of data files used by the system, etc.). Part of a series of entities used to model a system's requirements and design activities, proposed in Chapter 3.

LCOM Lack of Cohesion of Methods. LCOM is 'a count of the number of method pairs whose similarity is 0... minus the count of method pairs whose similarity is not zero' [31]. 'Similarity' here is assessed by use of a shared instance variable between a pair of methods. One of a series of complexity metrics suggested by Chidamber and Kemerer [31].

LOC Lines of code, a simple expression of size.

Method hiding factor The percentage of methods which are private or protected, out of all methods in a class. One of a series of complexity metrics (the MOOD set of metrics) suggested by Abreu *et al* [48].

Method inheritance The percentage of methods inherited, out of all methods in a class. One of a series of complexity metrics (the MOOD set of metrics) suggested by Abreu *et al* [48].

MPC Message Passing Coupling. The number of send statements in a class. One of a series of complexity metrics suggested by Li and Henry [94].

NA Number of Assumptions. Represents the number of Assumptions upon which any Outcome linked to a given Component depends. Part of a series of metrics used to assess a system's requirements and design activities, outlined in Chapter 3.

NCR Number of Conflicting Requirements. Number of Requirements that conflict with Requirements linked to Component. Part of a series of metrics used to assess a system's requirements and design activities, outlined in Chapter 3.

NDP Number of Decision Problems. Number of Decision problems which can be linked to a Component. Part of a series of metrics used to assess a system's requirements and design activities, outlined in Chapter 3.

NES Number of External Standards. Number of External standards that can be linked to a component, either via a Decision Outcome or via Requirement.

NFR Non-functional Requirements, or 'soft goals'. Requirements which are concerned with performance and quality of a system.

NOM Number Of Methods. The number of local methods in a class. One of a series of complexity metrics suggested by Li and Henry [94].

NOpt Number of Requirements. Number of options that can be linked to a Component via the Decision problems that have affected it. Part of a series of metrics used to assess a system's requirements and design activities, outlined in Chapter 3.

NOut Number of Decision Outcomes that affect a Component. Part of a series of metrics used to assess a system's requirements and design activities, outlined in Chapter 3.

NR Number of Requirements. A count of the number of Requirements directly linked to a Component via 'allocated-to' relationships. Part of a series of metrics used to assess a system's requirements and design activities, outlined in Chapter 3.

NSG Number of Soft Goals. A count of the number of Requirements which are soft goals and are directly linked to a Component via 'allocated-to' relationships. Part of a series of metrics used to assess a system's requirements and design activities, outlined in Chapter 3.

Option An entity used to represent a potential solution considered as part of a decision problem. Part of a series of entities used to model a system's requirements and design activities, proposed in Chapter 3.

Outcome The set of entities which are Internal Standards, Constraints or Decision Outcomes. These entities are generated as a result of a Decision problem. Part of a series of entities used to model a system's requirements and design activities, proposed in Chapter 3.

Polymorphism factor The percentage of attributes inherited, out of all attributes in a class. One of a series of complexity metrics (the MOOD set of metrics) suggested by Abreu *et al* [48].

Requirement An entity used to represent a requirement. Part of a series of entities used to model a system's requirements and design activities, proposed in Chapter 3.

RFC Response For a Class. This is a set of all methods which 'can potentially be executed in response to a message received by an object of that class'. One of a series of complexity metrics suggested by Chidamber and Kemerer [31].

RI Requirement Importance. The number of sub-requirements of a Requirement linked to a Component. Part of a series of metrics used to assess a system's requirements and design activities, outlined in Chapter 3.

Scenario An entity used to represent a simple statement of a user's needs, or basic use case. Part of a series of entities used to model a system's requirements and design activities, proposed in Chapter 3.

SIZE1 The number of semicolons in a class. One of a series of complexity metrics suggested by Li and Henry [94].

SIZE2 The number of attributes + number of methods in a class. One of a series of complexity metrics suggested by Li and Henry [94].

SLOC Source lines of code, a simple expression of size.

Soft Goal An entity used to represent a soft goal, or non-functional requirement (a requirement concerned with performance or quality). Part of a series of entities used to model a system's requirements and design activities, proposed in Chapter 3.

SOO Strengh of Option Opposition. Number of entities that oppose an Option which has been linked to a given Component. Opposing entities are Claims which are linked to an Option via the 'opposes' relationship. Part of a series of metrics used to assess a system's requirements and design activities, outlined in Chapter 3.

SOR Strength of Outcome Rationale. Number of rationale entities recorded for an Outcome, where rationale entities include Claims and Generators. Part of a series of metrics used to assess a system's requirements and design activities, outlined in Chapter 3.

SOS Strength of Option Support. Number of entities that support an Option which has been linked to a given Component. Supporting entities are Claims which are linked to an Option via the 'supports' relationship. Part of a series of metrics used to assess a system's requirements and design activities, outlined in Chapter 3.

SRR Strength of Requirements Rationale. A count of the number of Generators plus the number of Claims which support the Requirements to which a CComponent is linked. Part of a series of metrics used to assess a system's requirements and design activities, outlined in Chapter 3.

WMC Weighted Methods per Class. A count of the methods in a class, which can optionally be weighted using some other metric if required. One of a series of complexity metrics suggested by Chidamber and Kemerer [31]. Chidamber and Kemerer use 'unity' as a weighting, giving each method an equal weighting such that WMC is a simple count of methods in the class.

Appendices

Appendix A

Database Schema and Queries

A database schema diagram and SQL statements for generating metrics-related data are presented below.

A.1 Database schema

Details for tables in the database used to store and calculate metrics described in this thesis are shown below.

Table A.1: Columns for tbl_entity

tbl_entity	
Column	**Description**
id	unique identifier
name	entity name
is_of_type	indicate the type of entity
id	boolean flags whether entity is 'valid' (see Chapter 4)
from_design_doc	boolean flags whether entity was created based on data from the design document

Table A.2: Columns for tbl_entity_types

tbl_entity_types	
Column	**Description**
id	unique identifier
name	name of the entity type

The table tbl_files_full_listing is an import of the files as they are listed in the Subversion-generated change logs.

We can count changes to components by counting the number of times the component's name appears in Subversion's change logs. However, Subversion only lists the names of files which have been changed individually. Changes applied to a whole directory - such as a directory moved or a renamed - would not see the component's name appear in Subversion's logs; just the name of the parent directory would be output by Subversion in the change logs. We should like to include some types of changes that are applied to the parent directory of a component, so we search for them manually and store them in tbl_extra_changes. See Section 4.5.7 for more details.

Table A.3: Columns for tbl_files_full_listing

tbl_files_full_listing	
Column	**Description**
full_list_file_id	unique identifier
name	name of the file (including location)
action_code	a letter representing the action taken: e.g., 'A' for added; 'D' for deleted; 'M' for modified. Generated by Subversion
directory	boolean indicating whether this listing is a directory or a file
valid	boolean to indicate whether this listing is a valid file (see Section 4.5.2 for more on validity)
entity_id	the id of the entity which this file represents
change_id	Subversion's change number (e.g. 'r781')
change_no	Subversion's change number converted to a numeric value (e.g. '781')

Table A.4: Columns for tbl_extra_changes

tbl_extra_changes	
Column	**Description**
entity_id	id of the entity affected
change_no	Subversion version number under which the change occurred

Table A.5: Columns for tbl_metrics_tools

tbl_metrics_tools	
Column	**Description**
id	unique identifier
name	name of the metrics generating tool

Table A.6: Columns for tbl_metrics_generated

tbl_metrics_generated	
Column	**Description**
id	unique identifier
entity_id	id of the entity for which a metric value is recorded
value	value of the metric for this entity
selected	boolean indicating whether the metric for this entity is part of a 'dormant' set of metrics (see Section 4.5.6)
tool_id	id for the tool that generated this value
metric_id	id for the type of metric generated (LOC, CBO, etc.)
change_no	Subversion's version number of the entity being tested
location	the directory where the entity is located in this version of the system

Table A.7: Columns for tbl_metric_types

tbl_metric_types	
Column	**Description**
id	unique identifier
name	name of the type of metric (e.g., LOC, CBO, NOC, etc)

A.2 Database schema diagram

Figure A.1 shows the database schema with foreign keys indicated.

Table A.8: Columns for tbl_relationship_types

tbl_relationship_types	
Column	**Description**
relationship_type_id	unique identifier
name	name of the relationship type (e.g., 'affects', 'allocated-to')

Table A.9: Columns for tbl_relationships

tbl_relationships	
Column	**Description**
entity_key	id of the entity which is the source of the relationship
target_key	id of the entity which is the target of the relationship
relationship_type	id of the type of the relationship
relationship_id	unique identifier

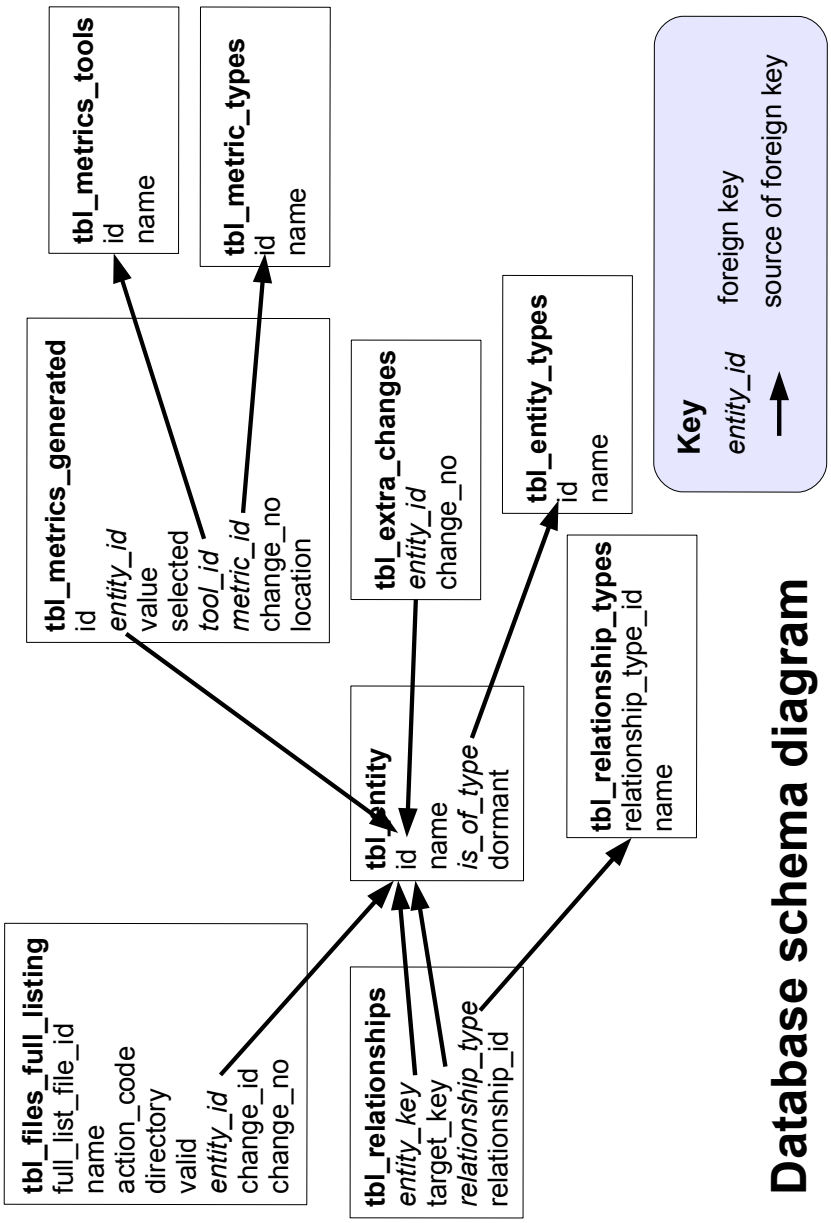

Figure A.1: Database schema diagram showing foreign keys

Appendix B

Validating links

Links between requirements and components ('allocated-to') and between design decisions for CARMEN were introduced retrospectively by examining code components and adding links between the component and the requirements they (helped to) implement and/or design issues which affected them.

The links were validated by showing a random selection of 25 components (representing approximately 10% of whole system of components) to CARMEN's development team to check the accuracy of the links. The selection of components, files and design issues is shown below.

/carmen/trunk/carmen-core/carmen-portal/portal/src/main/java/carmen/york/portal/gui/client/AboutDialog.java

Requirements	'The portal component MUST be a web portal'
Soft goals	Usability
Design issues	Affected by decision to provide a GUI.

/resources/wsPortal/serviceClient/src/main/java/uk/ac/york/carmen/portal/wsportal/serviceClient/WsdlClient.java

Requirements	'Service interfaces MUST be defined in WSDL.' 'Services SHOULD be dynamically deployable.' 'The endpoints of deployed services MUST be discoverable.' 'The portal component MUST be a web portal'
Soft goals	None in particular.
Design issues	The need to protect CARMEN from failures caused by an uploaded service. Whether to deploy services in a virtual machine and any constraints that implies (eg, only one service per machine, that any virtual machines are transparent etc). Decision to provide a GUI.

/carmen/trunk/carmen-core/carmen-portal_release_1.1.32/portal/src/main/java/carmen/york/portal/gui/client/NotificationEntry.java

Requirements	None at this stage.
Soft goals	None at this stage.
Design issues	None at this stage.

/carmen/trunk/carmen-core/carmen-portal_release_1.1.32/UploadApplet/src/main/java/carmen/york/portal/upload/ChunkUploadApplet.java

Requirements	'The portal component MUST allow appropriate users to add new digital artefacts to an existing experiment.' 'The portal component MUST allow users to upload workflows.' 'The portal component MUST allow users to upload services.' 'Resources SHOULD be protected against theft during transfer and storage' 'Resources SHOULD be protected against corruption during transfer' 'The Carmen System SHOULD be protected against malicious attacks' 'The Storage component MUST give authorised users the ability upload digital artefacts to the Carmen system where they are registered and stored in the Storage system.' 'It MUST be possible to store and retrieve the source code of Services where available.' 'The portal component MUST be a web portal' 'An agent MUST be able to exercise his rights.'
Soft goals	None in particular.
Design issues	Whether to make a centralised upload available Whether to provide restart after failure for uploads Decision to provide a GUI.

/carmen/trunk/carmen-core/carmen-portal/portal/src/main/java/Carmen/York/Portal/Gui/client/AnalysisPanel.java	
Requirements	'The portal component MUST allow users to view the output of workflows to which they have access rights.' 'Users MAY query provenance about derived results' 'The portal component MUST have a consistent look and feel.' 'The portal component MUST prevent users from accessing digital artefacts to which they don't have access rights.' 'Access to non-public data MUST be restricted to authorised agents.' 'The portal component MUST be a web portal' 'An agent MUST be able to exercise his rights.'
Soft goals	Usability.
Design issues	Decision to provide accountability to improve security. Decision to provide a GUI.

/carmen/branches/carmen-core/new-look-carmen-portal/portal/src/carmen/york/portal/gui/client/IViewContextChangeHandler.java	
Requirements	None in particular.
Soft goals	None in particular.
Design issues	None in particular.

/carmen/trunk/carmen-core/carmen-portal_release_1.1.32/portal/src/main/java/carmen/york/portal/gui/client/RegisterPanel.java

Requirements	'The portal component MUST have a consistent look and feel.' 'The portal component MUST allow users to register for a user account.' 'The User Management component MUST store user details for user registration purposes.' 'The User Management component MUST ensure that user credentials are unique.' 'Agents MUST be given access credentials on registration.' 'Agents MUST given a set of rights about Carmen data and services.' 'Details of registered users MUST be handled according to data privacy laws.' 'The portal component MUST be a web portal'
Soft goals	None in particular.
Design issues	Decision to provide a GUI.

/carmen/trunk/carmen-core/carmen-portal_release_1.1.32/portal/src/main/java/carmen/york/portal/gui/client/AdminPanel.java

Requirements	'The User Management component MUST provide for different categories of user, to allow system administrators.' 'The User Management component MUST allows admin users to create user accounts.' 'The User Management component MUST allow admin users to change user passwords.' 'The portal component MUST allow administrators to create accounts.' 'The portal component MUST allow administrators to reset passwords.' 'An agent MUST be able to exercise his rights.'
Soft goals	Usability.
Design issues	Decision to provide a GUI.

/carmen/trunk/carmen-core/carmen-portal_release_1.1.32/portal/src/main/java/carmen/york/portal/gui/client/FileChooser.java	
Requirements	'The portal component MUST be a web portal'
Soft goals	None in particular.
Design issues	Decision to provide a GUI.

/carmen/trunk/carmen-core/carmen-portal_release_1.1.32/portal/src/main/java/carmen/york/portal/gui/client/RecentDataPanel.java	
Requirements	'The portal component SHOULD display user's data around the concept of experiments.' 'The portal component MUST allow users to view their experiments.' 'The portal component MUST allow users to see the digital artefacts attached to their experiments.' 'The portal component MUST be a web portal' 'An agent MUST be able to exercise his rights.'
Soft goals	None in particular.
Design issues	Decision to provide a GUI.

/carmen/trunk/carmen-core/carmen-security/carmen-panos-security/VDMFrontEnd/src /vdmfrontend/VDMFrontEndPanel.java	
Requirements	None in particular.
Soft goals	Security.
Design issues	Decision to provide a GUI.

/carmen/trunk/carmen-core/carmen-portal_release_1.1.32/portal/src/main/java/carmen/york/portal/gui /client/INotifyable.java	
Requirements	None in particular.
Soft goals	None in particular.
Design issues	None in particular.

/carmen/branches/carmen-core/new-look-carmen-portal/portal/src/carmen/york/portal/gui /client/MFMenuPanel.java	
Requirements	None in particular.
Soft goals	None in particular.
Design issues	None in particular.

/carmen/trunk/carmen-core/carmen-portal_release_1.1.32/servlets/src/main/java/carmen/york/portal/servlets/LocalGetSRBFile.java

Requirements	'The Storage component MUST allow access to data either as complete files, or specific sections.' 'The Storage component MUST give authorised users the ability upload digital artefacts to the Carmen system where they are registered and stored in the Storage system.' 'The Storage component MUST enforce read only access to ingested artefacts. Writes 'enerate new data items, new metadata and unique identifiers.' 'The storage component MUST be capable of being distributed across multiple Cairns.' 'It MUST be possible to store and retrieve the source code of Services where available.'
Soft goals	None in particular.
Design issues	Decisions on how the workflow enactor requests access to data. Decision to use SRB and SQL.

/carmen/trunk/carmen-core/carmen-portal/portal/src/main/java/carmen/york/portal/gui/client/ServicesPanel.java

Requirements	'The portal component MUST have a consistent look and feel.' 'The portal component MUST allow users to search the metadata for services to enact on their data.' 'The portal component MUST be a web portal'
Soft goals	None in particular.
Design issues	Decision to provide a GUI.

/carmen/branches/carmen-core/new-look-carmen-portal/portal/src/carmen/york/portal/gui /client/ModelOfBuildData.java	
Requirements	'The portal component MUST be a web portal'
Soft goals	None in particular.
Design issues	None in particular.

/carmen/trunk/carmen-core/carmen-portal_release_1.1.32/portal/src/main/java/carmen/york/portal/gui /client/IDBLClickListener.java	
Requirements	None in particular.
Soft goals	None in particular.
Design issues	Decision to provide a GUI.

/carmen/trunk/carmen-core/carmen-portal_release_1.1.32/servlets/src/main/java/carmen/york/portal /servlets/ServiceException.java	
Requirements	None in particular.
Soft goals	None in particular.
Design issues	None in particular.

/carmen/trunk/carmen-core/carmen-portal_release_1.1.32/portal/src/main/java/carmen/york/portal/gui/client/IRightClickListener.java

Requirements	None in particular.
Soft goals	None in particular.
Design issues	Decision to provide a GUI.

/carmen/trunk/carmen-core/carmen-security/carmen-panos-security/PolicyCreator/src/java/PolicyManagement/PolicyCreationSEI.java

Requirements	None in particular.
Soft goals	Security.
Design issues	None in particular.

/carmen/trunk/carmen-core/carmen-portal_release_1.1.32/servlets/src/main/java/carmen/york/portal/servlets/EmptyResultsException.java

Requirements	None in particular.
Soft goals	None in particular.
Design issues	None in particular.

/trunk/Software/Portal/CarmenServlets/src/java/Carmen/York/GetMetaData.java

Requirements	'The System Integration component MUST maintain the session state of users.' 'The portal component MUST prevent users from accessing digital artefacts to which they don't have access rights.' 'The portal component MUST allow administrators to create accounts.' 'The User Management component MUST allows admin users to create user accounts.' 'The User Management component MUST provide for different categories of user, to allow system administrators.' 'The User Management component MUST allow admin users to change user passwords.' 'The portal component MUST allow administrators to reset passwords.' 'The portal component MUST allow users to manage their account e.g. change password and user details.' 'The User Management component MUST allow users to amend the details held about them.' 'The portal component SHOULD display user?s data around the concept of experiments.' 'The portal component MUST allow users to view their experiments.' 'The portal component MUST allow users set the security constraints of their experiments, and the digital artefacts in them.' 'The portal component MUST allow user to search the metadata for workflows.' 'The portal component MUST allow users to search the metadata for services to enact on their data.' 'The portal component MUST allow users to search the metadata for experiments, including those that they MAY not have access rights to.' 'Carmen MAY Keep Logs about access to data, services and workflows' 'Carmen MAY keep record of the userIDs, data lsids and service lsids, actions (action verbs and method calls)' 'Carmen MAY Store and maintain logs within its domain.' 'The User Management component MUST allow users to view the details held about them.'

	'The agent rights management system MUST be able to update agent's rights.' 'Allow metadata, and artefacts to be individually made public.' 'The portal component MUST allow users to manage access rights.' 'Agent's interactions with system MAY be logged.' 'Data usage MAY be logged.' 'The portal component MUST be a web portal' 'The User Management component MUST not allow users to view the details held about other users.' 'An agent MUST be able to exercise his rights.'
Soft goals	Decision to provide accountability to improve security.
Design issues	Decisions relating to how to get access to metadata (such as location) for services and/or data (e.g., suggestion to create method *RetrieveMetadata()* on p.10 of carmen-system-design.doc).

/carmen/branches/carmen-core/new-look-carmen-portal/portal/src/carmen/york/portal/gui/client/ViewContext.java

Requirements	'The portal component MUST have a consistent look and feel.' 'The portal component MUST be a web portal' 'Concerned with the soft goal of usability'
Soft goals	None in particular.
Design issues	Decision to provide a GUI.

/carmen/trunk/carmen-core/carmen-portal_release_1.1.32/portal/src/main/java/carmen/york/portal/gui/client/IntroductionPanel.java	
Requirements	'The portal component MUST be a web portal'
Soft goals	None in particular.
Design issues	Decision to provide a GUI.

/carmen/branches/carmen-core/new-look-carmen-portal/portal/src/carmen/york/portal/gui/client/ModelReadWriteBaseClass.java	
Requirements	None in particular.
Soft goals	None in particular.
Design issues	None in particular.

Appendix C

Validation Queries

The tests below were used to ensure that the metrics models were populated in accordance with some basic rules and that (as far as possible) no detectable errors were introduced into the data.

C.1 Requirements model

We checked that:

- Only requirements participate in the has-subrequirements relationship
- Only permitted entities participate in the 'generates' relationship. This is assumption, external standard or scenario for the source, and requirement, soft goal, internal standard, decision outcome or constraint as a target
- Only permitted entities participate in the 'conflicts-with' relationship (requirements should be both the target and the source)
- Only permitted entities participate in the 'allocated-to' relationship - this should be requirement as source and component as target
- Only permitted entities participate in the 'raises-issue' relationship (this should be a requirement, soft goal, decision problem, internal standard, outcome or constraint as the source and a decision problem as the target)
- Only permitted entities participate in the 'relies-on' relationship. This should be requirement, internal standard, outcome or constraint as the source and assumption as the target
- Only permitted entities participate in the 'supports' relationship (list shown in C.1).
- Only permitted entities participate in the 'oppose' relationship (list shown in C.2).

C.2 Design model

We checked that:

- Only decision problems and options participate in the 'has-option' relationship. This should be decision problem as the source and option as the target.
- Only appropriate entities participate in the 'affects' relationship. This should be internal standard, outcome, constraint or decision probelm as the source and component as the target.

Source	Target
Requirement or soft goal	Requirement or soft goal
claim	requirement or soft goal
claim	claim
claim	option
claim	internal standard, outcome or constraint
assumption, external standard, scenario	claim

Table C.1: Entities that legitimately participate in the 'support' relationship

Source	Target
claim	requirement
claim	claim
claim	option
claim	internal standard, outcome or constraint

Table C.2: Entities that legitimately participate in the 'oppose' relationship

- Only appropriate entities participate in the 'has-outcome' relationship. This should be decision problem as the source and internal standard, outcome or constraint as target.

C.3 General queries

The queries below are designed to ensure that data drawn FROM the CARMEN project and represented in the database follows some basic rules. We checked that:

- There are no orphaned entities (i.e., entities which do not participate with any other entities) which could indicate a problem area we need to revisit.
- There are no duplicates in the relationships table.
- There are no circular pairs of relationships (ie, duplicate entities exist in relationships table, with A related to B in one entry and B related to A with the same type of relationship in another entry. This is only a risk for relationships where source and target types are permitted to be the same.
- All options participate in the has-option relationship. There should be no options which do not participate in this relationship.
- All decision problems are raised by something.
- Internal standards, decision outcomes and constraints are generated by something.
- All claims are supporting or opposing something.
- All external standards and assumptions are linked to something. We don't check that scenarios are linked to something, as we decide that a scenario may legitimately not be attached to a design structure if that particular use case is not addressed by the system.
- No entity is linked to itself.

Appendix D

Validating design and requirements data

Presented here are diagrams which were created using information from design documents, and then shown to and discussed with CARMEN developers. The object was to ensure that the design entities created and the relationships between them were an accurate representation of ideas taken into account during the design process. Some changes were made to the system as a result of these discussions. The diagrams below represent earlier versions of our design and requirements models, although entities shown have not changed drastically. See 8.6.1 for a full discussion.

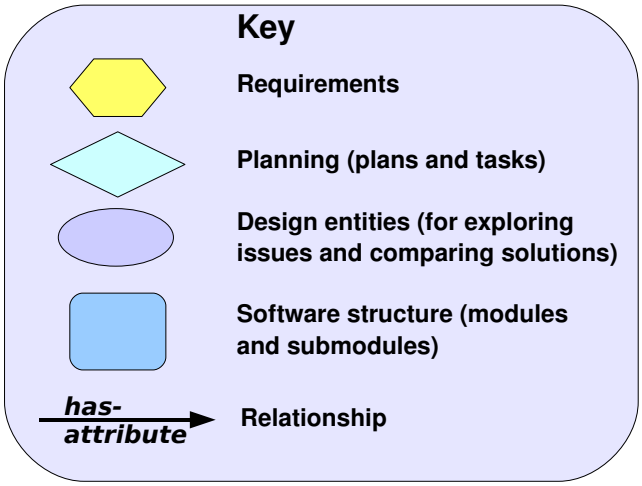

Figure D.1: Key for design and requirements validation diagrams

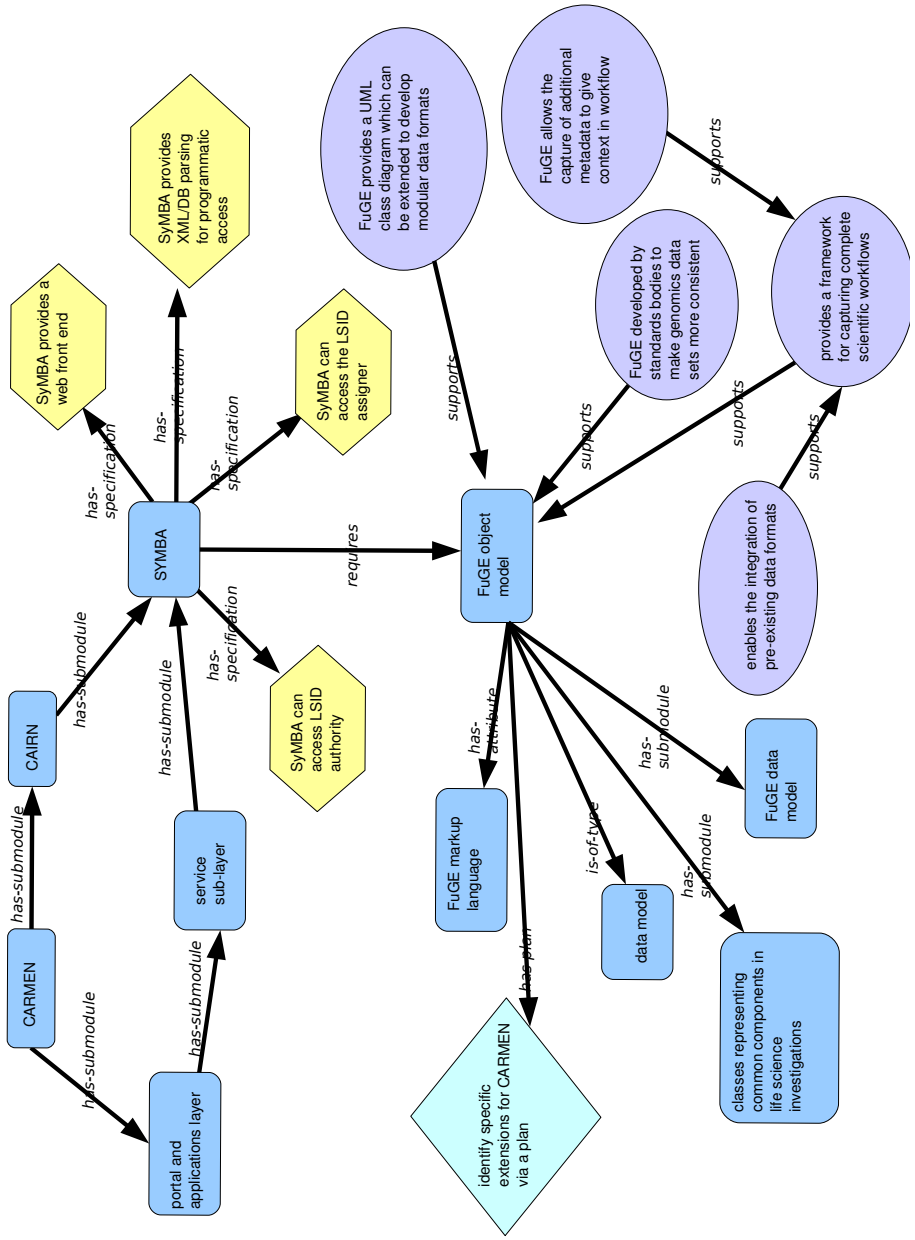

Figure D.2: Requirements and design entities related to the entity representing the component Symba

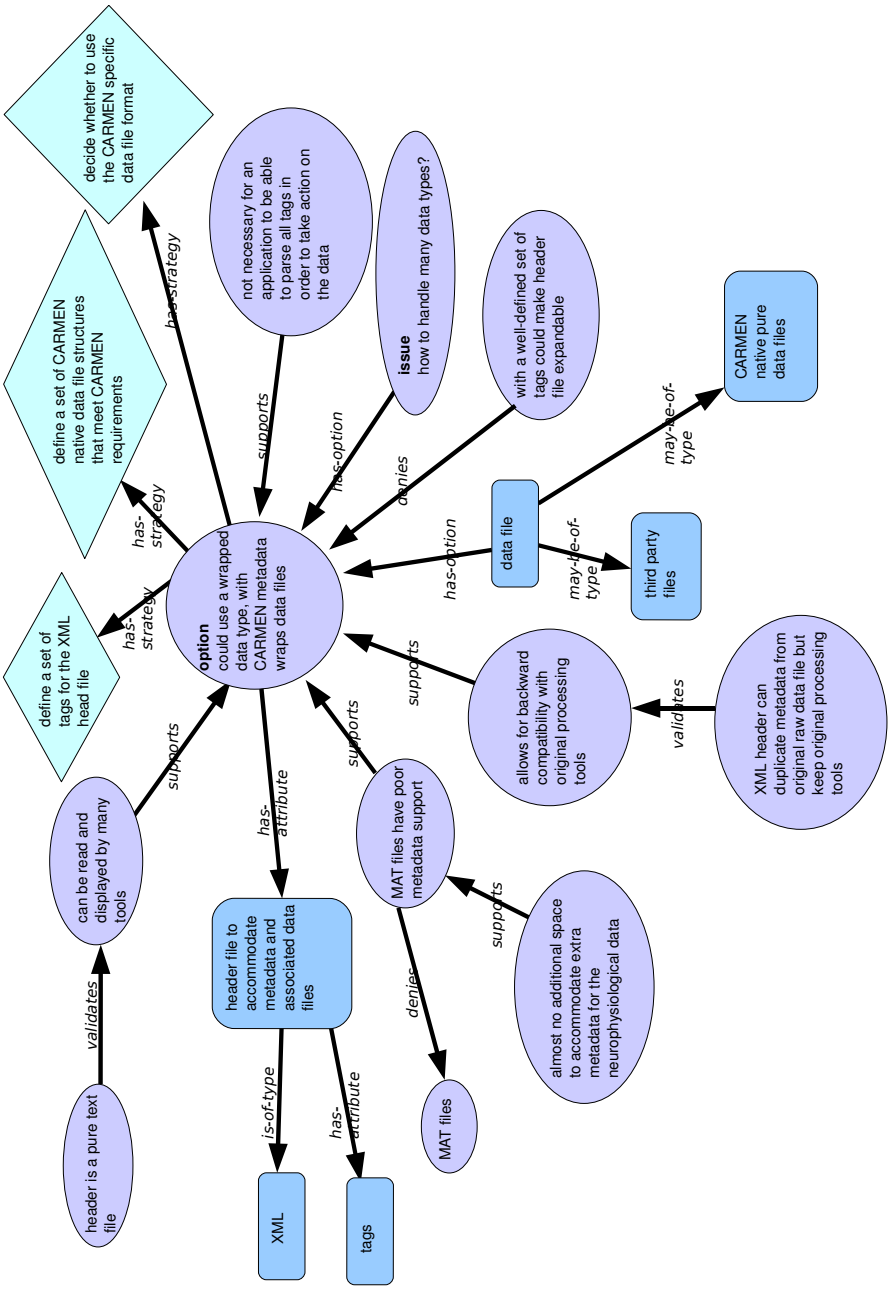

Figure D.3: Requirements and design entities related to the entity representing the claim that the header 'can be read and displayed by many tools' (part 1)

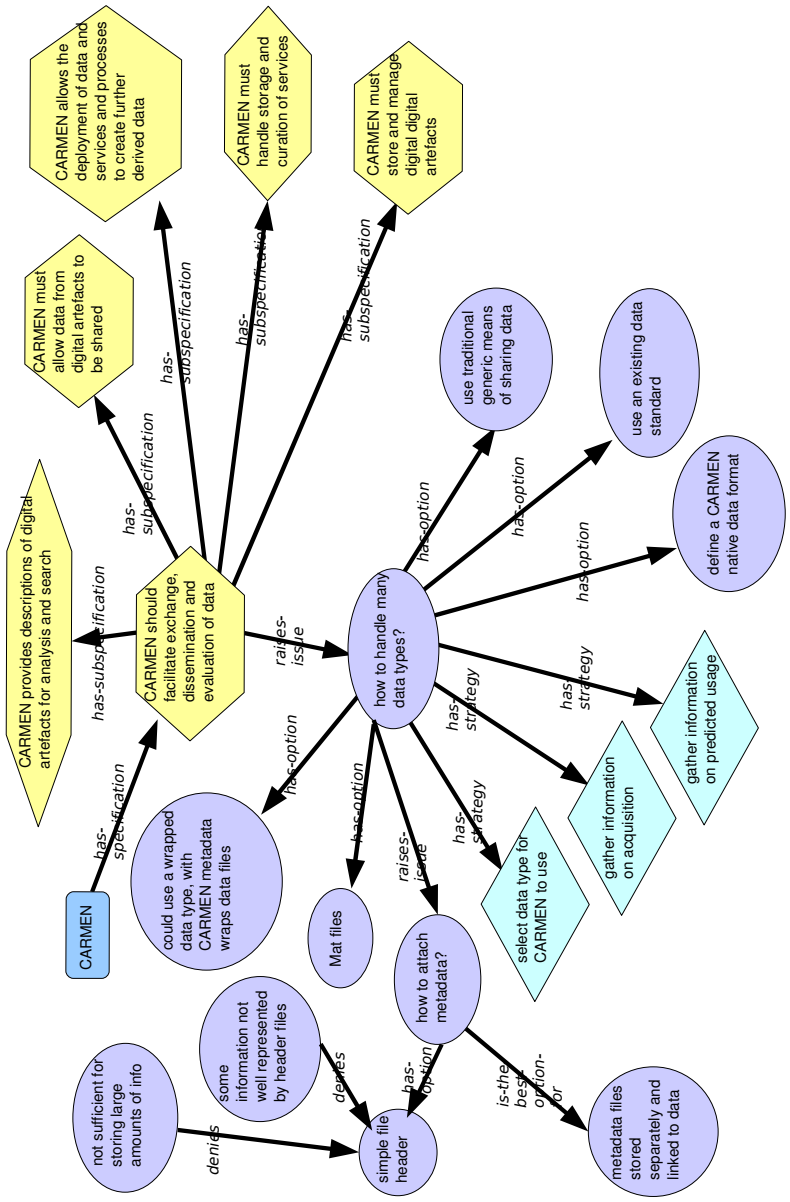

Figure D.4: Requirements and design entities related to the entity representing the claim that the header 'can be read and displayed by many tools' (part 2)

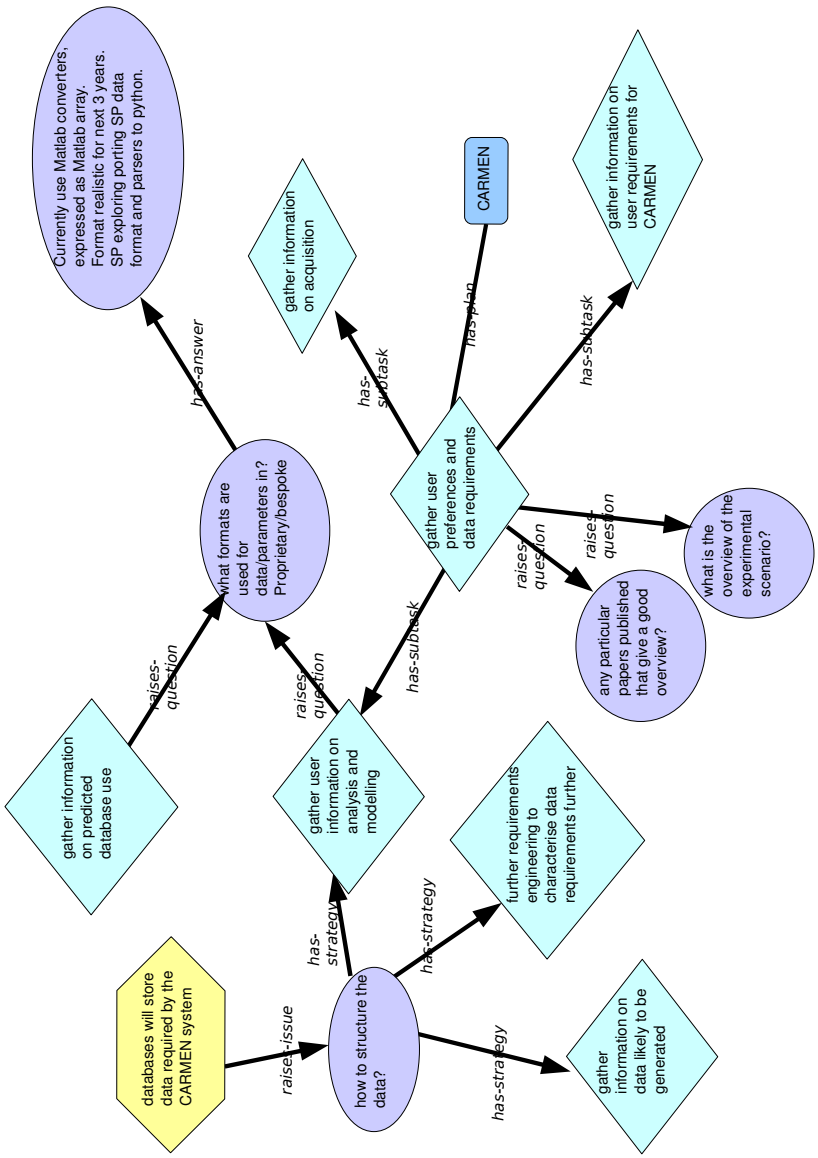

Figure D.5: Requirements and design entities related to the entity representing the claim 'Current use Matlab converters and format is expressed in a Matlab array...' (part 1)

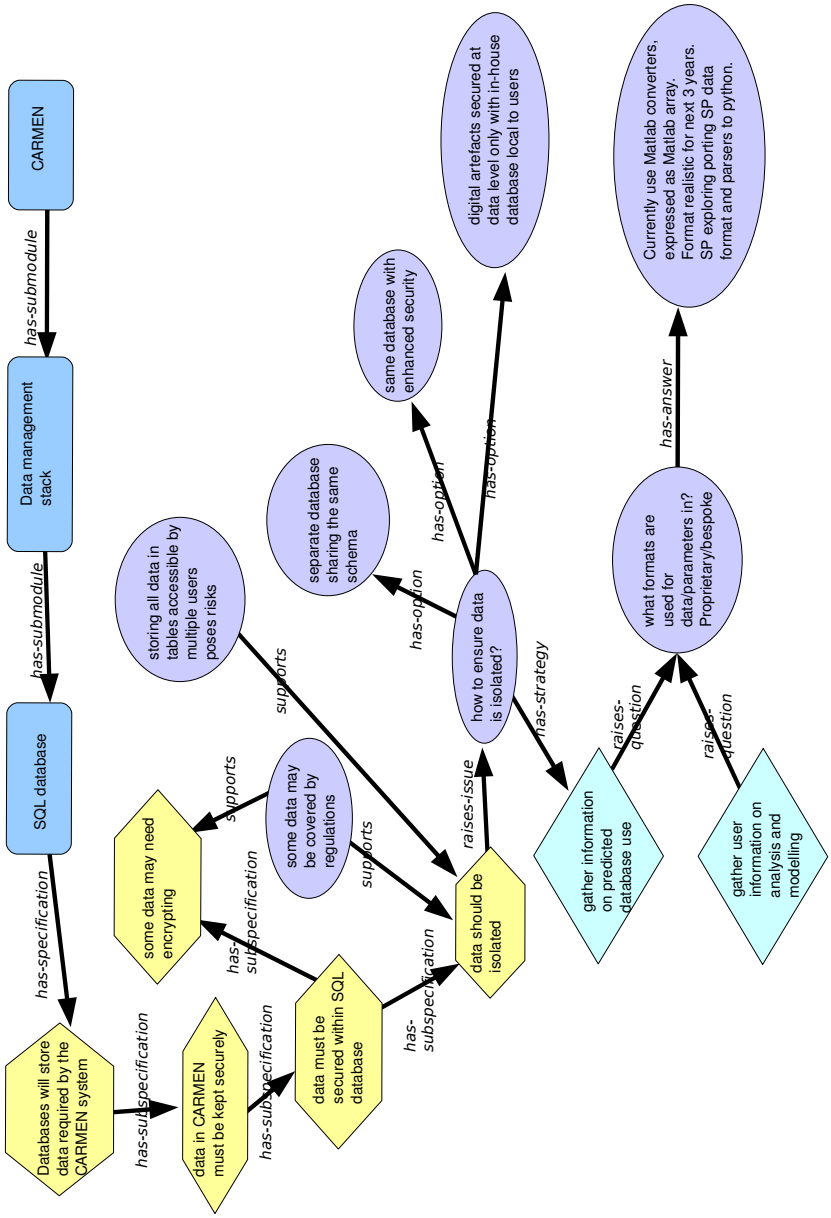

Figure D.6: Requirements and design entities related to the entity representing the claim 'Current use Matlab converters and format is expressed in a Matlab array...' (part 2)

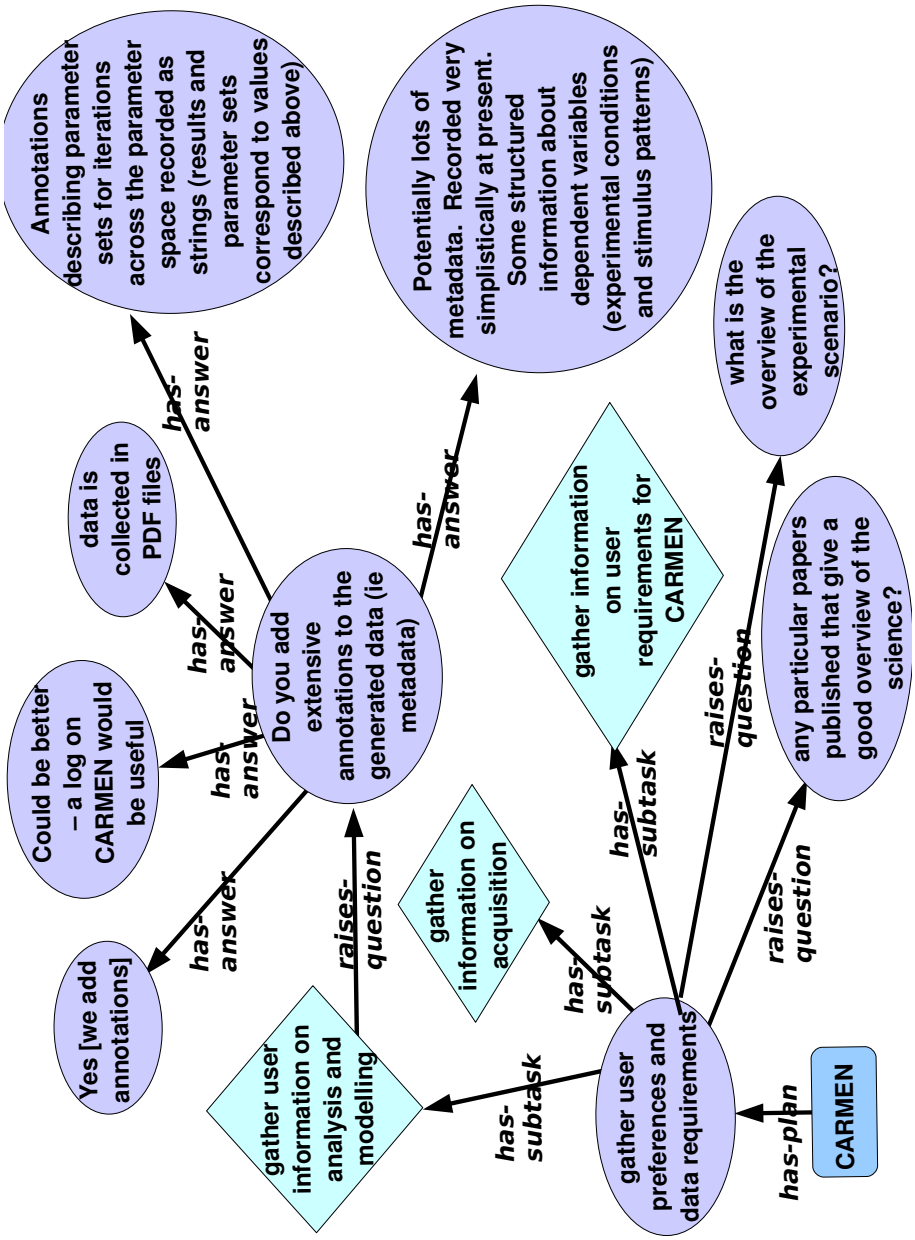

Figure D.7: Requirements and design entities related to the entity representing the claim 'Potentially lots of metadata...'

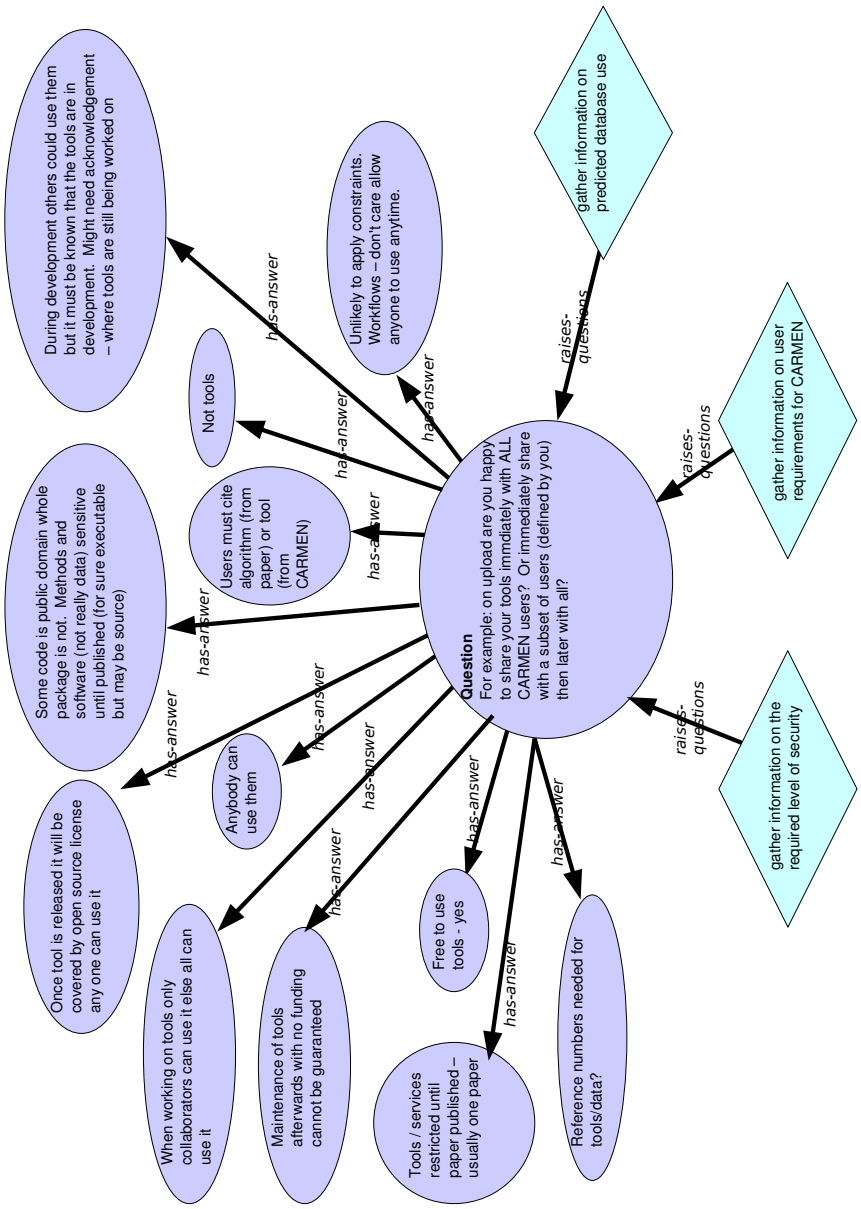

Figure D.8: Requirements and design entities related to the entity representing the claim 'Users must cite algorithm'

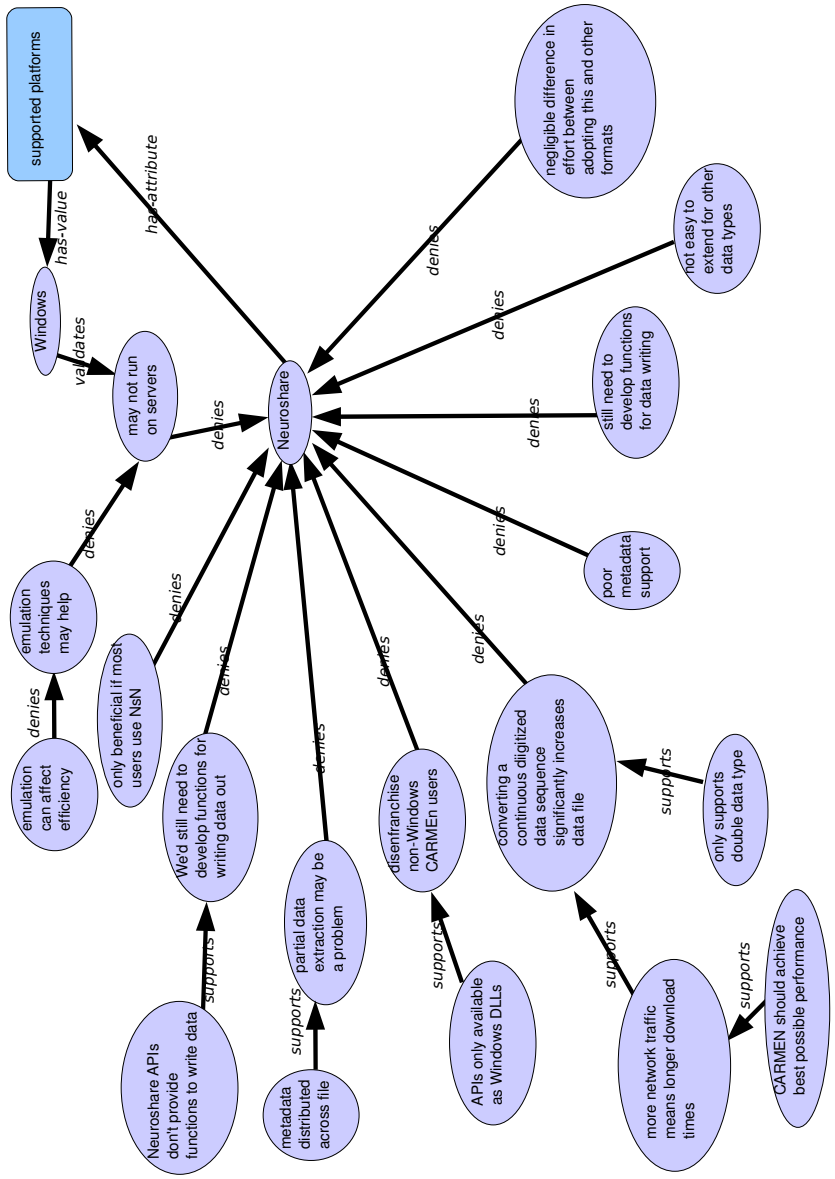

Figure D.9: Requirements and design entities related to the entity representing the claim 'Neuroshare APIs don't provide functions to write data' (part 1)

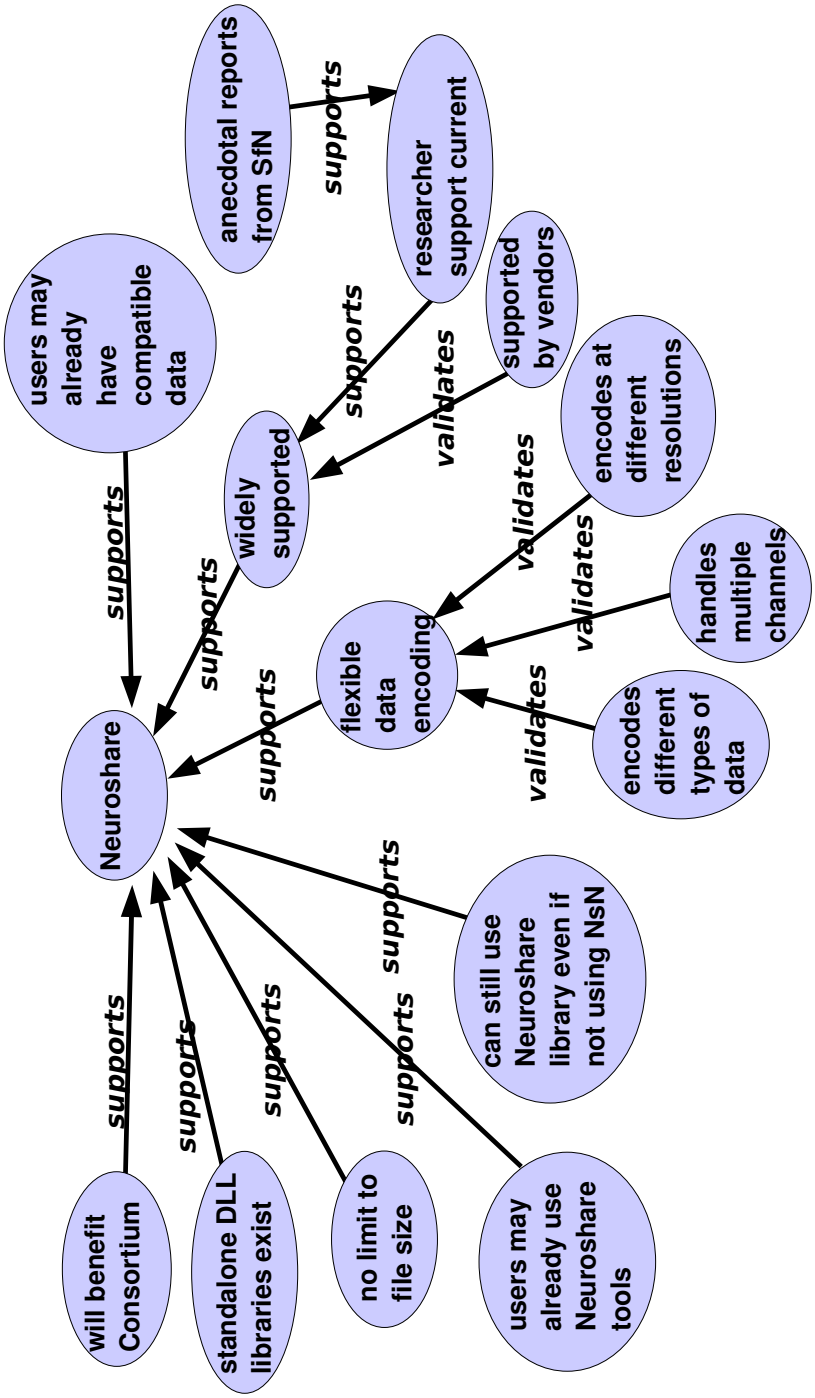

Figure D.10: Requirements and design entities related to the entity representing the claim 'Neuroshare APIs don't provide functions to write data' (part 2)

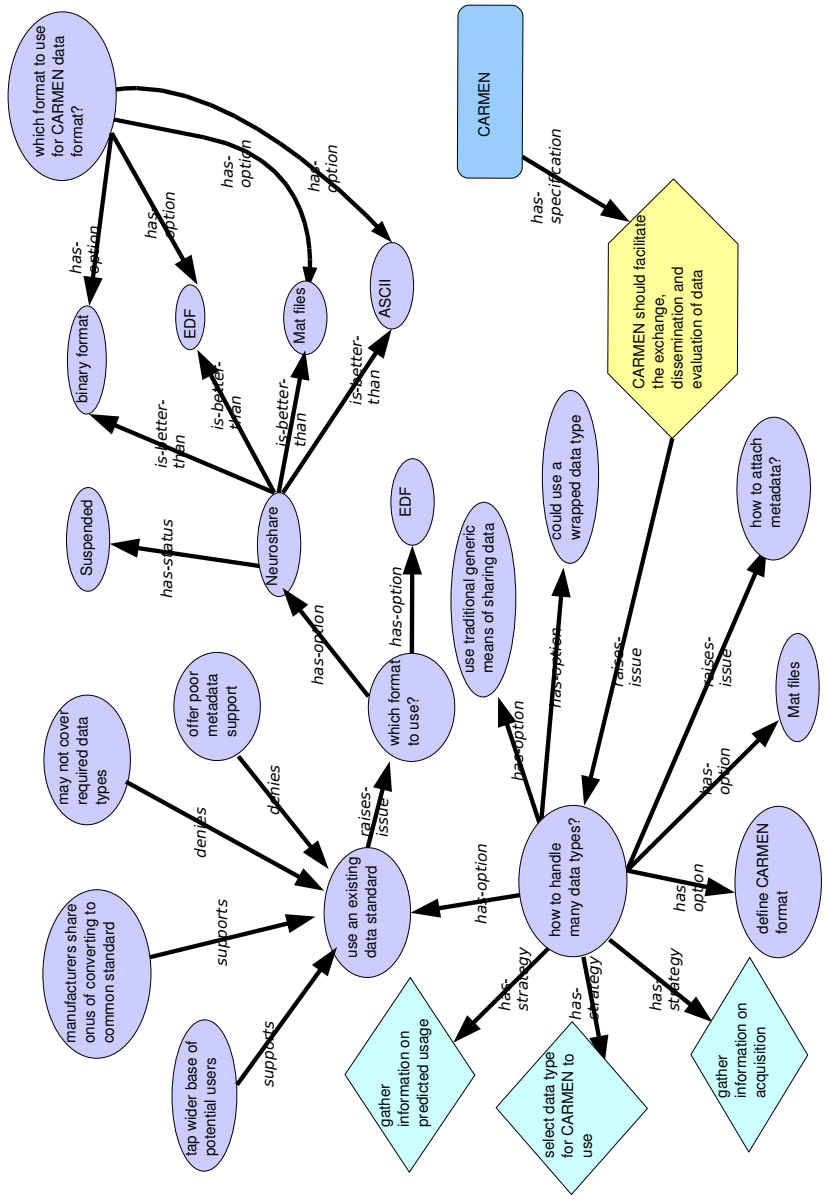

Figure D.11: Requirements and design entities related to the entity representing the claim 'Neuroshare APIs don't provide functions to write data' (part 3)

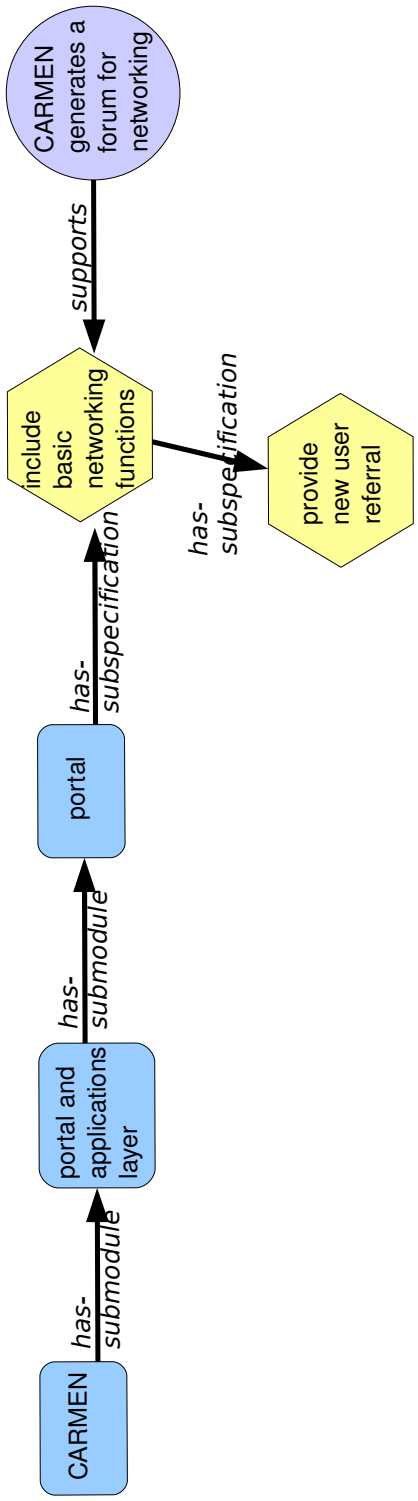

Figure D.12: Requirements and design entities related to the entity representing the claim 'Provide user referral'

Appendix E

Normality Tests on Input Data

Presented below are detailed outputs from Anderson-Darling normality tests conducted on existing metrics.

The test generates a *normal probability plot*, which plots the sample data versus the values we would expect to get if the data were normally distributed [133]. Data points from a normal distribution should follow the course of line closely; the curves shown in the graphs for our metrics is typical of non-normally distributed data. A p value is also generated (shown in the top right panel of each plot); this indicates the probability of a Type 1 error. A low value thus indicates that we reject the null hypothesis (the data is not normally distributed).

E.1 Metrics generated by CCCC

In this section we present the output generated by Minitab in carrying out Anderson-Darling normality tests on metrics generated by CCCC.

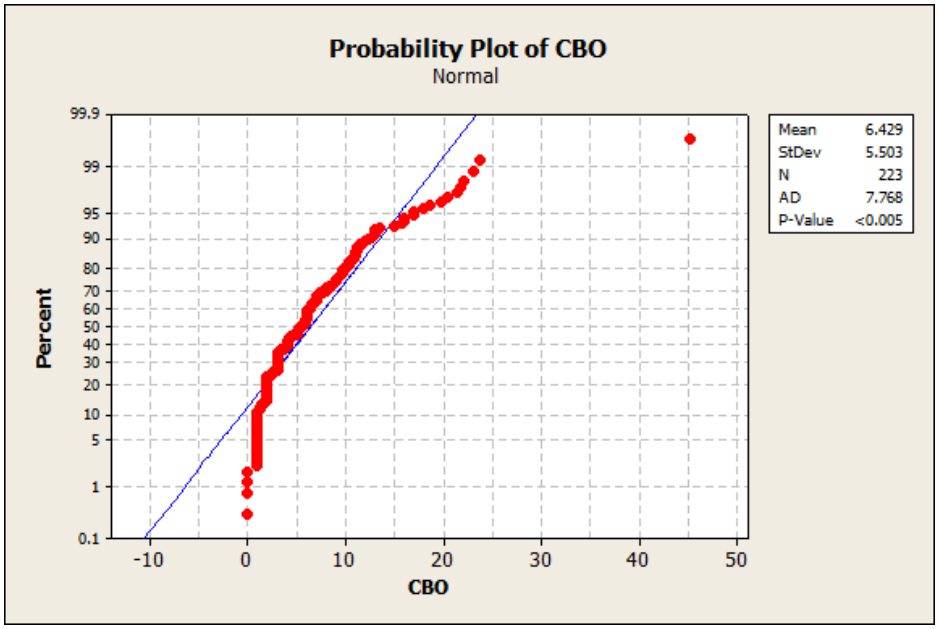

Figure E.1: An Anderson-Darling normality test on CBO

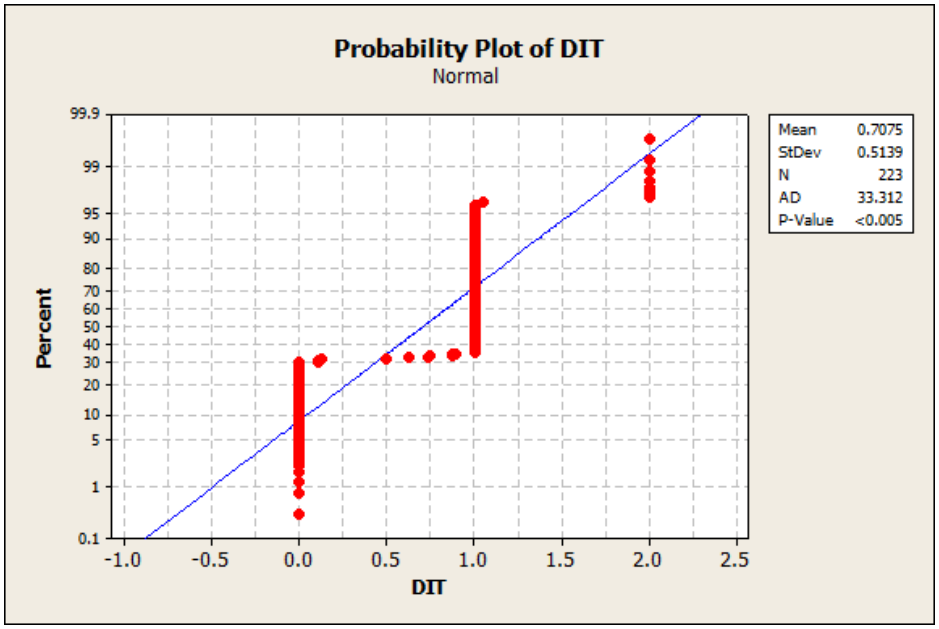

Figure E.2: An Anderson-Darling normality test on DIT

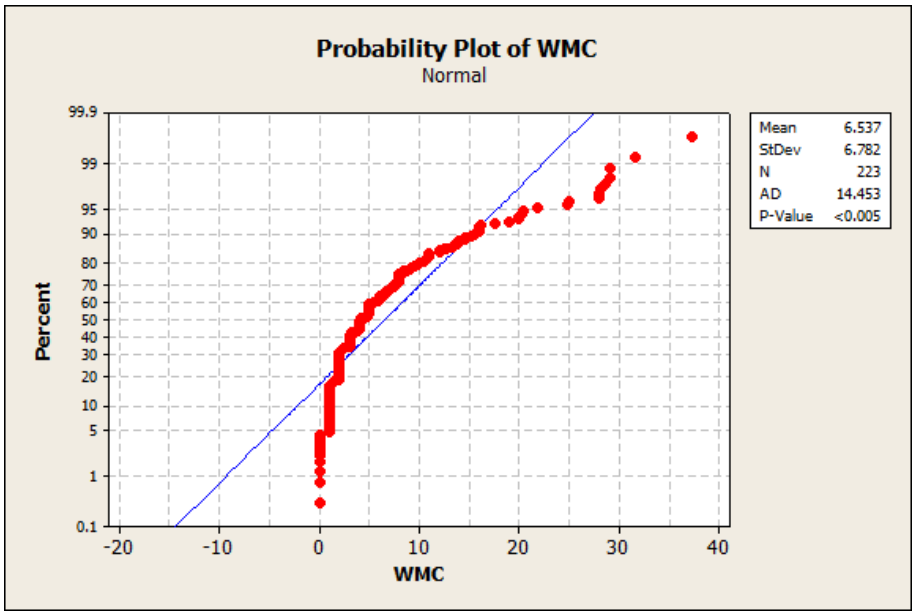

Figure E.3: Results of an Anderson Darling Normality Test on WMC

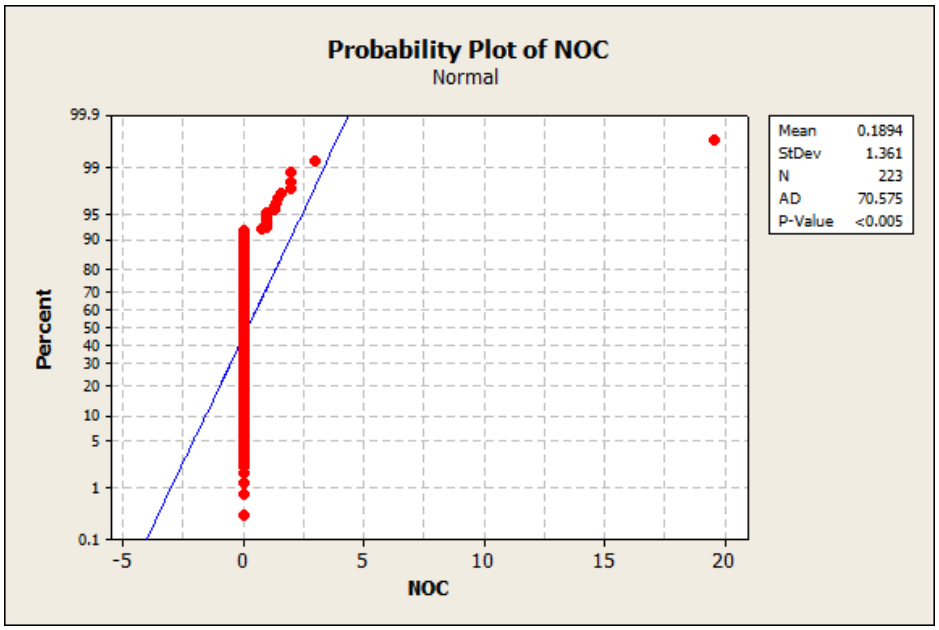

Figure E.4: An Anderson-Darling normality test on NOC

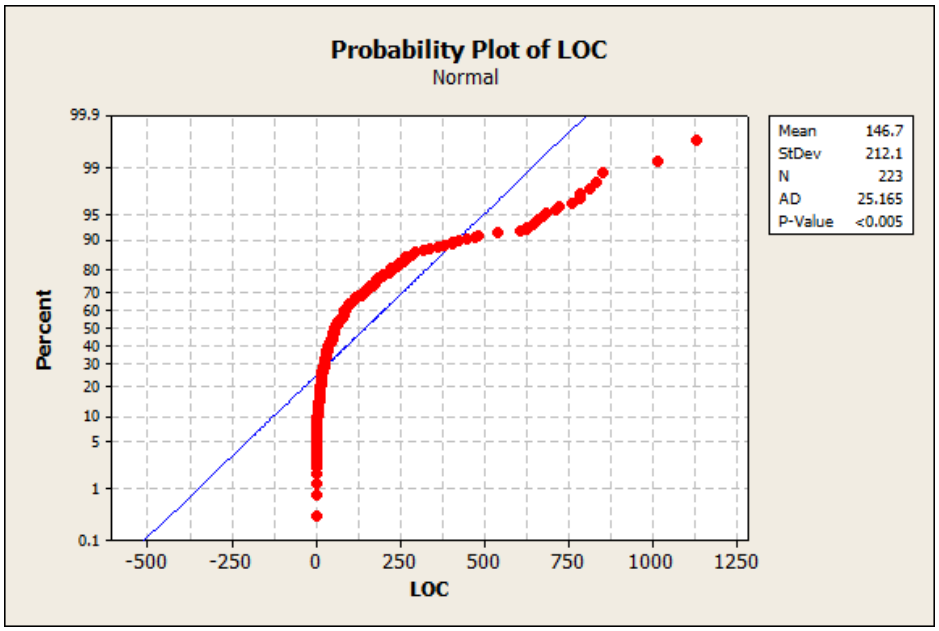

Figure E.5: An Anderson-Darling normality test on lines of code

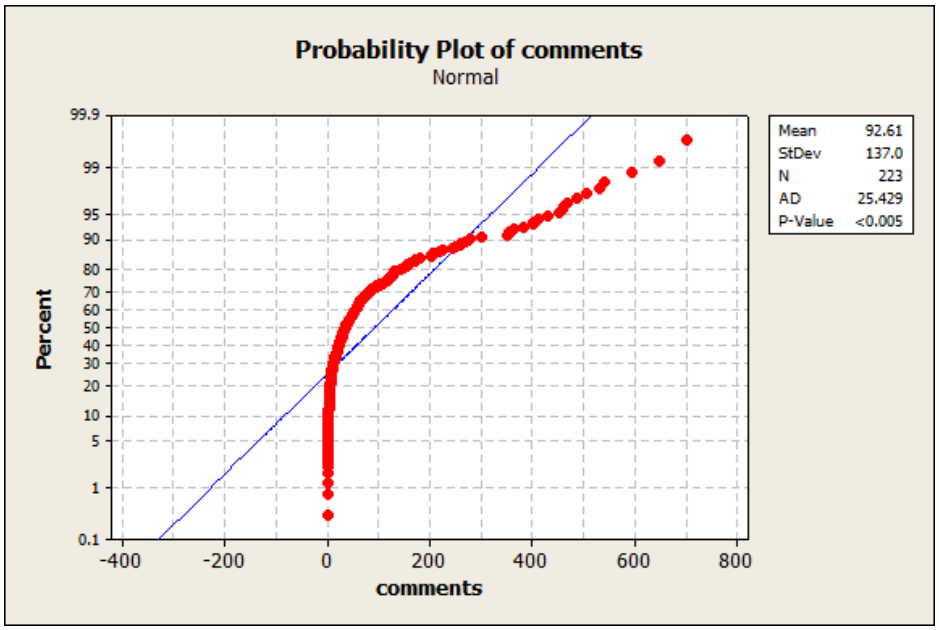

Figure E.6: An Anderson-Darling normality test on lines of code

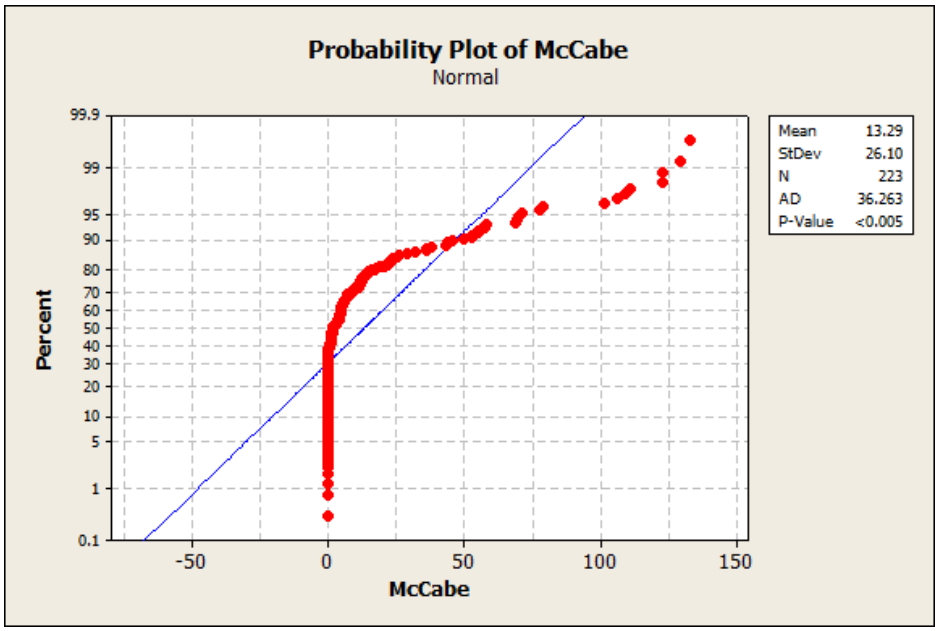

Figure E.7: An Anderson-Darling normality test on McCabe's cyclomatic complexity

E.2 Our metrics

In this section we present the output generated by Minitab in carrying out Anderson-Darling normality tests on values achieved for our metrics.

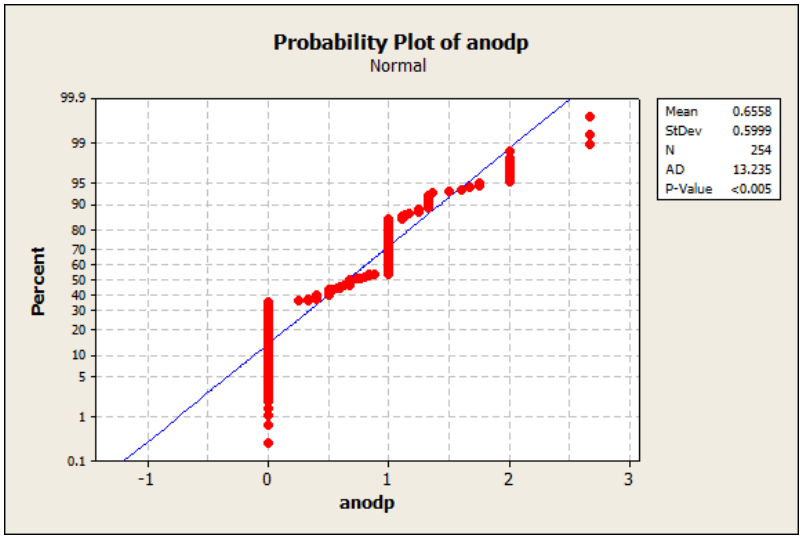

Figure E.8: An Anderson-Darling normality test on ANODP

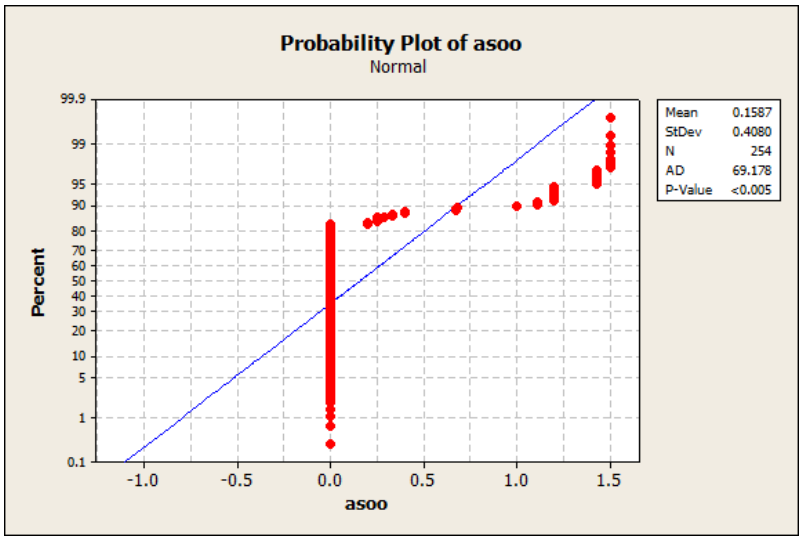

Figure E.9: An Anderson-Darling normality test on ASOO

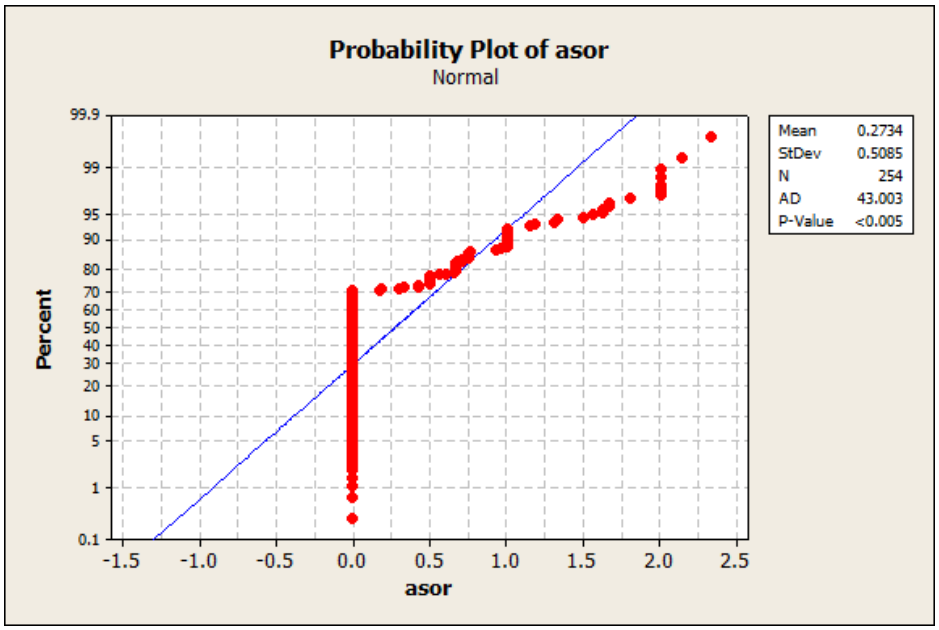

Figure E.10: An Anderson-Darling normality test on ASOR

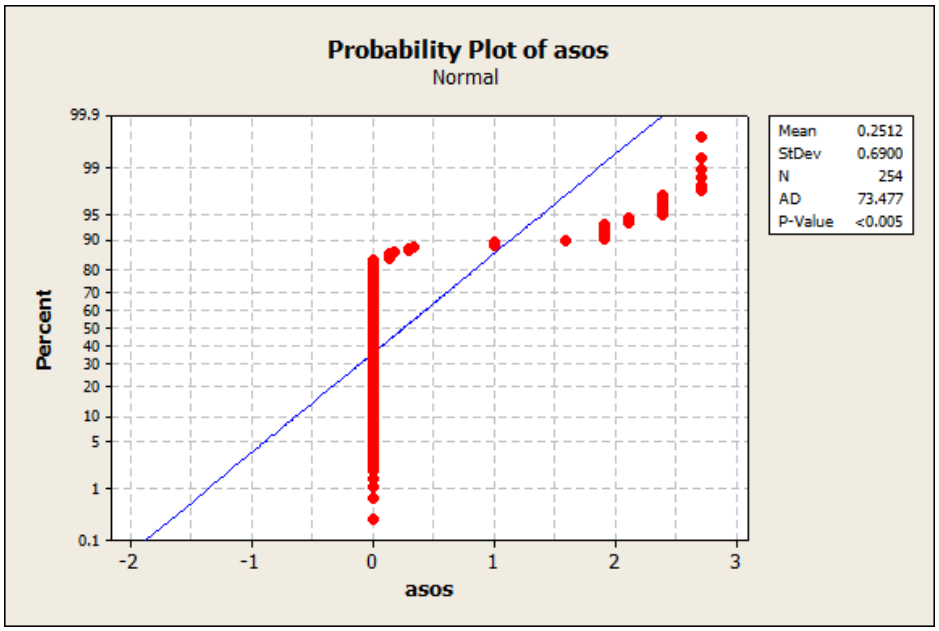

Figure E.11: An Anderson-Darling normality test on ASOS

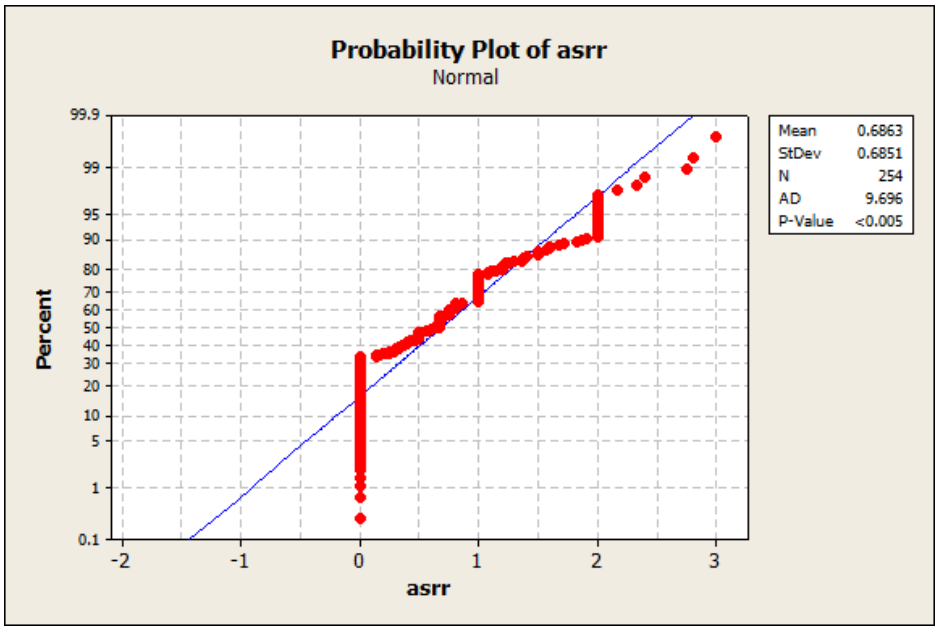

Figure E.12: An Anderson-Darling normality test on ASRR

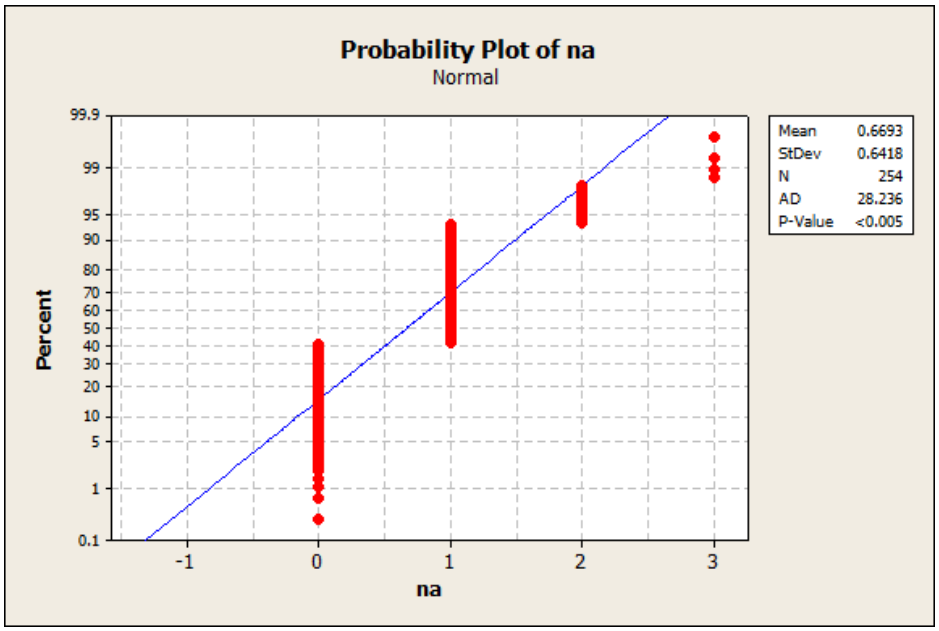

Figure E.13: An Anderson-Darling normality test on NA

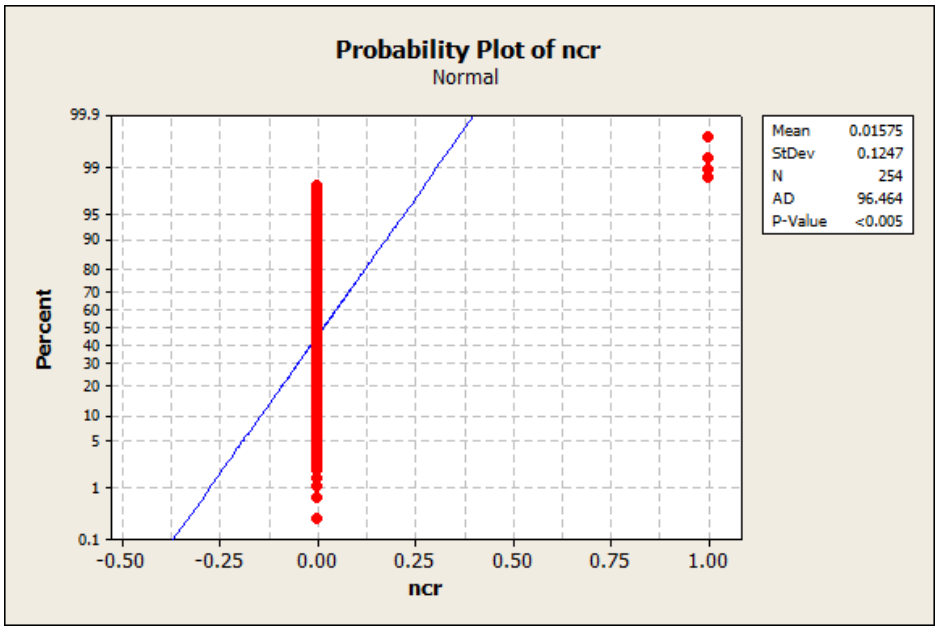

Figure E.14: An Anderson-Darling normality test on NCR

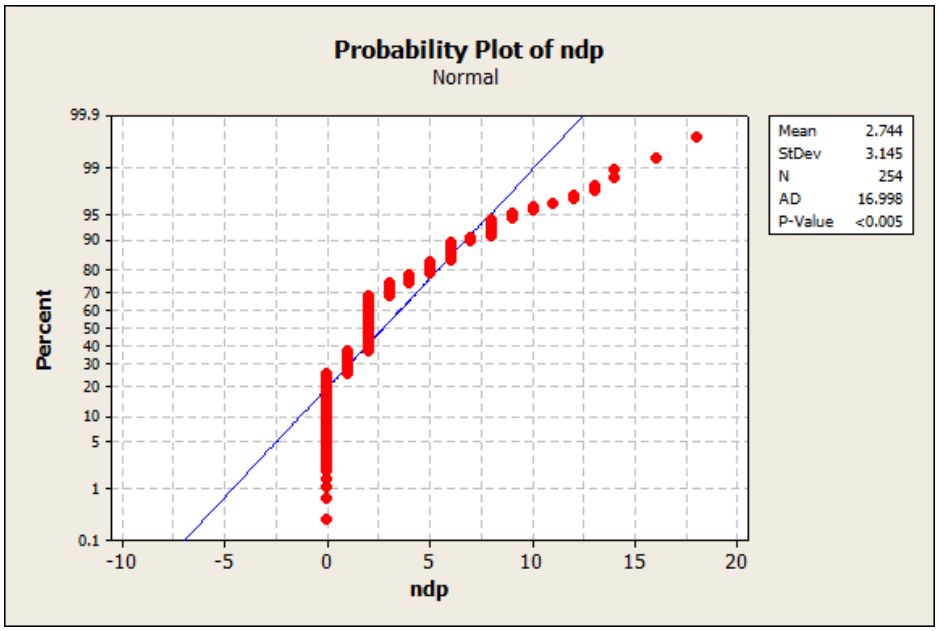

Figure E.15: An Anderson-Darling normality test on NDP

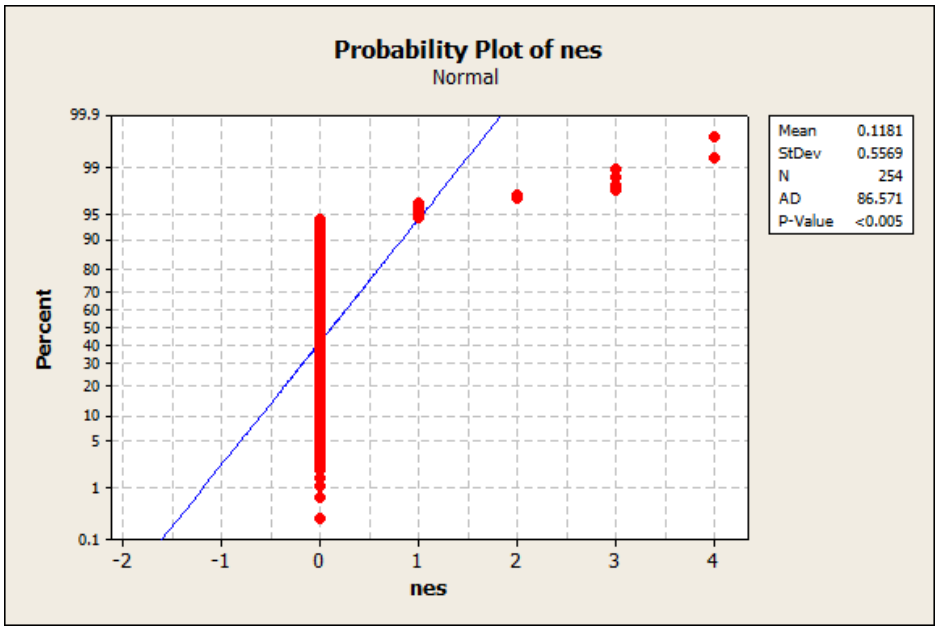

Figure E.16: An Anderson-Darling normality test on NES

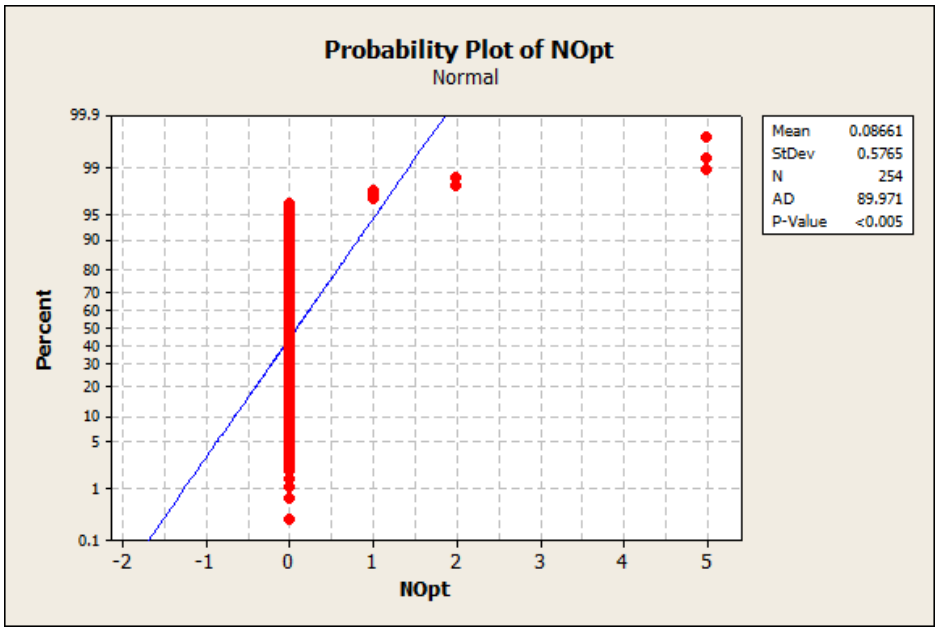

Figure E.17: An Anderson-Darling normality test on NOpt

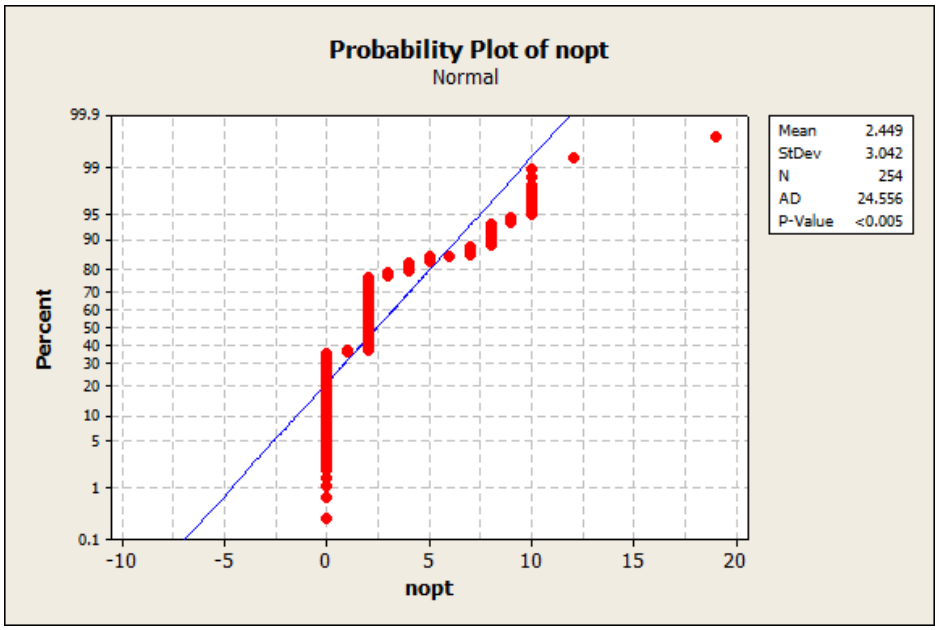

Figure E.18: An Anderson-Darling normality test on NOut

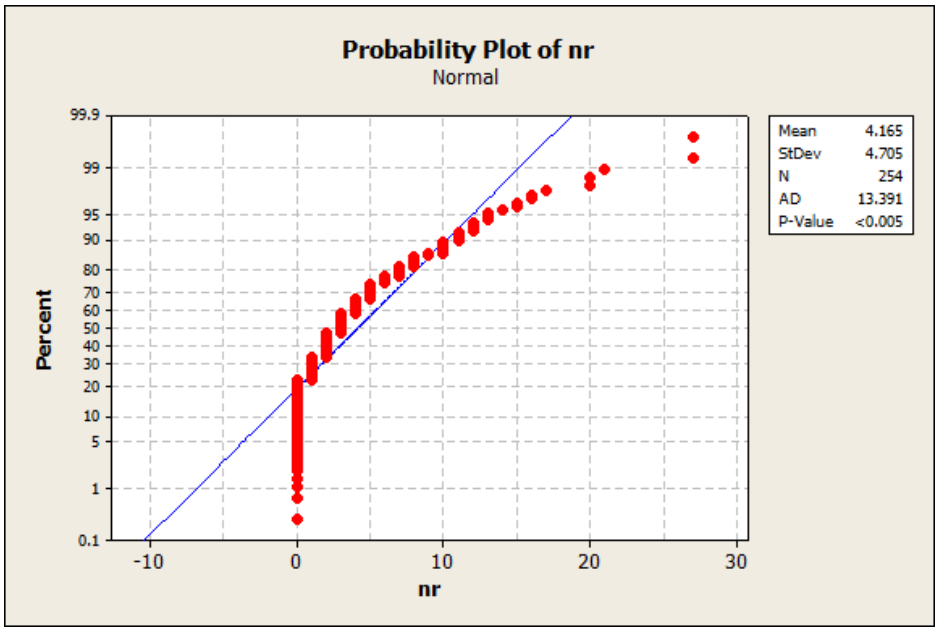

Figure E.19: An Anderson-Darling normality test on NR

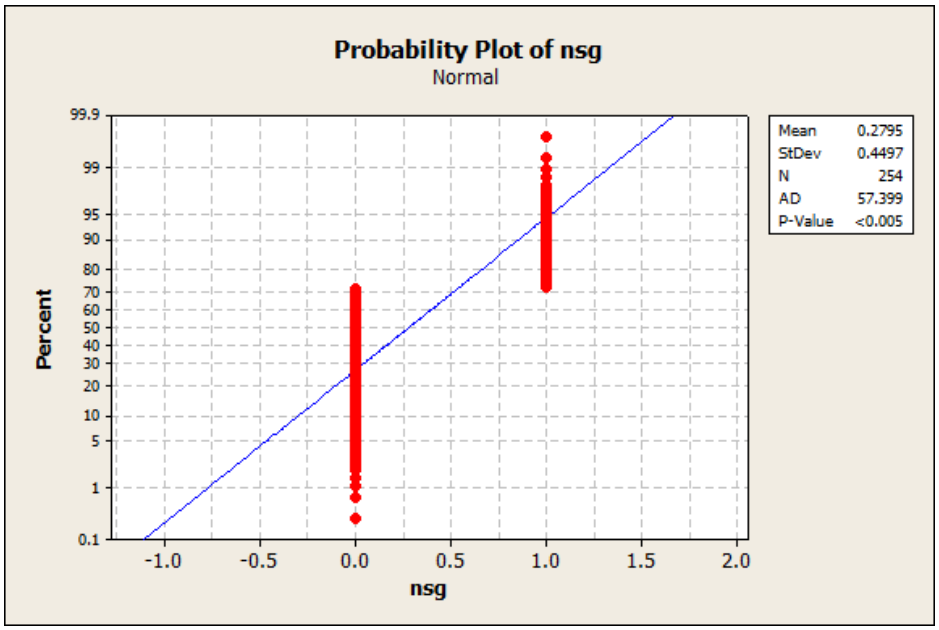

Figure E.20: An Anderson-Darling normality test on NSG

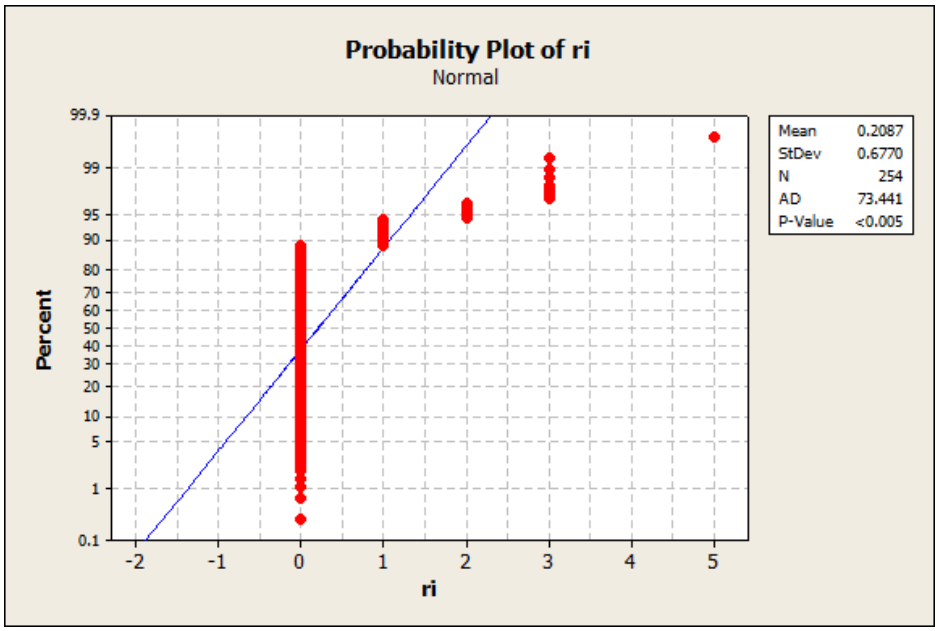

Figure E.21: An Anderson-Darling normality test on RI

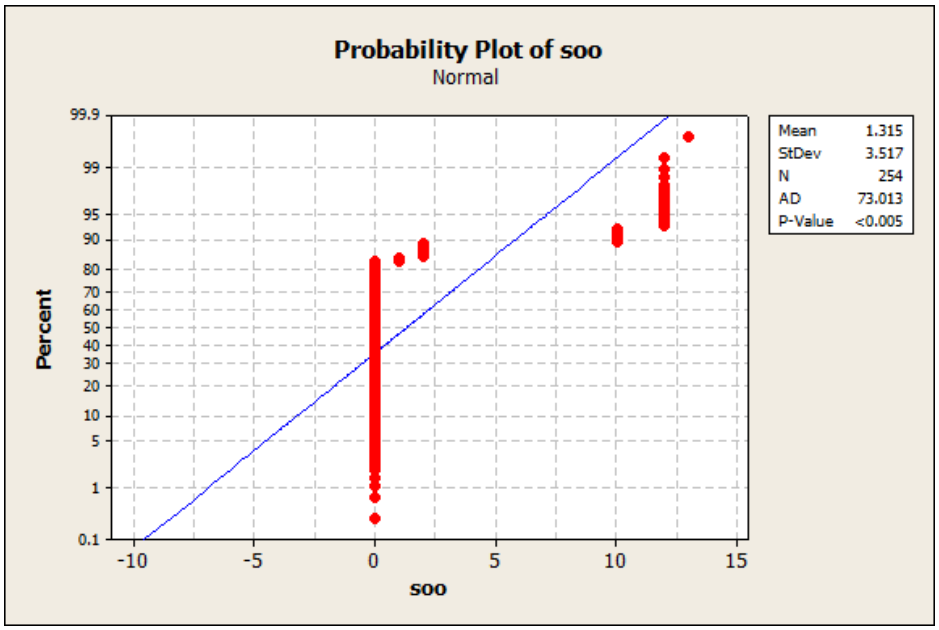

Figure E.22: An Anderson-Darling normality test on SOO

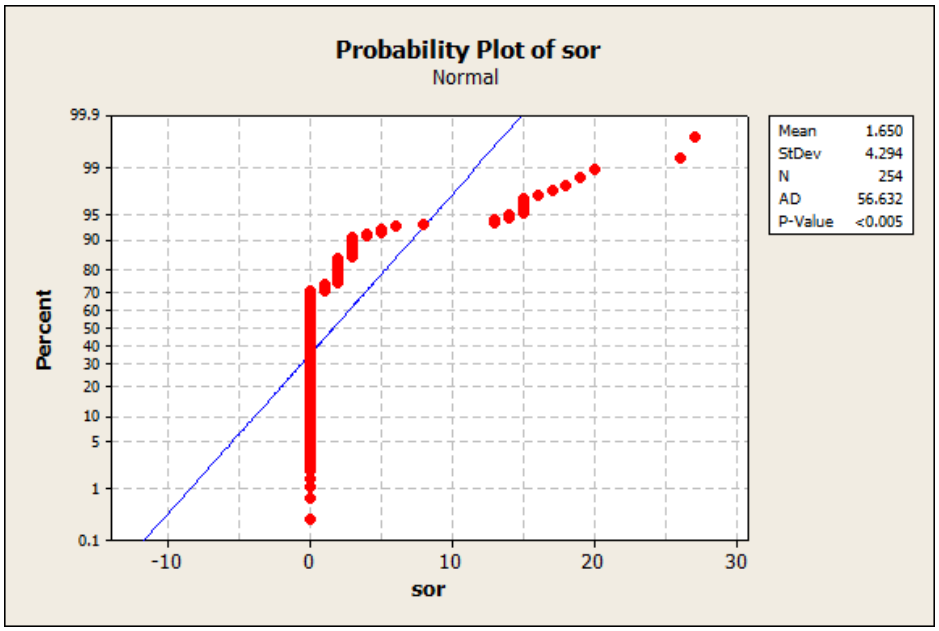

Figure E.23: An Anderson-Darling normality test on SOR

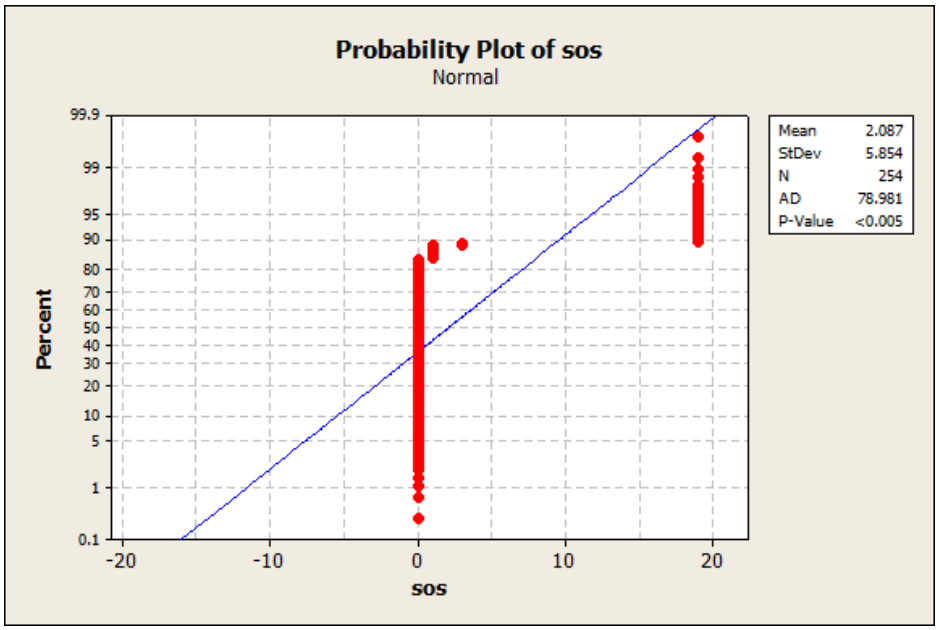

Figure E.24: An Anderson-Darling normality test on SOS

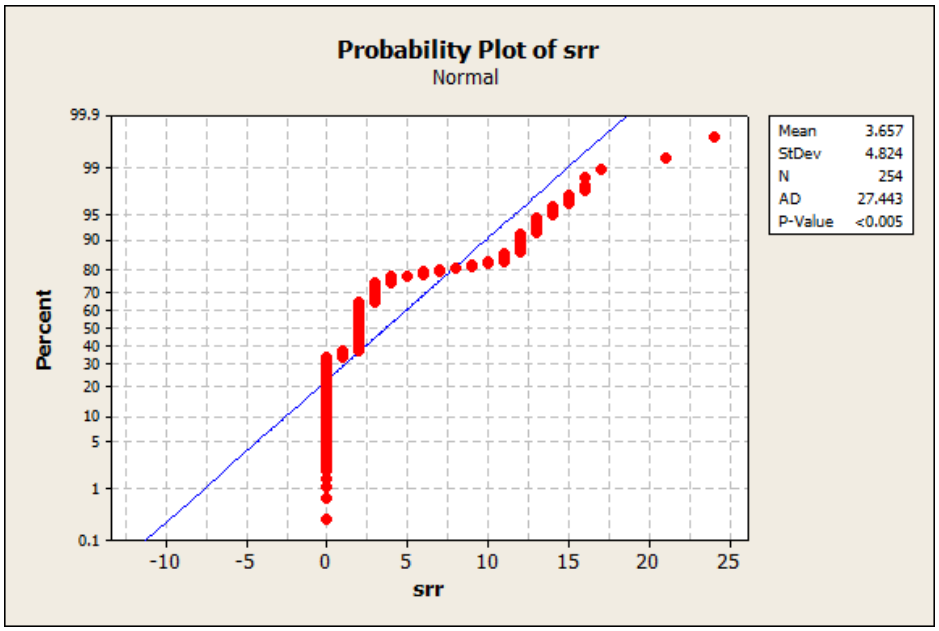

Figure E.25: An Anderson-Darling normality test on SRR

Appendix F

Multiple regression on existing metrics

In this section we present the output generated by Minitab during multiple regression tests performed on existing complexity metrics.

F.1 Metrics generated by CCCC

The stepwise alogorithm included four of the predictors and then terminated, not finding any other predictors with a sufficiently low p value to justify inclusion.

F.1.1 Stepwise regression

```
Stepwise Regression: changes versus CBO, DIT, ...

  Alpha-to-Enter: 0.15  Alpha-to-Remove: 0.15


Response is changes on 7 predictors, with N = 223


Step            1       2       3       4
Constant    6.728   3.967   2.528   2.386

comments   0.0556  0.0413  0.0249  0.0429
T-Value      8.78    5.52    2.73    3.50
P-Value     0.000   0.000   0.007   0.001

CBO                  0.64    0.57    0.60
T-Value              3.42    3.08    3.29
P-Value             0.001   0.002   0.001

WMC                          0.52    0.49
T-Value                      3.03    2.83
```

```
P-Value                       0.003   0.005

McCabe                                -0.116
T-Value                                -2.17
P-Value                                0.031

S             12.9    12.6    12.4    12.3
R-Sq         25.84   29.59   32.41   33.84
R-Sq(adj)    25.50   28.95   31.49   32.63
Mallows Cp    23.6    13.4     6.1     3.4


Regression Analysis: changes versus comments, CBO, WMC, McCabe

The regression equation is
changes = 2.39 + 0.0429 comments + 0.603 CBO + 0.486 WMC -
                                                    0.116 McCabe


Predictor      Coef  SE Coef      T      P
Constant      2.386    1.354   1.76  0.079
comments    0.04294  0.01228   3.50  0.001
CBO          0.6030   0.1832   3.29  0.001
WMC          0.4856   0.1715   2.83  0.005
McCabe     -0.11577  0.05330  -2.17  0.031

S = 12.2882   R-Sq = 33.8%   R-Sq(adj) = 32.6%


Analysis of Variance

Source           DF       SS      MS      F      P
Regression        4  16840.5  4210.1  27.88  0.000
Residual Error  218  32918.0   151.0
Total           222  49758.5


Source    DF   Seq SS
comments   1  12857.8
CBO        1   1864.2
WMC        1   1406.0
McCabe     1    712.5
```

F.1.2 'Best subsets' regression

The first column ('Vars') indicates the number of predictors in the model, and the predictors themselves are indicated with an 'X' in one of the columns at the end of the row. In between are shown the model's R-Sq value, the adjusted R-Sq value, and the *Mallows Cp* figure.

```
Best Subsets Regression: changes versus CBO, DIT, ...

Response is changes
```

											c		
											o		
											m	M	
											m	c	
											e	C	
					C	D	N	W	L	n	a		
			Mallows		B	I	O	M	O	t	b		
Vars	R-Sq	R-Sq(adj)	Cp	S	O	T	C	C	C	s	e		
1	25.8	25.5	23.6	12.922						X			
1	24.1	23.8	29.3	13.071				X					
2	30.1	29.5	11.6	12.572	X			X					
2	29.6	28.9	13.4	12.620	X					X			
3	32.4	31.5	6.1	12.392	X			X		X			
3	31.4	30.5	9.4	12.484	X					X	X		
4	33.8	32.6	3.4	12.288	X			X		X	X		
4	32.8	31.6	6.8	12.384	X			X	X		X		
5	34.1	32.6	4.6	12.293	X		X	X		X	X		
5	34.0	32.5	4.9	12.300	X			X	X	X	X		
6	34.3	32.4	6.0	12.305	X		X	X	X	X	X		
6	34.1	32.3	6.6	12.321	X	X	X	X		X	X		
7	34.3	32.1	8.0	12.333	X	X	X	X	X	X	X		

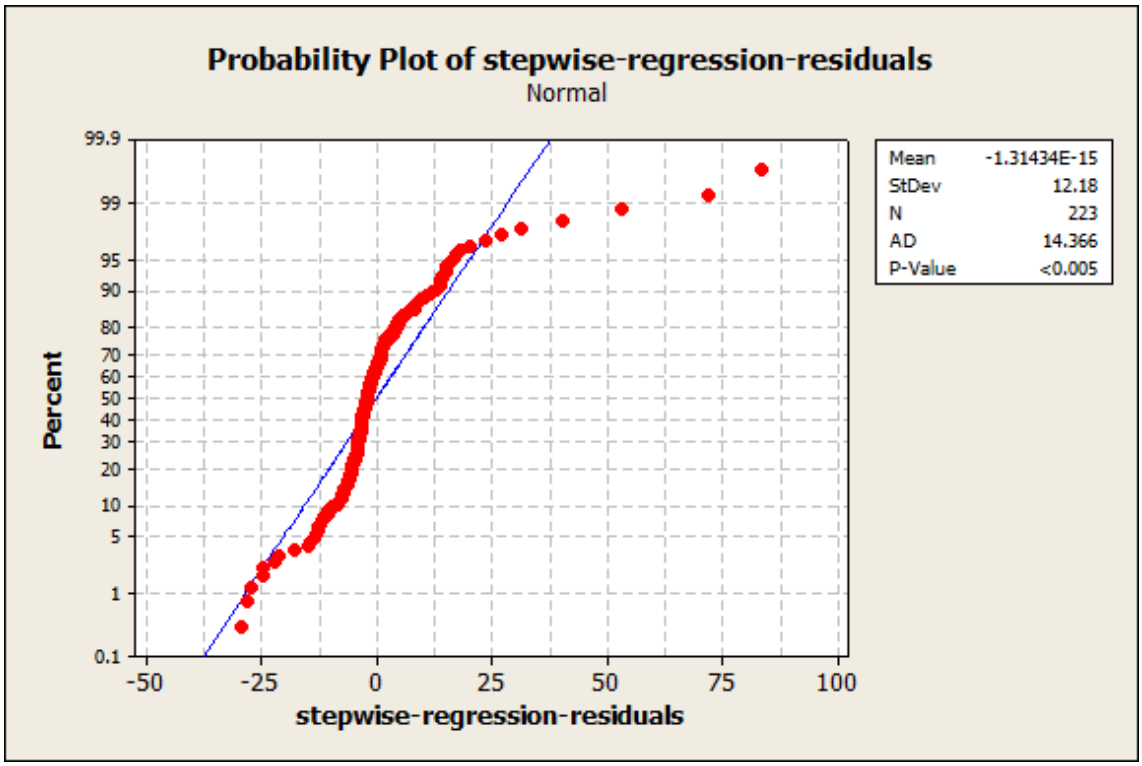

Figure F.1: Anderson-Darling normality test on residuals of a regression model using CBO, comments, WMC and MVG metrics generated by CCCC

Appendix G

Multiple regression analysis on our metrics

In this section we present the output generated by Minitab during multiple regression tests performed on our metrics.

G.1 Stepwise regression model

```
Stepwise Regression: changes versus nr, nsg, ...

  Alpha-to-Enter: 0.15  Alpha-to-Remove: 0.15

Response is changes on 18 predictors, with N = 254
```

Step	1	2	3	4	5	6
Constant	4.203	4.323	4.330	3.084	2.790	2.747
nr	1.31	1.16	1.30	1.22	1.09	1.07
T-Value	7.53	6.46	6.76	6.24	5.19	5.25
P-Value	0.000	0.000	0.000	0.000	0.000	0.000
nes		4.2	4.9	5.3	1.0	
T-Value		2.77	3.16	3.38	0.32	
P-Value		0.006	0.002	0.001	0.750	
asos			-2.6	-2.8	-4.4	-4.6
T-Value			-1.94	-2.15	-2.73	-3.09
P-Value			0.054	0.032	0.007	0.002
asrr				2.3	2.1	2.0
T-Value				1.92	1.69	1.66
P-Value				0.056	0.093	0.098
nout					0.81	0.94
T-Value					1.68	3.78

P-Value					0.093	0.000
S	13.0	12.9	12.8	12.7	12.7	12.7
R-Sq	18.38	20.80	21.97	23.11	23.98	23.95
R-Sq(adj)	18.05	20.17	21.03	21.87	22.44	22.72
Mallows Cp	10.2	4.5	2.8	1.2	0.4	-1.5

G.1.1 'Best subsets' regression model

```
Best Subsets Regression: changes versus nr, nsg, ...

Response is changes

                                                                a
                                              a     n   a     n n   a   a
                                          n s s n   o s s n   o o s s s s n
                       Mallows          n s r r c r u o o d n p d o o o o e
Vars  R-Sq  R-Sq(adj)     Cp       S  r g r r r i t r r p a t p s s o o s
   1  18.4       18.1     10.2  13.038  X
   1  15.3       15.0     20.1  13.282                    X
   2  20.8       20.2      4.5  12.869  X                                 X
   2  20.5       19.8      5.6  12.897  X           X
   3  23.1       22.2     -0.8  12.705  X           X               X
   3  22.9       21.9     -0.1  12.724  X           X                   X
   4  23.9       22.7     -1.5  12.661  X     X     X               X
   4  23.9       22.7     -1.3  12.666  X     X     X                   X
   5  24.3       22.8     -0.7  12.656  X X   X     X               X
   5  24.3       22.7     -0.5  12.660  X X   X     X                   X
   6  24.5       22.7      0.6  12.663  X X   X     X             X X
   6  24.5       22.7      0.7  12.666  X X   X     X                 X X
   7  24.8       22.6      1.9  12.668  X X   X     X               X X X
   7  24.8       22.6      1.9  12.669  X X   X     X             X X   X
   8  25.0       22.5      3.2  12.678  X X   X     X         X     X X X
   8  25.0       22.5      3.3  12.678  X X   X   X X               X X X
   9  25.2       22.5      4.4  12.681  X X   X     X         X X   X X X
   9  25.2       22.4      4.5  12.685  X X   X X X X       X     X X
  10  25.5       22.4      5.6  12.686  X X   X X X X       X X   X X
  10  25.5       22.4      5.7  12.688  X X   X   X X         X X   X X X
  11  25.7       22.4      6.7  12.690  X X   X X X X       X X X X X
  11  25.7       22.4      6.8  12.690  X     X X X X       X X X   X X X
  12  26.0       22.3      7.9  12.694  X X   X X X X       X X X   X X X
  12  26.0       22.3      8.1  12.698  X X   X X X X       X X X X X   X
  13  26.1       22.1      9.6  12.712  X X   X X X X X     X X X   X X X
  13  26.1       22.1      9.7  12.714  X X   X X X X       X X X X X X X
  14  26.2       21.9     11.4  12.732  X X   X X X X X X   X X X   X X X
  14  26.2       21.8     11.4  12.732  X X   X X X X X X   X X X X X   X
  15  26.3       21.6     13.0  12.749  X X   X X X X X X   X X X X X X X
  15  26.2       21.6     13.3  12.756  X X   X X X X X X   X X X   X X X X
  16  26.3       21.3     15.0  12.776  X X   X X X X X X   X X X X X X X X
  16  26.3       21.3     15.0  12.776  X X X X X X X X X   X X X X X X X
  17  26.3       21.0     17.0  12.802  X X X X X X X X X   X X X X X X X X
  17  26.3       21.0     17.0  12.803  X X   X X X X X X X X X X X X X X X
  18  26.3       20.7     19.0  12.830  X X X X X X X X X X X X X X X X X X
```

Appendix H

Scatterplots

Presented below are scatterplots showing relationships between our metrics and LOC (generated by CCCC).

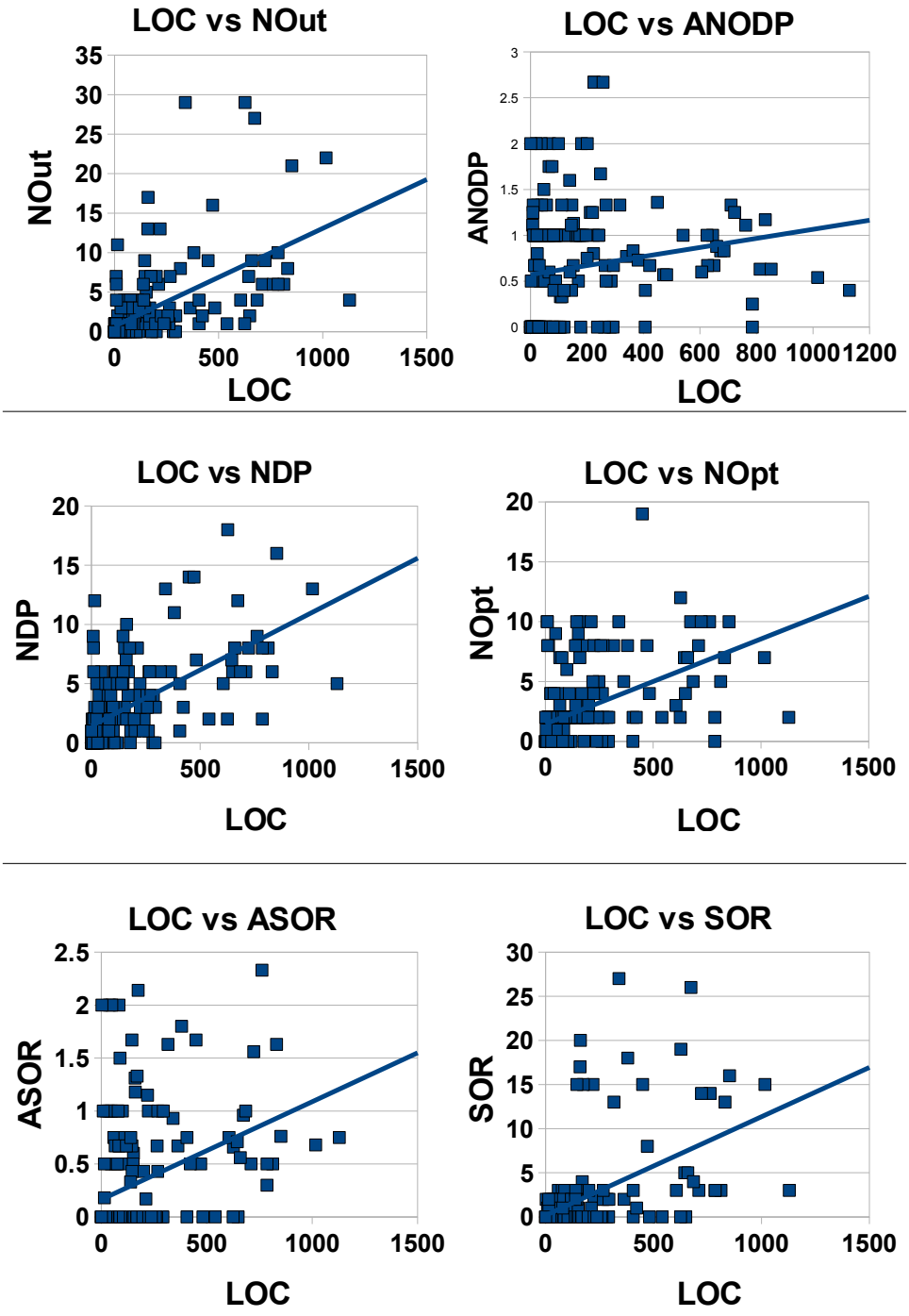

Figure H.1: Scattergraphs plotting our metrics against LOC (generated by CCCC). Graphs generated by Open Office Calc.

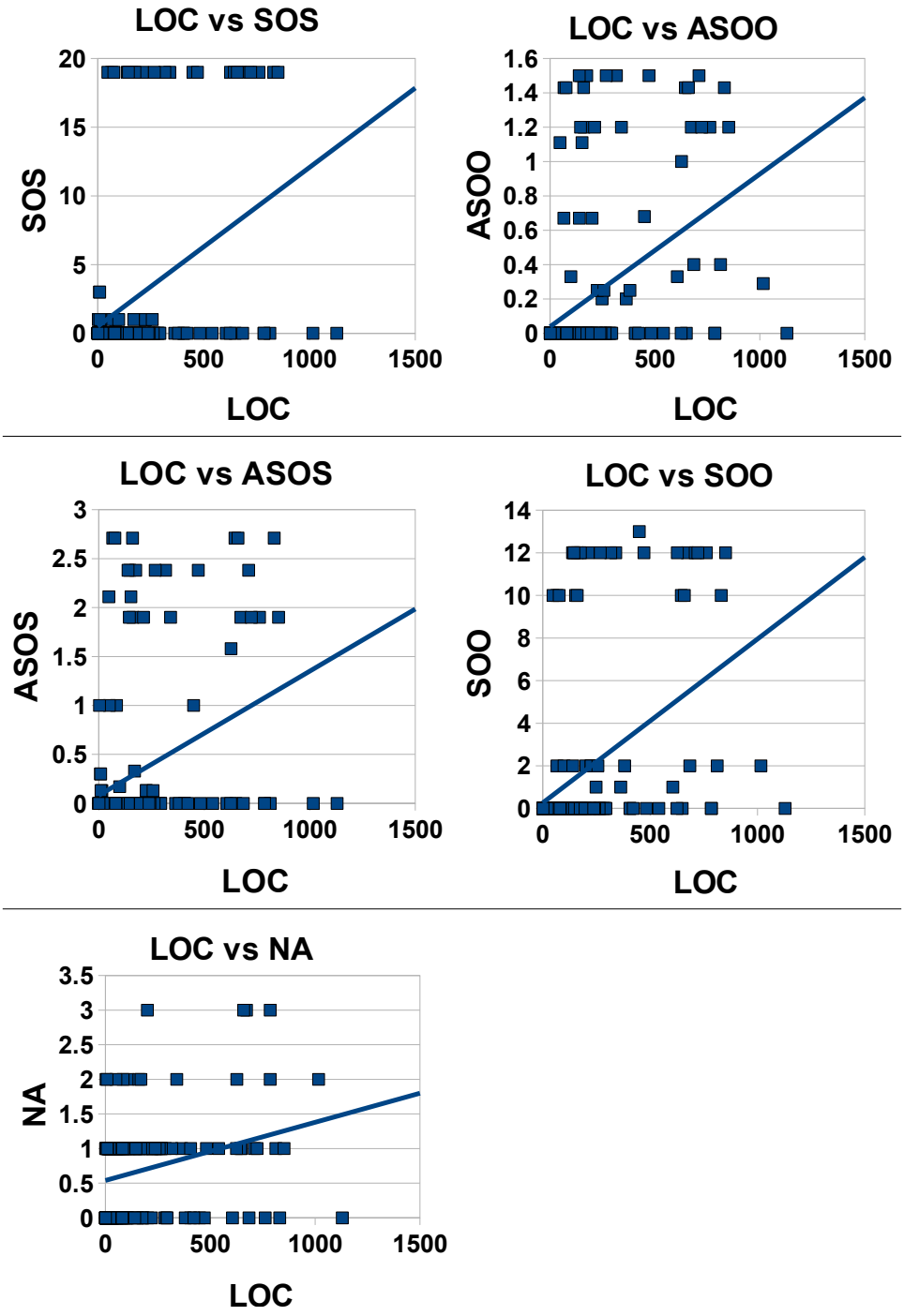

Figure H.2: Scattergraphs plotting our metrics against LOC (generated by CCCC). Graphs generated by Open Office Calc.

Appendix I

Evaluation of all metrics

The table below presents values calculated for all metrics in our study for: precision; recall; standard F-measure; precision-weighted F-measure; and recall-weighted F-measure (calculations are defined in Chapter 7).

Table I.1: Precision and recall values for all metrics

		Precision			**Recall**			**Standard F-measure**			**Precision Weighted F-measure**			**Recall Wght'd F-m**
Metric	**Total predicted**	**1.5%**	**7%**	**13%**	**1.5%**	**7%**	**13%**	**1.5%**	**7%**	**13%**	**1.5%**	**7%**	**13%**	**1.5%**
CCCC metrics individually														
CBO	20	0.05	0.25	0.35	0.25	0.24	0.21	0.08	0.24	0.26	0.06	0.25	0.31	0.14
DIT	148	0.02	0.1	0.17	0.75	0.71	0.76	0.04	0.18	0.28	0.03	0.12	0.2	0.09
NOC	205	0.02	0.1	0.16	1.0	1.0	1.0	0.04	0.19	0.28	0.02	0.12	0.19	0.09

Continued on next page

Table I.1 – continued from previous page

		Precision			Recall			Standard F-measure			Precision Weighted F-measure			Recall Wght'd F-m
Metric	**Total**	**1.5%**	**7%**	**13%**	**1.5%**	**7%**	**13%**	**1.5%**	**7%**	**13%**	**1.5%**	**7%**	**13%**	**1.5%**
WMC	44	0.05	0.23	0.39	0.5	0.48	0.52	0.08	0.31	0.44	0.06	0.25	0.41	0.17
LOC	29	0.03	0.52	0.66	0.25	0.71	0.58	0.06	0.6	0.61	0.04	0.55	0.64	0.11
comment	34	0.06	0.47	0.62	0.5	0.76	0.64	0.11	0.58	0.63	0.07	0.51	0.62	0.2
MVG	38	0.03	0.34	0.55	0.25	0.62	0.64	0.05	0.44	0.59	0.03	0.38	0.57	0.09
CPL	137	0.02	0.12	0.21	0.75	0.81	0.88	0.04	0.22	0.34	0.03	0.15	0.25	0.1
Our metrics individually														
NR	125	0.02	0.14	0.22	0.75	0.81	0.85	0.05	0.23	0.35	0.03	0.16	0.26	0.11
SRR	83	0.02	0.19	0.29	0.5	0.76	0.73	0.05	0.31	0.41	0.03	0.23	0.33	0.1
ASRR	103	0.03	0.16	0.24	0.75	0.76	0.76	0.06	0.26	0.37	0.04	0.18	0.28	0.13
RI	26	0.04	0.35	0.5	0.25	0.43	0.39	0.07	0.38	0.44	0.05	0.36	0.47	0.12
NOut	72	0.03	0.25	0.32	0.5	0.86	0.7	0.05	0.39	0.44	0.03	0.29	0.36	0.11
SOR	37	0.03	0.32	0.46	0.25	0.57	0.52	0.05	0.41	0.49	0.03	0.36	0.47	0.09
ASOR	39	0.03	0.31	0.36	0.25	0.57	0.42	0.05	0.40	0.39	0.03	0.34	0.37	0.09
NDP	140	0.03	0.14	0.22	1.0	0.95	0.94	0.06	0.25	0.36	0.04	0.17	0.26	0.13
NOpt	138	0.03	0.14	0.22	1.0	0.95	0.91	0.06	0.25	0.35	0.04	0.17	0.26	0.13
ANODP	131	0.03	0.15	0.22	1.0	0.9	0.88	0.06	0.25	0.35	0.04	0.17	0.26	0.14
SOS	25	0.04	0.2	0.36	0.25	0.24	0.27	0.07	0.22	0.31	0.05	0.21	0.34	0.12
ASOS	37	0.03	0.14	0.24	0.25	0.24	0.27	0.05	0.17	0.26	0.03	0.15	0.25	0.09
SOO	39	0.03	0.23	0.33	0.25	0.43	0.39	0.05	0.30	0.36	0.03	0.25	0.34	0.09
ASOO	39	0.03	0.23	0.33	0.25	0.43	0.39	0.05	0.30	0.36	0.03	0.25	0.34	0.09
NES	14	0.07	0.43	0.57	0.25	0.29	0.24	0.11	0.34	0.34	0.08	0.39	0.45	0.17
Combinations of metrics														

Continued on next page

Table I.1 – continued from previous page

		Precision			Recall			Standard F-measure			Precision Weighted F-measure			Recall Wght'd F-m
Metric	**Total**	**1.5%**	**7%**	**13%**	**1.5%**	**7%**	**13%**	**1.5%**	**7%**	**13%**	**1.5%**	**7%**	**13%**	**1.5%**
comments and ASRR	27	0.07	0.48	0.63	0.5	0.62	0.52	0.13	0.54	0.57	0.09	0.5	0.6	0.23
comments and NES	10	0.1	0.6	0.7	0.25	0.29	0.21	0.14	0.39	0.33	0.11	0.49	0.48	0.19
comments and SRR	27	0.07	0.52	0.63	0.5	0.67	0.52	0.13	0.58	0.57	0.09	0.54	0.6	0.23
comments and WMC	11	0.18	0.73	0.82	0.5	0.38	0.27	0.27	0.5	0.41	0.21	0.62	0.58	0.37
comments and CPL	28	0.07	0.5	0.68	0.5	0.67	0.58	0.13	0.57	0.62	0.09	0.53	0.66	0.23
CPL and NES	10	0.07	0.1	0.5	0.25	0.24	0.21	0.14	0.32	0.33	0.11	0.41	0.48	0.19
CPL and NOC	134	0.07	0.02	0.13	0.75	0.81	0.88	0.04	0.22	0.35	0.03	0.15	0.25	0.1
CPL and WMC	35	0.06	0.26	0.46	0.5	0.43	0.48	0.1	0.32	0.47	0.07	0.28	0.46	0.2
DIT and comments	25	0.04	0.44	0.6	0.25	0.52	0.45	0.07	0.48	0.52	0.05	0.45	0.56	0.12
DIT and NES	9	0	0.33	0.56	0	0.14	0.15	0	0.2	0.24	0	0.26	0.36	0
DIT and NOC	142	0.02	0.11	0.18	0.75	0.71	0.76	0.04	0.18	0.29	0.03	0.13	0.21	0.09

Continued on next page

Table I.1 – continued from previous page

		Precision			Recall			Standard F-measure			Precision Weighted F-measure			Recall Wght'd F-m
Metric	**Total**	**1.5%**	**7%**	**13%**	**1.5%**	**7%**	**13%**	**1.5%**	**7%**	**13%**	**1.5%**	**7%**	**13%**	**1.5%**
DIT and WMC	30	0.03	0.2	0.4	0.25	0.29	0.36	0.06	0.24	0.38	0.04	0.21	0.39	0.11
LOC and NES	9	0.11	0.67	0.78	0.25	0.29	0.21	0.15	0.4	0.33	0.13	0.53	0.51	0.2
LOC and WMC	9	0.11	0.78	0.89	0.25	0.33	0.24	0.15	0.47	0.38	0.13	0.61	0.58	0.2
NDP and ASRR	95	0.03	0.17	0.25	0.75	0.76	0.73	0.06	0.28	0.38	0.04	0.2	0.29	0.14
NDP and comments	31	0.06	0.52	0.65	0.75	0.76	0.61	0.11	0.62	0.63	0.08	0.55	0.64	0.21
NDP and CPL	99	0.03	0.17	0.28	0.75	0.81	0.85	0.06	0.28	0.42	0.04	0.2	0.33	0.13
NDP and DIT	118	0.03	0.13	0.21	0.75	0.71	0.76	0.05	0.22	0.33	0.03	0.15	0.25	0.11
NDP and LOC	28	0.04	0.54	0.68	0.25	0.71	0.58	0.06	0.61	0.62	0.04	0.56	0.66	0.11
NDP and NES	14	0.07	0.43	0.57	0.25	0.29	0.24	0.11	0.34	0.34	0.08	0.39	0.45	0.17
NDP and NOC	138	0.03	0.14	0.22	1.0	0.95	0.94	0.06	0.25	0.36	0.04	0.17	0.26	0.13
NDP and NOut	72	0.03	0.25	0.32	0.5	0.86	0.7	0.05	0.39	0.44	0.03	0.29	0.36	0.11

Continued on next page

Table I.1 – continued from previous page

		Precision			Recall			Standard F-measure			Precision Weighted F-measure			Recall Wght'd F-m
Metric	**Total**	**1.5%**	**7%**	**13%**	**1.5%**	**7%**	**13%**	**1.5%**	**7%**	**13%**	**1.5%**	**7%**	**13%**	**1.5%**
NDP and NR	113	0.03	0.15	0.24	0.75	0.81	0.82	0.05	0.25	0.37	0.03	0.18	0.28	0.12
NDP and SRR	82	0.02	0.2	0.29	0.5	0.76	0.73	0.05	0.31	0.42	0.03	0.23	0.33	0.1
NDP and WMC	36	0.06	0.25	0.44	0.5	0.43	0.48	0.1	0.32	0.46	0.07	0.27	0.45	0.19
NES and ASRR	11	0.09	0.36	0.55	0.25	0.19	0.18	0.13	0.25	0.27	0.1	0.31	0.39	0.19
NES and SRR	11	0.09	0.45	0.64	0.25	0.24	0.21	0.13	0.31	0.32	0.1	0.38	0.45	0.19
NOC and ASRR	102	0.03	0.16	0.25	0.75	0.76	0.76	0.06	0.26	0.37	0.04	0.19	0.28	0.13
NOC and comments	34	0.06	0.47	0.62	0.5	0.76	0.64	0.11	0.58	0.63	0.07	0.51	0.62	0.2
NOC and LOC	29	0.03	0.52	0.66	0.25	0.71	0.58	0.06	0.6	0.61	0.04	0.55	0.64	0.11
NOC and NOut	71	0.03	0.25	0.32	0.5	0.86	0.7	0.05	0.39	0.44	0.03	0.3	0.36	0.11
NOC and SRR	83	0.02	0.19	0.29	0.5	0.76	0.73	0.05	0.31	0.41	0.03	0.23	0.33	0.1
NOut and comments	29	0.07	0.55	0.66	0.5	0.76	0.58	0.12	0.64	0.61	0.08	0.58	0.64	0.22

Continued on next page

Table I.1 – continued from previous page

		Precision			Recall			Standard F-measure			Precision Weighted F-measure			Recall Wght'd F-m
Metric	**Total**	**1.5%**	**7%**	**13%**	**1.5%**	**7%**	**13%**	**1.5%**	**7%**	**13%**	**1.5%**	**7%**	**13%**	**1.5%**
NOut and NES	14	0.07	0.43	0.57	0.25	0.43	0.24	0.11	0.34	0.34	0.08	0.39	0.45	0.17
NOut and WMC	25	0.08	0.36	0.48	0.5	0.43	0.36	0.14	0.39	0.41	0.1	0.37	0.45	0.24
NR and comments	30	0.07	0.47	0.6	0.5	0.67	0.55	0.12	0.55	0.57	0.08	0.5	0.59	0.22
NR and NES	12	0.08	0.42	0.58	0.25	0.24	0.21	0.13	0.3	0.31	0.0	0.36	0.43	0.18
NR and NOC	124	0.02	0.14	0.23	0.75	0.81	0.85	0.05	0.23	0.36	0.03	0.16	0.26	0.11
NR and WMC	35	0.06	0.23	0.43	0.5	0.38	0.45	0.11	0.29	0.44	007	0.25	0.43	0.2
WMC and ASRR	29	0.07	0.24	0.41	0.5	0.33	0.36	0.12	0.28	0.39	0.08	0.26	0.4	0.29
WMC and SRR	26	0.08	0.31	0.5	0.5	0.38	0.39	0.13	0.34	0.44	0.09	0.32	0.47	0.24

Bibliography

[1] G. Antoniol, G. Canfora, and A. De Lucia. Estimating the size of changes for evolving object oriented systems: a case study. In *In Proccedings of the Sixth International Symposium on Software Metrics, METRICS '99*, pages 250–259. IEEE Computer Society, 1999.

[2] Erik Arisholm, Lionel C. Briand, and Audun Føyen. Dynamic coupling measurement for object-oriented software. *IEEE Transactions on Software Engineering*, 30(8):491–506, 2004.

[3] Paul Arkley and Steve Riddle. Overcoming the traceability benefit problem. In *Proceedings RE'05: Proceedings of the 13th IEEE International Requirements Engineering Conference*, pages 385–389, Aug-Sept 2005.

[4] Paul Arkley and Steve Riddle. Tailoring traceability information to business needs. In *Proceedings RE'06: Proceedings of the 14th IEEE International Requirements Engineering Conference*, pages 234–239, September 2006.

[5] Thomas Ball, Jung-Min Kim, Adam A. Porter, and Harvey P. Siy. If your version control system could talk ... In *Workshop on Process Modelling and Empirical Studies of Software Engineering*, 1997.

[6] R. K. Bandi, V. K. Vaishnavi, and D. E. Turk. Predicting Maintenance Performance Using Object-Oriented Design Complexity Metrics. *IEEE Transactions on Software Engineering*, 29:77–87, 2003.

[7] Victor R. Basili, Lionel C. Briand, and Walcélio L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, 1996.

[8] Victor R. Basili, Richard W. Selby, and David H. Hutchens. Experimentation in software engineering. *IEEE Transactions on Software Engineering*, 12(7):733–743, 1986.

[9] Victor R. Basili and David M. Weiss. A methodology for collecting valid software engineering data. *IEEE Transactions on Software Engineering*, 10(6):728–738, 1984.

[10] Kent Beck. *Extreme Programming explained: embrace change*. Pearson, 2000.

[11] L.A. Belady. Modifiability of large software systems. In M.M. Lehman and L.A. Belady, editors, *Program Evolution: Processes of Software Change*, volume 27 of *A.P.I.C Studies in Data Processing*, pages 355–374. Academic Press, 1985.

[12] L.A. Belady and M.M Lehman. An introduction to growth dynamics. In M.M. Lehman and L.A. Belady, editors, *Program Evolution: Processes of Software Change*, volume 27 of *A.P.I.C Studies in Data Processing*, pages 123–132. Academic Press, 1985.

[13] L.A. Belady and M.M. Lehman. Programming system dynamics or the metadynamics of systems in maintenance and growth. In M.M. Lehman and L.A. Belady, editors, *Program Evolution: Processes of Software Change*, volume 27 of *A.P.I.C Studies in Data Processing*, pages 99–122. Academic Press, 1985.

[14] Keith Bennett. Software evolution: past, present and future. *Information and Software Technology*, 38(11):673–680, 2000.

[15] K.H. Bennett and V.T. Rajlich. Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, pages 73–87, June 2000.

[16] D.M. Berry. The inevitable pain of software development, including of extreme programming, caused by requirements volatility. In *Proceedings of the International Workshop on Time Constrained Requirements Engineering (T-CRE)*, pages 9–19, September 2002.

[17] Jennifer Bevan and Jr E. James Whitehead. Identification of software instabilities. In *Proceedings of the 10th Working Conference on Reverse Engineering*, page 134, November 2003.

[18] Barry Boehm. Requirements that handle IKIWISI, COTS, and rapid change. *IEEE Computer*, 33(7):99–102, 2000.

[19] Barry Boehm, Alexander Egyed, Julie Kwan, and Archita Shah. Using the WinWin spiral model: A case study. *IEEE Computer*, 31(7):33–44, July 1998.

[20] Barry Boehm and Li Guo Huang. Value-based software engineering: A case study. *IEEE Software*, 36(3):33–41, 2006.

[21] Barry W. Boehm. A spiral model of software development and enhancement. *IEEE Computer*, 21(5):61–72, 1988.

[22] Shawn A. Bohner. Software change impacts: An evolving perspective. In *Proceedings, International Conference on Software Maintenance 2002*, pages 263–271, October 2002.

[23] Salah Bouktif, Yann-Gaël Guéhéneuc, and Guiliano Antoniol. Extracting change-patterns from CVS respositories. In *Proceedings of the 13th Working Conference on Reverse Engineering*, pages 221–230, October 2006.

[24] Lionel Briand, Sandro Morasca, and Victor R. Basili. Property-based software engineering measurement. *IEEE Transactions on Software Engineering*, 22:68–86, 1996.

[25] Lionel C. Briand, John W. Daly, and Jürgen K. Wüst. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on Software Engineering*, 25(1):91–121, 1999.

[26] Lionel C. Briand, Jürgen Wüst, and Hakim Lounis. Using coupling measurement for impact analysis in object-oriented systems. In *Software Maintenance, 1999. (ICSM '99) Proceedings. IEEE International Conference on*, pages 475–482, 1999.

[27] CCCC. User guide for CCCC. Downloaded from: http://sourceforge.net/projects/cccc, 2010. [Visited 27th April 2010].

[28] Ned Chapin, Joanne E. Hale, Khaled Md. Khan, Juan F. Ramil, and Wui-Gee Tan. Types of software evolution and software maintenance. *Journal of Software Maintenance: Research and Practice*, 13(1):3–30, 2001.

[29] M. Ajmal Chaumun, Hind Kabaili, Rudolf K. Keller, François Lustman, Département Iro, and Université De Montréal. A change impact model for changeability assessment in object-oriented software systems. In *In Proceedings of the Third Euromicro Working Conference on Software Maintenance and Reengineering*, pages 130–138, 1999.

[30] John C. Cherniavsky and Carl H. Smith. On Weyuker's axioms for software complexity measures. *IEEE Trans. Software Eng.*, 17(6):636–638, 1991.

[31] S. Chidamber and C. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.

[32] Shyam R. Chidamber, David P. Darcy, and Chris F. Kemerer. Managerial use of metrics for object-oriented software: An exploratory analysis. *IEEE Transactions on Software Engineering*, 24(8):629–639, 1998.

[33] Neville I. Churcher and Martin J. Shepperd. Comments on 'A metrics suite for object oriented design'. *IEEE Transactions on Software Engineering*, 21(3):263–265, March 1995.

[34] John Clarkson, Caroline Simons, and Claudia Eckert. Predicting change propagation in complex design. *Journal of Mechanical Design (Transactions of the ASME)*, 126:788–797, September 2004.

[35] Jane Cleland-Huang. Just enough requirements traceability. In *30th Annual International Computer Software and Applications Conference (COMPSAC 2006)*, pages 41–42, September 2006.

[36] Jane Cleland-Huang, Carl K. Chang, and Mark Christensen. Event-based traceability for managing evolutionary change. *IEEE Transactions on Software Engineering*, 29(9):796–810, 2003.

[37] Jane Cleland-Huang, Raffaella Settimi, and Oussama BenKhadra. Goal-centric traceability for managing non-functional requirements. In *Proceedings of the 27th International Conference on Software Engineering*, pages 362–371, 2005.

[38] Jane Cleland-Huang, Raffaella Settimi, Xuchang Zou, and Peter Solc. The detection and classification of non-functional requirements with application to early aspects. In *Proceedings RE'06: Proceedings of the 14th IEEE International Requirements Engineering Conference*, pages 36–45, September 2006.

[39] Jane Cleland-Huang, Grant Zemont, and Wiktor Lukasik. A heterogeneous solution for improving the return on investment of requirements traceability. In *12th IEEE International Conference on Requirements Engineering (RE 2004)*, pages 230–239, September 2004.

[40] Jeff Conklin and Michael L. Begeman. gIBIS: A tool for all reasons. *Journal of the American Society for Information Science*, 40(3):200–213, 1989.

[41] CARMEN Consortium. CARMEN: Code analysis, repository and modelling for e-neuroscience. http://www.carmen.org.uk/about, 2008. [Visited 6th June 2008].

[42] High Coolican. *Research Methods and Statistics in Psychology.* Hodder and Stoughton, 1990.

[43] Bill Curtis, Herb Krasner, and Neil Iscoe. A field study of the software design process for large systems. *Communictions of the ACM*, 31(11):1268–1287, 1988.

[44] David P. Darcy and Chris F. Kemerer. OO metrics in practice. *IEEE Software*, 22(6):17–19, 2005.

[45] Anne Dardenne, Axel Van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. In *Science of Computer Programming*, volume 20, pages 3–50, 1993.

[46] David N. Dickter and Mary Roznowski. *The Psychology Research Handbook: A Guide for Graduate Students and Research Assistants*, chapter A Basic Statistical Analysis, pages 208–218. Sage Publications, 1996.

[47] Norman R. Draper and Harry Smith. *Applied Regression Analysis*. John Wiley and Sons, 1998.

[48] Fernando Brito e Abreu, Miguel Afonso Goul ao, and Rita Esteves. Toward the design quality evaluation of object-oriented software systems. In *Proceedings of the 5th International Conference on Software Quality*, October 1995.

[49] Steve Easterbrook. Handling conflict between domain descriptions with computer-supported negotiation. *Knowledge Acquistion: An International Journal*, 3:255–289, 1991.

[50] Christof Ebert and Jozef De Man. Goal-centric traceability for managing non-functional requirements. In *Proceedings of the 27th International Conference on Software Engineering*, pages 553–560, 2005.

[51] Alexander Egyed. Determining the cost-quality trade-off for automated software traceability. *ASE*, 2005:360–363, 2005.

[52] Alexander Egyed, Paul Grünbacher, Matthias Heindl, and Stefan Biffl. Value-based requirements traceability: Lessons learned. In *15th IEEE International Requirements Engineering Conference, RE 2007,*, pages 115–118, 2007.

[53] Khaled El Emam, Saïda Benlarbi, Nishith Goel, and Shesh N. Rai. The confounding effect of class size on the validity of object-oriented metrics. *IEEE Transactions on Software Engineering*, 27(7):630–650, 2001.

[54] William M. Evanco. Comments on 'The confounding effect of class size on the validity of object-oriented metrics'. *IEEE Transactions on Software Engineering*, 29(7):670–672, 2003.

[55] Norman Fenton and Martin Neil. Software metrics: Roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, pages 357–370, June 2000.

[56] Harald Gall, Mehdi Jazayeri, and Claudio Riva. Visualizing software release histories: The use of color and third dimension. In *Proceedings of the International Conference on Software Maintenance*, page 190, 1998.

[57] Daniel M. German. An empirical study of fine-grained software modifications. *Empirical Software Engineering*, 11:369–393, 2006.

[58] Tom Gilb. Evolutionary development. *ACM SIGSOFT Software Engineering Notes archive*, 6(2):17, April 1981.

[59] Orlena Gotel and Anthony Finkelstein. Contribution structures. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, pages 100–107, March 1995.

[60] Orlena Gotel and Anthony Finkelstein. Extended requirements traceability: Results of an industrial case study. In *International Symposium on Requirements Engineering (RE97*, pages 169–178. Society Press, 1997.

[61] Orlena C.Z. Gotel and Anthony C.W. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of the First International Conference on Requirements Engineering*, pages 94–101, April 1994.

[62] T. R. G. Green. An introduction to the cognitive dimensions framework: Extended abstract of invited talk at MIRA workshop, Monselice, Italy. http://homepage.ntlworld.com/greenery/workStuff/Papers/introCogDims/index.html, November 1996. [Visited 23rd April 2008].

[63] Thomas Green and Alan Blackwell. Cognitive dimensions of information artefacts: a tutorial. http://www.ndirect.co.uk/~thomas.green/workStuff/Papers/, October 1998. [Visited 8th August 2008].

[64] David A. Gustafson, Joo T. Tan, and Perla Weaver. Software measure specification. In *SIGSOFT FSE*, pages 163–168, 1993.

[65] Ah-Rim Han, Sang-Uk Jeon, Doo-Hwan Bae, and Jang-Eui Hong. Behavioral dependency measurement for change-proneness prediction in UML 2.0 design models. In *COMPSAC '08: Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference*, pages 76–83, Washington, DC, USA, 2008. IEEE Computer Society.

[66] S.D.P Harker and K.D. Eason. The change and evolution of requirements as a challenge to the practice of software engineering. In *Proceedings of the IEEE International Symposium on Requirements Eingineering*, pages 266–271, January 1993.

[67] Rachel Harrison, Steve J. Counsell, and Reuben V. Nithi. An evaluation of the MOOD set of object-oriented software metrics. *IEEE Transactions on Software Engineering*, 24:491–496, 1998.

[68] Ahmed E. Hassan and Richard C. Holt. Predicting change propagation in software systems. In *Proceedings of the 20th International Conference on Software Maintenance*, pages 284–293, September 2004.

[69] Matthias Heindl and Stefan Biffl. A case study on value-based requirements tracing. In *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 60–69, 2005.

[70] Joel Henry and Sallie Henry. Quantitive assessment of software maintenance and requirements volatility. In *Proceedings of the 1993 ACM Conference on Computer Science*, pages 346–351, February 1993.

[71] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.

[72] George Hripcsak and Adam S. Rothschild. Agreement, the f-measure, and reliability in information retrieval. *Journal of the American Medical Informatics Association*, 12(3):296–298, 2005.

[73] LiGuo Huang and Barry Boehm. How much software quality investment is enough: A value-based approach. *IEEE Software*, 23(5):88–95, September/October 2006.

[74] Matthias Jarke. Requirements tracing. *Communictions of the ACM*, 41(12):32–36, December 1998.

[75] Stephen R. G. Jones. Was there a Hawthorne Effect? *The American Journal of Psychology*, 98(3):451–468, November 1992.

[76] Frederick P. Brooks Jr. *The Mythical Man-Month.* Addison Wesley, 1995.

[77] Batalia Juristo, Ana M. Moreno, and Andrs Silvéa. Is the European industry moving toward solving requirements engineering problems? *IEEE Software*, 19(6):70–77, December 2002.

[78] Hind Kabaili, Rudolf K. Keller, and François Lustman. Cohesion as changeability indicator in Object-Oriented systems. In *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*, pages 39–46, March 2001.

[79] Cem Kaner and Walter P. Bond. Software engineering metrics: What do they measure and how do we know? In *10th Interntional Software Metrics Symposium, (METRICS 2004)*, pages 1–12, September 2004.

[80] Tim Kelly and J.A.McDermid. Safety case construction and reuse using patterns. In *Proceedings of 16th International Conference on Computer Safety, Reliability and Security (SAFECOMP'97)*, September 1997.

[81] Tim Kelly and Rob Weaver. The goal structuring notation - a safety argument notation. In *Proceedings of the Dependable Systems and Networks 2004 Workshop on Assurance Cases*, July 2004.

[82] Barbara Kitchenham, Shari Lawrence Pfleeger, and Norman Fenton. Towards a framework for software validation. *IEEE Transactions on Software Engineering*, 21(12):929–944, 1995.

[83] Barbara Kitchenham, Lesley Pickard, and Shari Lawrence Pfleeger. Case studies for method and tool evaluation. *IEEE Software*, 12(4):52–62, 1994.

[84] Barbara A. Kitchenham, Shari Lawrence Pfleeger, Lesley M. Pickard, Peter W. Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. Preliminary guidelines for empirical research in software enginering. *IEEE Transactions on Software Engineering*, 28(8):721–734, 2002.

[85] Mark Klein. Capturing design rationale in concurrent engingeering teams. *IEEE Computer*, 26(1):39–47, January 1993.

[86] Geraldn Kotonya and Ian Sommerville. *Requirements Engineering: Processes and Techniques.* John Wiley, 1998.

[87] W. Kunz and H. Rittel. *Issues as Elements of Information Systems.* Center for Planning and Development Research, University of California at Berkeley, 1970.

[88] W. Lam and M. Loomes. Requirements evolution in the midst of environmental change: A managed approach. In *Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering*, pages 121–127, March 1998.

[89] W. Lam and V. Shankaraman. Managing change in software development using a process improvement approach. In *Proceedings of the 24th Euromicro Conference 1998*, volume 2, pages 779–786, August 1998.

[90] M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski. Metrics and laws of software evolution - the nineties view. In *METRICS '97: Proceedings of the 4th International Symposium on Software Metrics*, Washington, DC, USA, 1997. IEEE Computer Society.

[91] Meir M. Lehman. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9):1060–1076, September 1980.

[92] M.M Lehman. Program evolution. In M.M. Lehman and L.A. Belady, editors, *Program Evolution: Processes of Software Change*, volume 27 of *A.P.I.C Studies in Data Processing*, pages 9–39. Academic Press, 1985.

[93] M.M Lehman and F.N. Parr. Program evolution and its impact on software engineering. In M.M. Lehman and L.A. Belady, editors, *Program Evolution: Processes of Software Change*, volume 27 of *A.P.I.C Studies in Data Processing*, pages 201–220. Academic Press, 1985.

[94] Wei Li and Sallie Henry. Object oriented metrics which predict maintainability. Technical report, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, February 1993.

[95] B.P. Lientz, E.B. Swanson, and G.E. Tompkins. Characteristics of application software maintenance. *Communictions of the ACM*, 21(6):466–471, 1978.

[96] Rüdiger Lincke, Jonas Lundberg, and Welf Löwe. Comparing software metrics tools. In *ISSTA '08: Proceedings of the 2008 international symposium on Software testing and analysis*, pages 131–142, New York, NY, USA, 2008. ACM.

[97] Mikael Lindvall. Evaluating impact analysis - a case study. *Empiricak Software Engineering*, 2(2):152–158, 1997.

[98] Annabella Loconsole and Jürgen Börstler. An industrial case study on requirements volatility measures. In *Proceedings of the Asia Pacific Software Engineering Conference*, pages 249–256, December 2005.

[99] Richard Lowry. Concepts and applications of inferential statistics. http://faculty.vassar.edu/lowry/webtext.html, 2020. [Visited 30th June 2010].

[100] Lesek A. Maciaszek. *Requirements Analysis and System Design.* Addison Wesley, 2nd edition (2005) edition, 2001.

[101] Allan MacLean, Richard M. Young, Victoria M. E. Bellotti, and Thomas P. Moran. Questions, options, and criteria: Elements of design space analysis. *Human-Computer Interaction*, 6:201–250, 1991.

[102] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Cambridge University Press, http://nlp.stanford.edu/IR-book/, HTML edition, 2008.

[103] Jack E. Matson and Brian R. Huguenard. Evaluating aptness of a regression model. *Journal of Statistics Education*, 15(2), 2007.

[104] Elaine L. May and Barbara A. Zimmer. The evolutionary development model for software. *Hewlett Packard Journal[0018-1153]*, 47(4):39, 1996.

[105] Thomas J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320, 1976.

[106] John H. McDonald. Handbook of biological statistics. http://udel.edu/ mcdonald/statintro.html, 2009. [Visited 30th June 2010].

[107] Tom Mens and Serge Demeyer. Future trends in software evolution metrics. In *IWPSE '01: Proceedings of the 4th International Workshop on Principles of Software Evolution*, pages 83–86, New York, NY, USA, 2001. ACM Press.

[108] Audris Mockus and Lawrence G. Votta. Identifying reasons for software changes using historic databases. In *Proceedings of the International Conference on Software Maintenance (ICSM'00)*, page 120, October 2000.

[109] David S. Moore and George P. McCabe. *Introduction to the Practice of Statistics*. W.H. Freeman and Company, 4 edition, 1996.

[110] Sandro Morasca, Lionel C. Briand, Victor R. Basili, Elaine J. Weyuker, and Marvin V. Zelkowitz. Comments on "towards a framework for software measurement validation". *IEEE Transactions on Software Engineering*, 23(3):187–188, 1997.

[111] John Mylopoulos, Lawrence Chung, and Eric Yu. From object-oriented to goal-oriented requirements analysis. *Communictions of the ACM*, 42:31–37, 1999.

[112] R. Ba nares Alcántara and H.M.S. Lababidi. Design support systems for process engineering - II. KBDS: An experimental prototype. *Computers & Chemical Engineering*, 19:279 – 301, 1995.

[113] Bob Nau. Testing the assumptions of linear regression (course notes, mba course: Decision 411 forecasting, duke university, fuqua school of business). http://www.duke.edu/ rnau/testing.htm, 2010. [Visited 16th June 2010].

[114] N. Nurmuliani, Didar Zowghi, and Sue Fowell. Analysis of requirements volatility during software development life cycle. In *Proceedings of the 15th Australian Software Engineering Conference (ASWEC 2004)*, pages 28–37, April 2004.

[115] David L. Parnas. Software aging. In *ICSE '94: Proceedings of the 16th international conference on Software engineering*, pages 279–287, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[116] Charles J. Petrie. Constrained decision revision. In *Proceedings of the Tenth AAAI Conference*, pages 393–400, 1992.

[117] Mario Piattini, Marcela Genero, and Luis Jiménez. A metric-based approach for predicting conceptual data models maintainability. *International Journal of Software Engineering and Knowledge Engineering*, 11(6):703–729, 2001.

[118] Martin Pinzger, Harald C. Gall, Michael Fischer, and Michele Lanza. Visualizing multiple evolution metrics. In *Proceedings of the ACM Symposium on Software Visualization (SoftVis'2005)*, pages 67–75, St. Louis, Missouri, USA, 2005. ACM.

[119] G. Poels and G. Dardene. Distance-based software measurement: necessary and sufficient properties for software measures. *Information and Software Technology*, 42(1):35–46, 2000.

[120] Klaus Pohl. *Process-Centred Requirements Engineering.* John Wiley, 1996.

[121] Derek Robert Price and Ximbiot. CVS - concurrent versions system. http://www.nongnu.org/cvs/, 2003. [Visited 16th October 2008].

[122] B. Ramesh, T. Powers, and C. Stubbs. Implementing requirements traceability: A case study. In *Proceedings of the 2nd IEEE International Symposium on Requirements Engineering*, pages 89–95, March 1995.

[123] Balasubramanhiam Ramesh. Requirements traceability: Theory and practice. *Annals of Software Engineering*, 3(0):397–415, January 1997.

[124] Balasubramanhiam Ramesh. Factors influencing requirements traceability practice. *Communictions of the ACM*, 41(12):37–44, December 1998.

[125] Balasubramanhiam Ramesh and Matthias Jarke. Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*, 27(1):58–93, January 2001.

[126] Balasubramaniam Ramesh and Vasant Dhar. Supporting systems development by capturing deliberations during requirements engineering. *IEEE Transactions on Software Engineering*, 18(6):498–510, June 1992.

[127] Jacek Ratzinger, Thomas Sigmund, Peter Vorburger, and Harald C. Gall. Mining Software Evolution to Predict Refactoring. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*, pages 354–363, Madrid Spain, 2007. IEEE Computer Society.

[128] W. C. Regli, X. Hu, M. Atwood, and W. Sun. A survey of design rationale systems: Approaches, representation, capture and retrieval. *Engineering with Computers*, 16:209–235, 2000.

[129] Jason D. M. Rennie. Derivation of the f-measure. http://people.csail.mit.edu/jrennie/writing, February 2004.

[130] Suzanne Robertson and James Robertson. *Mastering the Requirements Process.* Addison Wesley, 1999.

[131] Colin Robson. *Real World Research.* Blackwell, 1993.

[132] Howard B. Rubenstein. The role of software architecture in software requirements engineering. In *Proceedings fo the First International Conference on Requirements Engineering*, page 244, April 1994.

[133] Barbara Ryan, Brian Joiner, and Jonathon Cryer. *MINITAB Handbook.* Duxbury, 5th edition (2005) edition, 2005.

[134] Phil Scholfield. Simple and sophisticated bonferroni adjustment. http://privatewww.essex.ac.uk/ scholp/bonferroni.htm, 2009. [Visited 25th June 2009].

[135] M. Chandra Shekaran. The role of software architecture in software requirements engineering. In *Proceedings fo the First International Conference on Requirements Engineering*, page 245, April 1994.

[136] Martin Shepperd and Chris Schofield. Estimating software project effort using analogies. *IEEE Transactions on Software Engineering*, 23(12):736–743, 1997.

[137] Jelber Sayyad Shirabad, Timothy C. Lethbridge, and Stan Matwin. Mining the software change repository of a legacy telephone system. In *Proceedings of the International Workshop on Mining Software Repositories (MSR 2004)*, pages 53–57, May 2004.

[138] Ian Sommerville. *Software Engineering*. International Computer Science Series. Addison Wesley, 6th edition (2001) edition, 1982.

[139] Peter Sprent and Nigel C. Smeeton. *Applied Nonparametric Statistical Methods*. Chapman and Hall, 4th edition (2007) edition, 2007.

[140] George E. Stark, Paul Oman, Alan Skillicorn, and Alan Ameele. An examination of the effects of requirements changes on software maintenance releases. *Journal of Software Maintenance: Research and Practice*, 11(5):293–309, 1999.

[141] Zoe Stephenson. *Change Management in Families of Safety-Critical Embedded Systems*. PhD thesis, Department of Computer Science, University of York, York, YO10 5DD, 2002.

[142] University of Glasgow Steve Draper, Department of Psychology. The hawthorne, pygmalion, placebo and other effects of expectation: some notes. http://www.psy.gla.ac.uk/ steve/hawth.html, 2009. [Visited 2nd June 2009].

[143] M.R. Strens and R.C. Sugden. Change analysis: A step towards meeting the challenge of changing requirements. In *Proceedings of the IEEE Symposium and Workshop on the Engineering of Computer-Based Systems 1996*, pages 278–283, March 1996.

[144] E. Burton Swanson. The dimensions of maintenance. In *Proceedings of the 2nd International Conference on Software Engineering*, pages 492–497, 1972.

[145] Tigris.org. Open source software engineering tools: Subversion. http://subversion.tigris.org/, 2006. [Visited 16th September 2008].

[146] C. van Koten and A.R. Gray. An application of Bayesian nework for predicting object-oriented software maintainability. *Information and Software Technology*, 48(1):59–67, 2006.

[147] Axel van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *Proceedings RE'01: 5th IEEE International Symposium on Requirements Engineering*, page 249, August 2001.

[148] Rajesh Vasa and Jean-Guy Schenider. Evolution of complexity in object oriented software. In *Workshop on Quantative Approaches in Object-Oriented Software Engineering (QAOOSE'2003)*, pages 1–5, July 2003.

[149] Michel Wermelinger and Yijun Yu. Analyzing the evolution of Eclipse plugins. In *Proceedings of the 2008 International working conference on mining software repositories*, 2008.

[150] Michel Wermelinger, Yijun Yu, and Markus Strohmaier. Using formal concept analysis to construct and visualise social hierarchies of software developers. In *Proceedings of the 2009 International Conference on Software Engienering (ICSE'09)*, 2009.

[151] Elaine J. Weyuker. Evaluating software complexity measures. *IEEE Transactions on Software Engineering*, 14(9):1357–1365, 1988.

[152] F.G. Wilkie and B.A. Kitchenham. Coupling measures and change ripples in C++ application software. *Journal of Systems and Software*, 52(2-3):157–164, 2000.

[153] Zhengchang Xing and Eleni Stroulia. Data-mining in support of detecting class co-evolution. In *Proceedings 16th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, June 2004.

[154] Robert A. Yaffee. *The Psychology Research Handbook: A Guide for Graduate Students and Research Assistants*, chapter A Basic Guide to Statistical Research and Discovery: Planning and Selecting Statistical Analyses, pages 193–207. Sage Publications, 1996.

[155] Eric S. K. Yu. Models for supporting the redesign of organizational work. In *Proceedings from the Conference on Organizational Computing Systems (COOCS'95)*, August 1995.

[156] Eric S. K. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE '97)*, pages 226–235, January 1997.

[157] Xiaoni Zhang, John Windsor, and Robert Pavur. Determinants of software volatility: a field study. *Journal of Software Maintenance: Research and Practice*, 15(3):191–204, 2003.

[158] Thomas Zimmerman, Peter Weißgerber, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software change. *IEEE Transactions on Software Engineering*, 31(6):429–445, 2005.

[159] Didar Zowghi and N. Nurmuliani. A study of the impact of requirements volatility on software project performance. In *Proceedings of the Ninth Asia-Pacific Software Engineering Conference (APSEC02)*, pages 3–11, December 2002.

[160] Horst Zuse. *A framework of software measurement.* Walter de Gruyter & Co., 1997.