Software Requirements Change Analysis and Prediction

by

Sharon McGee (B.Sc.)

A thesis submitted to the
School of Electronics, Electrical Engineering and Computer Science
in Queen's University Belfast
for the Degree of

Doctor of Philosophy

September 2013

Abstract

Software requirements continue to evolve during application development to meet the changing needs of customers and market demands. Complementing current approaches that constrain the risk that changing requirements pose to project cost, schedule and quality, this research seeks to investigate the efficacy of Bayesian networks to predict levels of volatility early in the project lifecycle. A series of industrial empirical studies is undertaken to explore the causes and consequences of requirements change, the results of which inform prediction feasibility and model construction. Models are then validated using data from four projects in two industrial organisations. Results from empirical studies indicate that classification of changes according to the source of the change is practical and informative to decisions concerning requirements management and process selection. Changes coming from sources considered external to the project are more expensive and difficult to control by comparison to the more numerous changes that occur as result of adjustments to product direction, or requirement specification. Although certain requirements are more change prone than others, the relationship between volatility and requirement novelty and complexity is not straightforward. Bayesian network models to predict levels of requirement volatility constructed based upon these results perform better than project management estimations of volatility when models are trained from a project sharing industrial context. This research carries the implication that process selection should be based upon the types of changes likely, and that formal predictive models are a promising alternative to project management estimation when investment in data collection, re-use and learning is supported.

Preface

Some of the research presented in this thesis has been published in conference proceedings and journal articles, as follows:-

1. McGee, S., Greer, D., Towards an understanding of the causes and effects of software requirements change: two studies, Journal of Requirements Engineering, June, 2012, Volume 17, Number 2, pp. 133-155.

2. McGee, S., Greer, D., Software Requirements Change Taxonomy: Evaluation by Case Study, Proceedings of the IEEE International Requirements Engineering Conference, pp. 25-34, Trento, Italy, September, 2011. (Distinguished paper award)

3. McGee, S., Greer, D., Sources of Software Requirements Change from the Perspectives of Development and Maintenance, International Journal on Advances in Software, pp. 186-200, Sep 2010.

4. McGee, S., Greer, D., A Software Requirements Change Source Taxonomy, IEEE International. Conference of Software Engineering Advances (ICSEA), 00 51-58, IEEE Computer Society, September, 2009. (Best paper award)

Acknowledgements

It has been a great pleasure to have undertaken this work under the guidance of my supervisor, Des Greer. His continual support, kindness and sense of humour fostered openness and honesty which has smoothed the progress of this research. Without some gentle steering, it would have been easy to have given mileage to a distracted mind. I particularly appreciate his open door, and readiness to engage in the co-authoring of papers with enthusiasm and attention to detail. His knowledge of all things empirical and technical was shared happily, and most gratefully received. Thanks must also go to my second supervisor, Professor Bell for his constant encouragement and inspirational conversations that offered much food for thought.

Many hours were spent with industrial practitioners during the course of these empirical studies. I am deeply indebted to all of them personally for their immense effort and continued commitment despite other claims upon their time. This research is entirely based upon the results of their endeavours. I am also grateful to the organisations involved for the data provided.

To my family – what can I say? I hope that team McGee continues to support each other with the level of patience and care that I have been given. My husband, Vernon, made this research entirely possible, from beginning to end, from inside to outside, from top to bottom. Thank-you.

Table of Contents

Table of Figures

List of Tables

Chapter 1 Introduction

Successful software development depends upon an ability to respond effectively to changes to requirements. Project managers are challenged to deliver software that accommodates technical advancement and improved awareness of customer need, while constraining the negative impact of volatility upon project cost, schedule and quality. Uncertainty regarding the likelihood, nature and consequences of change frustrates decision making. Consequently, mismanagement of changes to requirements that occur during software development is recognized as a major cause of project failure [1][2]. Press coverage highlighting the high, and sometimes fatal cost of changing requirements in industry [3], is responded to by calls from researchers such as Pfleeger [4] who recommends that "We must find a way to understand and anticipate some of the inevitable change we see during software development".

There are various change management approaches within Requirements Engineering that aim to maximise value delivered to the customer while minimising risks associated with unplanned changes. These include the use of creative methods to more fully explore customer need [5], crafting adaptable architectures that cost-effectively facilitate future change [6], and risk management that explicitly addresses requirements change [7]. These approaches have an associated cost, and do not address the uncertain nature of change. Adopting a more agile development process reduces the need to be change-averse and welcomes the opportunity to add value by addressing changes as they arise. It has been argued though, that this approach has limitations and environmental pre-requisites which make close adherence to agile principles challenging, and potentially risky in certain circumstances [8]. Indeed, the UK software development contract procurement process is somewhat contraindicative to agile practices, given the need for clear upfront requirements in order to bid for project work. It has also been observed that project managers in Germany consider that he need and expenditure patterns of their customers is the foremost consideration of process selection [9].

An approach that may be considered a compromise between tight control and flexibility is project planning that takes into account predicted levels of volatility. Awareness of the types of changes that may be expected, as well as identification of the more change-prone requirements will support such constrained flexibility in the planning process. This approach has three main advantages. Firstly, schedule and cost may be bound to estimates that are augmented with anticipated volatility, allowing flexibility of application scope. Secondly, decisions regarding the most appropriate development process will benefit from an understanding of the types of changes that are likely to occur. Thirdly, identification of more volatile requirements near to the start of the development life cycle enables the design of the system such that architectural components with more volatile requirements can be decoupled.

While there are many studies that predict change-prone software after release, research intending to discover more volatile requirements early in software development is limited. Proposed requirements engineering techniques encourage looking to the future to assess requirement stability [10][11]. Empirical research concludes that attributes such as the size of a use case document [12], and a particular requirements type classification [13] correlate with volatility. A more cause-based study examined how proposed government regulation rules evolve into final rules and thus ascertained those that were more likely to change [14]; requirements based upon such government rules are likely to be more volatile. Each of these studies is practically useful in certain circumstances, but do not thoroughly explore the character and nature of requirements volatility.

The overall goal of this research is to improve decision support for requirements handling and architectural planning during software development by clarifying some of the uncertainty associated with changing requirements. Working with an industrial partner, a series of empirical studies investigate the causes and consequences of software requirements change. The results of these studies inform the development of predictive mechanisms that facilitate change anticipation early in the software development lifecycle. By comparison to other research in this area, the models rely on causal influences upon volatility that relate to the software engineering environment, and are grounded in tested hypotheses regarding the ways in which requirements change. As such, this research contributes to knowledge of the nature of software requirements change, the extent to which it can be anticipated, and the benefits and weaknesses of formal models of prediction.

1.1 Motivation and Context

For many years, it has been considered that effective management of change is fundamental to the timely delivery of valuable software [15][2]. There are many avenues of research aiming to support change management. The first is concerned with assessing the negative consequences of change. The inclusion of requirement change in risk registers [7][16] can be complemented by reactive measures to predict the cost impact of change [17][18], or the propagation of change to other areas of the system [19]. In practice, most software development management environments include change control processes such as review boards and change repositories [20]. Approaches that proactively manage changing requirements fall into three broad categories which are illustrated in Figure 1.1.

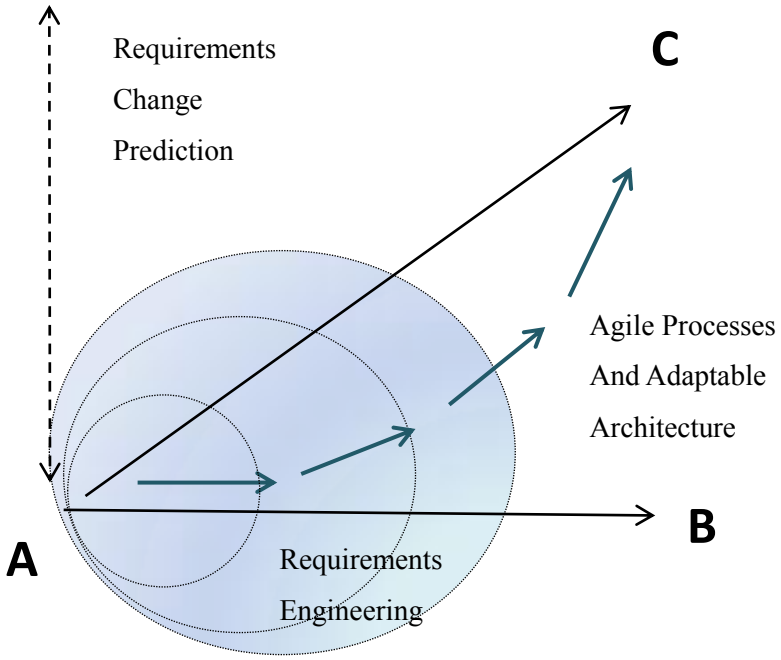

Figure 1.1 REQUIREMENTS CHANGE MANAGEMENT APPROACHES

A project that begins at point A with a full set of clearly defined requirements will progress, as specified, towards delivery of a set of requirements at point B unless changes are accepted. Upon software delivery (point B), it is possible that the customer will have been better served with a different set of requirements (point C). Requirements Engineering techniques are

concerned with exploring customer needs and should accommodate a range of potential requirement sets, that would include those at both point B and C along with many other possible outcomes. Agile processes begin with no definite end-point, and deliver software iteratively, thereby engineering opportunities for feedback and a general incremental movement toward point C. Change prediction does not make an assessment of requirement direction, but uses project and environment factors available at point A to either estimate the extent of the difference between point B and point C, or determine which requirements are more likely to vary from point B. Adopting one of these approaches does not preclude use of the others. Instead they complement one another, though each has distinct research opportunities and limitations which are outlined briefly below:-

1. Constrain future change through Requirements Engineering techniques.

Attempts to reduce the possibility of future change during requirements elicitation include the use of creative methods to encourage exploration of requirements from different perspectives which may reveal hitherto unknown possibilities [5] [10]. Similarly, 'Futurology' attempts to predict the direction of evolution by using techniques such as change visualisation to help create visions of future need [11]. These avenues of research have proved fruitful, in that new directions for application development have been discovered, though it is arguable that requirements will continue to change as these new directions are more fully understood during the development process. Requirements inspections that examine the language used in Requirements Engineering documents for inconsistency and ambiguity lead to increased accuracy and thus reduce the likelihood of future change [21]. However, such inaccuracies are only one of a number of causes of changes to requirements [22].

2. Embrace change through agile processes and architectural flexibility.

A significant volume of research has addressed the management of change by means of software development process [23], with change-embracive agile methods at the forefront. However, these are not without constraints since their scalability to larger projects or those considered business-critical is still considered a challenge [2]. MacCormac's [24] work supports the view that the selection of development practices and processes should be founded upon the specific types of uncertainty faced. This is reflected by Moynihan [25], who argues that project managers use different strategies to cope with different uncertainty-related issues. Proposed by

Boehm [26] in an effort to discourage a one-size-fits-all approach, an agility-discipline framework supports assessment of the suitability of traditional or agile techniques. However, one of the deciding factors is an estimate of the volatility that will be experienced, which has been observed to be a challenge beyond the capabilities of most experienced project managers [27].

In recognition of the need for flexible architecture that will support changing requirements, Parnas's [28] early work on information hiding encourages architectural modularization that facilitates ease of architecture and code modification by encapsulating volatile areas. Nuseibeh [29] considers the needs of incremental software delivery and proposes that developers evolve architecture and requirements concurrently. Subsequently, much research effort in this area has addressed issues involved in designing adaptable architecture, including quality, cost and modelling techniques [6]. Boehm [2] warns that developers rarely achieve a change-driven design by considering which requirements are likely to change, and therefore there is a tendency to over-engineer and create larger systems which can be more abstract and therefore costly to develop, maintain and manage. Recent studies recommend the use of explicit modelling of uncertainty to inform architectural design decisions, when there is a gap in domain knowledge or disagreement between stakeholders [30]. However, Thakurta [20] observes that in practice, architecting the product to withstand change is one of the least used approaches to volatility management.

3. Predict Change

This area of research includes studies that attempt to predict the measure of overall volatility that can be expected [12][31][32], and the identification of more change prone requirements [14][10][33]. Currently these studies either examine correlative relationships rather than causal influences, or are focused upon a narrow definition of requirements change. These studies are reviewed in detail in Section 2.2. A prediction of volatility will not supersede the use of change management approaches discussed above, since it will not provide Requirements Engineers with better knowledge regarding customer need, nor explicitly reduce future change costs.

Prediction of expected volatility and identification of change prone requirements can inform decisions concerning all approaches to change management. Decisions regarding the most suitable process will benefit from an understanding of the changeability of requirements, and the

identification of change prone requirements will allow focus of creative methods, or costly architectural de-coupling upon volatile areas. In addition, a prediction of volatility will support release management decisions, project resourcing and cost estimates. In a survey to discover how project managers respond to the risk of requirements volatility, the prediction of volatility was not found to be an approach taken by practitioners [20].

Developments in the use of subjective measurements in Software Engineering [34][4], coupled with successful implementations of causal Bayesian networks [35][36], offer a possible means by which software requirements change prediction early in the development lifecycle may be achieved. Research using Bayesian networks to predict effort or quality use attributes of the software engineering environment that are considered causal to cost or defects and calibrate models using expert judgement [37] [38]. Many benefits have been reported including the ability to use subjective measures, handle uncertainty and make predictions with incomplete data [39]. Importantly, by comparison to other means of prediction, Bayesian networks facilitate the reasoning from effect to cause such that is possible to use networks to explain why a certain outcome may have occurred. Increasing awareness and attention paid to the discipline of empirical research in software engineering supports both the investigation of the causes and effects of software requirements change, and provides guidelines for the conduct of model validation [40][41][42].

1.2 Contribution

This research has made contributions to academic knowledge in the following areas, some of which has been published in conference papers and journal articles (listed in the Preface):-

1. Requirements Engineering.

    a) Requirements change classification.

    Through empirical study a cause based change classification was derived which provides a means to monitor, understand and analyse volatility. This classification builds upon previously published classifications, and provides a novel way to understand changes occurring during software development and maintenance. Importantly, by contrast to other classifications, was evaluated and found to be informative and practical.

b) The causes of requirements change

Through literature review and practitioner involvement a more exhaustive inventory of the causes of requirements change has been produced, including those in the environment, and attributes of the requirements themselves. The observation that change discovery depends upon the disposition of the requirement to change (requirement uncertainty) and also upon change discovery effort (change trigger) has implications for the prediction of volatility.

c) Empirical knowledge regarding requirements change.

Case Studies examining requirements volatility during the development lifecycle reveal the cost, value and controllability of requirements change during industrial software development.

d) An exploration of the feasibility of volatility prediction by formal methods.

The derived change classification, examination of change causes, and challenges concerning formal model calibration, contribute to knowledge concerning the practical feasibility of change prediction, and some of the limitations involved.

e) Prediction of change-prone requirements.

Affected through use of Bayesian networks and validated in an industrial setting, results suggest that it is possible to predict requirements volatility due to changes arising from within the software engineering environment in 70% of cases provided models are trained from a project sharing industrial context.

2. Empirical Software Engineering.

a) Contribution to debates concerning the efficacy of formal models for prediction.

Models were validated in two industrial contexts and compared with project manager estimation. Comments can be made regarding the importance of context for model calibration, and the possibility that project manager estimation can improve with experience.

3. Bayesian Networks.

   a) Comparison of model accuracy between expert defined calibration and models trained from data.
   Three scenarios of testing included models trained though expert judgement, models trained from data sharing industrial context, and models trained using data from a project in other organisation. Models calibrated using expert judgement produce predictions of volatility no better than expert estimation of volatility, carrying implications for Bayesian network usage where data is not available for model training.

1.3 Thesis Overview

Chapter 2 reviews relevant research, and opens with a discussion of related terminology such as evolution, volatility and requirements change. Such terms are frequently used in Software Engineering research but there is no common consensus as to their meaning. Their intended usage for the remainder of this thesis is clarified, allowing ease of comparison with other studies. Research relating to requirements change prediction, measurement, and analysis is examined, together with relevant research in predictive models for Software Engineering. Thus knowledge gaps and valuable research opportunities are exposed, which are summarised in a research problem statement at the end of the chapter. The following research challenges are identified:-

RC1. Clearly define context and terminology, especially software development, maintenance and evolution.
RC2. Investigate the causes of requirements change.
RC3. Derive an informative measurement of requirements change.
RC4. Examine feasibility of change prediction by formal model.
RC5. Build models relating causes of change to volatility.
RC6. Validate models for accuracy and evaluate models for usability.

The Solution approach is presented in Chapter 3, which firstly draws high level research questions relating to the goal of requirements change prediction from the problem statement and research challenges identified in the previous chapter. The means by which these questions are addressed, including empirical methods and Bayesian Belief Networks, are outlined along with rationale for their usage. Benefits of these methods for both research and industry are discussed, as well as potential limitations. The process of empirical research, theory building, model implementation and validation, is illustrated by way of a research framework which also links research questions to the subsequent 5 chapters.

Chapter 4 describes the process of deriving an informative taxonomy of change sources, and presents the resulting classification. Five change domains were identified, each comprising a number of change cause constructs such as 'New stakeholder' or 'Analyst skill' drawn from literature. It was observed that such change sources can either be described as 'uncertainties' (for example, requirement complexity or novelty), or triggers (for example, change to business processes). The five change domains, of *market, customer organisation, vision, requirements specification,* and *solution* can be applied generically to any software engineering context. The original taxonomy, comprising change causes applicable to software development, is extended in a subsequent study to encompass change sources specific to software maintenance. Combined with a metric for volatility which takes into account cost as well as frequency, this classification allows requirements change measurement to be specified as a multi-dimensional formula.

The empirical evaluation of this taxonomy is presented in Chapter 5. A case study examining 282 changes occurring during an industrial project of 16 month duration is explained and the results presented. Detailed research questions are defined which to answer to the needs of industry and research. The data collection protocol, and validation process is described, and results presented. Results indicate that changes coming from each of the domains in the taxonomy had different characteristics. For example, it was discovered that changes coming from the domain of *customer organisation* are generally more costly and harder to control, though they are also thought to be more valuable to the business by comparison to changes arising due to increased understanding of the requirement specification. Changes are also discovered by different project activities. Customer testing more often discovers *vision* changes, whereas tasks related to programming are more likely to realise *requirement specification* changes. This implies that different types of changes may be better managed by techniques and

processes that reflect the characteristics of the change taxonomy. Classifying change in this way therefore supports management decisions regarding process and technique selection, as well as schedule, cost and delivery planning.

Chapter 6 furthers the investigation of the causes and consequences of change by investigating attributes of requirements for correlation with volatility. During data specification, a metric for requirements volatility was defined which was used in subsequent studies. Using the same project as the previous case study, complexity, novelty and dependency data for 240 requirements was collected. While the observation from previous empirical studies that a small percentage of requirements were the focus of most changes is borne out (in this case 80% volatility was spread over 31% of the requirements), many of the results are not straightforward. Requirements dependency is positively correlated in all change domains, but attributes such as novelty exhibited both positive and negative correlation depending upon the change domain. The implication of these results is that the causal influences upon requirements change are complex and varied.

This complexity is reflected in the causal Bayesian networks described in Chapter 7, which describes the structuring, calibration and verification of the models with our industrial partner through a series of workshops and interviews. Model design is based upon the results of the previous studies and this theoretical foundation is reviewed at the beginning of the chapter. Since previous studies identified that changes coming from the domains of *market* and *customer organisation* have influential factors outside of the immediate software engineering environment, only causal models for the domains *vision*, *requirement specification* and *solution* are derived. Methods and rationale for net construction are described as well as the variables and models themselves. Initial verification procedures by way of walk-throughs and sensitivity analysis are undertaken to ensure that the networks behave as they are designed.

Chapter 8 describes the validation of the Bayesian network models with industrial data. Two tranches of validation are presented. The first examines the capability of the models to predict the total level of requirements volatility in four industrial projects and compare the results with that of expert estimation. In this exercise, three cases were considered: i) models that have been calibrated using only expert opinion, that is, those models arising from the construction process in the previous chapter; ii) those utilizing the learning capability of Bayesian Nets to

parameterize the models from a project sharing industrial context; iii) models parameterized with data from a project with alternative industrial context. The second validation tranche examines the potential of the models to predict changes to requirements for a project in-flight; prediction validation occurred at the end of each project phase, whereupon the models were recalibrated and used to predict changes during the subsequent project phase. Subsequent analysis of usability issues and limitations provides an evaluation of the benefits to industry of the models in their current state. Results indicate that models trained from a project sharing industrial context yielded accuracy levels of roughly 70%. However, those whose calibration is defined by experts and those trained from a project with alternative context performed no better than expert estimation. The phase by phase validation accuracy improved from 40% to over 70% illustrating the ability of the models to learn as project development progresses. Model usability revealed that the cost of data collection and maintenance required by the models may be off-set against added functionality such as the prediction of cost and/or defects. Practitioners considered that there was benefit in maintaining the model data for analysis purposes, and reasoned that the prediction of requirements volatility would be particularly useful in situations where there were many stakeholders or an immature project team.

Chapter 9 provides a summary of all research undertaken and highlights the most significant results and contributions to academic knowledge. A review of progress towards the goal of volatility prediction, including research limitations, reveals potential areas for further research. These are introduced briefly, alongside additional directions that would be of benefit to empirical Software Engineering research, such as software project context classification, and improvements/support of estimates of subjective factors.

Chapter 2 Related Research

This chapter reviews related research in the area of requirements change management, analysis and prediction. This includes empirical research investigating the causes and effects of software requirements change, and approaches to change management. The differing perspectives on volatility measurement are examined with respect to the reasons for the use of such a metric for requirements change. Previously published change classifications are reviewed and compared, and attempts to predict volatility or identify change prone requirements are discussed in detail. The use of formal models is also discussed given the recent debate regarding their efficacy for prediction. The chapter begins with a review and clarification of the frequently used terminology.

2.1 Terminology Review

2.1.1 Software Evolution, Development and Maintenance.

Lehman's [43] [44] influential and continuingly relevant work on software change brought the term evolution into common research usage. Defined as "the dynamic behaviour of programming systems as they are maintained and enhanced over their lifetimes" [45], Belady & Lehman are deliberately inclusive of all stages of the software lifecycle, including initial development [43][46]. Subsequent to that work, authors have applied the term to development [47], used it as a synonym for maintenance [48][49], and proposed that it refers to a period of time between initial development and servicing [50]. Noting that the term lacked a standard definition, Bennett & Rajlich [50] sought to clarify its meaning by asking the question "What is maintenance?" and proposing a staged model for the software lifecycle. Their theory derived model promotes the latter view that software enters a phase of evolution following initial delivery and stops evolving once it is no longer feasible to make requirements changes. Subsequently the software enters a period of servicing when only minor corrections are made. Bennett & Rajlich claim that from a research perspective each stage has "different technical solutions, processes, staff needs and management activities". Therefore empirical research should firstly ensure that context is specified, and secondly explore the best solution for each stage. Interestingly, in a retrospective examination, Lehman & Ramil [43] observed that their empirical research supported the staged model.

The term maintenance has been defined by the IEEE [51] as "The modification of a software product after delivery to correct faults, to improve performance or other attributes or to adapt the product to a modified environment". As argued by Godfrey & German [52], this definition is not representative of all post-delivery activity, and the semantic inference of the term evolution more closely reflects the changing nature of software, and in particular accounts for requirements changes. Nonetheless, the term maintenance is still used widely, though not consistently. Kitchenham [53] developed an ontology of maintenance in which two scenarios are outlined. The first scenario, more commonly understood as evolutionary development, is included since in this instance the incremental nature of software delivery necessarily implies that there is a portion of software in the post-delivery phase. The second scenario represents the case where activity concerning software change is facilitated by a maintenance organisation

distinct from that of development. The second of those is the more traditionally accepted view of maintenance and the context of much ‘maintenance’ research. As an interesting aside, Basilli [54] considers software re-use and surmises that from a re-use perspective all development can be considered maintenance due to the prevalence of components usage. Chapin [55] assert that a classification of requirements change types, more traditionally ascribed to maintenance, can equally be applied to software development, and that this project nomenclature is relevant only in so far that it is prevalent in industry. Indeed, in that environment, deciding whether a project is ‘maintenance’ or ‘development’ is merely a question of project funding and contractual agreement. Supportive of this contention is the observation that the maintenance process ontology from Kitchenham [53] is derived from and bears direct semblance to a development process ontology proposed by de Fablo [56]. The activities involved in managing change (evaluation, impact analysis, approval, implementation, regression testing) and the supporting processes of configuration management, requirements traceability and release planning are beneficial elements of change management, irrespective of life-cycle phase. However, Chapin [57] also assert that the level of effort consumed by these activities depends upon whether they occur in a development or maintenance environment, and that recognition of the differences between the two phases will lead to more realistic measurement and work evaluation. Kemerer [49] suggest that the types of changes seen during longitudinal post-delivery studies are not homogeneous. Further empirical research may reveal predictable patterns of evolutionary change which would contribute to knowledge regarding the software lifecycle.

It is apparent therefore that there is some commonality of change process and activity shared amongst projects in phases termed development, evolution and maintenance. However, Bennett [50], Chapin [55] and Kemerer [49], advocate the benefit of differentiating between life-cycle phases. This is echoed by Ebert [58] and Thakuta [20], who emphasise the need for situation specific approaches to dealing with requirement volatility. Clearly specifying context, meaning and intent of the terminology used is important for research assessment and applicability.

2.1.2 Requirements Evolution, Change, Volatility, and Instability

In a recent systematic review of research on software evolution, Li [59] observed that terminology used to describe changes to requirements and software lack formal definition, and that there are at least five synonyms in common usage. These include creep, churn, change volatility and evolution. Therefore, the term 'evolution' can be used to define a life-cycle phase, a phenomenon that affects software, or an activity. Making an insightful distinction between software evolution which happens post implementation and requirements evolution which happens during the entire life-cycle of a software application, Ernst [60] further suggests that requirements evolution only include unanticipated changes and not adaptations or improvements. These types of changes commonly fall under the umbrella term 'requirements change' [61][13][62][18][22], though more commonly this term is accorded to the period of development, and includes additions, modifications and deletions. While volatile requirements are change-prone, the term volatility is often used as the metric to measure requirements change after it had happened, as is churn and creep, either during project development [12], or maintenance [48]. The terms instability and uncertainty can also refer to a measurement for the inherent potential of a requirement to change before the change occurs [13] [16]. The terms evolution, change and volatility, can also refer to changes occurring as a result of requirements change [63], or as a result of faults [64]. Thus studies investigating 'change-prone' or less stable software modules, may be referring to those that are likely to change to accommodate new or better understood customer need [14] [10], or that they are defect-prone [64], or both [65] [66].

Given the disparity of meaning, and potential for misunderstanding and assumption, the following discussion offers clarification of these terms when it is possible, to discern meaning in related research.

2.2 Empirical Analysis of the Nature of Requirements Change

Many empirical studies investigating the phenomena of software evolution (see [67]), concentrate upon changes made between consecutive software product releases. By comparison, there are few experience reports of requirements change during software development, though together they provide much insight as to the nature of requirements change that occur between elicitation and delivery. Such empirical studies report the results of either surveys or case studies which investigate the causes, consequences, timing or management of changes. Alternatively, studies may test specific hypotheses regarding techniques for change management. While the studies discussed here make use different empirical methods and often project specific data that has been collected for the purposes of project management rather than research, it is still possible to draw inference from their combined research efforts.

2.2.1 Causes, Consequences and Timing of Change

In a systematic review published in 2011, Bano [22] remarks that it is surprising how few empirical studies provide evidence concerning the causes of requirements change. Making no distinction between life the cycle stages of post-delivery and development, Bano identifies five studies, from which were extracted a list of causes. These are then characterised as ‘essential changes’ due to customer need, and ‘accidental changes’ that arise due to specification vagaries. Taken together, the causes of requirements change are varied and include customer and market driven demand, inaccurate specification, insufficient evaluation of the business case, and increased understanding of the software team. Alternative cause-based classifications are proposed by other researchers, and are discussed fully in section 2.4.

An empirical study of over 300 project managers revealed that projects with higher levels of maturity, such as those employing the capability maturity model at level 2 or higher, observed lower levels of volatility [68]. Use of process techniques such as requirement reviews also lowered levels of volatility, though somewhat surprisingly, prototyping raised levels of late requirements change. The use of ethnographic techniques was also associated with increased levels of volatility. Mirroring these results, Zowghi [69] found that variables such as frequent communications between users and developers, and usage of a definable methodology in

requirements analysis and modelling, have a positive influence on the stability of requirements. It was also observed that the involvement of higher numbers of user representatives increased levels of volatility. These results raise questions regarding the timing and level of user involvement for requirements elicitation. Interestingly, Thakurta [9] reports from survey data that projects employing a Waterfall process more often had high mid to late stage volatility by comparison to more agile techniques. However, it is also noted that decisions regarding process were often determined by the customer due to financial considerations or other organisational factors. Thus project managers are sometimes unable to engineer flexibility in developmental processes which would support effective change management [9]. Over half of the projects under consideration in this study employed a Waterfall process and less than 7% were 'fully agile'. The remainder adopted a hybrid approach with elements of both waterfall and agile. Importantly, of the 239 projects under investigation in this study, 23 failed, citing an inability to manage project scope as one of the main contributors. It was found that 84% projects were threatened because of requirement variation. From two sets of case study results, Chudge [70] reaches the conclusion that the relationship between the software developer and the customer is fundamental to the effective management of changes, such that disagreements and change escalation do not frustrate the course of project progression. While this research was undertaken a number of years ago, it is interesting to set these results beside recent observations by Thakurta [20] which illustrate the continuing significance of the impact upon project performance of relationships between customer and software provider.

Findings from both Ferreira [68], and Zowghi [69], indicate that volatility has a significant impact on schedule and cost. In a case study performed at Rolls Royce, Nolan [16] notes that between 25% and 69% of requirements will change between specification review and implementation, and that scrap and rework is, on average, 50% of the original estimated project cost. Interestingly, in this case study, only 16% of the changes were made by the customer, the remainder being generated by the development team. However, Nurmuliani [71] notes that changes 'external' to the system, that is, deriving from the business were significantly more expensive than internal changes, though they were less in number. In this case study, during the course of software development, the number of changes decreased, but the total cost of change remained constant. The implication is made that there is an increased average change cost as project work progresses and the cost of re-engineering rises. However, Weiss [72] observes that more than 75% of changes took a day or less to make, and that relatively few changes resulted in errors. In 1987, Boehm [73] wrote that finding and fixing a problem after delivery costs 100

times more than fixing the problem in early requirements phases, an observation which lead to the derivation of an exponential cost of change curve, and became the rationale behind thorough upfront requirements analysis. Beck [74] contradicts this, claiming a flattened cost of change curve, which other agile proponents explain by suggesting that the traditional sequential approach requiring documentation and change control upkeep, is the cause of the high escalation ratio [75], or that principles such as continuous design and refactoring ensure that change costs remain stable [76]. Until change can be measured such that the amplitude is independent of cost, and further empirical analysis undertaken, the reasons for a possible rising change cost will remain debateable.

Weiss [72] further discerns that while changes are non-localised with respect to individual components, they are localised with respect to subsystems, with the majority of changes being made in one or two subsystems. From a different perspective, Nakatani [77] consider that different types of requirements mature at different times in the development process, and recommend the categorisation of groups of requirements according to maturation.

In addition to impact upon cost and schedule, there are other equally compromising though less tangible consequences of change, such as compromised quality, or team morale. Javed [78] reports that design changes made during coding and testing have a significant impact upon the numbers of high severity defects that affect the major functionality of the system. Using simulation based on a causal model of requirements change, a comprehensive empirically informed study predicts the effects of requirements changes upon effort, schedule, requirements defects and team morale [79]. Though there is no indication of the models ability to accurately predict cost, its application of sociological factors such as morale, productivity rate and customer satisfaction, is in marked contrast to other research in this area, and highlights a potential limitation of purely cost-focused impact analysis.

2.2.2 Change Management

In a survey of project managers Thakurta [20] identified that 65.9% of the respondent organisations have an organisational plan for handling changing requirements that involved change control boards and change repositories. This percentage rose to 85% when only projects of maturity level of 2 and above were considered. Of 13 change management activities, the most commonly used was' the involvement of the customer', while 'architecting for change' was the least common approach taken. In a case study involving 246 projects in a single company in the telecoms domain, Ebert [58] showed that use of a single technique, such as requirements traceability had no impact upon schedule delays, and that only a combination of process related techniques and stakeholder involvement led to improvements to schedule over-run.

In a survey examining approaches taken by Irish project managers to managing the risk of changing requirements, Moynihan [25] identified 34 risk factors, and discovered that project managers determine the best means to manage changing requirements by a consideration of the type of risk being faced. Factors pertaining to the environment and organisation should be included in risk assessment in addition to product features. These findings are corroborated by MacCormac [24], who examined data from 29 projects to ascertain the effect of certain management techniques upon project performance. The most significant outcome was that approaches such as architectural flexibility or early market feedback had a positive influence upon project performance, but only in certain situations. The conclusion drawn was that the use of specific development practices is based upon their usefulness in resolving the specific types of uncertainty faced.

In collaboration with industry, Lam [80] provides a set of metrics and action plans that together form a framework for change risk mitigation and management. While practical guidance is offered and applicability considered, there is no evidence for the validation of such an approach in practice. Nolan [16] also recommends and outlines a risk management approach and makes the interesting contention that 80% requirements uncertainty can be predicted at project launch through careful consideration of the requirements and their environmental influences. Since this is refuted by other authors [27], further empirical investigation may reveal whether it is the method of risk handling, or significant software engineering contextual factors that facilitate such early prediction.

Taken together, the results of these studies clearly indicate that there are a number of different approaches to change management, and that managers adopt an approach that best reflects the character of the risk faced. However, the risks are varied and, as yet, there is no research which correlates technique effectiveness with specific types of risk.

2.3 Requirements Change Measurement

As mentioned briefly in section 2.1, there are two perspectives of the measurement of requirements volatility, each of which has a number of different means of measurement. The first is a measure of the potential of the requirement to change, which is often referred to as instability or change-proneness. The second is the measure of changes to requirements measured after the fact, which is most commonly referred to as volatility and can have varying scope from requirement, to component, to entire project. It is also used to refer to the measurement of changes at any point in the project lifecycle. This discussion focuses upon measurement of volatility after the fact in alignment with the goal of this research since analysis and prediction of requirements change will rely upon a measure that can be validated. There are two sources of research which promote understanding of volatility measurement: empirical studies that are based upon industrial data relating to volatility, and research that advocates the use of certain algorithms or methods to quantify change.

In empirical studies, volatility can be measured by a simple count of the number of changes (additions, deletions, and amendments), over a specified period of time. This has been used as a dependent variable in the prediction of volatility [12], to assess impact upon software defects [78], and as a means for risk assessment [81] [82]. However, the extent to which this metric is meaningful is questionable. Intuitively, change x costing £4,000 added to change y costing £150 yielding volatility = 2 is not that informative. Thakurta [9] notes that this is the most common way that practitioners measure change data, though they regularly use change cost for assessment and monitoring. Measuring this count relative to the number of original requirements is also common in industrial studies, and has been utilised in empirical surveys [68] , analysis of industrial projects [83] [84], as a recommended metric to support software maintenance management [85] and for change anticipation [13].

The case for including a dimension of amplitude in a metric for volatility is strongly made by Barry [48], who analysed change repositories holding 250,000 software maintenance changes in order to gather knowledge about change causes and patterns. Jones [15] proposes that a measure of volatility is indicative of the health of a project, and uses function point change as the basis of measurement. Furthering this work from a theoretical perspective, Kulk [86] recommends that a function point changes be calculated as a compound measure, relative to the changing size of a software system, since changes usually result in increasingly larger systems. An interesting approach was suggested by Nidumolu [87], who determined to find standard ways to assess project performance, and concluded that volatility should be measured along three dimensions: the number of changes, the level of diversity (how much the changes differ from one another) and the degree of analysability (how easy the change is to implement). This approach is approved by Zowghi [69], who assess the effect of volatility upon project performance, though there is little indication of how these measures may be used in practice. Alternative means of volatility measurement include an index of deep changes, such as database or architectural changes, to total change effort [88], or using a range of complexity metrics to describe the extent of change [89].

The intention of the more complex definitions of volatility is to add sematic value to the metric so that it is informative to project management. Broadly, a measurement of requirements change may be used to assess the health of a project, or inform requirements management.

2.3.1 Volatility and the health of a project.

Through an extensive empirical study, Kulk [86] examines Jones [15] discernment of a ceiling for healthy volatility indicative of a projects ability to succeed – around 5% – and concludes that this is a helpful measure only when certain project characteristics are present. A much higher level of volatility might be perfectly acceptable on projects whose developmental processes are focused not only on productivity, but also on value-enhancing change. In a study purposefully juxtaposed to those seeking to determine the negative consequences of change, Davis [90] investigates the negative effects of *not* making the change and makes the argument that as customer need changes, corresponding changes must be made to software requirements to lessen the risk of product failure. Therefore in order that a metric for requirements change reflect the health of a project it cannot be expressed exclusive of product value and project characteristics.

It seems plausible also, that there exists the notion of 'good' change and 'bad' change. Should the proportion of changes occurring due to poor communication, assumption, ambiguity or political conflict be higher than those resulting from business improvement or opportunity, this may indicate an unhealthy project regardless of the absolute measure of volatility.

2.3.2 Volatility and requirements change management.

Whilst accepting that a knowledge of volatility may support decisions regarding the need for flexible software development processes, a project-level metric will not be informative to the questions of architectural modularity, requirements prioritization, or elicitation technique selection. To this end, it will be necessary to determine which parts of the system are more likely to change, and in what way. Moynihan [25] argues that describing requirements uncertainty along dimensions such as uncertainty deriving from users, analysis, utilisation and *adding them up* is of little value since project managers use different strategies to cope with different uncertainty-related issues. Changes made in response to software usage, for example, would be managed differently to those made as a result of competitor activity. Studies designed to classify requirements change, beyond 'addition, deletion and modification [83] [91][80] confirm that requirements can change for different reasons and in different ways. Correspondingly, many different ways have been proposed to classify change.

2.4 Requirements Change Classification

Making no distinction between development and maintenance, Benestad [67] reviews the literature on software evolution, and notes that one of the primary objectives for empirical studies is to classify requirements change. A traditional classification of change during software development includes the categories add, modify and delete which is, in many cases, sufficient for the needs of research [68] [82][92][93][71]. A number of alternative classifications have been proposed, focused upon software maintenance, development, or both, which often have the intention of meeting different objectives.

Much empirical and theoretical work focused upon software maintenance reuses or builds upon Swanson's classification [94] which includes corrective, adaptive and perfective changes.

Chapin [13] provide a thorough review of literature referring to maintenance change types, and propose a comprehensive classification combining a focus upon activity (such as enhancement, performance improvement, and consideration of future maintenance) with what is being changed (such as interface, properties and business rules). This dual approach to classification echoes that of Kemerer & Slaughter [14] who intend their observations to contribute to a theory of software evolution and derive their change classification from empirical data of 25000 requirements changes. With the objective of providing a list of change types to support impact analysis, regression testing, estimation or software re-use, a number of classifications have been proposed which primarily concern what is being changed. These include domain specific business features [15], object-oriented elements (such as classes, interfaces, methods) [16], and a distinction between deep structure and semantic changes [95][96] [17].

By contrast, a classification which will support the needs of change anticipation must give consideration to the cause of the change. Harker [91] divide empirically gathered requirements changes into five categories depending upon the source of the change: i) fluctuations in the organization and market environment; ii) increased understanding of requirements; iii) consequences of system-usage; iv) changes necessary due to customer migratory issues or v) changes due to adaptation issues. Based on Harker's study, an appraisal by Sommerville [97] includes compatibility requirements relating to business process change in place of migratory and adaptation issues. Working from data held in a change control database within an industrial setting, Nurmuliani [83] catalogues volatility by type (addition, modification, deletion), origin, and reason for change. Noting that most change requests used in the study had little information about the reason for change, a further study was undertaken using card sorting to classify the recorded change [62]. This resulted in a list of 'super-ordinate constructs' classified by reason for change – product strategy, hardware/software environment changes, scope reduction, design improvement, missing requirements, clarification changes, testability and functionality enhancement. A later study [71] added change sources of internal and external. From a theoretical view point, Perry [98] discusses the dimensions of change and concludes that software development imitates the 'real world' by the creation of a 'model' from which we abstract an 'understanding of system requirements'. These, he states, are subsequently implemented upon a foundation of sometimes weak 'technical theory'. A genre of studies related to requirements engineering risk and uncertainty is also of relevance. Mathiassen [7] classify requirements engineering risks by reliability, complexity and availability, and relate these to appropriate techniques.

As can be seen, there are many different ways to classify requirements change, each designed to suit a particular purpose. It would seem at first sight that studies to date have little commonality. However, this may be due to the different contextual basis of the studies, or perhaps that classification was established at different levels. It is possible, for example, that Nurmuliani's [62] change reason of 'missing requirement' is included within Harker's [91]change source of 'increased understanding'.

2.5 Requirements Change Prediction

Research concerned with the prediction of volatility or change-prone (often defect-prone) requirements, often uses available source code metrics and is therefore suited to the prediction of volatility subsequent to the software having been written. Among such studies are Bhatti [64], who uses graph topology mining to analyse patterns of changes between subsequent releases of software to indicate more fault-prone modules. Similarly, and with promising results, Eski [99] analyses modifications to sequences of open source projects to discover the relation between object oriented design metrics and changes in software. Applying the resultant predictive mechanism to software during the last phase of software development facilitates the identification of change-prone classes, though no distinction is made between the types of changes that are being predicted be they functionality changes or defect fixes. A probabilistic approach is taken by Sharafat [33] who uses dependencies observed in UML diagrams in addition to source code metrics to predict change-prone modules in the next software release. Research seeking to broaden the range of factors that may help to identify change-prone modules post-release revealed that software maintenance managers observed clusters of changes which often coincided with previous periods of time when no changes were made [65]. While something can be learned about approaches to prediction, this research is of limited value to software development since it is focussed upon post-release software where a different change dynamic is observed [67].

Research attempting to identify change prone requirements during software can be found in the work by four groups of researchers:-

1. Bush [10] intends to provide decision support for "targeting design for change" by envisaging a number of possible future scenarios for each requirement relating to the goals and the environment of the software system. A stability assessment is enumerated based upon scenario confluence. Such a creative examination of potential changes to requirements is a valuable Requirements Engineering technique, though one that is yet to be validated as a means to predict volatility.

2. Postulating and testing the theory that different types of requirements are more or less volatile, Lim [13] asserts that categorising requirements can help identify change-prone areas. From the most stable to the most change-prone the categorization is: 'patterns', 'functional constraints', 'non-functional constraints', 'business policies and rules'. While there was some difficulty assigning a category to requirements, since they often comprised more than one, the volatile nature of business rules and in particular rules related to government policy was also affirmed by Maxwell [14]. In this series of studies Maxwell [14] successfully predicted changes to government regulation were successfully predicted, allowing requirements engineers and designers to focus upon the more stable product areas. Though this represents an important type of change, and has significant impact upon software development, there are many other reasons for software change that would need to be addressed in order to fully identify change-prone requirements.

3. Mao [100] engineered a Neural Network to predict the distribution of requirements changes in each of the life-cycle phases of a traditional waterfall project. The model assumed a static distribution of changes in projects complaint with a mature development process and was tested on a small project; it is unclear how this model would scale to larger industrial projects of varying maturity.

4. Loconsole [27][12] investigated correlation between volatility (as a measure of requirements change) and a number of metrics relating to use case documents. While no claim was made to causality, it was observed that the size and number of revisions of a document were significantly positively correlated with requirements volatility, implying

that modularisation of requirements documentation would reduce volatility. It is possible, however, that this correlation may be explained, for example, by the complexity of a requirement. In this case it could be the complexity that 'causes' both the increased size of the document, and the level of volatility.

By contrast to the predictive approach by correlation, a number of researchers are seeking to develop and evaluate more complex causal relationships [101][102][103]. In particular, Fenton[103][104], whose causal models were engineered to predict defect rates, argues that many models using small numbers of prediction variables ignore 'causal' factors such as programmer ability, or design quality. There have been attempts to identify factors that are causal to requirements change during software development which opens the way for predicting change early in the developmental life-cycle [68] [22]. These include 'softer' factors pertaining to process, people, product and environment, which, while subjective and therefore more difficult to validate, are becoming increasingly recognised as a valuable source of empirical data [4].

2.6 Predictive Models in Software Engineering

Research in software engineering has been exploring prediction systems, typically to estimate costs and defects, for over 40 years [105]. However, there is still some debate regarding their efficacy since there is no accepted 'best approach' and there are a plethora of prediction studies whose results are difficult to generalize to contexts other than those of the original study [106] [49]. This makes it difficult to assess results and encourage industrial application. Shepperd [105] argues that empirical studies must ensure that the results are at least better than guesswork, and Jørgensen [108][109] has written a series of articles to improve expert estimation on the basis that it is at least as accurate as formal models for prediction of software cost. Other studies have arrived at conflicting conclusions regarding the capability of formal predictive mechanisms to be similarly efficacious in different software development contexts. Kitchenham [110] examined the reliability cross company v's within company estimation models and found that of seven studies, three found no difference between results of cross company and within company predictions , and four found cross company results to be significantly worse. Zimmerman [111] ran 622 cross-project predictions using models based upon logistic regression, and found that

only 3.4% actually worked, though to be considered to 'work' required a success rate of at least 75%. They also observed that bi-directional testing was not successful, and models calibrated one on project may accurately predict another's defects, but the same is not true in reverse. In a much more optimistic review by Menzies [112], who examined the value of using static code attributes to predict defect measures, concluded that defect predictors yield a mean probability of detection of 71% and are therefore "demonstrably useful". Further, efforts have been made to improve the quality of empirical validation and encourage study replication beyond initial context [113] [114]. Menzies [115] makes recommendations regarding the goal of predictive models to admit to conclusion stability, where an effect that is seen on one project, is replicated in other situations. In a review of software fault prediction studies, Catal [116] stated that the use of machine learning algorithms rather than statistical methods or expert opinion give rise to better fault predictors since they are more easily transferable to new SE contexts and perform better with larger datasets.

However, often large datasets aren't readily available in practice, and one such machine learning technique, that of Bayesian networks, has an important advantage in the ability to build models using only expert opinion, data, or a combination of both [117]. Coupled with their visual appeal, their ability to reason in a forward direction (predict changes), and also in a backward direction (why might we have seen these changes?) answers to the need identified by Shepperd [105] and Penta [118] to attend to usability issues in predictive mechanisms. Consequently the use of Bayesian networks has increased [36]. Penharker [119] reported that Bayesian networks performed "competitively" against neural networks and regression trees for development effort estimation and in a comparative study Mendes [120] showed that they out-performed other methods (forward stepwise regression, case based reasoning, classification and regression trees) when applied to web estimation.

While not yet applied to requirements volatility, Bayesian networks have been used extensively to predict product factors related to quality and cost estimation [121] [37]. Recent Bayesian network implementations include Radlinsky [122] [123] who furthers the prediction of defects by integrating several factors relating to software quality, and Okutan [124] who uses the sensitivity analysis faculty of Bayesian networks to identify the most influential factors upon defect prediction. Wagner [125] predicts the effects of facts relating to the organisation, environment and system upon activities involved in software maintenance to produce a set of quality indicators. To predict the effects of changes, Mirarab [126] bases variables upon

software code elements such as classes and methods, in order to assess the propagation of changes from one to another. Bayesian nets have also been applied to schedule variance [127], cost estimation [128] [119] [38], and project velocity [129]. Radlinski [36] reports that less than half of Bayesian networks that are published are not validated using empirical data, and instead use scenario walk-throughs, or what-if analysis. Of those that are, relevant to this study are Fenton [37] who reports a 93% correlation between predicted and actual defect count models calibrated using only expert judgment, Hearty [129] who observed a Mean Residual Error of 0.51 which improved to 0.026 upon model training, and Stamelos [38] who produced a correct prediction in 52% of cases.

Fenton [35][39][37][130] undertook a number of studies to predict software defects that use a causal rational for variable selection and topology construction. There are good reasons to strive for a causal structure, the most significant of which is diagnosis – that is, when defects are observed, the models can be used to explain why that might be the case. Of importance for software engineering research, Fenton notes that a causal approach will increase the likelihood of better results when moving from one empirical context to another. Jensen [131] adds that we cannot predict the effect of interventions without a causal structure. In other words, if our network was used in a 'What if' capacity to determine the effect of changing the level of analyst skill, we will only intuit genuine affects if the nature of the links are causal. In the case of correlation between the size of a Use Case document and requirements volatility [12], intervention to reduce volatility by reducing document size, will not necessarily have the desired effect, since there are many ways to reduce document size, for example, the removal of pages, which are not necessarily causal to a reduction in volatility. Rather, it is some quality of the Use Case document which is the operating causal factor. Fenton [103] illustrates the difference between correlation and causality by investigating correlation between complexity and defect rate. Three conflicting results were obtained; positive correlation, negative correlation, and no significant correlation. To explain these results, the argument follows that without considering the effort with which testing is undertaken, the relationship between complexity and defect rate is arbitrary [104]. More time spent on good quality testing will result in fewer defects even in more complex modules. A causal emphasis upon model structure will therefore require the consideration of both subjective and objective data. Drawing a parallel between defects and change, it may also be the case that using techniques that 'drive out' change will result in more changes being discovered, and that less change-inducing methods will leave residual uncertainty to be discovered during user acceptance testing.

2.7 Problem Statement

The goal of this research is the prediction of software requirements change in the early stages of software development. This will provide an alternative to Requirements Engineering change management approaches, especially when more agile processes can't be employed. However, examination of current knowledge has revealed a number of areas of uncertainty, which may need to be addressed in order that derived models are based upon a sound understanding of requirements change. In addition, there are concerns regarding the practicality and relevance of such models which mean that industrial participation is important. Listed below is a summary of issues, challenges and opportunities identified in the previous discussion of related research.

RC1. Clearly define context and terminology.

Given the alternative usage of terms such as requirements change, evolution, development, maintenance, volatility etc. it will be necessary to first clarify the meaning and context inherent in the terminology used. This will determine research applicability and also simplify comparison with related research. While terms such as evolution, change and volatility are semantic concerns, a definition of software development and/or maintenance may benefit from contributions from industrial practitioners to ensure mutual understanding.

RC2. Investigate the causes of requirements change

Empirical studies have revealed a considerable number of causes of requirements change. It will be necessary to collate these in a coherent way for consideration as candidate influencing factors in predictive models. Since it has been observed that some requirements change more than others subsequent to software release, it seems plausible that certain requirements also are more prone to change during software development. However, post-release studies often rely upon source code metrics that are unavailable in early lifecycle stages. Other than a single study that makes correlation between a certain requirement type classification and volatility, there is no evidence for any particular characteristic of a requirement that makes it more susceptible to change. This will require further investigation in order to provide predictions of volatility for each requirement.

RC3. Derive an informative measurement of requirements change.

As discusses in section 2.2.2, a common view amongst researchers is that project managers manage change according to the nature of the particular risk faced. The provision of a prediction of volatility must therefore reflect the character of potential changes if it is to be informative to change management activities, process selection and scheduling. A possible way to achieve this is to express volatility as a profile construct consisting of a change type plus a quantitative measure of volatility. A classification based upon management technique would best support project planning, though one based upon the cause of the change would align with the goal of change prediction. Current change classifications are disparate and there is no indication that they are meaningful as a guide to action. In addition, there is no clear understanding regarding techniques that are better suited to particular types of changes. In order that a metric for change be indicative of the health of a project, some notion of change severity (cost or lines of code) must conveyed by the numeric portion of a change profile construct.

RC4. Examine feasibility of change prediction by formal model.

Since so few studies exist that attempt to predict volatility early in the project life-cycle, this could be because it is unfeasible to do to, either because change is entirely uncertain, or the models too complex. Existing research is contradictory, with some studies indicating that project managers are unable to forecast volatility, while another study reports that though risk assessment, it is possible to anticipate 80% requirements change. Further investigation of the phenomena of requirements change may reveal the extent that change prediction is possible, and therefore the constraints upon formal predictive models.

RC5. Build models relating causes of change to volatility.

Previous studies have shown that elements of process and aspects of relationships between customers and software providers affect volatility. However, it is interesting to note that customer involvement can both increase and decrease volatility. Model selection may therefore need to support both scenarios. Industrial support will illuminate such phenomena and guide model construction and calibration. It will be necessary to adhere to a causal framework as closely as possible to facilitate explanation as well as prediction and support the transition from one empirical context to another.

RC6. Validate models for accuracy and evaluate models for usability.

To ensure that results are practicable, viable and relevant, industrial collaboration should be sought for model validation. Since current research is ambivalent about the efficacy of formal models, it will be necessary to compare results with available alternatives. This may mean the implementation of alternative means of prediction, or comparison with project managers estimation of volatility. Also, consideration should be given to software engineering context, applicability of results, usability and cost benefit analysis.

An examination of related research in the area of software requirements change analysis and prediction has identified a number of challenges which will be addressed in this research. The following chapter derives research questions from these challenges and introduces the approaches taken to answer them which include empirical methods and Bayesian networks.

Chapter 3 Solution Approach

Having reviewed related research and identified a number of research challenges, this chapter introduces the research questions derived from those challenges, and the methods employed to answer them. A research framework sets out the process of theory building through empirical study, and subsequent Bayesian network construction based upon the results of these studies. Empirical methods are explained and an overview of Bayesian networks is provided. The research questions are summarised and presented alongside the thesis chapters, and methods which address them.

3.1 Research Framework

The research approach is illustrated in Figure 3.1. Overall, the approach taken is to employ empirical methods and knowledge engineering techniques to explore the phenomena of software requirements change and thus develop a theory upon which predictive models can be based. Bayesian Net models are then constructed in collaboration with industry, tested within an industrial setting and evaluated for usability and applicability. The results and implications of model validation and evaluation add to the theory of requirements change realised by this research.

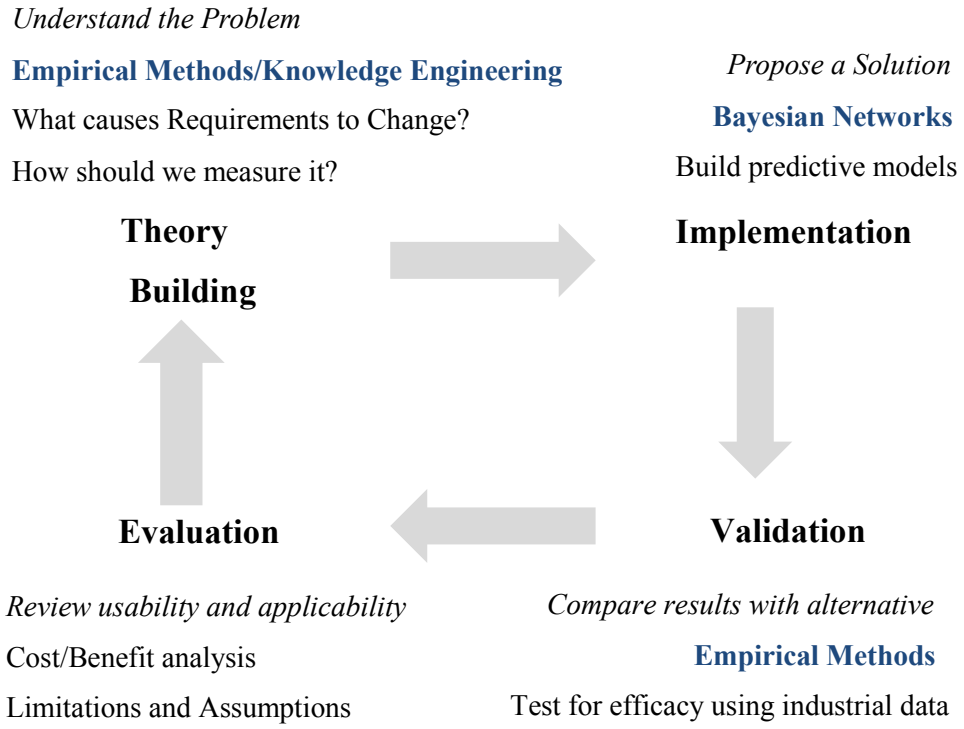

Figure 3.1 RESEARCH FRAMEWORK

3.2 Terminology

Due to the need identified in Chapter 2 to explicitly define terminology, the intention and meaning of the terms used are clarified thus:-

'*Change*' in the context of requirements refers to changes made to requirements (not software) for any reason. This includes errors in documentation/elicitation (not software errors), as well as changes arising from new awareness of customer need. It can apply to any type of requirement including those that deliver business functionality, as well as those that attend to quality issues such as maintainability or failure management.

'*Volatility*' is the name given to the metric for requirements change. Terms such as creep and churn are avoided.

'*Evolution*' in the context of requirements refers to changes to any requirement at any time during the life-cycle of the product from inception to demise. When used in the context of software, or system, it refers to a new release of previously released software

'*Uncertainty*' in the context of requirements refers to their disposition to change. This can also be referred to as the susceptibility of a requirement to change.

The terms Development and Maintenance are defined by industrial practitioners and their definitions are explained in Chapter 4.

3.3 Research Questions

Listed in Table 3.1 is a list of questions posed in this research, alongside the associated research challenges identified during a review of the literature. There is not a straightforward one to one relationship, since in some circumstances, a challenge is addressed through more than one question. For example, the challenge to examine feasibility of prediction (due to conflicting results from published studies) is in part responded to by increased understanding of change through classification (RQ2). In addition, this issue is reconsidered during model construction

(RQ4) and again following evaluation (RQ6). A brief description of each research question is followed by a discussion of the methods used to address them.

Table 3.1 RESEARCH QUESTIONS

Research Challenge	**Research Question**
Theory Building	
RC1 RC2	RQ1 What are the causes of requirements change?
RC1 RC3 RC4	RQ2 How can change be classified and measured in order that it is informative to project management?
RC2	RQ3 Are there attributes of requirements that make them more susceptible to change?
Implementation	
RC4 RC5	RQ4 Can we model requirements change in such a way that we can predict change?
Validation	
RC6	RQ5 Are model predictions more accurate than an available alternative?
Evaluation	
RC4 RC6	RQ6 How usable are the models?

Put simply, a solution to the problem of requirements change prediction would involve the provision of a mathematical model linking causes of requirements change to a measurement of volatility. The questions that arise from such a proposition are therefore concerned with i) understanding the nature of the cause, ii) the measurement of the effect, and iii) the relationship between the two.

Drawing from the challenges related to these three concerns, the first obvious question is 'RQ1- What are the causes of Requirements Change?' A classification based upon change source will facilitate future analysis of the relationships between cause and effect. Therefore the subsequent

question ‘RQ2 - How can change be classified and measured in order that it is informative to project management’ uses ‘cause’ as the focus for classification. The resulting classification is also examined from the perspectives of development and maintenance, thus addressing the research challenge of clearly differentiating between these two life-cycle stages. Evaluation of the taxonomy by case study provides evidence for whether this classification can be considered to be ‘informative’.

The third research question ‘RQ3 - Are there attributes of requirements that make them more susceptible to change?’ turns once more to the causes of requirements change, and examines the hypothesis that some requirements are more change prone than others. Should this be the case, in addition to causes relating to process and environment, there may be attributes of the requirements themselves that make them more susceptible to change. Requirements attributes that are considered by involved industrial practitioners to be influential to change are collected and examined for correlation with volatility.

The combined findings of these studies result in a comprehensive understanding of the nature of software requirements change. This informs the scope, and theoretically underpins the construction of predictive models which are designed in collaboration with industry practitioners in answer to the question ‘RQ4 - Can we model requirements change in such a way that we can predict change?’ Initial review by means of sensitivity analysis and model walk-throughs ensures that they perform as expected.

To answer the question ‘RQ 5 - Are they better than an available alternative?’ data from a further 3 case studies compares model accuracy with that of project management estimation. The final question ‘RQ6 - How usable are such models’ considers in detail the benefits, limitations, assumptions and cost of model usage and contributes to the important research challenge of ensuring that models are usable, practical and at least as good as any available alternative.

3.4 Empirical Methods

Perry [132] argues that although all software development is based upon some underlying process that contains specific stages, decisions regarding tools and management approach are not based upon hard evidence. Instead, managers base decisions upon personal experience, or organisational tradition. It has been noted that research in the field of software engineering lags behind what is happening in industry in certain areas [133], and that, as a discipline, it has failed to produce the models and analytical tools common in other sciences [132]. Shepperd [115] proposes that the use of empirical methods encourages software engineering to move from an advocacy based discipline to one that stands upon evidence. Perry [132] and Sjøberg [134] add that empirical research builds partnerships between industry and practice which ensures relevancy of research direction as well as the transfer of knowledge. Promoted by many researchers, including Basili [135], Kitchenham [136], Pfleeger [40], Fenton [137] and Wohlin [42], the employment of empirical studies in software engineering research has grown steadily in the last 20 years. Empirical research falls into two broad categories. Quantitative research is based upon numerical data which can be gathered through formal experiment, case study or survey and be analysed by statistical tests. Qualitative research is often description based, such as diary accounts or interview transcriptions. Each supports one or more of the following motives for empirical research in Software Engineering:-

1. Exploration.

   Exploratory research is conducted to discover more about a research question that is not yet clearly defined, and typically employs qualitative methods such as interviews, focus groups and pilot studies. Such studies seek to shed light upon a phenomena so that hypotheses and study design for subsequent in-depth studies can be defined [138]. This may bring to light unforeseen issues that should be examined [42]. Explorative research may also uncover the ways in which research can best answer to the needs of industry, and therefore steer research in directions that promote greater correspondence between research and practice [139]. Thus new methods and tools can be based upon the requirements of industry, just as software applications meet the requirements of their users.

2. Theory testing.

   When hypotheses and study design are clearly defined, quantitative methods such as surveys, case studies, and formal experiments facilitate the testing of hypotheses and promote theory building. This is mainly concerned with quantifying and testing a cause-effect relationships, or comparing two study groups for different behaviours [42]. When combined with qualitative methods, the findings of case studies and experiments may be strengthened and explained through what is known as triangulation [140]. That is, the same result is observed through different, unrelated methods. The results of such studies help build a theoretical basis for describing activities and phenomena related to Software Engineering.

3. Method and tool validation.

   Runesen [41] argues that the analytical research paradigm is not sufficient for investigating real life issues, and that empirical methods using data gathered during software development in an industrial setting are necessary for the validation of new tools and methods. This can be achieved by a combination of both quantitative and qualitative methods. Case studies of industrial projects can be the vehicle by which project data is collected and analysed to determine the efficacy of tool use. Post mortem analysis can employ a range of qualitative means, such as interviews or analysis of documentation to ascertain the value of tool or method usage [42].

4. Improve Software Engineering Work practices

   Empirical methods are also useful from an industrial point of view to inform project retrospectives thus promoting empirically founded learning which may complement the tacit knowledge acquired by practitioners during software development [42]. This encourages the transfer of knowledge through a company, and often leads to insights that affect future decision making. Project planning and process selection can be guided by empirical findings as well as organisational tradition, constraints imposed by customers and the personal preferences of practitioners.

A series, or combination, of studies is often necessary to answer a number of questions related to the goals of the research [132]. Sequential study families also allow investigation of related issues as they become known, and result in a deeper understanding of the important factors pertaining to a particular issue of interest [139]. In addition, they afford an opportunity to improve study design, data collection protocol and data quality, though experiential learning. Thus reliability and validity of results can be improved in succeeding studies [139].

The empirical studies involved in the theory building portion of this research comprise a family of studies whose collective aim is to understand the causes and effects of software requirements change. Preliminary exploratory studies determine scope and direction for the three subsequent empirical investigations. The empirical methods used in this research are described below; details of their execution, including organisational and project context are included in the relevant chapter.

3.4.1 Preliminary exploratory Studies

Unstructured interviews are those that seek to investigate a phenomena without any predetermined set of questions. The benefits of such an approach include the potential to discover important information that had not been considered relevant by the interviewer. In addition, they help to build rapport with industrial partners. However, the information gathered is not easily analysed or compared with that of other interviewees. For that reason, it is appropriate to use them in an exploratory context, where both researcher and practitioner are exploring the issues, determining scope and outlining research design. Questionnaires have standardised answers that make it easy to compile information gathered from a number of subjects. A pilot survey involves asking a small number of people to complete the survey and review that process to discover any difficulties and assess whether it is beneficial to proceed with a larger survey effort. Interviews and a pilot study are undertaken to assess the scope of research that would meet the goal of requirements change prediction, and also understand the influences upon requirements change and practical complexity of the software engineering environment.

3.4.2 Workshops

In requirements engineering, group elicitation techniques such as workshops aim to foster stakeholder agreement and buy-in [141], and are a mechanism whereby individuals can make collaborative decisions through leadership of a facilitator [5]. In Software Engineering research they are a useful medium for negotiation, debate and discussion amongst participants in order to reach a consensus of opinion regarding a particular issue. Workshops complement interviews and surveys that investigate an individual's experience or preferences. In situations where a number of practitioners contributed to the knowledge acquisition effort that would result in a single cohesive outcome (such as the classification of requirements change) workshops are employed.

3.4.3 Case Studies

Case studies are an empirical method aimed at investigating contemporary phenomena in their context [41]. Yin [142] states that "[a] case study should be used when a how or why question is being asked about a contemporary set of events over which the investigator has little or no control". They can employ either quantitative methods, where numeric data regarding a specific entity or situation is collected, or qualitative techniques that involve words, pictures or diagrams. Case studies are normally used in Software Engineering to monitor projects, or particular activities within the software development life-cycle [42], and can be used on an exploratory basis, or seek to investigate relationships between variables of interest. In the case where relationships are explored, key factors are identified and data is collected for a specific purpose. Collection protocol is defined prior to case study execution. Ideally project selection is achieved through random sampling of an industrial context of interest. However, in practice this rarely the case and instead projects are those that practitioners and organisations are willing to make available [143].

Runeson [41] review case study execution processes and terminology from other sources such as Wohlin [42] and Yin [142] and adapt them for Software Engineering research. They provide recommended practices and an extensive empirically derived and evaluated 50 point checklist to support case study research. These include items relating to the design of the case study, data collection, analysis and reporting. In as far as is practically possible, these guidelines have been

followed for all the case studies reported in this research. Two case studies are undertaken as part of the theory building stage of research. An additional 3 case studies are used for model validation. Case study context, design, data collection protocol and analysis procedures are described in the relevant chapter.

3.4.4 Card Sorting

Knowledge Engineering is a discipline that involves integrating knowledge into computer systems in order to solve complex problems normally requiring a high level of human expertise [144]. As such, there are range of tools and techniques that support the acquisition and formalising of knowledge. Card sorting is a knowledge elicitation technique which involves categorizing a set of cards into distinct groups according to a single criterion. Each card represents a construct which can be expressed in words or pictures, and participants are invited to place them into related groups. The categorisation may be left to the participants (open sort) or pre-determined (closed sort). Maiden & Rugg [5] suggest that card sorting is one of the most suitable techniques for acquiring knowledge of data (in contrast to knowledge of behaviour or process). Further, Rugg and McGeorge [145] argue that card sorting overcomes one of the disadvantages of other methods of categorisation since this uses Likert-type measurements to capture participant responses and is not well suited to nominal scale data. However, one of the challenges of card sorting is that it does not lend itself easily to statistical analysis [146]. Other more semantic disadvantages include the need for careful selection and naming of cards in order to ensure cross participant construct understanding, and the potential disparity of group labelling during open sorting. However, the use of extensions such the Delphi method (each participant iteratively improves a proposed hierarchy) [147] can overcome some of these difficulties. Most analogous to this approach is affinity diagramming which is similar to card sorting except that the focus is upon reaching a consensus, and therefore consists of a single card sorting exercise with a number of participants. However, by contrast to singular participant card sorting, taking this approach will mean that the differences in participant perspectives will be lost. Salient amongst the advantages of card sorting are its simplicity, focus on participants terminology, and ability to elicit semi-tacit knowledge [146]. A special edition of the journal ‘Expert Systems’ in 2005 was dedicated to the subject and it has widespread use in psychology, knowledge engineering and requirements engineering. Accordingly, single participant card sorting with supporting aforementioned workshops for terminology understanding and analysis consensus was deemed an appropriate approach to the derivation of a taxonomy of change sources.

3.4.5 Goal Question Metric approach and UML Modelling

The Goal question metric approach was first recognised by Basili [148] as a means to identify data for feedback and evaluation in Software Engineering. It places the focus for data specification upon identifiable goals, and therefore the data to collected is defined in a top down fashion. This method can be similarly employed in Software Engineering research for empirical studies to answer particular questions that address research goals, and is used to identify the data for the first two case studies.

The Unified Modelling Language (UML) is a general purpose suite of modelling notations that can be used to create a visual representation of a software system or domain from a structural or behavioural perspective. Of the available diagrams, the object diagram and activity diagram are used during the first case study to capture the organisation of requirements, and the activities that translate them from concept to software. These diagrams are not an end in themselves, but are used to support understanding and facilitate communication with industrial partners, which are two of the benefits of UML modelling [149]. For further discussion of the use of UML modelling in Software Engineering and an overview of the different diagrams, see [149].

3.4.6 Validity of the Results of Empirical Methods

Following Yin [142], Runeson [41] insists that the limitations of study validity are explicitly stated alongside study results. Four aspects of validity are identified which may limit the trustworthiness of the results. These are listed below, and referred to as necessary in the discussion section of each chapter describing an empirical study:-

1. *Construct validity* reflects the extent to which the data specification is similarly understood by all parties involved, and accurately meets the needs of the research question. Ensuring common understanding between participants can be addressed through interviews, data reviews, and piloting of the research instrument (such as questionnaires). Employment of the Goal Question Metric approach described in section means that the data is specified in accordance with research goals and objectives.

2. *Internal validity* is of concern when causal relations are examined, given that a third invisible (confounding) factor which is not included in the study may affect the results. As a rule of thumb, conclusions based on experiment design that manipulates the independent variable, rather than examine correlation have a higher internal validity. For studies based upon correlation, it is important therefore to consider uninvolved circumstances or situations which may be responsible for the effects seen.

3. *External validity* is the extent to which the results of a study can be generalized to other situations or software engineering contexts. This is a particular challenge for empirical software engineering research, since comparison between industrial context is difficult due to the complexity of factors involved. Study replication in alternative contexts is the only way to widen external validity. Ideally this study should be replicated firstly within a similar context for results comparison before applying it in alternative software development environments. Subsequently widening software development context reflects Sjøberg's recommendation [134] to "formulate [research] scope relatively narrowly to begin with and then extend it gradually".

4. *Study reliability* is defined as "the extent to which the data and analysis are dependent upon specific researchers." [41]. If a study were to be repeated by another researcher, the results should be identical. Threats to this aspect of validity are, for example, if it is not clear how data was specified and collected. Data collection protocol and specification is detailed for each case study undertaken.

3.5 Bayesian Networks

There are number of alternative techniques that are used for prediction in Software Engineering, including methods based upon machine learning and those reliant upon statistical procedures. The next section provides a review of the literature (until 2010 when technique selection was made) that compares predictive techniques, and discusses the rationale for the selection of Bayesian Networks as the means to predict software requirements change. An overview of Bayesian Networks is illustrated using a fictional model.

3.5.1 Prediction Tools and Methods Comparison

A number of studies compare the techniques used in SE prediction models [150] [103][151] [152]. Table 3.2 provides a summary of the most commonly used techniques, their advantages and limitations. Case based reasoning (CBR) , Delphi and Work Breakdown Structure (WBS) are all performed by experts. In the case of CBR, previous software is used in an analogous capacity to help determine new estimates, while Delphi and WBS focus solely upon the current system. The Delphi method arrives at estimation through group consensus, whereas WBS breaks down work products to progressively smaller units of work in order to estimate at a high level of granularity. Regression analysis is a statistical method that examines relationships between variables, and having determined a formulaic correlation, applies that to new situations where some of the variables are unknown.

All of these authors [150] [103][151] [152] agree that formal methods moving from simple regression based models towards those that are able to incorporate complex non-linear relationships are more likely to yield more accurate results. However, there is no unanimous defence of a single technique. Boehm [150] refers to Bayesian Networks (BN's) as a composite technique combining the advantages of standard regression with the (non-expense related) benefits of CBR and models based upon expert judgement. This is because a distinctive feature of Bayesian networks is that they make inferences using both sample data and expert judgement. In view of the overall goal of this research, a brief comparison will focus upon commonly used formal methods of estimation that handle complex non-linear relationships, namely neural networks, Bayesian networks, and system dynamics.

Table 3.2 COMPARISON OF PREDICTION TECHNIQUES

Technique	Estimation Application	Advantages	Limitations
Case Based Reasoning	Effort	Akin to human thinking	Case Data hard to acquire
Expertise Based (Delphi, WBS)	Cost	Inexpensive	Depends upon 'expert' availability and skill
Regression (Least Squares, Robust)	Cost, quality, volatility	Simplicity Ease of use,	Requires linear relationships, Fits model to historical data, Needs large datasets
Neural Networks	Cost, Project Schedule escalation, Reliability	Accuracy, Non-linear variable interaction	Over-fitting, Invisible reasoning process, Large training datasets
System Dynamics	Cost, Change Impact, Staffing needs, schedules	Two-way variable interaction, Dynamic changing factor values	Difficult to calibrate
Bayesian Belief Networks	Cost, Quality, Risk, Process Selection	Models Uncertainty, Backward and forward evidence propagation, Combines expert judgment with data	Conditional probability definition time-consuming, Acyclic

Neural Networks are the most commonly adopted alternative to regression for software cost estimation. Most commonly used are 'feed-forward' networks which are acyclical with information passing in a forward direction [153]. A number of input variables are related to a number of output variables via one or more hidden node layers. The network learns by adjusting the weights to decrease the distance between its predicted output and the actual output of an

existing dataset. Thus 'trained' the model can be used with new sets of input data to produce output predictions.

Baysian Networks (BNs) [154] provide a decision-support framework for problems involving uncertainty, complexity and probabilistic reasoning. A directed acyclic network connects nodes representing variables via arcs representing dependencies between variables. Each variable has a pre-defined set of allowable values and associated with each is a conditional probability table denoting inter-variable dependency. With BNs, it is possible to articulate expert beliefs about the dependencies between different variables and to propagate, in both a forward and backward direction, the impact of evidence on the probabilities of uncertain outcomes, using Bayesian probability theory.

System Dynamics [152] is a continuous simulation modelling methodology whereby model results are displayed as graphs of information that change over time. Models are represented as networks with positive and negative feedback loops between nodes representing variables. Variables dynamically change according to the flows along the arcs (feedback loops), at a rate determined by pre-specified calibration.

Despite the accuracy of Neural Networks, – Wittig has reported estimation accuracies of within 10% for effort estimation - Gray [151] maintains that they may be subject to the same statistical problems as those of regression such as there being too many free parameters for the data size. It has been suggested by Idri [153] that the main reason for their lack of industrial use is that the mechanism linking cause to effect is invisible to a user. Project managers have a tendency to mistrust cost prediction models unless the predictions either reflect their prior belief or the reasoning is transparent and intuitive [108]. Hence presenting a prediction of requirements change without explanation of the underlying reasoning may be prohibitive to project management trust. Subsequent research addressing this problem means that it may now be possible to provide evidence of a neural network's working transition [155]. However, this ability is already inherent within BN technology. An advantage that BN technology offers over that of neural nets is that reverse inference is also possible. Therefore, given evidence of a high level of, say, change due to poor communication, the probability that the 'wrong user' is involved can be determined.

System Dynamics (SD) also provides mechanisms for capturing uncertainty related to complex systems [156] and there has been increasing interest in exploring its use in the study, control, and management of the software development processes [157]. However, they are particularly difficult to calibrate and results can be misleading [150]. They are also subject to the same validation challenge that results from calibration of a model in one context and subsequent application in another. However, a BN's capacity to learn project specific relationships will address this validation issue. By contrast to costs and defects whose actual values become known following development, change requests begin to arrive early in the project lifecycle, increasing the potential to learn relationships from data early in the project life-cycle.

Fenton [103] compares a number of techniques and concludes that BN's more readily incorporate software process concepts and are adaptable to data availability and quality issues. Fundamentally they reflect the notion inherent in Bayesian statistics of a focus upon the nuances and knowledge of a particular environment, including expert opinion, rather than inferring behaviour entirely from aggregated general statistics. As noted by Singpurwalla [158] software is a one-of-a-kind entity, 'almost identical conditions' required for the application of frequentist statistics don't exist. "The objective nature of frequentist statistics is anathema to the spirit of intuition and inspiration that is necessary for addressing software engineering problems. Personal experience is vital" [158]. As discussed in section 2.6 'Predictive models in Software Engineering', BN's have been seen to perform competitively against available alternatives [119].

The rich functionality afforded by Bayesian nets means that not only are they a coherent and mathematically sound way of reasoning with uncertainty in both a forwards and backwards direction [159], but they also facilitate domain exploration through 'what if' scenarios and sensitivity analysis. These faculties, coupled with visual appeal and an ability to handle missing data, make them an attractive solution vehicle for the Software Engineering environment [160]. As a summary, the benefits of Bayesian networks underlying their suitability for the prediction of requirements change are :-

- Explicit quantification of uncertainty
- Modelling causal relation between variables
- Combining prior knowledge and data.
- Enabling reasoning from effect to cause as well as from cause to effect
- Possibility of overturning previous beliefs in the light of new data (learning)

- Ability to make predictions with incomplete data
- Combining subjective and objective data
- Enabling users to arrive at decisions that are based on visible auditable reasoning.
- Specification of complex relationships using conditional probability statements
- Use of ‘what-if?’ analysis and forecasting of effects of process changes
- Easier understanding of chains of complex and seemingly contradictory reasoning.

3.5.2 Overview of Bayesian Networks

Bayesian Networks (BN’s) are now fairly well established as practical representations of knowledge for reasoning under uncertainty, as demonstrated by an increasing number of successful applications in domains such as medical diagnosis and prognosis, planning, vision, information retrieval, and natural language processing [161]. A BN consists of a graphical structure, encoding a domain's variables (nodes) and the relationships between them (arcs). As a directed acyclic graph, the arcs are directed such that nodes are hierarchical and may have both children and parents. Each node has an associated set of allowable mutually exclusive predefined categorical or discretised continuous values. A probability of occurrence is assigned to each value which represents the degree of belief that a variable will take a certain value. Associated with each node is a conditional probability table which captures the relationship between a node and its parents. When the value of a variable is known, probability calculus and Bayes theorem is used to propagate this evidence throughout the network, thereby updating the strength of belief in the occurrence of the unobserved events.

Illustrated by a simple fictional example in Figure 3.2 and Figure 3.3, a small Bayesian Network contains variables pertaining to requirement changes, each of which can take one of a set of predefined states; in this case the states are representative of a Likert scale from Very Low to Very High. In Figure 3.2 no evidence has been entered, and the relationships between nodes have been defined such that without evidence both ‘specification accuracy’, and ‘number of changes’ are most likely to be Medium. Evidence has been entered on the greyed-out nodes in Figure 3.3, and this evidence propagated to all other nodes in the net. In this example, evidence of a very low review quality coupled with a very low specification accuracy results in a 70.2% probability that the number of changes will be ‘Very high’. Also, this evidence has propagated in a backwards direction and the probabilities of each parent node have also been revised. In this

case, a ‘Very low’ specification accuracy increases the belief that the complexity and novelty are ‘High’ or ’Very high’, and that the analyst skill is ‘Low’. The revised beliefs are known as ‘posterior probabilities’. This propagation is affected through the nodes’ conditional probability tables (CPT), an example of which, for the node ’Number_of_changes’ is illustrated in Table 3.3. Each combination of particular values of the parent nodes ‘specification accuracy’ and ‘review quality’, yield a probability for each of the possible states of ‘number of changes.

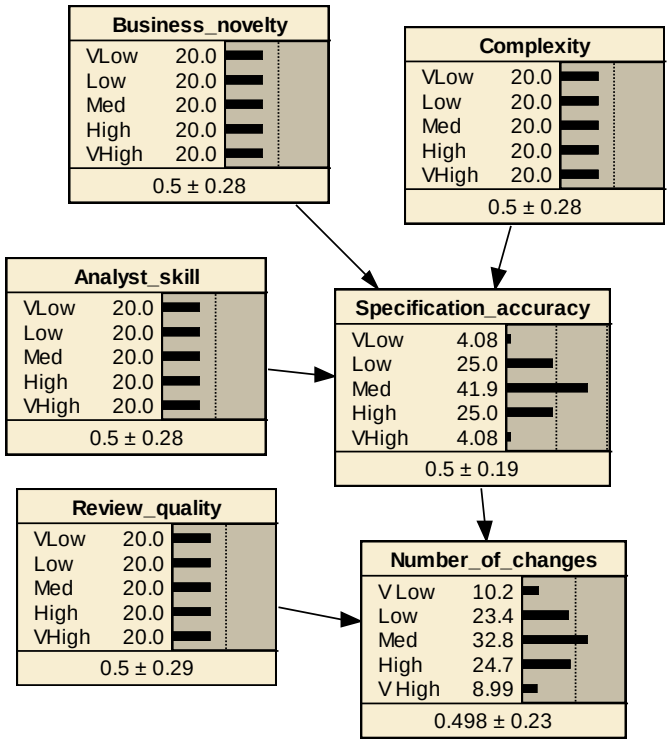

Figure 3.2 BAYESIAN NETWORK WITH NO EVIDENCE

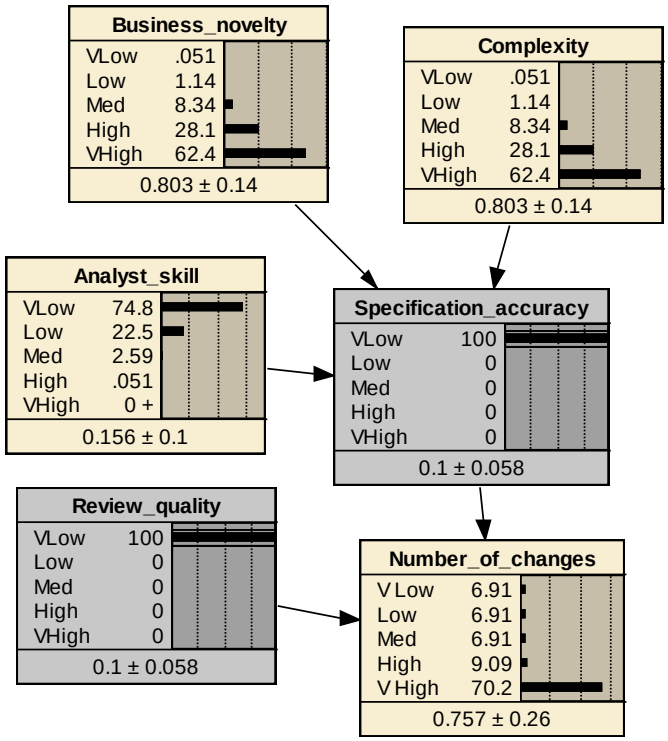

Figure 3.3 BAYESIAN NETWORK WITH EVIDENCE

Table 3.3 EXAMPLE CONDITIONAL PROBABILITY TABLE (CPT) FOR THE NODE 'REQUIREMENT UNCERTAINTY'

Specification_accuracy	Review_quality	V Low	Low	Med	High	V High
VLow	VLow	6.912	6.912	6.912	9.089	70.175
VLow	Low	6.67	6.67	6.675	35.959	44.027
VLow	Med	6.667	6.667	12.439	64.35	9.877
VLow	High	6.667	6.687	41.664	38.304	6.678
VLow	VHigh	6.667	8.263	69.24	9.163	6.667
Low	VLow	6.668	6.668	6.762	47.356	32.545
Low	Low	6.667	6.667	9.676	68.925	8.065
Low	Med	6.667	6.711	41.121	38.825	6.677
Low	High	6.667	8.896	63.095	14.676	6.667
Low	VHigh	6.677	35.48	44.393	6.783	6.667
Med	VLow	6.667	6.667	10.842	63.622	12.202
Med	Low	6.667	6.68	38.96	41.005	6.688
Med	Med	6.667	11.831	65.924	8.911	6.667
Med	High	6.67	27.996	51.789	6.878	6.667
Med	VHigh	11.649	64.644	10.374	6.667	6.667
High	VLow	6.667	6.752	38.625	41.219	6.738
High	Low	6.667	9.708	66.41	10.549	6.667
High	Med	6.726	38.812	41.099	6.697	6.667
High	High	9.269	66.316	11.081	6.667	6.667
High	VHigh	34.24	45.678	6.747	6.667	6.667
VHigh	VLow	6.667	8.609	68.28	9.777	6.667
VHigh	Low	6.673	37.626	42.344	6.69	6.667
VHigh	Med	10.221	67.52	8.926	6.667	6.667
VHigh	High	49.933	30.023	6.703	6.671	6.671
VHigh	VHigh	65.372	14.356	6.757	6.757	6.757

From probability theory, if variables *A* and *B* are not independent then the belief in *A* given that *B* is known is the conditional probability:-

$$P(A|B) = P(A, B) / P(B).$$

In this nomenclature, P (A|B) means the probability of A given B, P (B|A) means the probability of B given A, P (A) means the probability of A and P (B) means the probability of B. A and B can be events, states, conditions or anything to which a probability can be assigned. This formula simply shows the degree of belief in the state of *A* when the state of *B* is known. Likewise, the probability of B given A can be calculated in the same manner, yielding what has come to be known as Bayes Rule:

$$P(A|B) = P(B|A)\, P(A)/ P(B)$$

Equation 3.1 BAYES THEORUM

It is this formula that forms the basis of the underlying mathematics which calculates the related joint distribution of probabilities over all nodes in the network, given 'evidence'. This is commonly referred to as probabilistic inference and there are many algorithms available within Bayesian Net tools to do this [162]. They rely on an assumption related to the relationship between the variables that may be explained as "When the present is known, the future is independent of the past". To illustrate, while none of the states of the variables in Figure 3.2 are known, having increased certainty in 'business_novelty' will change our belief in 'specification_accuracy' which will further change our belief in 'number_of_changes'. However, when the state of 'specification_accuracy' is already known, subsequent knowledge of 'business_novelty' will not have any affect upon our belief regarding the 'number_of_changes'. Effectively, this path of probability propagation is blocked by the knowledge of 'specification_accuracy'. This is formally known as conditional independence, and it supports probability partitioning and algorithm efficiency. The conditional independencies evident in the net structure must be reflective of the real world in order that posterior probabilities can be trusted. While, for meaningful results, the assumption of conditional independence must be reflected in the real world represented by the model structure, the arcs need only reflect variable dependence rather than cause-effect relationships. However, there are good reasons to strive for a causal structure, the most significant of which is diagnosis – that is, when a level of

requirements uncertainty is observed, we can use the model to explain why that might be the case.

Broadly, there are two distinct components of Bayesian network Design - Structure definition, and the specification of the CPT's. In practice they are often intermingled as examining the conditional independencies can constrain the way the real world may be represented, and in conditions where the number of rows in a CPT is too large, this may result in a review of the structure. There is much research concerning algorithms which can learn both structure and/or CPT's from data. However, as has been noted, data in sufficient quantity or quality is rarely available in the SE environment, meaning that structure definition is most often achieved through the use of experts [36]. This involves dependency modelling of the domain in question, often through the use of white-boards. This task of eliciting from domain experts the variables of importance and the relationships between them is comparable to knowledge engineering and, although it may require significant effort, is generally considered achievable [163]. There is no established defined methodology for Bayesian network design, and it is generally agreed that there is no one 'correct' model [164]. Often an iterative approach is taken until model structure most closely reflects the real world given the algorithmic constraints [165] Most Bayesian networks in the SE environment are structured using expert judgment [36], though notable exceptions are Okutan [124], who derived structure from data in an open source repository, and Bibi [166], who used data from the available COCOMO dataset. Fenton [167] notes that Bayesian networks constructed using expert guidance significantly outperformed those built by data alone. There are alternative approaches to manual variable selection and topology development. While most authors include factors relating to process, product and people, some make topology design decisions based on some aspect of the software development life-cycle [128] [129]. Yet others base variables upon software artifacts [126], or activities [125].

In principle the data to generate CPTs can come from the environment, or from literature sources. However, often the variables selected as the most influential have not been collected or previously examined in other research and/or have been taken from populations whose characteristics differ seriously from the domain in question [163]. While it is possible that data may be collected following structure definition, Radlinsky reports that in two thirds of cases, experts are involved in model CPT specification [36]. This can be time-consuming since, as can be seen from Table 3.3, a node with 5 values and 2 parents requires 125 probability values, and this number increases exponentially with the number of parents. Fortunately there are ways to

reduce the effort, which fall into three main categories. The first involves reducing the number of parents by introducing 'latent' variables, thereby changing the structure. The second uses probability distributions which are arrived at through calculations which are often based upon parental independence of causal influence. The third uses techniques such as the probability wheel to help experts visualize the probabilities involved [163].

3.6 Summary of Research Questions and Methods

A summary of the research questions described above in section 3.3 can be found in Table 3.4, alongside the methods used and the Chapter that describes the research conducted to answer the question. Under the research questions, in italics, is the product of the research effort, which informs subsequent research.

Table 3.4 RESEARCH QUESTIONS, METHODS AND CHAPTERS

Chapter	Research Question	Method
4	RQ1 What are the causes of requirements change? RQ2 How can change be classified and measured in order that it is informative to project management? Research Outcome: *Requirements Change Taxonomy*	Literature Review, Preliminary exploratory research (interviews, pilot survey), Card sorting, Workshops
5	RQ2 How can change be classified and measured in order that it is informative to project management? Research Outcome: *Requirements Change Taxonomy evaluated*	Case Study 1
6	RQ3 Are there attributes of requirements that make them more susceptible to change? Research Outcome: *Layered Causal account of Software Requirements Change*	Case Study 2
7	RQ4 Can we model requirements change in such a way that we can predict change? Research Outcome: *Bayesian net models to predict change*	Bayesian Networks Workshops

Chapter	Research Question	Method
8	RQ5 Are model predictions more accurate than an available alternative? RQ6 How usable are these models? Research Outcome: *Bayesian Net Models validated* *Theory and Models evaluated for usability*	Case studies 2, 3, 4 & 5 Post Mortem Analysis

Having derived the research questions from challenges identified in the literature, and presented the means by which they will be addressed, the next five chapters describe the research undertaken, the results and a discussion in light of relevant research.

Chapter 4 Change Classification

Toward the ultimate research goal of requirements change prediction, this chapter describes the first of a series of empirical studies that investigate the causes and consequences of software requirements change. By combining evolution, maintenance and software lifecycle research, a definition of software development and maintenance is derived, which provides the foundation for empirical context and comparison**.** Previously published change 'causes' pertaining to software development are elicited from the literature, consolidated using expert knowledge and classified using card sorting. The resulting change trigger taxonomy constructs are initially validated using a small set of requirements change data. Methods, execution protocol and results are presented as well as a discussion of findings in light of related research. This chapter addresses the following research questions:-

RQ1 What are the causes of requirements change?

RQ2 How can change be classified and measured in order that it is informative to project management?

Chapter Contributions to Academic Knowledge

a) Requirements change classification.
   Through empirical study a cause based change classification was derived which provides a means to monitor, understand and analyse volatility.
b) The causes of requirements change
   A more exhaustive inventory of the causes of requirements change has been produced, including those in the environment, and attributes of the requirements themselves.
c) Comments on the feasibility of volatility prediction.

4.1 Introduction

For the purpose of change management, it is generally recommended that change requests are held in a database with attributes such as 'origin' and 'change type' [8]. Therefore, an obvious starting point in the investigation of the causes of software requirements change would be to analyse existing change control databases. However, it has been observed that reasons for change are insufficiently recorded *for the purpose of analysis*, and there is no standard 'list' of causes that is commonly adopted in industry [9]. Standardizing data collection across multiple projects regardless of life-cycle phase will not only satisfy research purposes, but also provide a means by which industrial software providers can take ownership of empirical opportunities. The first objective of this research is therefore to identify a common set of change cause constructs.

It is commonly understood that classifying requirements changes will inform project management decision making and process review [67]. In a review by Benestad [67] three primary objectives for empirical studies of requirements change are identified, among them the classification of software evolution. A traditional classification of change during software development includes the categories add, modify and delete. A number of alternative classifications have been proposed, focused upon software development, maintenance, or both, which often have the intention of meeting different objectives. These are discussed in section 2.4 'Requirements Change Classification'. Due to the divergence of change sources compiled in these studies, none of the classifications could be considered exhaustive and did not meet the needs of this research. Therefore, the second objective of this study is to classify the change source constructs in such a way that is reflective of industrial need and understanding. Previous requirements change classifications, together with requirements change causes derived from empirical studies provide a collection of change constructs upon which to base change classification. This classification of requirements change sources should be useful as a pick-list (along with other pre-defined attributes) in change diaries across multiple projects for the purpose of analysis in line with the research goal of requirements change prediction.

Viewing software evolution as a continuum from conception to demise is a perspective purported by some researchers [45], though much empirical effort is bounded by a clear distinction between initial development and post-implementation [34][44][35]. Pfleeger's [7]

recommendation that "We must find a way to understand and anticipate some of the inevitable change we see during software development" is complemented by Bennett and Rajlich's [35] encouragement to focus upon empirically founded predictive models of maintenance. A third objective is to examine the causes of requirements change to assess their relevance to software development and maintenance. This will firstly require that the terms 'development' and 'maintenance' be defined. Initial definitions derived from a combination of literature sources is reviewed by our industrial partner. Change source constructs and classification are examined in light of these definitions.

Readers should note that the results of this study were published prior to the systematic review conducted by Bano [22], in which the taxonomy presented in this chapter is cited (see the preface for papers published as a result of this research).

This study sets out to meet the following objectives:-

1. Define the terms 'maintenance' and 'development'.
2. Identify the sources of requirements change such that future analysis is possible.
3. Provide a classification of requirements change in line with the goal of change prediction.
4. Examine classification relevance for both software development and maintenance.

4.2 Empirical Context and Execution

This study comprises a series of empirical investigations in collaboration with an industrial partner; between 1 and 6 project managers were involved at each stage. Our industrial partner in this research employs 300 staff, has offices in England and Ireland, and delivers IT solutions to clients across both the public and private sectors. Most of their contracts involve a single customer and roughly 80% of these relate to government contracts. Of importance to collaborative research, their involvement is supported by both upper and middle management, and reflects their stated initiative to become a centre of project management excellence.

4.2.1 Empirical Methods

To explore the scope and complexity of the problem, and decide upon appropriate and effective research methods, a number of initial investigations were undertaken. Three unstructured interviews, during which project managers in the main reflected upon their current project, demonstrated the need for a focus for 'memory-jogging'. A subsequent self-administered questionnaire exposed difficulties with change construct interpretation and understanding, and a review of a change database revealed that not all changes were recorded, particularly those relating to the technical solution. Therefore, methods were sought that would maximise the opportunity for consensus building, provide a visual basis for brainstorming, and maximise the potential for knowledge sharing and exchange.

In requirements engineering, group elicitation techniques such as workshops aim to foster stakeholder agreement and buy-in [141], and are a mechanism whereby individuals can make decisions through the consensus building leadership of a facilitator [5]. In view of this, they were used to familiarize all participants with the constructs, come to a common understanding of their meaning, and reach a consensus of opinion at the end of the study regarding the structure of the taxonomy to be used. The unit of analysis is our industrial partner organization. Participants were sampled from the company's Project Managers and Maintenance Engineers by convenience, within the stratum of those with at least 12 years of experience in IT.

Single participant card sorting with supporting workshops for construct consolidation and consensus building was the approach taken to derive a taxonomy of change sources. See section 3.4 for details of interviews, focus groups and card sorting.

4.3 Software Project Categorisation

In order to accommodate and compare sources of change pertaining to all phases of the software lifecycle, it is first necessary to clearly define what we mean by development and maintenance. Noticing that there is some terminological disparity in the literature [50], a character based project categorization is founded upon existing studies. It is from this basis that understanding is established between academic and industrial research team members, and it also provides the context for cause construct consideration and comparison.

4.3.1 Initial proposal from Literature review

While an exhaustive account of the comparison between development and maintenance is out of the scope of this study, the categorisation illustrated in Table 4.1 was derived for the purposes of this and future empirical studies. It combines the staged model proposed by Bennett & Rajlich [50], Kitchenham's maintenance scenarios [53] and Chapin's classification of change types [55]. The division between development and maintenance was drawn to reflect the importance of the factors relating to team knowledge, stability and responsibilities [53] [57], coupled with the distinct contractual governance prevalent during 'product upkeep' and 'servicing'.

As shown in Table 4.1, the broader distinction between software development and maintenance is further categorised into two development scenarios and two maintenance scenarios which derive from the staged model of Bennett [50]. Development can either be initial development before any code is delivered to the customer, or iterative delivery, when there is a portion of the software system in development, and a portion in usage. Though Kitchenham [53] makes no distinction in types of maintenance projects Bennett [50] propose that maintenance can be in a stage of product upkeep when both requirements enhancements and corrections are undertaken, or in a phase referred to as servicing when only corrections are made. Both maintenance scenarios fall under a contractual arrangement with a customer for administration and change scheduling and costing. The knowledge requirements are different for each scenario, with domain and technical requirements only assumed and required for both development scenarios. Staff roles are clearly delineated as being development or maintenance, with the exception of iterative delivery, when staff can be involved with the initial development of software modules or with changes to those that have been delivered. User support is a significant feature of both types of maintenance projects, but is only a concern of development during iterative delivery, when feedback is the mechanism for future changes.

Table 4.1 DEVELOPMENT AND MAINTENANCE PROJECT CATEGORISATION

	Development		**Maintenance**	
	Development	Iterative Delivery	Product Upkeep	Servicing
Naming Convention	Initial Development[1]	Evolution[1] Maintenance Scenario A[2]	Evolution[1] Maintenance Scenario B[2]	Servicing[1] Maintenance Scenario B[2]
Staff Roles	Pre-delivery only	Pre and Post-delivery.	Post-delivery only	Post-delivery only
Software Engineer Knowledge	Domain and project-specific technical knowledge inherent	Continuity of domain and project-specific technical knowledge.	Some Domain and project-specific technical knowledge required but not assumed.	Domain and project-specific technical knowledge not required or assumed.
User Support	N/A	Feedback through requirements analysis activities	Help/Support Desk Service Level Agreement	Help/Support Desk Service Level Agreement
Types of changes	All types	All types	All types	Corrective

[1] Bennett & Rajlich [35]
[2] Kitchenham [46]

4.3.2 Project Categorisation Clarification

With one project manager present, the proposed project categorisation was reviewed. Two post-delivery support contracts were examined and it was noticed that small changes termed ‘enhancements’ were permitted under the terms of both contracts provided that they did not exceed an agreed (contract-specific) cost ceiling. These would be undertaken by a member of the organisation’s maintenance team and scheduled in accordance with maintenance priorities.

Provision was made in both contracts for further enhancements, whose costs were estimated to be in excess of the ceiling, which would require the agreement of a further contract. This work would be undertaken by a dedicated software development project team.

Under the proposed project categorisation, the enhancement work falling under the maintenance contract would be termed ‘maintenance’ while the work requiring further funding arrangements would fall under ‘development’. Since both cases would involve requirements changes made to post-delivery software, this supports Chapin’s comment that industrial naming convention is a matter of budget considerations [42], and highlights the potential for confusion when understanding the context of research studies. It was emphasised by the project manager that any development project emerging from a maintenance contract would necessitate more depth of requirements analysis processes than that required by the ‘mini projects’ (taking only a few weeks) undertaken under the terms of the maintenance contract. The project categorisation as proposed was used in the remainder of the study.

From Table 4.1 and qualitative review, the scenarios of ‘product upkeep’ and ‘servicing’ are both included within software maintenance. Software development is distinguished from maintenance by the following definition of software maintenance:-

Maintenance projects are those that:-

1. Employ staff whose work assignment is distinct from that of pre-delivery development, and whose application domain knowledge is not assumed.
2. Operate under a clearly defined support contract.
3. Involve activities of product correction to production software
4. Involve activities of enhancement to production software where the cost is not in excess of a contractually agreed amount.

4.4 Taxonomy development for Software Development

This section describes the process of initial taxonomy development for the lifecycle stage of software development. Upon agreement of the proposed categorisation, a consideration of the sources of requirements changes observed during initial software development informs the organisation of an initial change source classification. This is followed by further study incorporating sources of change associated with maintenance projects, including those that pertain to software development with iterative delivery.

4.4.1 Development Change Source Constructs

Electronic keyword searches were performed to assemble candidate academic papers, industrial articles, and text books. Citations referring explicitly to requirements change/evolution sources/causes/uncertainty/creep/risk were followed in a forward direction in search of the initial source. This resulted in a total of 73 papers and text books which were reduced to a final 14 sources by the criteria 'software development' with 'discovered empirically' or 'seminal work/text book'. As 'seminal work' was subjectively assessed, this cannot be considered a systematic review. However, without this criterion, papers such as 'Issues in requirements elicitation' [168] would not have been included, which may have devalued the review.

During the collation of change source constructs it became apparent that reasons for change such as 'diverse user community' and 'New tools/technology' were often gathered together under the umbrella term 'cause' [91][97] [169]. Clearly there is a distinction between *uncertainty* giving rise to change and events that *trigger* a change. Whilst an event can lead to a change without preceding uncertainty, uncertainty cannot result in a change unless an event resolves or intervenes to mitigate the risk of uncertainty. It could be argued that change is 'caused' by a combination of uncertainty and trigger, although in reality causation cannot be proved to arise from one, other or both due to the presence of confounding environmental factors. Accordingly, uncertainties and triggers, collectively referred to as sources of change, were separated. This separation was not difficult since in most cases the semantics of the constructs related to an *event* (trigger) or a *situation* (uncertainty).

4.4.2 Initial Workshop – development construct consolidation

The first workshop taking 2 ½ hours introduced the constructs to 3 project managers and each construct was clarified for meaning. In so doing, constructs sharing a similar meaning were amalgamated, and those represented by other constructs at a finer level of granularity were removed. Additionally, constructs such as 'New Functional Feature', which would necessarily arise as the consequence of resolved uncertainty or opportunity were also removed. The most debated of the constructs was 'changes following prototyping'. Though quoted as a cause of change, it was the opinion of the participants that this change source should be thought of as a technique, having no more causal significance than other techniques such as 'requirements inspections'. Either all techniques should be included and constructs added accordingly, or constructs pertaining to increased understanding should represent the techniques. The final consensus favoured the latter argument, though the addition of technique constructs remains a question for further research. Four triggers were added and as a result of this process, the number of constructs was reduced from 73 to 46. Making the distinction between trigger and uncertainty was confirmed both to be viable and useful, since triggers could more easily be attributed to requirements changes. The constructs are listed in Appendix1 under the headings 'Development Trigger Construct' and 'Development Uncertainty Construct'. What remained was to classify the triggers and assign uncertainty constructs accordingly, thus endorsing the classification and confirming that uncertainties had corresponding change events.

4.4.3 Participant Card Sorting

Individual card-sorting ensured that the opinions and contribution of each project manager were represented. The process was first validated by a pilot card sort with 1 project manager and 1 researcher. Each card sorting session was audio-recorded and reviewed, and photographs were taken of card classifications. This process took between 45 minutes and 1 ½ hours.

Each of the 23 development trigger constructs (as they appear in the Appendix) was written on a card and assigned a random number. Six participants were asked to classify the triggers according to their source. All participants classified the triggers into between 3 and 5 categories, and there was homogeneity between the classifications, although in all cases they were named

differently. For example, one project manager referred to ‘ownership’ of the categories; another used process labels such as ‘customer interface’ and ‘Requirements engineering’. Naming convention aside, 14 of the 23 constructs were placed in the same pattern by all participants, that is, co-resided in 3 groups. Notably, differences of card placement related to degree of granularity of classification. For example, 4 participants grouped ‘increased customer understanding’ and ‘first engagement of customer’ alongside constructs relating to understanding the technical solution. The classifications of the remaining 2 project managers conveyed the importance of distinguishing between changes that arose due to increased understanding of the problem, and those relating to the technical answer to that problem. Only 1 classification, illustrated the distinction between market factors and those concerning the customer organisation, the remainder considering them similarly ‘external’ to the project. One of the participant’s card sorts of the development trigger constructs is illustrated in Figure 4.1

Figure 4.1 CARD SORT – A SINGLE PARTICIPANT SORTS THE DEVELOPMENT TRIGGER CONSTRUCTS

4.4.4 Second Workshop- Consensus building

Four project managers attended a second workshop lasting 3 ½ hours. Stimulating and interesting discussion resulted in a unanimously agreed trigger taxonomy to which uncertainty constructs were attributed.

Beginning with 3 untitled groups containing a total of 14 trigger constructs, it remained to come to a consensus of opinion regarding the remaining 9. As observed by one of the participants, the granularity differences were a matter of perception. For example, as a project manager, constructs such as ‘market stability’ or ‘customer organisation strategic change’ were equally external to their control. However, from the perspective of a customer, this is perhaps not the case. Figure 4.2 illustrates the card sorting during consensus building and adding of uncertainty constructs. The final taxonomy was built according to the maximum variance of classifications made during the card sorting procedure. Consequently, a taxonomy comprising 5 groups was derived and agreed.

Figure 4.2 CARD SORT – CONSENSUS BUILDING AND ADDING UNCERTAINTY CONSTRUCTS

These groups comprised the change domains illustrated in Table 4.2. The five change domains were named *market, customer organisation, project vision, requirements specification* and *solution*. Table 4.2 also contains a brief description of each domain. Uncertainty constructs were attributed to their associated domain. At this stage several additional uncertainty constructs were added. Most notable amongst these were technical uncertainty, and technical complexity of solution. Though considered general project risks [26], they had not previously been recognised as a source of requirements change. This may be because they do not alter the vision of the problem, but rather the way in which the problem is addressed. Nonetheless, as discovered by Curtis [169], 'creeping elegance' is a source of change and a risk to budget and schedule slippage. There was some debate about the positioning of 'project size'. Initially considered to be a risk to change in all domains, it was further reasoned that size has an effect, due to the increased difficulty of conceptualizing the problem. Therefore 'size' was placed in the domain of project vision.

Table 4.2 SOFTWARE REQUIREMENTS CHANGE DOMAINS

Change Domain	**Description**
Market	Differing needs of many customers, government regulations, external to project.
Customer Organisation	Strategic direction of a single customer, customer organisation considerations, external to project.
Project Vision	Problem to be solved, product direction and priorities.
Requirements Specification	Specifying the requirements of the established problem.
Solution	Technical answer to problem.

.

4.5 Software Requirements Change Source Taxonomy for Development

The resulting taxonomy is shown in Figure 4.3. The reader is referred to appendix 1 for full construct tracing from research origin to construct consolidation and comparison. The change domains relate to both triggers and uncertainties. There is a many to many relationship between the uncertainties and triggers within each change domain and in many cases a 'chain' of uncertainties may culminate with a trigger event. For example, increased availability of project stakeholders would raise the quality of communication which may lead to increased knowledge of the business area and the identification of an incorrect requirement. Such links between the uncertainties and triggers are more obvious in the domains of vision, requirement specification and solution. In the domains of customer organisation and market, the uncertainties are much more imprecise, and have a broader set of consequences.

A single uncertainty in the domain of '*customer organisation*' of 'stability of customers business environment' could be associated with many factors, and lead to any of the change triggers, such as 'company reorganisation' or 'change in political climate'. An examination of the triggers and uncertainties associated with each change domain further clarifies the meaning of change domain nomenclature. '*Solution'* includes influences upon architectural considerations which would in turn have an effect upon requirements, though this may relate primarily to non-functional requirements such as performance or maintainability. '*Requirements Specification'* relates to the process of transposing an in-articulated requirement to a state that is directly translatable to code. The related uncertainties and triggers are therefore concerned with the quality and efficacy of this process. As such this domain refers to the resolution of uncertainty, ambiguity, inconsistency and error that can arise during specification. The domain of '*project vision'* is concerned with the changing needs of the customer beyond those that may be considered inaccuracies. These refer to newly identified opportunities or decisions to alter the way in the software will deliver a specific requirement. There is some overlap between the domains since constructs pertaining to understanding (knowledge and skill) are included in both the domains of *vision* and *requirement specification*.

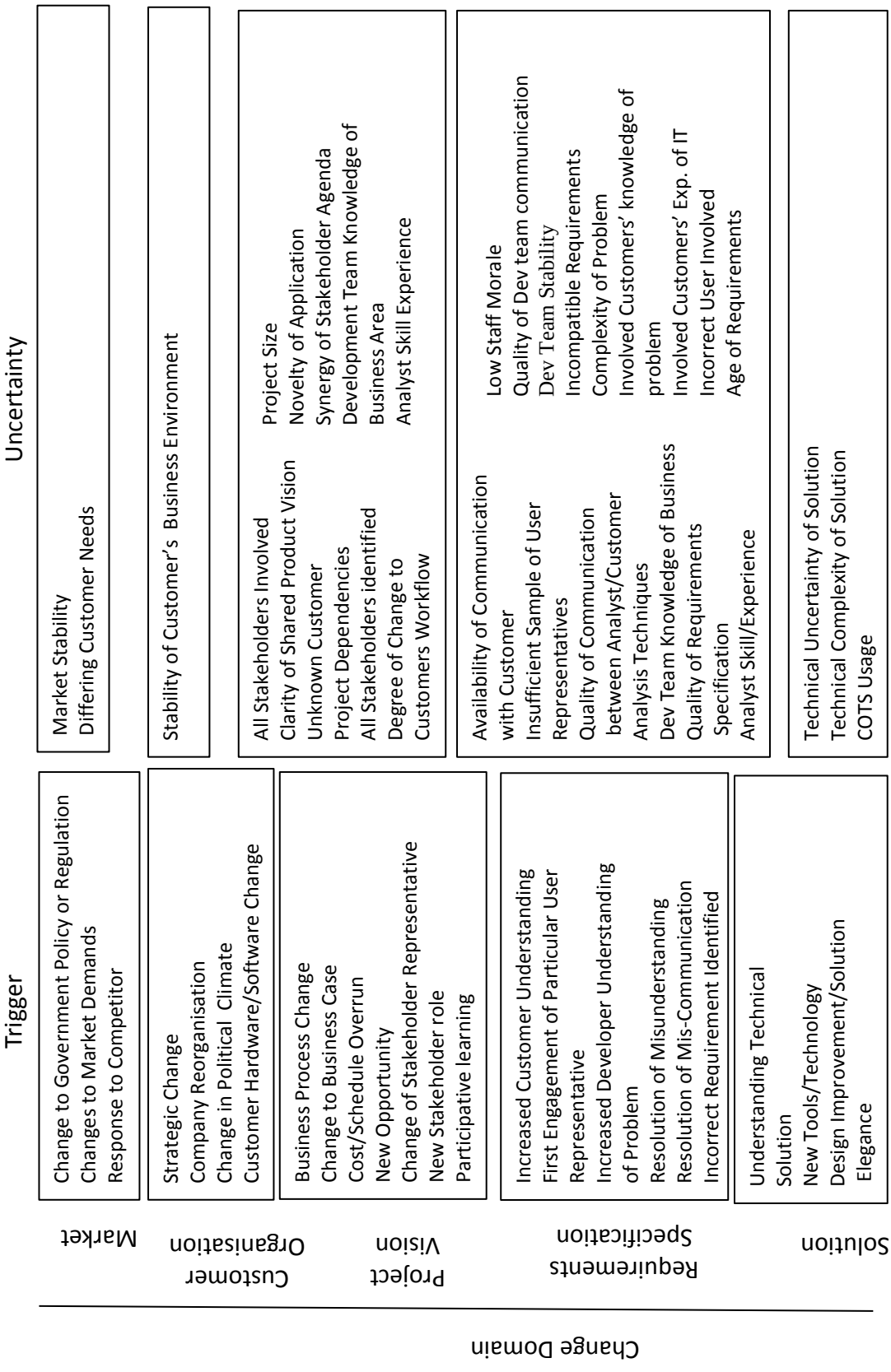

Figure 4.3 SOFTWARE REQUIREMENTS CHANGE SOURCE TAXONOMY FOR DEVELOPMENT

While the knowledge and experience of the software analysts has an effect upon changes from both domains, the customers' knowledge and experience of IT has an influence only upon changes from the domain of *requirement specification*. The structure and strategic direction of the organisation in which the software will be deployed can have an influence upon the requirements of any software system residing therein. Such matters are included in the domain of '*customer organisation'* which was unanimously considered to be external to the software development environment. There are fewer triggers and uncertainties in this domain, and even less in the domain of '*market',* similarly external to the software development environment and related to market demands and government policy.

4.6 Taxonomy Extension for Software Maintenance

Extension of this taxonomy to include constructs pertaining to maintenance firstly requires the identification, consolidation and classification of maintenance change source constructs. This is then compared with the previous classification for software development in order to derive a common classification wherein the constructs are deemed to relate to development, maintenance, or all stages of the software life-cycle.

4.6.1 Maintenance Change Source Constructs

Of the initial 73 papers, 11 contained references to post-delivery requirements change causes. Having established a project categorisation, there was difficulty applying it to other studies since none of them made reference to contract conditions or staffing arrangements. Only the criteria 'changes to production software' was used. Interestingly there were significantly fewer empirical studies examining sources of requirements changes post-delivery than during development, despite the high proportion (75% [50]) of enhancement work carried out during that time. It was noted that many studies examining risks or uncertainty within the maintenance environment were exploring risks to maintenance *change productivity* rather than *change likelihood*. Perhaps this indicates support for Kitchenham's argument [53] that one of the major differences between development and maintenance is that development is requirement-driven and maintenance is event-driven. In their words, "This means that the stimuli (i.e., the inputs)

that initiate a maintenance activity are unscheduled (random) events". Perhaps prohibitive to investigation is the limited value that an exploration of change causes would yield, should the contention be empirically proven that maintenance change is stimulated by random events. Once again, separation of trigger and uncertainty presented no difficulties.

4.6.2 Third Workshop - Maintenance construct consolidation

Consolidation of maintenance constructs during a workshop consisting of a researcher and 2 maintenance team members, taking 2 hours, followed the same process as development constructs, reducing an initial set of 45 constructs to 10 triggers and 12 uncertainties (see appendix 1). This is in marked contrast to the number of constructs concerning development projects elicited from the literature. Many of these constructs ignited lengthy discussion. Of particular note was that many of the uncertainty constructs were likely to introduce error rather than requirements change. Those in that category included 'maintenance team instability' and 'maintenance team knowledge'. By contrast, these team-related constructs had been considered sources of requirements change during development. It was believed that the perceived more limited business knowledge required by maintenance engineers coupled with the reduced need for requirements analysis processes to implement 'mini changes' meant that these team attributes had no significant effect upon requirements changes. Also interesting was the observation that some uncertainties such as 'economic climate' altered a projects capacity to make change, rather than invoking change. The construct 'system usage' was removed since it was seen as an 'activity' during which an alternative change source may manifest (such as 'increased understanding'), rather than a cause of change itself. This bears comparison to the removal of techniques during the development construct consolidation. However, this is differentiated from 'domain change due to system use' which is a change in the way that the organisation works due to system usage. 'Deferred requirement' was also removed since this was considered a planned later enhancement rather than a source of requirements change. The 9 added constructs included Commercial Off-the-shelf Software (COTS) usage, which was felt to be a contributor to requirements change, due to the need to react to new COTS opportunities and release functionality. 'Number of interfaces' and 'Number of functions' were and distinguished from system complexity, as it was thought that system complexity doesn't in itself lead to changes of requirements during maintenance, though it would during development when requirements are still being understood. However, the number of interfaces and functions in a

requirement would determine which portion of the software system is most often used. System usage would engender requirements change during maintenance. 'Function Usage' was also added since some software modules are used more frequently than others.

4.6.3 Fourth Workshop - Card Sorting (Maintenance)

Since the intention was to discover if sources of change during maintenance and development projects could be similarly classified it was decided to perform one closed card sort [146] within a workshop setting. Provided with the change domains derived previously and described in Table 4.2, two maintenance engineers were asked to ascribe the maintenance change constructs to one, many, or none of the change domains.

The participants found the trigger constructs easy to attribute, though some of the uncertainty constructs resided in both *requirements specification* and *project vision*. A higher number of users, or a high level of function usage may uncover opportunities to improve the way in which the system requirements have been implemented, or reveal new desires and needs. Similarly, the discussion surrounding 'project size' during the consolidation of constructs pertaining to software development, 'system age' was initially thought to reside in all domains. However, further consideration led to the conclusion that, while an older system is more likely to require functional updating without changes to the surrounding market or customer environment, the system could retain value in its current state. By itself, the age of a system will only affect performance or data storage issues requiring *solution* maintenance. The term 'semantic relativism' described by Heales [88] as 'generation of language construction' was placed in the domain of project vision, although the participants felt that as a concept it had less relevance than the other uncertainties, and was difficult to evaluate. No constructs remained unplaced.

4.6.4 Fifth Workshop – development and maintenance taxonomy consolidation and comparison

During this workshop, taking 3 hours, both project managers and maintenance engineers were brought together to compare and consolidate the two previously derived taxonomies. The following agenda items were agreed:

1. Identify and consolidate corresponding maintenance and development constructs
   a) within the same domain, and
   b) within alternative domains.
2. Review constructs to determine if those located in a single taxonomy related equally to both.

4.6.5 Maintenance and Development Construct Consolidation.

Eight of the 11 consolidated maintenance related triggers, and 6 of the 12 remaining maintenance related uncertainties were semantically synonymous, though named differently, to development related constructs. Those that resided within the same change domain retained the naming convention used in the software development change source taxonomy. There was some discussion regarding the naming and placing of 'Number of interfaces' and 'number of functions' which had been placed in both the domains of *project vision* and *requirements specification* by the maintenance engineers. These represented factors contributing both to 'project size' and 'logical complexity of problem' residing in the domains of *project vision* and *requirements specification* respectively. The ensuing discussion led to the recognition that while these constructs embodied a similar concept, they gave rise to change for two different reasons. The 'logical complexity of problem' affected the capability of the development team to understand and model the problem, and therefore changes would ensue. However from the perspective of the maintenance team it increased the likelihood for change discovery during maintenance, since they were used more often. All constructs remained though labelled as applicable to either development or maintenance.

4.6.6 Development and Maintenance construct review

The first task was to examine maintenance constructs to ascertain if they also applied to software development. Of the additional maintenance constructs identified, a number of sources of requirements change were deemed equally applicable to software development. However, it was felt that 'semantic relativism', 'response to gap in market', 'presence of a competitor' and 'quality control during development' were deemed applicable to *initial* software development

only. When taking iterative development into consideration, the constructs relating to system usage would also become relevant. It was argued by the project managers that from their perspective 'alter performance' is a (non-functional) requirement change that would happen in response to a market or customer need and was therefore not a cause of requirements change. From the perspective of maintenance, 'alter performance' represented the pro-active changes made to deter system degradation or promote further usage. Therefore 'design improvement/solution elegance' was a more appropriate construct. Those remaining solely within the realm of software maintenance related only to system age. Most interestingly, the construct of 'No of users.' was added by the maintenance managers during maintenance construct consolidation, having been removed by the software development managers during the consolidation of the development constructs. This is because the 'No of users' *using* a system would give rise to maintenance change, while the number of users *specifying* a system would have no effect (provided the correct users were involved and they agreed).

Consideration was also given to the development constructs not included within the maintenance classification. Many of the constructs pertaining to software development applied also to maintenance. Indeed, it was agreed that, aside from 'cost/schedule overrun', only those constructs relating to the ability to understand the problem related solely to software development. It was recognised that this is the case when the definition of maintenance is understood to be that defined in 4.3, with only small requirement enhancements allowed under the terms of a contract. It was observed that many of the sources, particularly those in the domains of *market* and *project vision* would result in the initiation of a new product release that would fall under the nomenclature of software development. So while the change may be incurred during maintenance, it will be realised by a software development team. Confirming the insight arising from the discussion regarding project size, a number of the uncertainties relating to software development were applicable also to maintenance, though with a distinct difference in effect. For example, during development high quality of communication with customers affected the clarity of the shared understanding of the problem, thereby *reducing the likelihood* of subsequent requirements changes. By contrast, during maintenance the quality of communication *increased the probability* of change recommendation, and hence had an effect upon system longevity.

4.7 Software Requirements Change Source Taxonomy

The final taxonomy is illustrated in Figure 4.4. Since the maintenance constructs fitted well within the previously Those constructs marked '(D)' apply solely to initial development, while those marked '(M)' are relevant only to maintenance. Those denoted '(M/I)' are relevant both to maintenance and also to development with a component of iterative delivery. The constructs that relate to software usage following development fall into this category. Relevance to a software life-cycle stage is dependent upon the definitions of software development and maintenance described in section 4.3.

The domains of '*market*' and '*customer organisation*' are very similar to the taxonomy derived for only the developmental stage of software development. Indeed only the construct of 'Response to gap in market' has been added which was deemed relevant to all life-cycle stages. Noteworthy though, is the comment from the participant that changes arising from these change domains would usually result in a new development project, so it could be argued that neither of these domains are significant change drivers for maintenance projects. In the domain of *vision* maintenance only constructs relate either to software usage which includes 'no of interfaces' and 'no of functions' in a requirement. In this instance 'no of users' refers to the number of people using the software, not those involved in specification. Constructs in the domain of *vision* that participants felt were relevant only to initial development included those that were concerned with the skill and knowledge of the analyst, and the cost/schedule constraints during development.

There are similar differences between constructs relevant to software development or maintenance in the domain of *requirement specification,* where maintenance only constructs relate to system usage, while development only constructs are mainly concerned with development team quality. In addition the complexity of the problem was deemed to be a development only concern. This observation is mirrored in the domain of *solution*, where 'system age' is influential upon requirements change for software maintenance only.

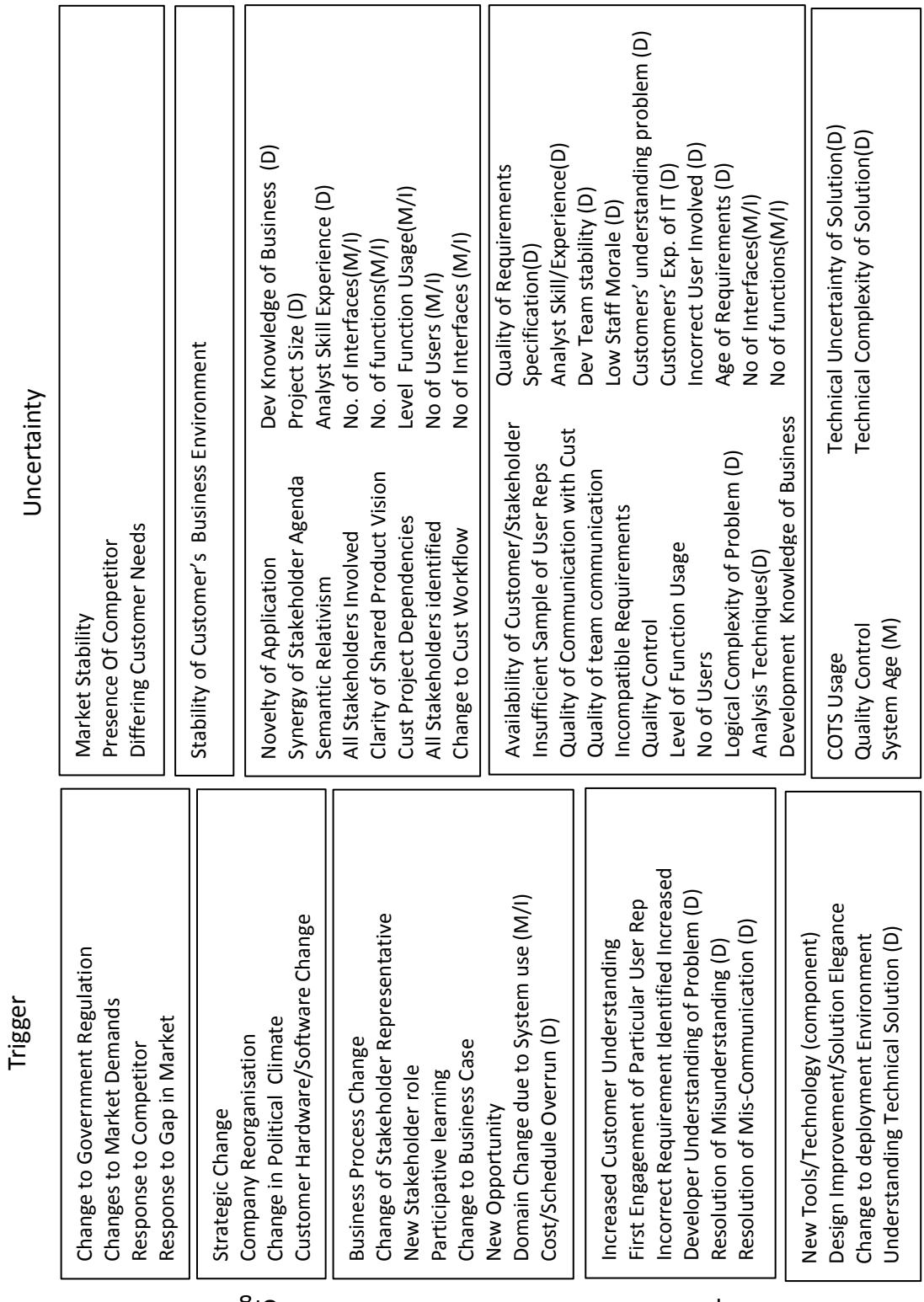

Figure 4.4 SOFTWARE REQUIREMENTS CHANGE SOURCE TAXONOMY

4.8 Validation of Change trigger Constructs

The capability of the change trigger constructs to describe the source of a change was initially validated by one of the participating project managers who used a small sample of changes (13) across two development projects to ensure that each had a corresponding trigger which accurately reflected the source of change. No changes were made at this stage and identification of the correct change trigger was not considered difficult. No analysis was attempted on this small sample, though it was reported that most of the triggers came from the domains of *vision* and *requirement specification*.

4.9 Discussion of the Software Requirements Change Source Taxonomy

Having derived a project categorisation based upon the work of Kitchenham [53], Bennett & Rajlich [50] and Chapin [55] (refer to Table 4.1), the taxonomy derived in this study verifies that many requirements change sources are similarly relevant to development and maintenance. This supports Bennett's [50] observation that software evolves during both iterative development and maintenance. The differentiation presented by Kitchenham [53] between the two scenarios is also reflected in this study. Sources of change arising due to continued understanding of the requirements are attributable to iterative delivery (scenario A), while those relating to system age are relevant only to product upkeep and servicing (scenario B). However, this observation relies upon a definition of maintenance that includes only minor enhancements, which are represented in Bennett's [50] model, not as a lifecycle stage, but as an iterative element of evolutionary product versioning. The results refute Kitchenham's contention that maintenance changes are event driven while changes during software development are requirement driven [53]. The separation of triggers and uncertainties and their pertinence to both development and maintenance, reveals that changes during software development can be equally reactionary to external events. The pro-active approach to maintenance described by one of the maintenance engineers in this study suggests that maintenance changes, like those during software development, aren't entirely event-driven either, but transpire as a result of a combination of uncertainty, event and pro-active change discovery. Whilst the change sources illustrated in the taxonomy indicate the similarities between development and maintenance,

further exploration of the consequences of the uncertainties may reveal differences to project risk.

The constructs in the taxonomy bear little synergy with change reasons derived by Nurmuliani [21] as many of these reasons such as 'missing requirement' and 'new functional feature' were considered to be consequences of other events, rather than sources of change. By comparison, there is some resemblance to the classification of change sources defined by Harker [91]. In particular, a combination of *market* and *customer organisation* domain sources equate to their 'mutable' class defined as "changes that arise in response to demands outside the system". This is reflective of a later observation by Nurmuliani [22], that some changes are 'external' to the system . By making the distinction between changes that occur in response to market demands, and those answering to customers' organisational considerations, the taxonomy developed here reflects the difference between customer-driven and market driven software development. Harker's 'emergent' requirements, "direct outcomes of the process of engagement in the development activities", correspond to constructs in both the *project vision* and *requirements specification* domain. In differentiating between *project vision* and *requirements specification* domains we are recognising the difference between variation in the product to be developed and change due to better understanding of the problem. This is an important distinction as it can support decisions regarding requirements elicitation techniques and rigour of documentation.

There are no analogous domains within this taxonomy for the remainder of Harker's categories. These include prototyping or system usage, adaptive requirements and migration requirements, which were reasoned to be techniques, activities, or new requirements. Sommerville's classification [97], while including 'mutable', emergent' and 'consequential' change (system usage) also removes adaptive and migration requirements. Instead 'change to business process' form a category which is included here in the project vision domain, since these types of changes result in a change of product direction. The *solution* domain has no direct parallel in any classification but reflects the reality that changes to the technical solution, though perhaps less visible, pose a risk to timely development.

While there are some differences in contained constructs, requirements availability as defined by Mathiassen [7] corresponds to *requirements specification* although constructs relating to

requirements complexity and reliability are included in both *customer organisation* and *project vision.* That said, further categorising these domains according to reliability and complexity would allow the findings of both studies to be combined, thus relating technique to change source domain.

A comparison can be drawn between this taxonomy and classifications of change types during maintenance [55]. Excluding error handling, the taxonomy derived here includes constructs in the *solution* domain relating to perfection and adaptation while enhancements are further classified according to the remaining change domains. There is an encouraging parallel with Perry's software development domains [98]. While the 'real world' is represented here in both the *Market* and *customer organisation* domains, Perry's 'model of the real world', 'derived specification' and 'underlying theory' correspond closely to *project vision*, *requirements specification* and *solution* respectively. Thus, to an extent this study corroborates Perry's theoretical model with empirical evidence, and furthers understanding of the nature of the domains.

4.9.1 Review of Results Validity

This research serves to meet the academic objectives, and also respond to concerns of the industrial practitioners taking part. This dual interest affords many advantages, not least by ensuring that the research has practical value, but also constrains the scope and industrial context to which the results may be applicable. Limitations of study validity are described in section 3.4.6. Those relevant to this study are reviewed below:-

Construct Validity

This type of validity is concerned with the understanding of the constructs used in the study. This is addressed by the collaborative approach taken with involvement of six practitioners whose objective to come to a common understanding ensures that the constructs have an agreed meaning.

External Validity

No claims can be made with regard to validity in industrial contexts beyond the boundaries of this study, and in particular to projects employing alternative delivery models such as service oriented and cloud computing. However given that the constructs were drawn from a variety of empirically based studies, it is plausible that the results apply to projects similarly adhering to a more traditional development lifecycle. Also, the individual practitioners involved bring several years of project managers experience between them which is not confined to a particular industrial context.

Study reliability

Given the disparity between both terminology and published change taxonomies combined with the debate among the participants of this study, it could be argued that change classification is by nature a subjective assessment, and therefore that results may be different each time the card sorting exercise was performed. However, given that the classification mirrors that of other researchers, there is some confidence that study replication would produce the same results. Initial constructs are provided here, with full traceability from the original source, along with methods description such that it should be possible to easily replicate this study.

4.10 Conclusion and Implications for Research

The study described in this chapter set out to identify and classify and compare the causes of requirements change during software development and maintenance. Expert knowledge of experienced project managers and maintenance engineers was used to consolidate and classify change source constructs elicited from the literature. This resulted in a taxonomy that was both applicable to development and maintenance, though there was some differences in construct inclusion. The taxonomy contained the change domains *market*, *customer organisation*, *vision, requirements specification* and *solution,* and made a distinction between events (triggers) that cause change, and situations that may mean that requirements are uncertain (uncertainties).

Of particular significance to industry is that by comparison to the descriptive or uncertainty based nature of previous work, the clearly defined change cause constructs within each change source domain allow comparative source data to be attributed to change databases. Therefore it

would be possible to assess the impact on the project of a particular change source such as 'new stakeholder' or 'first engagement of customer representative', giving software providers some empirical data with which perhaps to leverage customer involvement. Further, it would be possible to assess the level of change in each change source domain. Should, for example, a high proportion of changes come from the domain of *project vision*, this would indicate the vulnerability of the 'problem' to change, thereby empirically illustrating the need for more 'agile' creative processes. By contrast, a high proportion of requirements specification changes may suggest the need for increased quality control or alternative specification techniques. Of interest was the observation by the project managers that the structure of the domain also reflects the amount of control they have of the uncertainties, with least control at the top - 'Market' and tighter control at the bottom – 'Solution'.

This observation carries implications for change prediction feasibility, and is reflected by the significantly fewer change constructs in domains that are considered external to the software engineering environment. This infers that less is known about the uncertainties and triggers that may give rise to *market* or *customer organisation* changes. Since the constructs were derived from a combination of literature and industrial sources, the conclusion drawn is that for both research and practice the focus upon requirements change is upon changes that arise from sources inside of the software engineering environment. The observation that requirement changes can arise due to either events or on-going situations will have an impact upon formal models of change prediction. That 'quality of communication' can both lower the possibility of change (during development) and raise the likelihood of change (during maintenance) may mean that predictive models be calibrated to suit a particular life-cycle stage. Given the collaborative nature of this research, and its immediate applicability, it has a high level of relevance.

Having met the study objectives of identifying the causes of change, differentiating between those relevant to development and maintenance, and classifying changes according to change source we have answered **RQ1** 'What are the causes of requirements change?' and partially answered **RQ2 '**How can change be classified and measured in order that it is informative to project management?'. What remains is to formulate a metric for requirements change, and most importantly, assess whether the software requirements change source taxonomy is informative to project management. The next chapter addresses the latter, with an empirical enquiry designed to assess whether the taxonomy represents different groups of requirements.

Chapter 5 Taxonomy Evaluation

Having proposed a classification for software requirements change that is based upon the source of the change, this chapter describes a collaborative case study designed to investigate whether the taxonomy is useful and informative to project management. Attributes of changes such as cost and value that are considered significant to decision making are identified by our industrial partner, and examined with respect to the domains in the software requirements change source taxonomy. Data specification, collection protocol and industrial context are presented here with the results and discussion. This study addresses the following research question:-

RQ2 How can change be classified and measured in order that it is informative to project management?

Chapter Contributions to Academic Knowledge

a) Requirements change classification.
   By contrast to other classifications, an empirical evaluation of the requirements change source taxonomy found it to be informative and practical.
b) Empirical knowledge regarding requirements change.
   A Case Study examines requirements volatility during the development lifecycle reveal the cost, value and controllability of requirements change.
c) An exploration of the feasibility of volatility prediction.

5.1 Introduction

Requirements change classification is commonly believed to be useful to the process of software engineering [67]. Towards the research goal of requirements change anticipation, a change classification has been proposed which may provide the foundation for a predictive mechanism. This taxonomy distinguishes between changes arising from 'market', 'organisation', 'project vision', 'specification' and 'solution'. A predictive facility based upon this classification would therefore provide a prediction for each type of change, rather than a single point estimate. However, as yet no (known) study has set out to test the assertion that change classification is informative. Indeed, industrial studies have confirmed that, in practice, organisations do not classify change beyond 'add', 'modify', and 'delete' [83]. Therefore, for a predictive mechanism to be based upon the proposed taxonomy, it is necessary to first ensure that the product of such a model of prediction would be informative and reflective of industrial practice. Since the taxonomy developed in the previous study was drawn from the experience of project managers, there is some confidence that it encapsulates the characteristics of change. Nevertheless, empirical support for that argument will ensure that models are based upon a strong evidential foundation.

Should it be the case that changes falling within each stratum of the proposed classification exhibit significantly different characteristics that would require particular managerial considerations, the conclusion can be drawn that classifying change in this way is informative to project management. The objectives of this study are:-

1. Determine a list of factors that are considered important to the management of requirements change.
2. Examine changes to ascertain if those falling within a particular change domain have similar values for those important factors, and if they vary significantly between change domains.

Since the data required is particular to this study and not available in public datasets, a case study is undertaken wherein case study data, in accordance with study objectives, is specified before commencement of research.

5.2 Empirical Context and Execution

5.2.1 Organisation

This study shares the same industrial context as the study described in the previous chapter. In summary, the organisation employs 300 staff, has offices in England and Ireland, and most software development is related to government initiatives.

5.2.2 Project

The project of interest in this study is in the government sector, has an estimated cost in excess of a million pounds, comprises on average 15 software developers and analysts, and follows a traditional waterfall lifecycle. Beginning in April 2009, the project was completed in August 2010 and data was collected during the entire development lifecycle. Since the software development work was the result of a successful tender, at the commencement of the project, the requirements made available to the software provider during that tendering process became the basis of the initial requirements specification effort. There were four main stakeholder groups involved, comprising the software provider and three departments on the customer side.

5.2.3 Case Study

The case study was designed prior to the commencement of the software development project, and conducted during the entire lifecycle. It was designed in accordance with the case study guidelines outlined by Runeson [41] which reviews terminology from other sources such as Wohlin [42] and Yin [142]and adapts them for Software Engineering research. The case study is a single unit studies, in which the unit of analysis in the change. See section 3.4.3 for details of case studies. This case study was supported by the Goal Question Metric Approach and UML modelling. An explanation of these activities can be found in section 3.4.5.

5.3 Identification of Significant Change Attributes

The academic objective is to discover if the requirements change source taxonomy can provide a meaningful and practical basis for change classification and measurement. At the same time there is an immediate business objective to improve visibility and understanding of requirements change. Effort was therefore required to clearly identify research variables, and define mutually expedient case study data. With our industrial partner, the Goal Question Metric (GQM) approach [148] was largely adhered to in order to firstly articulate the change attributes (the questions) and secondly identify case study data. The Goal Question Metric Approach was operated initially in a focus group setting consisting of a researcher and 2 project managers. Past change data were used as the basis of discussion, and this was supported by UML modelling of project processes and work products, which enabled the identification of the possible values of the variables under study.

The most obvious candidates for attributes of change that are significant for change management are cost and value. Cost may determine whether or not a change is made, and affects scheduling and delivery. Value to the customer is important for product usability, acceptance and prioritisation. The selection of additional change attributes related to management issues and concerns that would provide our industrial partner with an empirical basis for future decisions concerning change management approaches. Discussion revealed that project managers and senior management were interested in the portion of changes that were made as a result of an opportunity to add value, compared with those made to correct an error in the specification of the requirement. Management, both contractually and administratively, was particular to these two types of changes. In addition, changes that required the input of a larger number of stakeholders meant that agreement was harder to discern and often the process of making the change was thwarted by differing needs, and required negotiation. Project managers reasoned that the activity during which changes were found was also of interest, since an understanding of the means by which changes are discovered may affect decisions concerning process and technique. For example, if particular types of changes were discovered by, say, user testing, then that activity could be introduced earlier in the project life-cycle. Lastly, changes that were felt to be ‘controllable’ by project management, that is, identifiable within the confines of the software engineering environment were considered less of a risk to project success, since effort could be made to discover such changes as early as possible. By comparison, changes arriving ‘out of the blue’

were more concerning, could not be addressed by decisions concerning process or technique selection, and were unavoidable.

Change attributes included in this study are:-

i. Change cost
ii. Change value
iii. Proportion of opportunity vs. defect related change
iv. The number of stakeholders involved
v. The activities during which changes are found
vi. The level of project management control

5.4 Data Specification

As well as supporting the needs of the academic objective, the data to be collected will also replace the company's existing change control database and be used for project retrospective analysis. The selection and practical implementation of metrics to answer the research questions was not straightforward. In the main a pragmatic approach was taken, which often required compromise between research and practice. The data specified (excluding those relevant only to practice such as originator, dates etc.) alongside their allowable values are illustrated in Table 5.1. Although the inclusion of metrics for change KLOC or function points would have benefitted research by providing a means to assess the size of the change, it was considered by practitioners to be too labour intensive. The addition of the data item 'phase' was therefore necessary for the analysis of cost comparison since average change costs may increase as the project progresses due to rework, rather than change size (see section 2.2.1 for a discussion of change costs). Four phases, commonly assigned to software development were appropriate in this case. Consequently, the project phases are 'requirements analysis' (Req), 'design and code' (D&C), which includes unit testing, 'system test' (Systest) and 'user acceptance test' (UAT). These were defined as distinct project stages since the end of each stage required formal mutual sign off by both the customer and provider. Cost was measured in days and defined as the difference between the unused previous estimate (if it existed) and the revised effort in days required to implement the requirement including additional requirement analysis. Expressing value in monetary

terms was impossible in most situations, so a Likert scale, subjectively assessed, was employed instead. Project management control was measured similarly. Opportunity and number of stakeholders were easily defined. The additional data items 'change trigger' and 'domain' were added to relate changes to the change domain. No classification scheme had been previously used by the company, though ad-hoc reasons for change had been included in descriptive text.

Table 5.1 DATA SPECIFICATION FOR EVALUATION OF THE REQUIREMENTS CHANGE SOURCE TAXONOMY

Name	**Description**	**Allowable Values**
ID	Unique Identifier	
Trigger	Change Source Trigger Eg. Change to business case, Increased customer understanding, New technology available.(Nominal Objective)	A complete list can be found in section 4.7
Domain	Change Source Classification. This was derived where possible from the trigger using the taxonomy in section 4.7 and reviewed. (Nominal, Objective).	Market, Organisation, Vision, Specification, Solution.
Phase	Project phase when change identified (Nominal Objective)	Requirements Analysis (Req), Design and Code (D&C), System Test (SysTest), User Acceptance Test (UAT).
Discovery Activity	Activity during which change was identified (Nominal, Objective)	Illustrated in Table 5.2.
PM_control	Project manager's control of change identification (Ordinal, Subjective)	Very low, low, med, high, very high.
Stakeholders	Number of stakeholder roles involved agreeing the change (Ordinal Objective)	One, Two, > Two.
Cost	Change cost expressed in days (Ratio, Subjective)	
Value	Business value to the customer (Ordinal, Subjective)	Very low, low, med, high, very high.
Opportunity?	Opportunity or defect (Nominal, Subjective)	Opportunity, Defect.
Description	Free text – qualitative	

It will be noted that many of the data items are subjective measures. Whilst appreciating the limitations imposed by non-objective measurement upon the analytical significance of results, the collection of subjective measures is becoming more widely accepted and advocated. [34] [4]. Given the need to define an agreed and standardized list of activities, UML modelling sessions led by the researcher and involving project managers as available, gave rise to the production of a prototype activity diagram and a domain model, which were used to identify and specify the list of project activities. Both models were used as research support artefacts in later studies to support shared understanding of the organisations processes.

Table 5.2 lists the activities with a short self-explanatory definition provided by the participating organisation. The table also includes the phase in which these activities took place. Since a project phase refers to a period of time rather than a particular endeavour, the activities therein aren't always obviously related. For example, all requirements definition activities, except for 'Define Technical Requirements' were completed during the phase of 'Requirements Analysis'. The definition of technical requirements took place during 'Design and Code'. Similarly, 'Specify UAT's occurred during 'Design and Code' rather than 'User Acceptance Testing' as might be expected.

Table 5.2 PROJECT ACTIVITIES

Activity	**Description**	**Project Phase**
Provide Business Case	A business case is usually a financial argument justifying the cost of the build and operation of a computer application. It can also be a regulatory argument.	Req
Define Goals	Goals are measurable objectives that the project is looking to achieve, usually tied to business case.	Req
Define Vision	A Vision is a communication artefact, used to articulate to all stakeholders, including the development team, the intent of the solution, what is it looking to achieve.	Req
Derive Initial Requirements	The initial requirements are requirements for the solution, at a high level, focusing on the scope of the solution rather than the fine detail.	Req
Define Functional	Functional Requirements describe how the solution functions, defined at a level to allow processes to be	Req

Activity	Description	Project Phase
Requirements	defined, software built and then verified that it does meet those requirements.	
Define Technical Requirements	Technical requirements are requirements for the solution focusing on how it needs to operate to required tolerances (speed, load, security, etc.), separate from the business functionality it needs to provide. (non-functional requirements)	D&C
Define Quality Requirements	Describes what checks and balances need to be in the development process to ensure the right solution is created as well as checks and balances in the solution to ensure it meets operational requirements. (non-functional requirements)	Req
Balance Requirements	Ensuring that competing requirements e.g. cost vs. security vs. functionality don't contradict each other.	Req
Approve Requirements	Confirmation from all customer-side stakeholders that the solution as defined will meet their objectives.	Req
Define Manual Processes	Define how the software will be interacted with and how activities not related to the software but part of the overall solution are carried out.	Req
Derive System Requirements	Identify hardware and software needed for the solution to operate as required.	Req
Specify Scenarios	Define discreet scenarios describing how different personas / roles will use the solution to carry out a function as part of their job.	Req
Define Architecture	Define what hardware, frameworks and structures will be needed for the solution to operate and meet the requirements.	D&C
Build & Unit test	Create code and documentation and ensure that the code, documentation and processes work in isolation to specified requirements.	D&C
System Test	Ensuring that the solution meets the defined requirements (technical and business) and additional ensuring that solution does not behave in an unexpected manner.	Systest

Activity	Description	Project Phase
Specify UAT	Define what checks, tests and inspections will be needed to assure stakeholders that the solution meets the requirements and Vision.	D&C
Perform UAT	Carry out checks, tests and inspections on the solution to ensure it meets the requirements (stated and unstated).	UAT
Implement Solution	Make the finished solution (software, hardware and operational processes) available to the target audience to be used in a "production" manner.	UAT

5.5 Data Collection Protocol and Validation

As changes were discovered, data was collected on a spread-sheet, by either the project manager or the senior analyst. Initially bi-monthly meetings took place to review the changes gathered though these became less frequent, due in part to the urgency of project delivery. The data was owned by the company until project sign-off, whereupon the company removed any company-confidential data before transference for research.

5.5.1 Data Validation

Effort was made to ensure that correct values have been entered against each change record. Observer triangulation was applied in the case of 'cost', 'value' and 'opportunity?' by the customer and project manager and remaining data items by project manager and senior analyst. Methodological triangulation between the qualitative 'change description' and the quantitative factor was achieved during the change review meetings with a researcher and project manager. A number of changes, randomly selected, were reviewed at these meetings. Roughly 60% of changes were re-examined, though data quality was high and only a small percentage of changes were amended, usually due to completion of missing data items.

5.5.2 Data Review Process

During the data review meetings, in addition to data validation, the trigger placement within each change domain was reviewed and the taxonomy amended as required. For example, the change trigger 'New Market Technology' was added to the domain of *market* to differentiate it from the trigger 'New technology' residing in the *solution* domain. Also, some overlap between triggers in the *specification* domain and those in the *vision* domain were identified. For example, 'Increased Customer Understanding' could change both the v*ision* and the *specification*. In these cases a referral was made to the overall domain definition; if a change was being made as a result of the 'problem' changing, the change was attributed to the domain of *vision*, else to the domain of *requirements specification*. In doing so, no problems were reported. Perhaps not surprisingly, particularly towards the latter phases of the project the customer and software provider experienced increased difficulty in coming to agreement about whether the change represented an opportunity or a defect. This was evidenced in cases where the customer was expecting something implicit or assumed within the agreed documentation. Therefore an allowable value - 'Undefined' - was added to the 'Opportunity?' data item.

Changes firing in a certain change domain may affect down-stream work- products in addition to the work product associated with that domain. For example, change triggered in the domain of *requirement specification* may change the solution work product, but will not fire a consequent *solution* change trigger. Therefore each change domain can be thought of as independent, even though the changes triggered in one domain may have a chain affect and require changes to many work products.

5.5.3 Data Analysis Methods

5.5.3.1 Analysis Procedures

Descriptive tables and graphs are complemented by statistical procedures to test for differences in factors between change domains. Procedures were selected on the basis of underlying distribution and variable scale assumptions. Data pertaining to change cost did not follow a normal distribution (see results section 5.6.2 'The Cost of Change') and many of the data items have a nominal (categorical) scale as indicated in Table 5.1. What follows

are short descriptions of the appropriate statistical test, and the research questions that they are employed to address.

5.5.3.2 Hypothesis Testing

The Kruskai Wallis test allows comparison of groups of data scores (ordinal or scale type), and tests whether the scores could be thought to come from different groups, that is, that there is a significantly different central tendency for each group. Simply put, when data does not conform to a normal distribution, (as is the case with the change costs), using this test is one of the ways groups can be compared without reference to mean values. This test uses score rankings in place of actual scores to perform the statistical test. Post-hoc procedures include examination of pairs of groups to determine where the main differences lie (Mann Whitney test). These tests will be used to examine the change costs observed for changes within each change domain. The Chi-squared test looks for relationships between two categorical variables, by comparing the observed frequencies in certain categories with expected frequencies. This test is appropriate for examining the ordinal scale for value as well as the nominal variables selected to represent managerial considerations, and was used for the remainder of the variables. The reader is referred to [170] for details of these tests.

5.6 Results

An overview of the frequency and timing of changes is followed by analysis of the variables included in this study.

5.6.1 Overall Look at Changes during the Developmental Lifecycle

From project inception to delivery, over a period of 16 months, a total of 282 requirements changes were recorded, at a cost of 2405.5 days effort which represents more than 50% of the final project cost of 4222 days. Table 5.3 illustrates the phase during which these changes were discovered, and the change source domain. Since the project followed a strict traditional waterfall process, the phases are temporally contiguous.

Table 5.3 NUMBER OF CHANGES PER PHASE PER DOMAIN.

	Req.	D&C	SysTest	U.A.T.	Total
Market	0	1	0	0	1
Organisation	30	4	0	0	34
Vision	15	1	1	7	24
Specification	22	58	5	102	187
Solution	0	33	3	0	36
Total	67 (24%)	97 (34%)	9 (3%)	109 (39%)	282

As can be seen, a high proportion of these changes occurred during the 'User Acceptance Testing' and the 'Design and Code' phases of the project. Since this was a project intended for a particular customer rather than a market-based initiative, it is not surprising that there is only 1 *market* change (which related to following market trends in COTS usage). This change (costing 30 days effort) was removed for all subsequent analysis, and means that this study is limited to the examination of the remaining four change domains. In addition, changes involving only requirement deletions (12) at zero cost are excluded from future analysis, reducing the total number of changes considered from 282 to 269. As a different view, change frequencies by change domain and life-cycle phase are illustrated in Figure 5.1. Notably, all *organisation* change was discovered during the requirements analysis or design and code stages of software development, with by far the greater number occurring during analysis. This is by direct contrast to changes arising in the domain of *requirements specification* where the majority of changes were discovered during U.A.T. *Vision* changes occurred primarily in the stages with customer involvement, namely requirements analysis and U.A.T. *Solution* changes were nearly all discovered during design and code. Some *specification* changes were also realised during this stage.

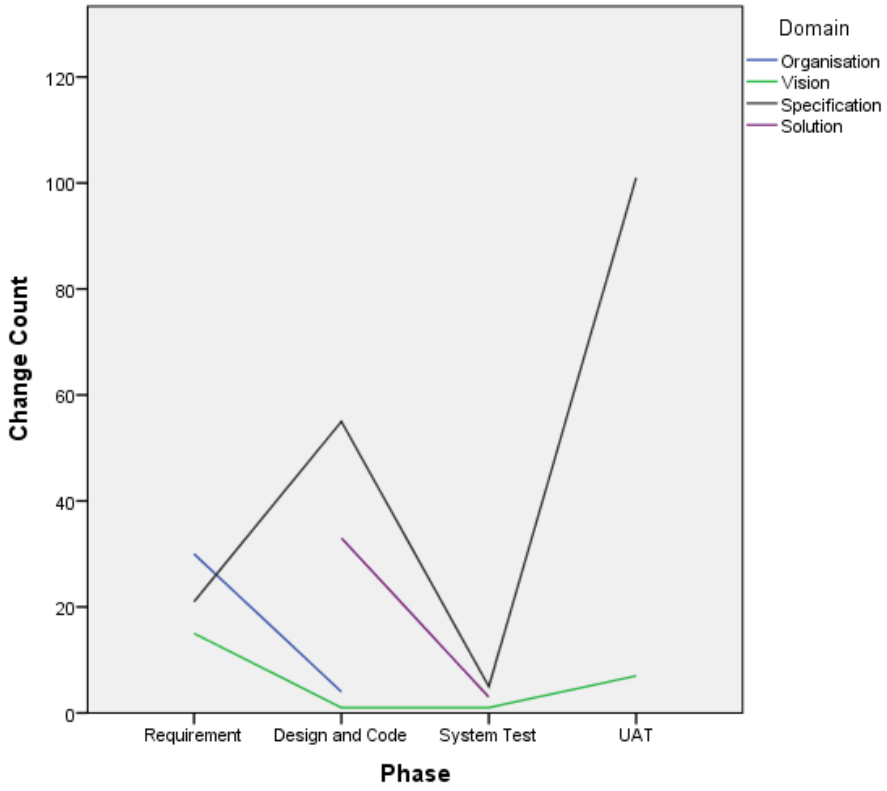

Figure 5.1 CHANGE FREQUENCIES BY CHANGE DOMAIN AND PROJECT PHASE

5.6.2 The Cost of Change

The analysis of change cost discounts the 12 deleted requirements. Figure 5.2 illustrates the frequencies of change costs for the entire project across all change domains. Change cost is not normally distributed (Shapiro-Wilk W = 0.669, p< 0.001), and is highly positively skewed due to the lower limit of zero cost being fixed. Since examination of mean values for cost is therefore less meaningful, future analysis uses the median values as representation of central tendency. As can be seen the majority of changes cost less than 20 days of effort, with the highest recorded cost being 70 days, of which there are two changes both in the domain of *customer organisation.* The median change cost is 4 days, though less than 20% changes took a day or less to make.

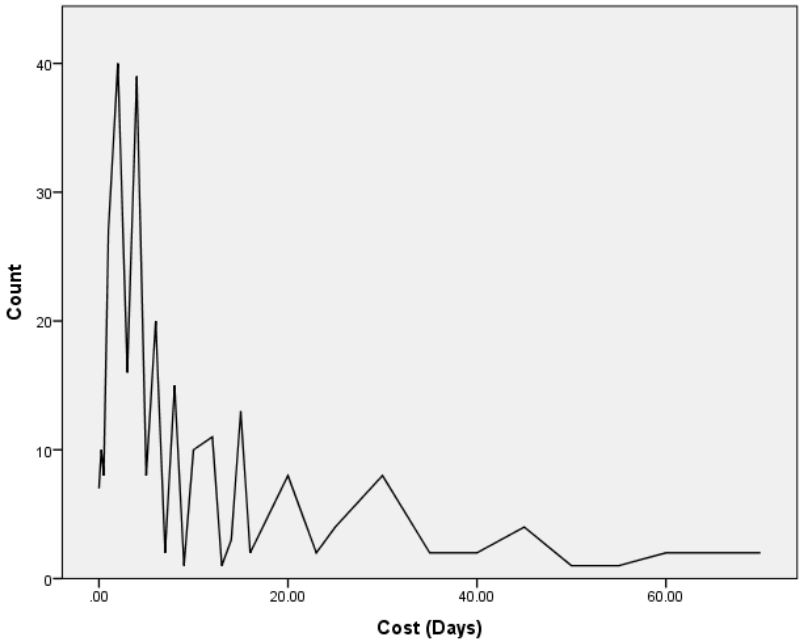

Figure 5.2 FREQUENCIES OF CHANGE COSTS ACROSS ALL CHANGE DOMAINS.

Table 5.4 shows a breakdown of these costs as they pertain to project phase and change domain. Although the most significant cost was experienced during the initial phase of requirements confirmation, in line with the number of changes discovered, a high percentage of change cost occurred during User Acceptance Testing (38%). By far, the largest percentage of cost came from the *specification* domain which represented half of the total change cost (excluding the market change) throughout the development life-cycle (49%).

Table 5.4 CHANGE COST PER PHASE PER DOMAIN

	Req	**D&C**	**SysTest**	**UAT**	**Total**
Organisation	638.0	64.0	0.0	0.0	702.0
Vision	266.0	5.0	2.0	163.0	436.0
Specification	193.9	222.0	4.5	737.0	1156.5
Solution	0.0	78.0	2.5	0.0	81.0
Total	1097.0 (46%)	369.5 (16%)	9.0 (0.4%)	900.0 (38%)	2375.5*

** Excludes Market change at a cost of 30 days*

Figure 5.3 illustrates a comparison between median (represented by bars) and total cost (represented by a line) for each change domain. While total costs are significantly higher in the *specification* domain, the medians of these costs illustrate that on average changes due to *organisation* changes are the most expensive, followed by changes to the *vision,* then *specification*, and with the lowest average cost being in the *solution* domain. The Kruskai Wallis test indicated that the differences are significant (H(3) =75.038 , p < .001) to the extent that these changes could be thought of as coming from different groups. While this indicates that there is a difference overall, it does not inform us of where the major differences lie. Performing selected Mann Whitney tests to test for differences between adjacent domains reveals that the median of the domain of *organisation* does not vary significantly to that of *vision* (U = 229.5, z = -1.787, p > 0.05) , but that *vision* differs from specification (U = 851.500, z= -4.879, p <0.001) and similarly *specification* differs from *solution* (U=1901, z= -4.006, p < 0.001). Since costs change over time, it is useful to explore the differences in domain costs for each phase of the project.

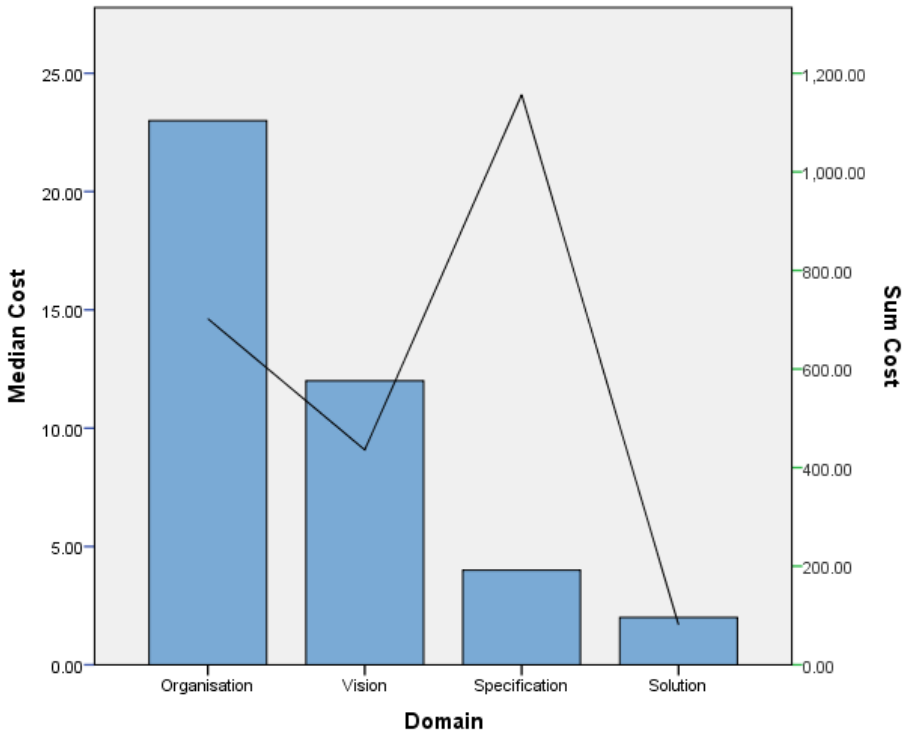

Figure 5.3 MEDIAN AND SUM OF CHANGE COSTS FOR EACH CHANGE DOMAIN

Figure 5.4 shows median change costs for each domain in each phase. It can be seen that the trend in all domains is generally reflective of the results we saw when all phases were included, with the most expensive changes occurring in the *organisation* domain (median 23

days) and the least expensive in the *solution* (median 2 days). Costs tend to fall in the second and third phases and in the case of the *vision* and *specification* domain rise sharply during User Acceptance Testing. From quantitative data alone, it is impossible to assess whether this rise in cost is due to increased change size (more function points per change) or rework of existing code and architecture. Performing the Kruskai Wallis test on phase one data alone indicates that there is significant difference in the costs in the three domains of *organisation, vision* and *specification* ($H(2) = 15.239$, $p < 0.001$). Similarly, overall cost medians are significantly different in phase two ($H(3) = 10.692$, $p < 0.05$). As can be seen though, there is no difference in this phase between costs in the domains of *specification* and *solution*. It is not possible to do median comparisons for phases three and four due to insufficient data.

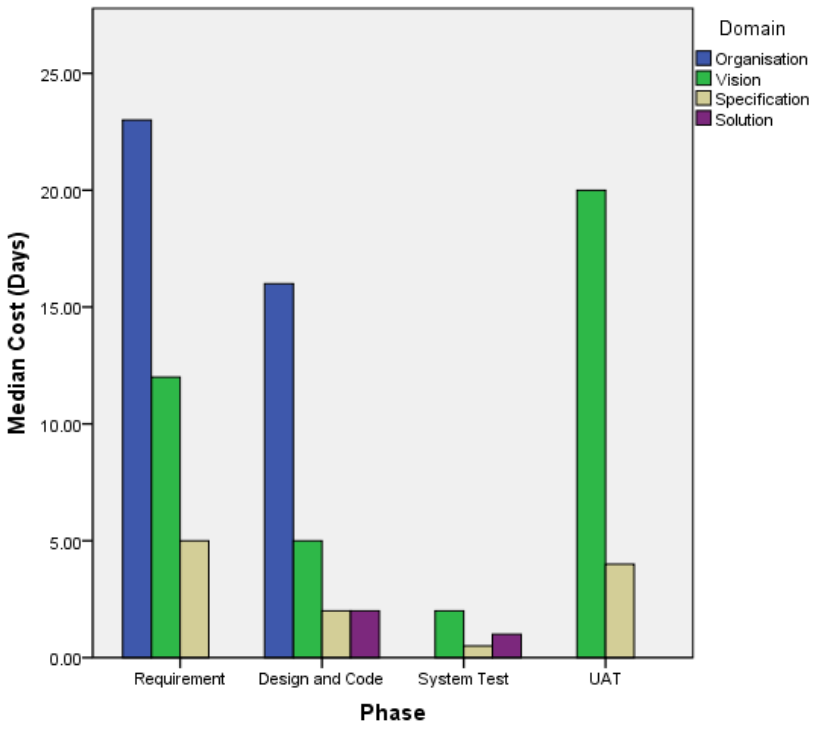

Figure 5.4 MEDIAN CHANGE COSTS FOR EACH CHANGE DOMAIN IN EACH PROJECT PHASE.

As can be seen, change costs are not consistent across change domains; the most expensive changes come from the domain of *organisation* and costs decrease through the domains of *vision, specification* and *solution*.

5.6.3 The Value of Change

Table 5.5 illustrates the proportion of subjectively assessed value accruing from these changes across the change domains. Just over half of all changes are of a very low value (51%), and of those the greatest proportion (74%) was in the *specification* domain. The highest value changes are only in the domains of *organisation* and *vision*. By contrast most of the changes in the solution domain are of very low value (91%). It must be remembered though, that 'value' in this context refers to business value to the customer. Changes coming from the *solution* domain may be of indirect value as they affect the quality of the entire solution. It may be the case also, that they are not fully understood by the customer. A chi-squared test (performed with value 'High' and 'Very High' changes added together due to low frequencies in these groups) reveals that there is an uneven distribution of values across the four domains ($x2 (9) = 144.354$, $p < 0.001$).

Table 5.5 CHANGE VALUE PER DOMAIN

	Value					
	Very Low	Low	Med	High	Very High	Total
Organisation	0	7	9	5	6	27
Vision	2	9	11	0	2	24
Specification	102	64	13	3	0	182
Solution	33	2	1	0	0	36
Total	137 (51%)	82 (30%)	34 (13%)	8 (3%)	8 (3%)	269[1]

From the perspective of business value, these requirements changes could be thought of as coming from different groups according to the change domains specified. The highest value changes come from the domain of *organisation* and the lowest from *solution.*

[1] market change and changes representing requirements deletions removed in this and subsequent tables.

5.6.4 Opportunity/Defect

Changes can represent opportunities to enhance functionality as well as correction of previous errors. Table 5.6 illustrates how these are spread across the change domains. Changes representing an opportunity comprise the majority of changes in the domains of *organisation* (89%), *vision* (75%) and *specification* (57%), and represent a total cost of 1677 days effort. This is somewhat surprising since the definition of the change domain *requirements specification* relates to the process of translating the needs of the customer into a format that provides the basis for programming. Thus, this domain not only caters for the resolution of ambiguity and inconsistency, but also improved provision for agreed requirements.

Table 5.6 NUMBERS OF CHANGES BY DOMAIN CATEGORISED AS OPPORTUNITY, DEFECT OR UNDEFINED

	Opportunity	Defect	Un-defined	Total
Organisation	24	2	1	27
Vision	18	5	1	24
Specification	104	62	16	182
Solution	13	20	3	36
Total	159 (59%)	89 (33%)	21 (8%)	269

Defects, costing a total of 559 days effort are more often the cause of change in the *solution* domain which represents (56% of the total cost of solution changes). When the customer and software provider have not been able to arrive at an agreement about whether the change represents an opportunity or a defect is has been referred to as 'Undefined'. In this case, most of these changes related to assumptions regarding functionality implementation methods. They represent a small proportion of all changes (< 10%), have a cost of 139.5 days effort, and are mostly in the *specification* domain. The chi squared test is significant (X^2 (6) = 21.662, p = 0.001) confirming that there is an uneven distribution of opportunity change across these change domains

Across change domains, is there a significant difference in the proportion of opportunity vs. defect related change. These results show that the proportion of changes representing an opportunity as opposed to a defect are not evenly spread across domains. Opportunity

change is more often seen in the domains of *organisation* and *vision*, while defects predominate the domains of *specification* and *solution.*

5.6.5 Number of Stakeholders

As the software provider was considered a stakeholder, changes involving only one stakeholder were either those that required decisions to be made without customer involvement, or those where a single customer stakeholder group was able to make changes that required only agreement rather than negotiation with the software provider. Table 5.7 illustrates stakeholder groups involvement in each change domain. In all domains there is greater proportion of changes requiring more than one stakeholder group. (89% of *organisation* changes, 96% of *vision* changes, 90% of *specification* changes and 56% of *solution* changes).

Table 5.7 CHANGES CATEGORISED AS NUMBERS OF STAKEHOLDER GROUPS INVOLVED IN AGREEING CHANGE PER DOMAIN

	Stakeholder Groups			
	1	2	3 or more	Total
Organisation	3	18	6	27
Vision	1	15	8	24
Specification	19	149	14	182
Solution	16	20	0	36
Total	39 (14%)	202 (75%)	28 (10%)	26

However, in the domains of *organisation* and *vision*, there are proportionally more changes requiring the involvement of three or more stakeholders (22% and 33% respectively) compared with the *specification* domain (8%) and *solution* (0%). In the *solution* domain we see a greater proportion of single stakeholder changes (44%) than in any other domain. A chi squared test indicates that there is dissimilarity in these domains when considering the numbers of stakeholder groups usually involved in the change ($X^2(6) = 50.795$, $p < 0.001$). Interestingly, median costs also rise as the number of involved stakeholders increases. The median cost when one stakeholder group is involved is 2 days effort, compared with 4 days

for 2 stakeholders and rising sharply to 10 days for 3 or more stakeholder groups. Since this costs includes any time required to re-analyse the requirement, the increased cost could, in part be explained by the observation from the practitioners that when more than stakeholder is involved, negotiation is often necessary.

Across change domains, is there a significant difference in the number of stakeholders involved. More often a higher number of stakeholders are involved with *organisation* and *vision* change.

5.6.6 Discovery Activity

The project activities during which the changes were found are shown in Table 5.8. In the main, change discovery is clustered in three main activities: 'Define functional requirements', 'Build and Unit test' and 'Perform UAT'. Changes are also discovered during activities related to the specification of technical and system requirement and system testing. Perhaps surprisingly, there was only 1 change discovered during 'Balance Requirements' which was the activity designed to ensure that that requirements did not contradict one-another.

A comparison between change domains reveals that although a high proportion of *specification* changes were discovered during UAT (55%), many of the *organisation* changes (63%) and *vision* changes (46%) were discovered during the 'Define Functional Requirements' activity. *Solution* changes in the main were discovered during build and test (64%). Note that Specify UAT's was during the D&C stage of the software development lifecycle.

A visual analysis of these changes, presented in Table 5.8 would suggest that changes in different domains are discovered during different activities in the developmental lifecycle. However, while these results were of interest to our industrial partner, there is insufficient data to perform a chi squared test for inequality of change discovery activity spread amongst domains. Consequently, this question was revisited and the activities grouped in order to reduce their number.

Table 5.8 CHANGE DISCOVERY ACTIVITY PER CHANGE DOMAIN

	Organisation	Vision	Specification	Solution	Total
Provide Business Case	0	0	0	0	0
Define Goals	0	0	0	0	0
Define Vision	1	1	0	0	2
Derive Initial Reqs	0	0	0	0	0
Define Functional Reqs	17	11	14	1	43
Define Technical Reqs	1	1	3	0	5
Define Quality Reqs	0	0	0	0	0
Balance Reqs	0	0	1	0	1
Approve Bus Reqs	1	0	0	0	1
Define Manual Processes	0	0	0	0	0
Derive System Reqs	4	2	5	2	13
Specify Scenarios	0	0	2	0	2
Define Architecture	1	0	0	2	3
Build and Unit Test	0	1	25	23	49
System Test	0	1	5	8	14
Specify UATs	0	0	26	0	26
Perform UAT	0	7	101	0	108
Implement Solution	0	0	0	0	0
No Activity	2	0	0	0	2
Total	27 (10%)	24 (9%)	182 (68%)	36 (13%)	269

This resulted in the four activities contained in Table 5.9. Demo (demonstration) was understood to be any communication or review by any stakeholder of a part of the application, either by presentation or prototype demonstration to any number of people. ‘Translate’ described the process of using a previously created work product to create a new one, for example creating code from specification. ‘Test’ is any activity related to testing, and ‘external’ is an event outside of the remit of the project manager.

A high percentage (59%) of *organisation* change was discovered through external activity, whereas all *solution* changes were discovered through translation activities. *Vision* Changes are more frequently discovered through demonstration activities (50%). A high percentage of *specification* changes (63%) were discovered through activities related to testing. A chi squared test confirms that there is a significant difference between activities which discover change in each of the domains ($X^2(9) = 265.4$, $p < 0.01$).

Table 5.9 CHANGE DISCOVERY EVENT PER CHANGE DOMAIN

	Event				
	Demo	Translate	Test	External	Total
Organisation	7	4	0	16	27
Vision	12	7	1	4	24
Specification	7	61	114	0	182
Solution	0	36	0	0	36
	26 (10%)	108 (40%)	115 (43%)	20 (7%)	269

Across change domains, is there a significant difference in the activities during which changes are found.

5.6.7 Project Management Control

As stated, the process followed in this project adhered to a waterfall approach wherein attempts are made to define all requirements at the beginning of the project. 'Project Management control' captures a subjective assessment by the project manager regarding the ease by which these changes may have been discovered earlier. It was felt that some changes would have been impossible to find (pm control = 'Very low') even with improved techniques. An example of a change such as this is changing the list of Internet browsers that the system was intended to be compatible with, following an organisational study of browser usage. By contrast those that the project manager believed may have been uncovered with more time, or different techniques (pm control = 'Very high') would include changes such as screen layout modification. Such changes may have been uncovered through application walk-throughs or by use of prototypes.

These results, illustrated in Table 5.10, indicate that all of the changes over which the project manager has the most control lie within the domains of *specification* and *solution*. There is a proportionally greater volume of 'Very low' control change in the domain of *organisation* (26%) than in *vision* (4%), *specification* (2%) and *solution* (3%). As it stands the data is insufficient to perform a chi squared test. However, when pm control = 'Very low' & 'Low' and pm control = 'High' & 'Very high' are compressed into single categories, the data meets the criteria necessary for this test and is significant ($X^2(6) = 85.113$, $p < 0.001$) indicating

that in general the level of project management control differs according to the domain from which the change arises.

Table 5.10 EXTENT OF MANAGEMENT CONTROL OVER CHANGES PER DOMAIN

	Project Management Control					
	Very Low	Low	Medium	High	Very High	Total
Organisation	7	4	16	0	0	27
Vision	2	3	18	1	0	24
Specification	3	13	126	31	9	182
Solution	1	1	5	18	11	36
Total	13 (5%)	21 (8%)	165 (61%)	50 (19%)	20 (7%)	269

Across change domains, there is a significant difference in the level of project management control. It was considered that a higher proportion of *solution* and *specification* changes could have been discovered earlier by the use of alternative approaches or techniques, while much *organisation* and *vision* change would have occurred regardless of analysis effort.

5.7 Discussion

Overall, requirements changes amounted to over >100% of the original project cost, which means that the project more than doubled in size. This is greatly in excess of Nolan's [16] observation that on average changes amount to 50% original project cost, and Jones [15] conclusion that changes represent between 14% and 26% original project size. The observation by Weiss [72] that over 75% changes took less than a day to make was also not borne out, with less than 20% of changes taking a day or less and the median change cost being 4 days. It is not clear why these results are not in line with other studies, though context, and timeliness will all affect the outcome.

The analysis of data collected in this study has allowed us to assess whether there is any correlation between the change taxonomy groups and change attributes reflecting change cost, value, stakeholder involvement, discovery activity, project management control, and whether the changes represented an opportunity to add value or a correct an error. Results indicate that there is a clear distinction between changes falling into the classifications in

this taxonomy. Not only do changes arising due to customer *organisation* changes cost more on average and accrue more value but they also generally involve the agreement of a higher number of stakeholder roles and were considered by the participants to be more difficult to uncover. This mirrors the observation made upon derivation of the change source taxonomy in section 4.10 , that by comparison to other change domains, relatively little is understood (by the software provider) about uncertainties that give rise to change in the domain of *organisation*. This is in stark contrast to *solution* changes which are in the main controllable and less costly than changes from other sources. However, the fact that customers rated *solution* changes to be of very low business value perhaps belies a lack of understanding of how these types of changes support business functionality. These results are in accordance with those of Nurmuiani [22] who observed that changes coming from sources external to the project require more effort to implement.

As well as differences in cost and value, there are also differences of management considerations between changes due to *vision* changes and those coming from *specification.* While it was possible to uncover changes from *specification* issues during Build and Test, any *vision* changes not already discovered during requirements specification were not found at this stage and remained until User Acceptance Testing. The large proportion of changes that were discovered during User Acceptance Testing at the end of the development life-cycle (39%) support Thakurta's [9] observation that projects employing a waterfall process often have high mid to late stage volatility.

Bano [22] refers to two types of changes; 'accidental changes' and 'essential changes'. These categories may be identified by referring to them as an opportunity to add value or attend to a defect fix. It is interesting that all domains with the exception of *solution* gave rise to a higher proportion of opportunity change, so therefore there is no direct comparison between accidental and essential changes with elements of the change taxonomy. Since the software provider and the customer were not always in agreement as to whether a change represented an opportunity or a defect, so the classification presented by Bano [22] may not be easy to apply in practice. This was more evident in the domain of requirements specification where changes arise due to the resolution of ambiguity and inconsistency. While both the domains of vision and requirements specification admitted to a higher proportion of opportunity change, the difference was much more marked in the *vision* domain with very few changes attending to errors.

Since this study is the first attempt to assess the informative value of any means of classification, it is not possible to compare the relative efficacy of this taxonomy with others.

5.7.1 Review of Results Validity

This data used in this study was defined using the GQM approach in line with study objectives. However, in some instances, a pragmatic approach requiring compromise between the needs of academia and industry was required. The following discussion reviews the relevant limitations to validity outlined in section 3.4.6.

Construct Validity

Since the data was discussed and specified through meetings with researcher and practitioners, there is a shared understanding of the meaning of the data items. They were defined specifically to answer the questions in these studies. While the results rely upon subjective measures, the participants in this study can be considered experts since their knowledge of both the business and the software development domain is extensive. All participants have at least 10 years of experience in their respective software development roles and at least 5 years of experience in software application development within the government sector.

External Validity

No claims to external validity can be made, in that without study replication, it is impossible to draw conclusions about the application of results to software development environments that are different to the study context described here.

Study Reliability

Attempt has been made to ensure study reliability which is defined as “the extent to which the data and analysis are dependent upon specific researchers.” [41]. Details of data specification and collection protocol are provided for use by other researchers. Efforts were made to encourage a high level of data quality through data reviews with a practitioner and a researcher and cross validation (where a second industrial participant checks the data collected).

5.8 Conclusion and Implications for Research

This case study was designed to investigate if the software requirement change source taxonomy is an informative basis for change classification. The involved project followed a traditional waterfall lifecycle, lasted 16 months and had a total of 282 requirements changes at a cost of 2405.5 days effort from a total project cost of 4222 days. Six characteristics of change were identified as being significant to change management and all (non-deletion) changes were examined in light of the change domain from which they originated. The change factors considered significant were:-

i. Change cost
ii. Change value
iii. Proportion of opportunity vs. defect related change
iv. The number of stakeholders involved
v. The activities during which changes are found
vi. The level of project management control

The results are summarised in Fig. 13. The arrow indicates the tendency for increasing cost, value and opportunity change from the *solution* domain through the *specification*, *vision* to the *organisation* domain. At the same the level of project management control is decreasing. While this study did not investigate changes arising from the domain of *market*, it has been included here for completeness in lighter shading. In addition there is a notable difference in the activities during which the changes are discovered. *Organisation* and *vision* changes are more often discovered in activities related to requirements analysis, and user testing, whereas it is possible to discover *specification* and *solution* changes during the design and code activities. There is no direct mapping between a requirement and an element in this taxonomy. A single requirement can be thought to comprise a slice consisting of elements of all 5 domains in differing proportions depending upon the developmental phase and position within the requirements hierarchy. Any requirement is therefore subject to change arising from any change source domain.

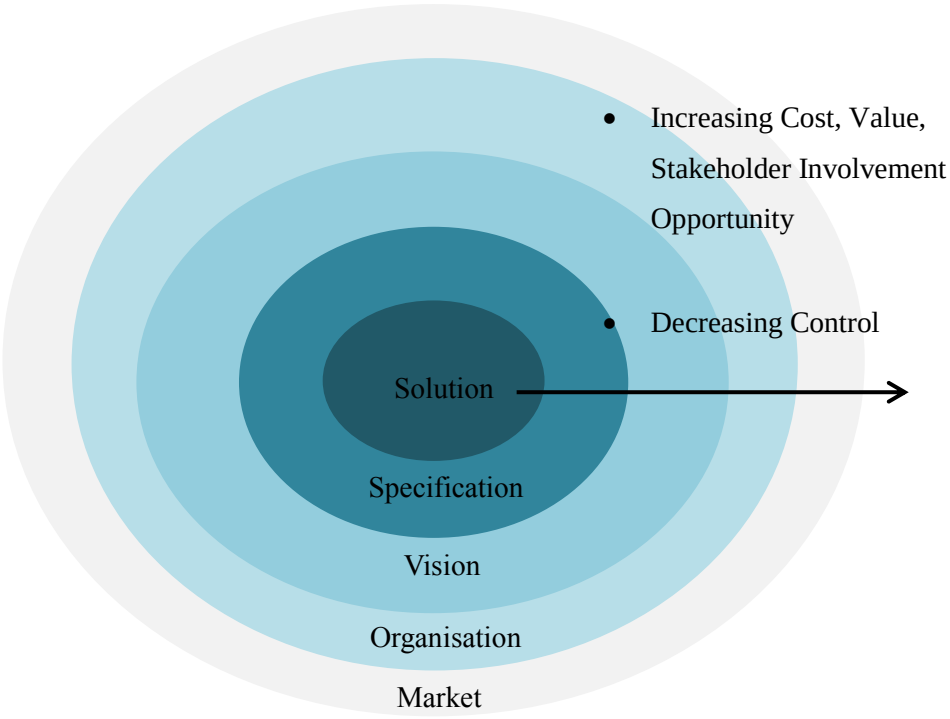

Figure 5.5 CHARACTERISTICS OF THE REQUIREMENTS SOURCE CHANGE TAXONOMY.

The implication of these results is that the management approach to change management and assessment of risk to project schedule, cost or quality should be reflective of different type of changes. For example, the high number of *vision* changes discovered during hands-on system usage during User Acceptance Testing may provide empirical support for the use of more agile techniques such as early prototyping or iterative development. Indeed, while it may be the case that agile techniques assuage late vision change, the observation that many *specification* changes were discovered during build and test may imply that the onus is upon analysis techniques as well as process procedures to reduce the types of changes that arise from *specification* issues. Maintaining change data in this way across multiple projects would allow software providers to assess the efficacy of analysis techniques and guide future process selection decisions.

Since a higher proportion of *organisation* and *vision* changes represent an opportunity to enhance previously agreed functionality as opposed to the correction of defects, the taxonomy also captures the notion that some change should be encouraged, and some types of change avoided. Despite the concerning fact that this project increased in size by over 100% due to requirement changes, over 70% of these changes represented an opportunity to enhance previously agreed functionality rather than correct errors. The often quoted "requirements change is a risk to project cost and schedule" [20] [15] [12] must be balanced by the fact that much change is value-driven and therefore there is a risk to business value

should cost and schedule considerations take priority. Classifying change in this way will illuminate this issue in project retrospectives and support constraint of some change and encouragement of another.

As a contribution towards the goal of requirements prediction, the results of this study underline the importance of classifying change in a way that is informative to project management. Analysis of results have also further clarified the meaning of the elements of the taxonomy and confirmed that should it be possible to align predictions with the change domains, such a prediction would offer more benefit than a single point change estimate. However, since there was some overlap in the change trigger constructs, future case studies that use change data will use the change domain instead of a trigger construct (unless a preference is specified by the practitioners).

The results also highlight the potential constraints to prediction using current knowledge. To uncover environmental factors that contribute to uncertainty associated with higher risk of customer *organisational* change, it would be necessary for project analysts to broaden the scope of application analysis to wider organisational concerns. In order to predict changes to compliance requirements, which would fall into the domain of 'Market', Maxwell needed to take the loci of the predictive influences well outside of the software engineering environment [14]. Traditionally, the change domains of project vision, requirements specification and solution are the focus of software application engineering. Although some research suggests that inclusion of market and customer organisation analysis during requirements elicitation can improve application value, the current UK software procurement process begins with project vision.

Having met the study objectives of determining a list of factors of change that are significant for change management decisions, and investigating whether the requirements change source taxonomy is an informative way to classify change, the answer in the affirmative leads us to the next stage of research. What remains of research question '**RQ2** How can change be classified and measured in order that it is informative to project management?' is to provide a metric for volatility that will be used alongside the taxonomy in order to measure change as a profile construct. Environmental, project and product factors that influence change have been identified, but there may also be attributes of the requirements themselves that render then more susceptible to change than others. Therefore the next study provides a metric for volatility, and addresses the question '**RQ3** Are there attributes of requirements that make them more susceptible to change?'

Chapter 6 Change Prone Requirements

The findings from empirical studies described in the previous chapters have resulted in the identification of an exhaustive list of change source constructs, and the derivation of an informative classification of software requirements change based upon these constructs. However, it may be the case that in addition to changes coming from the software environment, inherent in requirements themselves may be attributes that render them more change-prone. This chapter proposes a simple metric for change measurement, and describes a case study that examines attributes of requirements such as novelty or complexity for correlation with volatility. Correlations for each of the domains in the taxonomy are also examined to ascertain if there are differences in patterns of volatility between change domains. This study addresses the following research questions:-

RQ2 How can change be classified and measured in order that it is informative to project management?

RQ3 Are there attributes of requirements that make them more susceptible to change?

Chapter Contributions to Academic Knowledge

a) The causes of requirements change

A case study further investigates the causes of requirements change through investigation of the correlation between volatility and attributes of requirements themselves.

6.1 Introduction

Towards the ultimate goal of requirements change prediction, it is necessary to fully investigate the causative factors that may influence levels of requirement volatility. Thus far, research towards that goal has focused upon attributes of the project, process, product and environment that may give rise to change. Notable among these, and present as constructs in the change source taxonomy (see section 4.7,) are product attributes of novelty (novelty of application) and complexity (how difficult the problem is to define). At product level, these constructs were placed in two separate domains of the taxonomy, respectively *vision* and *requirements specification*. However, these attributes can be considered to describe an individual requirement, and may vary from one requirement to the next. Encouraged by research that claims that some requirements are more change-prone than others [10][13][14], the primary objective of this study is to ascertain if such requirements attributes can be indicative of levels of volatility. This begins an exploration of the relationship between the causes and effects of software requirements change. Further, it may be the case that requirements that are susceptible to change from one domain may also be disposed to change in other domains. For example, should it be the case that requirements that change due to *organisational* changes are also typically prone to *vision* changes, early indications of *organisational* change will increase a belief in future *vision* changes. Therefore, patterns of volatility across change domains will be investigated. In addition to novelty and complexity, there may be other requirement attributes that should be considered. Therefore, it is first necessary to determine a list of candidate requirements attributes. In order to investigate correlation with volatility, a metric is defined that facilitates statistical testing and is easy for our industrial partners to collect.

The objectives of this study are therefore:-

1. Determine a quantitative measure for volatility.
2. Determine a list of requirements attributes that are considered to affect requirements volatility.
3. Determine if requirements that are susceptible to change from sources in one domain are also disposed to changes coming from other domains.
4. Examine correlation between requirements attributes and volatility.
5. Determine if identified patterns of correlation are evident in all change domains.

6.2 Empirical Context and Execution

This case study shares industrial and project context with the previous case study, details of which can be found in section 5.2.1 and section 5.2.2 respectively. The unit of analysis in this case study is the requirement. There were three industrial participants in this study, the project manager, technical architect, and lead analyst. An explanation of case study research can be found in section 3.4.3.

6.3 Requirements Volatility Measurement

A metric for volatility for use in this study has two requisites. Firstly, it must facilitate statistical analysis, and secondly, it must be practical and easy to collect by industrial practitioners. Of volatility metrics proposed in the literature, the use of function points [86], measures of user need diversity [87], and an index of deep change to total change [88] can be eliminated for practical reasons since they cannot be easily collected by our industrial partners (see section 2.3 for a discussion of proposed volatility metrics).

A simple measurement of changes to requirements involves counting the number of requirement changes over a period of time, thus expressing a change frequency [81]. However, since changes can vary considerably in terms of the effort involved, a metric based solely upon frequency does not take into account the associated cost. An approach taken by a number of researchers is to formulate the frequency as a ratio to the total number of requirements [71][13][68]. This may go some way to address that weakness, though a metric based upon the amplitude of the change better communicates the risk to project cost [86][171]. Following Barry, who advocates that a metric for change reflect change amplitude as well as frequency, the approach taken in this study, and thereafter, is to define requirements volatility in terms of change cost. Importantly, this is expressed relative to the original estimate, so that a change of 4 days to a requirement whose original estimate was 20 days will be less volatile (20%) than a change of 4 days to a requirement whose original estimate was 2 days (200%).

Change cost was defined in a previous study (see section 5.4) as the difference between a revised cost estimate in days for a requirement given changes, and the unused portion of the original cost estimate. For example, a requirement is originally estimated to take 10 days. After working on it for a period of 2 days, a change is discovered. The revised effort estimate is a further 15 days. The change cost is calculated as 15- (10-2) = 7.Thus, for a requirement with original cost C, a revised cost of C', and used portion of previous estimate UC,

$$\text{Change cost} = \ C'_n - (\ C - \ UC)_n$$

The sum of all these change costs, expressed as a percentage ratio of the original estimate is the value for volatility. Hence

$$V_r = 100\ x\ \frac{\sum_{n=1}^{n=N} C'_n - (\ C - \ UC)_n}{C}$$

Equation 6.1 VOLATILITY OF A REQUIREMENT

This metric satisfies the needs of research in that it is easy to collect and calculate, and reflects the amplitude of changes.

6.4 Identification of Requirement Attributes

Rather than software-application specific attributes such as 'financial system interface', the intention was to identify generic qualities that could be attributed to any set of software requirements, so that results may support the formulation of general predictive models. Once again using the GQM approach [148], a project manager and senior analyst identified requirement qualities that, in their opinion, were most likely to give rise to change. A potential list of six attributes were identified, and reduced to requirements dependency, complexity, and novelty in order to reduce the data collection effort. Those not included in this study are framework (COTs) usage, business criticality and number of involved stakeholders. In a previous study (see section 4.7) complexity was seen to have a both a technical and a business faculty. Therefore complexity was defined as two separate attributes. Similarly, although novelty had not been previously explicitly defined as having a technical element, the change source construct 'uncertainty of solution' reflects the notion of novelty. A requirement that is entirely new to the business may be satisfied by technology that is well understood and often used by the software provider. Alternatively, a requirement

well understood and perhaps already met by existing software, may now be re-engineered using techniques that are new to the software provider. Requirement dependency was thought to influence the level of volatility since, in addition to changes it may incur, a requirement that is dependent upon others has a greater likelihood to change when a related requirement changes. The requirement attributes under investigation in this study are:

i. Requirement Dependency
ii. Requirement Business Complexity
iii. Requirement Technical Complexity
iv. Requirement Business Novelty
v. Requirement Technical Novelty

6.5 Data Specification

This data was collected solely for the purpose of research, so ease and efficiency of collection was an important consideration during data specification. The research goal is to make predictions of requirements change early in the project life-cycle, thus allowing adaptation of software processes and requirements handling techniques. At this stage, source code metrics are not available, and requirements may or may not have been documented. Therefore, prediction will be reliant upon the subjective estimates of project managers and analysts. The attributes of this study are therefore defined with this in mind. Rather than a Likert scale for each subjective data item, a short description for each allowable value 1-5 was defined by our industrial collaborators enabling agreement between individual data collectors. It is not certain that these descriptions would be fitting for other projects or organisation, but they support faster data collection in this case study by more than one practitioner. The data items and descriptions of subjectively measured dependency, novelty and complexity can be found in Table 6.1.

Table 6.1 DATA SPECIFICATION FOR REQUIREMENTS

Name	Description	Allowable Values and indicative meaning
ID	Unique Identifier	
Change ID	Change Identifiers	
Cost Estimate	Effort estimate of original requirement expressed in days. (Ratio, Subjective)	
Dependency	Pervasiveness of requirement (Ordinal, Subjective)	1. Requirement has no dependency to any others 2.Requirement has dependencies with non-functional requirements 3.Requirement has dependencies to other requirements in a tight functional area 4.Requirement had dependencies to other requirements in a tight functional area and consumes services from generic components 5.Requirement is a consumer of many other requirements throughout the solution
Business Complexity	Difficulty of requirement expression by the customer (Ordinal, Subjective)	1. Simple 2. Some thought needed 3. Can be done whilst multi-tasking 4. Challenging 5. Very Challenging
Technical Complexity	Difficulty of technical solution (Ordinal, Subjective)	1. Simple 2. Some thought needed 3. Can be done whilst multi-tasking 4. Challenging 5. Very Challenging
Business Novelty	How novel is the requirement to existing business processes? (Ordinal, Subjective)	1. Part of Day Job 2. Occasionally used 3. Occasionally used but not in this job 4. Familiar with concept 5 Entirely new idea
Technical Novelty	How novel is the technical requirement to the design, build and test teams? (Ordinal, Subjective)	1. Very similar to something done before by many of the team. 2. Some of the team have done something similar 3. We have examples but no direct experience 4. We are aware that something similar has been done before but we have no examples 5. We believe we may be able to do this but we aren't sure how
Cost Total Changes	Total cost of all changes made to that requirement (Ratio, Subjective)	
Domain Change Cost	Cost of changes in each domain made to that requirement (one data item for each domain) (Ratio, Subjective)	

As described in section 6.3 above, the change cost is defined as a sum of the revised estimate for any work remaining minus the unused portion of the previous estimate. There are therefore five cost data items, comprising a total change cost and additional change costs for each of the domains of *organisation, vision, specification* and *solution*.

6.6 Data Collection, validation and analysis

Case study data was assembled upon project completion, by the project manager and leading analyst. Change data had been collected in a previous study (see section 5.4), which included change costs. However, the requirements to which those changes applied had not been specified explicitly and instead requirement details were contained within the qualitative change description. A separate data source contained requirement details, and these two sources were combined through examination of the qualitative change descriptions. This process took approximately 4.5 hours. There was a many to many relationship between requirements and changes, so for simplicity, change cost was divided evenly amongst involved requirements. Having linked requirements to changes, it was necessary to collect the requirement attribute data described in Table 6.1. This data was extracted from requirements specification documentation by the project manager and technical architect who each worked independently on a subset of requirements. These were entered into a spread-sheet, and in situations where there was uncertainty about individual data items, it was agreed by all parties that they would be left blank, and reconsidered during data validation.

6.6.1 Data Validation

Once the data in spread-sheet form was made available for research purposes, data was checked visually for completeness. Since both practitioners were familiar with all of the requirements, data items left blank by one participant were requested from the other participant, and a small sample of data was also sent for cross-validation. Once this validation process was complete, volatility calculations were made for each requirement according to the equation specified in Equation 6.1 in section 6.3. There were five such data items; a measure of total volatility, and one for each of the domains of *organisation*, *vision, requirements specification* and *solution.*

6.6.2 Data Analysis Procedures

As in the previous study, descriptive tables and graphs are complemented by statistical procedures to test correlation between requirements attributes and volatility. Procedures were selected on the basis of underlying distribution and variable scale assumptions.

Spearman's Rho is a statistical test based upon ranks rather than scores, and tests for correlation between two sets of data that do not adhere to a normal distribution. This test is suitable for ordinal data such as the requirements attributes described in Table 6.1. It is possible to test for positive or negative correlation (two tailed), or positive correlation only (one tailed). A one tailed test is used to compare change measurements, and also determine if requirements that have a high degree of change in one domain also have a similarly high level of change in other domains. A two tailed test is used to explore correlation between requirement dependency, complexity and novelty, and volatility. Refer to [170] for details of this test.

6.7 Results

An overall look at the number and frequency of changes is followed by a comparison of measurements of volatility, and examination of change patterns across change domains, and an investigation of the correlation between requirements attributes and volatility.

6.7.1 Overall look at changing requirements

The developed application had a total of 240 requirements, of which only 40 were change free. Per requirement, change counts range from zero to 16 with the mean being 3.5. Figure 6.1 shows the frequency of changes made to requirements.

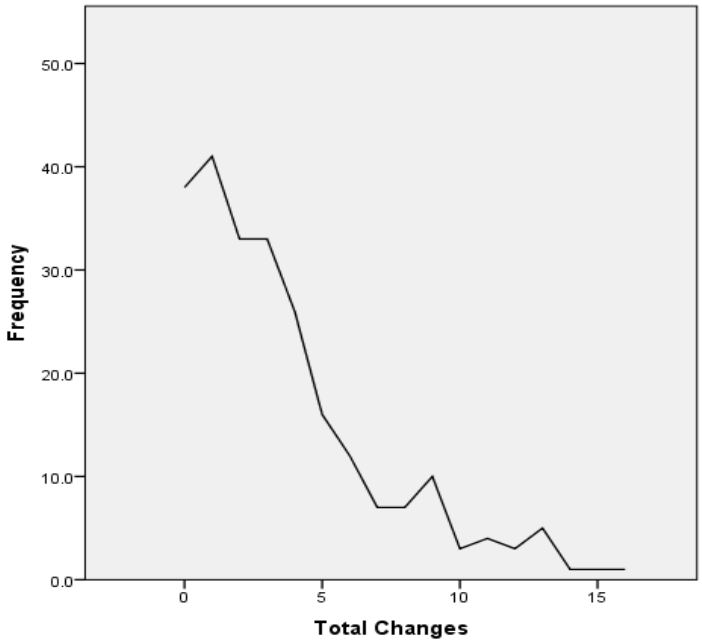

Figure 6.1 FREQUENCY OF TOTAL CHANGES PER REQUIREMENT

6.7.2 Comparison of volatility, change count and change cost.

Volatility, change count and change cost are all measures of requirements change that have the potential to convey information regarding project health, changeability and risk [1]. A frequency shape similar to that of change count illustrated in Figure 6.1, is observed upon examination of the volatility of requirements and the cost of change as illustrated in Figure 6.2. Thirty requirements experienced volatility greater than 500%, and the highest volatility recorded was 1831%. Total change cost for a single requirement varied from 0.2 days effort to 116.4 days effort

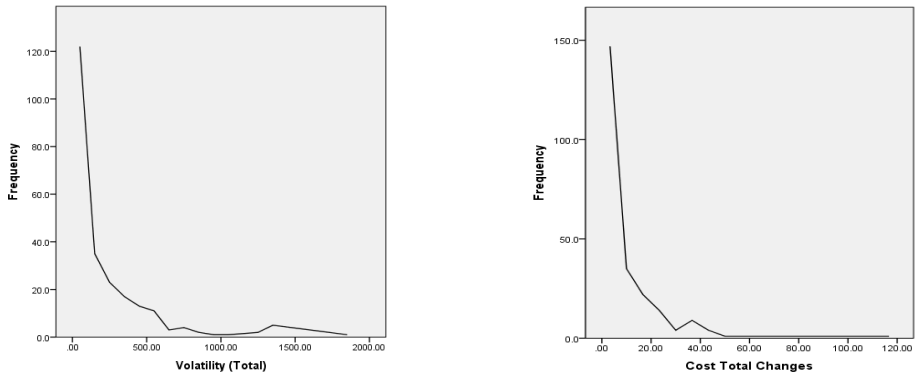

Figure 6.2 FREQUENCY OF TOTAL VOLATILITY AND TOTAL COST OF CHANGES FOR EACH REQUIREMENT

Table 6.2 illustrates the relationship between all change measures. As can be seen, a pairwise comparison between all change measures reveals that in all cases a significant ($p <= 0.01$) positive correlation can be observed. Therefore, any of these measures will provide project managers with an indication of the overall level of requirements change.

Table 6.2 CORRELATION BETWEEN MEASURES OF CHANGE

Change Measure	**Correlation coefficient (Spearman's Rho 1 tailed)**	**Significance**
Volatility with cost.	0.797	p<0.1
Volatility with No. of changes	0.698	p<0.1
Change cost With No. of changes.	0.859	p<0.1

Further analysis by means of example illustrates the different qualities that each measure conveys. Consider three requirements from this case study:-

a) Requirement A has the highest number of changes: 16 changes, incurring a total cost of 60.3 days effort, and a volatility of 430.5%.
b) Requirement B has the highest change cost: 13 changes incurring at total of 116 days change effort yielding a volatility of 171%.
c) Requirement C has the highest volatility: 6 changes, at a cost of 36.6 days and a volatility of 1831%.

From this example, it would seem that a change count is a more accurate reflection of the amplitude of the change, since the requirements with the higher change counts also had the higher change costs. The requirement with the highest volatility – requirement C - has the lowest change count, and also the lowest change cost. A closer look reveals that the original estimate for requirement C was 2 days, while the estimate for requirement B was significantly higher at 68 days. It is likely, therefore that requirement C, having gone from 2 days estimated work to a total of 38.6 days, bears little semblance to its original specification. By contrast, requirement B has changed from 68 days to 164 days effort, which is also a substantial change, but less extensive by relative magnitude. In this sense the measurement of volatility reflects the changeability of a requirement, and also expresses change amplitude which is relative to the original specification.

It can be seen, therefore that by comparison to either a simple change count, or a change cost, relative volatility communicates better the changeability of the requirement. Being tethered to change cost through its calculation may also mean that it is more robust in terms of its capability to express change cost than a simple change count. However, the original estimate must be taken into account when reviewing volatility figures from the perspective of project cost and schedule risk assessment.

6.7.3 Patterns of requirements volatility across change domains.

Table 6.3 provides a summary of change counts and volatility. The number of changes, number of involved requirements, and the lower and upper levels of change count and volatility are shown for each change domain. Thirty-four *organisation* changes involved a total of 76 requirements. Similarly, 24 *vision* changes affected a total of 56 requirements. By comparison, 187 *specification* changes involved a much greater number of requirements, 186. It would seem therefore that *specification* changes are at a higher level of granularity, meaning that a single *organisation* or *vision* change affects a greater number of requirements than a *specification* change.

Table 6.3 NUMBERS OF CHANGES AND VOLATILITY IN EACH CHANGE DOMAIN

	No of Changes	No Of Reqs	Change range	Volatility Range
Organisation	34	76	0-5	0-1,284%
Vision	24	56	0-6	0-600%
Specification	187	186	0-12	0-501%
Solution	36	59	0-4	0-791%

Requirements also change more often from *specification* sources. On average a requirement changes 2.5 times from *specification* changes, by contrast to an average of less than 1 in each of the other change domains. A much higher percentage of requirements (75%) underwent changes coming from the domain of *specification* compared to other domains where between 23% and 31% requirements were affected. *Organisation* volatility has a greater range than other change domains, with 9 requirements having *organisation* volatility

greater than 700%. Only 1 requirement had similarly high *solution* volatility, and there were no requirements that had *vision* or *specification* changes of this magnitude.

Relating to Boehm's observation [73] that 80% of maintenance effort was absorbed by 20% software modules, a Pareto analysis reveals that 80% change cost actually involves 31% of requirements. Interestingly, 80% of *organisation* and *vision* related rework was attributed to less than 11% of the requirements. This is in contrast to the 36% of requirements contributing to 80% of the *specification* domain change cost.

Further scrutiny of this Pareto analysis reveals that no single requirement contributes to 80% of the change cost in all change domains. However, a one-tailed Spearman's Rho test[2] reveals that there is a weak but significant correlation between volatility in the domain of *organisation* and all other change domains. [(Vol(org) and Vol(vision) = 0.38, $p < 0.05$; r (Vol(org) with Vol(req Spec)) = 0.21, $p < 0.05$; r (Vol(org) with Vol(sol)) = 0.14, $p < 0.05$]. In other words, a requirement that experiences *organisational* volatility, is more likely to also have volatility coming from sources in the other change domains. There is no significant correlation between volatility in the remaining domains. For example, a requirement experiencing *vision* volatility is not necessarily also subject to *specification* domain changes. Therefore, with the exception of *organisation* changes, requirements that have a high volatility in one domain do not generally have a high volatility in other domains.

6.7.4 Requirement Dependency

Figure 6.3 shows the distribution of median volatility by Requirement Dependency for all changes. Also on this graph, as in all the requirement attribute graphs to follow, the change count is also shown. In Figures 6.3-6.11 the bars represent the median volatility, and the lines indicate the change count. In Figure 6.3, both median volatility, and change count are increasing as levels of requirements dependency increase. This is reflected in the Spearman's Rho test which identifies significant positive correlation between dependency and Volatility. Vol(tot) [$r = 0.587$, $p < 0.01$]. A further exploration of the domain specific volatility reveals a similar pattern.

[2] For simplicity, the results of the Spearman's Rho test replace 'r' with the associated Volatility description.

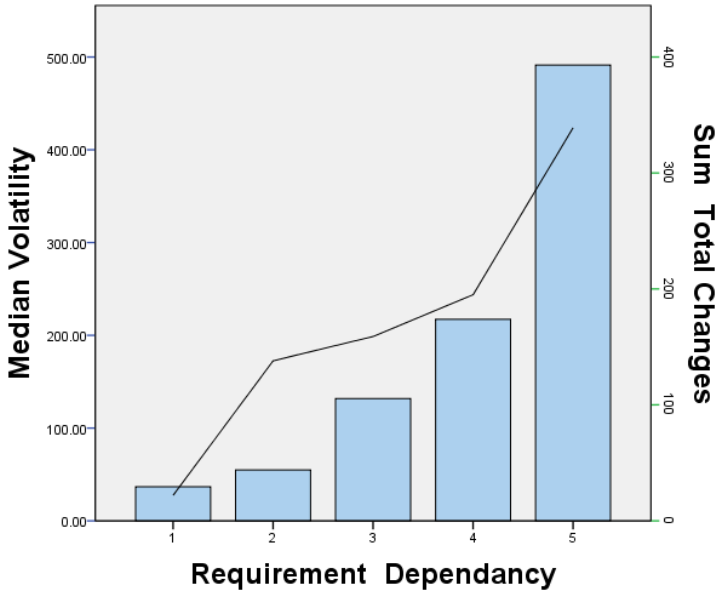

Figure 6.3 MEDIAN TOTAL VOLATILITY FOR REQUIREMENT DEPENDENCY

Figure 6.4 illustrates median volatility and change frequencies for all four domains where it can be seen that requirements with higher levels of requirement dependency have higher organisation, vision, specification and solution volatility. A Spearman's Rho confirms that requirements dependency is positively correlated with volatility in all domains [Vol(Org)=0.563, $p<0.01$; Vol(Vis) =0.380, $p<0.01$; Vol(Req Spec) =0.342, $p<0.01$; Vol(Sol) = 0.203, $p<0.01$]. The strongest correlations are with total volatility and *organisation* volatility. This mirrors the observation that *organisation* change tends to be at a higher level of granularity, affecting a greater number of requirements.

These results indicate that requirements dependency positively correlates with requirements volatility in all change domains.

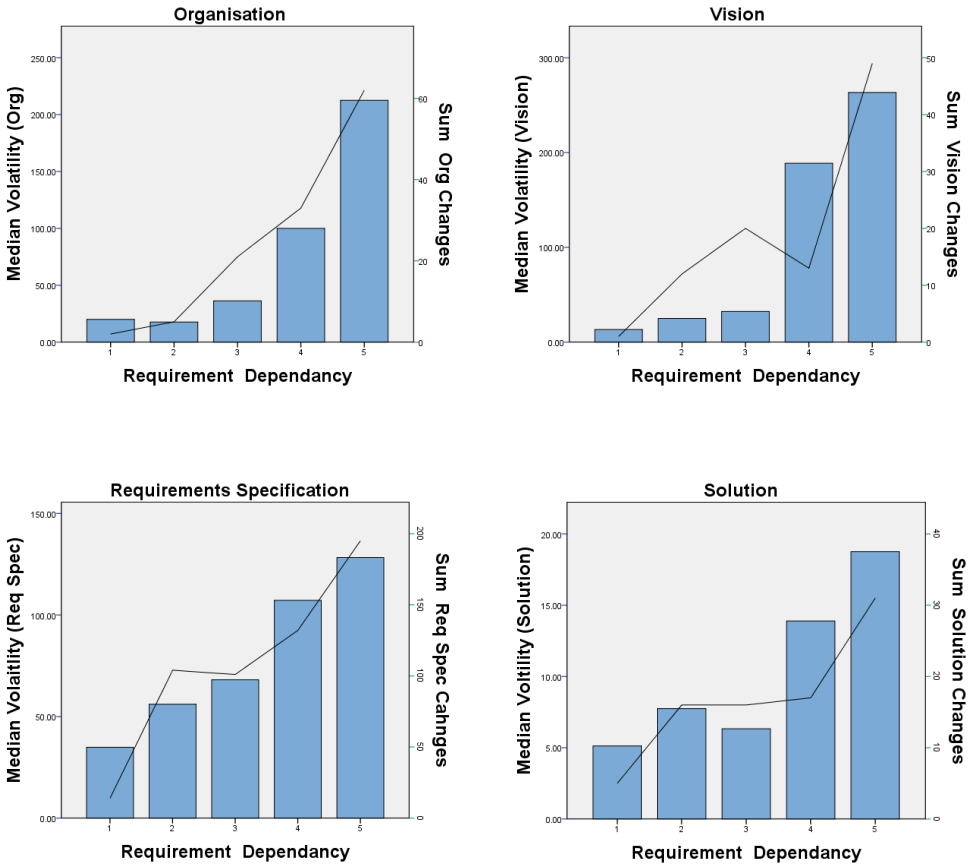

Figure 6.4 MEDIAN DOMAIN VOLATILITY FOR REQUIREMENT DEPENDENCY

6.7.5 Business Complexity

There is a less clear relationship between business complexity and volatility. Indeed, as can be seen in Figure 6.5, frequency of changes is highest when business complexity is at level 2 ('some thought needed' See Table 6.1). Median total volatility for each requirement is highest when business complexity is at level 1. A Spearman's Rho test confirms that there is no significant correlation between business complexity and total volatility [Vol(Tot)=-0.76,p>0.05]. Further analysis of volatility in each change domain reveals a similar outcome [Vol(Org)=0.07, p>0.05; Vol(Vis)=0.06, p>0.05; Vol(Req Spec)=-0.71, p>0.05; Vol(Sol) = 0.07, p>0.05].

These results indicate that there is no correlation between business complexity and volatility in any change domain

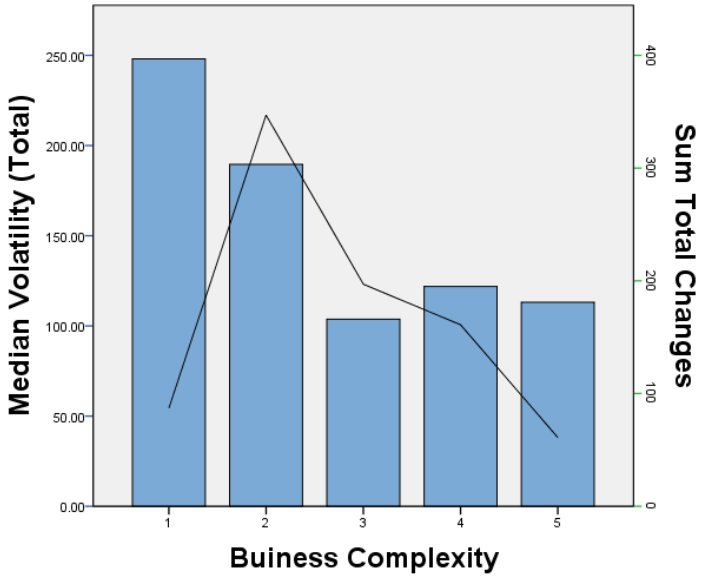

Figure 6.5 MEDIAN TOTAL VOLATILITY FOR BUSINESS COMPLEXITY

6.7.6 Technical Complexity

Once again, the distribution of volatility for technical complexity is not straightforward. Figure 6.6 would seem to suggest that less technically complex requirements are changing more frequently and have higher rates of volatility. The Spearman's Rho test confirms total volatility is weakly negatively correlated with technical complexity [Vol(Tot)=-0.17, $p<0.05$].

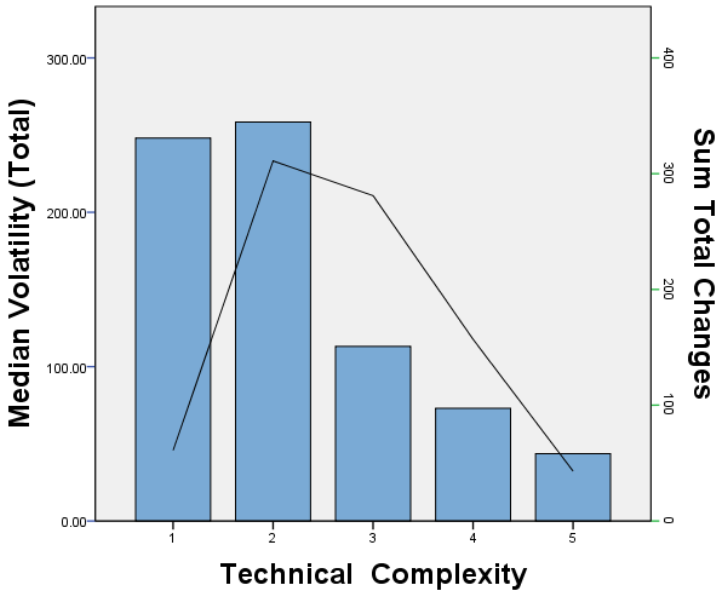

Figure 6.6 MEDIAN TOTAL VOLATILITY FOR TECHNICAL COMPLEXITY

An examination of the correlations between technical complexity and volatility in other change domains reveals that there is a similar negative correlation between technical complexity and specification volatility [Vol(Req Spec)=-0.21,p<0.01]. This is illustrated in Figure 6.7. There is no significant correlation in the remainder of the domain volatilities. [Vol(Org)=0.1,p>0.05; Vol(Vis)=0.10, p>0.05; Vol(Sol) = 0.144, p<0.05]

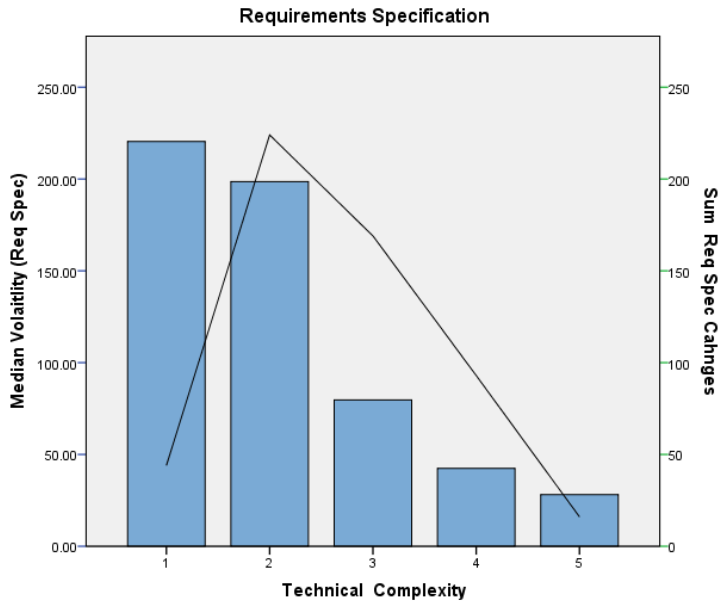

Figure 6.7 MEDIAN REQUIREMENTS SPECIFICATION VOLATILITY FOR TECHNICAL COMPLEXITY

These results indicate that technical complexity negatively correlates with total volatility and specification volatility but there is no correlation in any of the other change domains.

6.7.7 Business Novelty

The somewhat confusing distribution for volatility with business novelty is illustrated in Figure 6.8.This not only reveals that though, in general, volatility is lower for less novel requirements, the frequency of change is higher. A Spearman's Rho indicates that there is no correlation between business complexity and total volatility [Vol(Tot)=0.08, p >0.05].

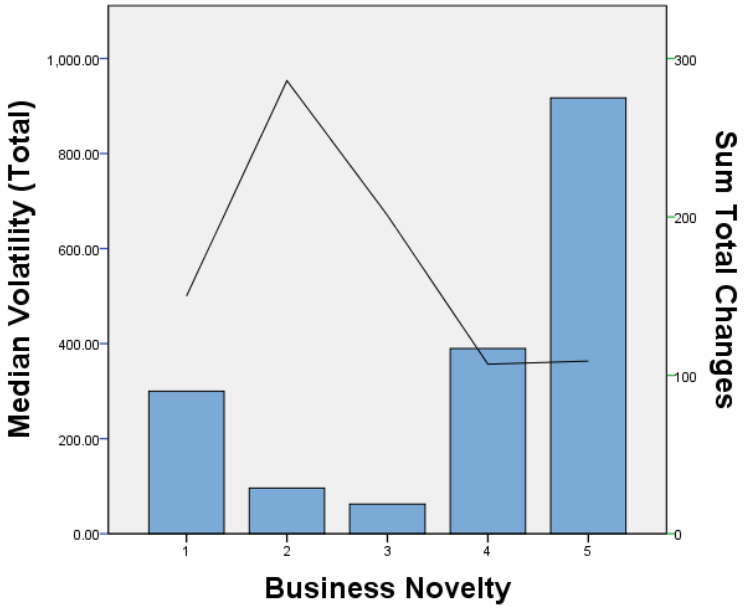

Figure 6.8 MEDIAN TOTAL VOLATILITY FOR BUSINESS NOVELTY

Further investigation of change domain volatility reveals a somewhat contradictory pattern of correlation. Figure 6.9 illustrates the relationship between domain volatility and business novelty. While higher business novelty corresponds to higher median volatility and frequency of change in the domain of *organisation*, the opposite is observed in the domain of *requirements specification*. Here we see that lower levels of business novelty give rise to higher levels of volatility. A Spearman's Rho confirms a moderately strong positive correlation between *organisation* volatility and business novelty and a weaker negative correlation with *specification* volatility [Vol(Org)=0.423, $p<0.01$; Vol(Req Spec) = -0.25, $p<0.01$]. However, there is no clear relationship in the domains of vision or solution. [Vol(Vis) = 0.39, $p>0.05$; Vol(Sol) = 0.27, $p>0.05$].

Requirement Volatility is positively correlated with Business novelty in the domain of organisation, negatively correlated in the domain of requirements specification, and has no significant relationship with total volatility, vision volatility or solution volatility.

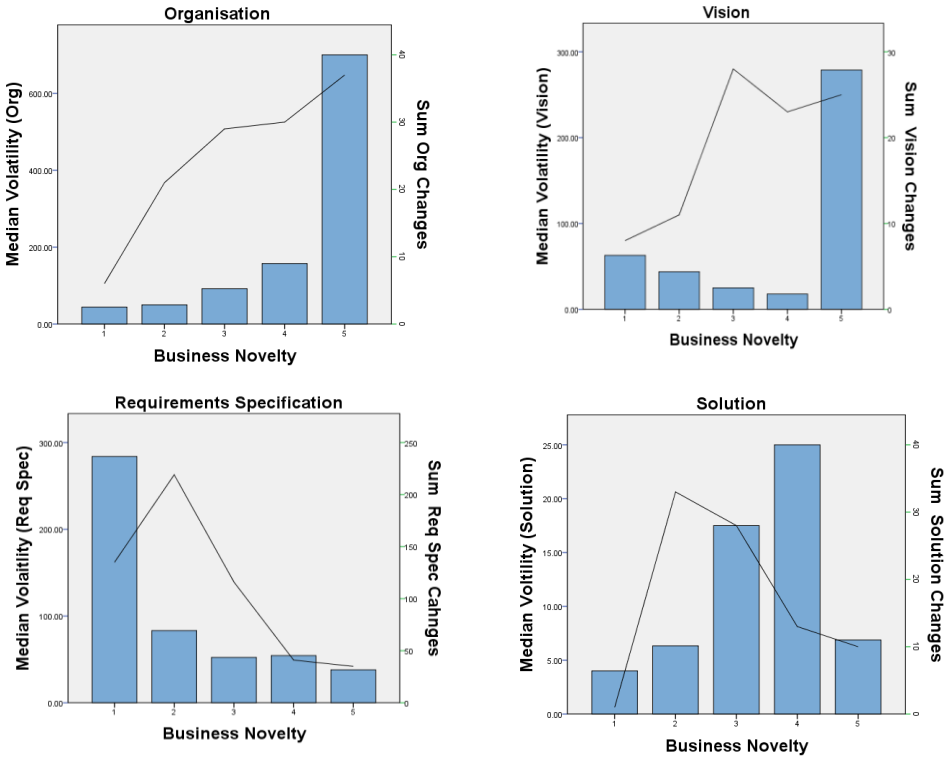

Figure 6.9 MEDIAN DOMAIN VOLATILITY FOR BUSINESS NOVELTY

6.7.8 Technical Novelty

A more straightforward distribution of volatility is observed when we consider the attribute technical novelty. It can be seen from Figure 6.10 that requirements with a higher technical novelty have higher overall volatility. This is confirmed by a Spearman's Rho indicating that technical novelty is correlated with total volatility [Vol(Tot)=0.28 , p <0.01].

However, a Spearman's Rho test examining volatility in each of the change domains is illuminating in that the only change domain where volatility even weakly positively correlates with technical novelty is the requirements specification domain . [Vol(Org)=0.07, p>0.05; Vol(Vis) = 0.07, p>0.05; Vol(Req Spec) =0.362, p<0.01; Vol(Sol) = 0.06, p>0.05].. This is illustrated in Figure 6.11. There is no significant correlation in any other domain.

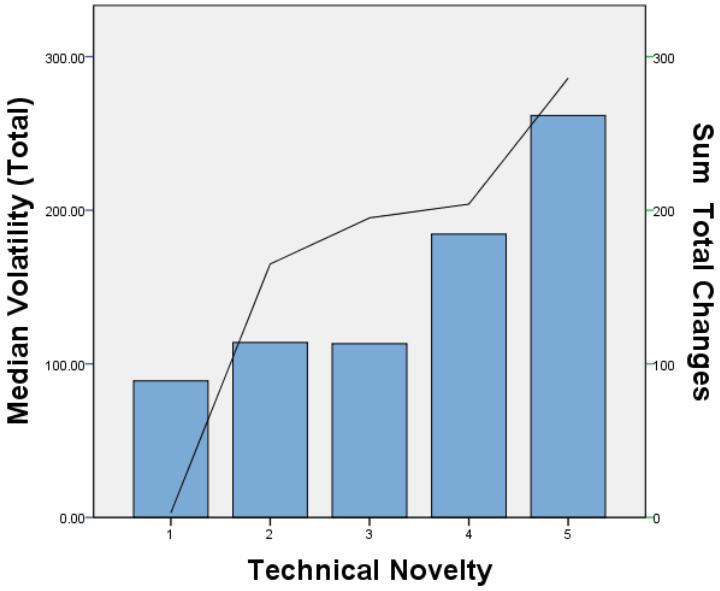

Figure 6.10 MEDIAN TOTAL VOLATILITY FOR TECHNICAL NOVELTY

Technical novelty positively correlates with requirements volatility. However, this is a weak correlation and only evident in the domain of specification.

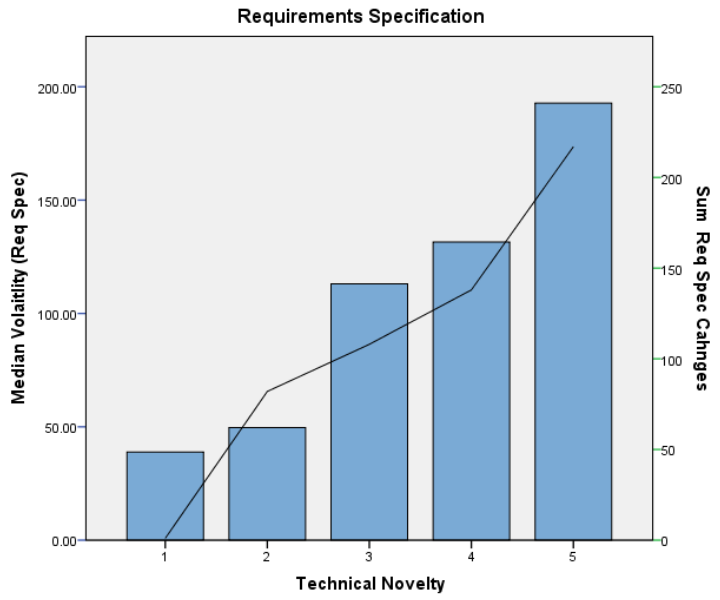

Figure 6.11 MEDIAN REQUIREMENTS SPECIFICATION VOLATILITY FOR TECHNICAL NOVELTY

6.8 Discussion

The metric most commonly used in industry to measure software requirement change is a simple count of the number of changes [20]. Despite intuitive reasoning which would question the ability of such a metric to convey the cost of changes, the results of this case study indicate the change count is related to change cost, and therefore a candidate for use in future studies. However, requirements changeability is better expressed through a relative cost, as illustrated by a simple example, and may therefore more closely reflect the notion of project health in terms of out of control change as defined by other researchers [86][15]. Nonetheless, since volatility is change cost expressed relative to the original estimate, larger values for volatility don't necessarily imply greater risk to project cost, and risk must be assessed with respect to original requirement effort estimates.

Given that requirements that have a high number of changes in one domain, but not others implies that requirements themselves are more susceptible to different types of changes.
However, these results cannot provide an explanation for that observation. Apart from requirement dependency, which positively correlated with all measures of volatility, some the results are counterintuitive. That business complexity bears no correlation to requirements changes raises the possibility of a confounding factor, and mirrors the observation by Fenton [29] that complexity alone has an arbitrary relationship to software defects. Most interesting, however, is the result that technical complexity is negatively correlated with volatility. Less technically complex requirements are changing with more frequency and cost with respect to their original cost estimation. Taken together these results indicate that a prediction of changing requirements will not be achieved based solely upon the requirement attributes examined in this study. Further consideration of more complex causal factors, such as the process factors and analysis techniques that have been found to correlate with requirements volatility [9] [8], and also levels of effort and ability thought to influence the likelihood of defects [30]. Lim's approach that categorises requirements by types such as non functional constraints, business policies and rules etc. (see section 2.5) [13] may also be worthy of further investigation.

The results of this case study deepen our understanding of the distinction between the change domains contained within the requirements change source taxonomy. The implication from the results is that changes coming from sources of *organisation*, *vision*, *specification* and *solution* are affecting different groups of requirements. Requirements with a higher level of business novelty change more often and with a higher change cost only

from changes coming from the domain of *organisation*. We may infer that in addressing *organisation* change, there is increased certainty that novel requirements meet business needs, and as a result are less prone to other types of changes. This inference is supported by the contrasting observation that requirements with a lower level of business novelty are changing more frequently from *specification* changes. No such inference can be made about requirements with a higher level of technical complexity since there are no instances of a positive correlation with volatility. However, the only domain in which there was a negative correlation was *specification*, which, as we discovered in a previous study (see section 5.6.4), has a higher instance of change from defects rather than opportunity. Fenton's argument [29] that more complex requirements may be afforded more resource, and therefore are likely to contain fewer (code) defects, is one possible explanation for these results.

6.8.1 Review of Results Validity

As in the previous case study described in chapter 4, the data used in this study was defined using the GQM approach in line with study objectives. The following discussion reviews the relevant limitations to validity outlined in section 3.4.6.

Construct Validity

Since the data was discussed and specified through meetings with researcher and practitioners, there is a shared understanding of the meaning of the data items. All participants have at least 10 years of experience in their respective software development roles. All subjective measures involving cost (those collected and those derived), were agreed between two practitioner roles. The remainder of the subjective measures in the second case study were collected by only one participant. However, a subset of data collected by each practitioner was sent to another practitioner for validation, which resulted in no changes.

Internal Validity

Internal validity is to be considered when causal claims are made. Although this study investigates the potential for requirements attributes to 'cause' requirements volatility, the complexity of the results would indicate the presence of other factors that need to be taken into consideration. In this instance, awareness of the presence of confounding factors, especially in the light of other research, enriches our understanding of results.

External Validity

No claims to external validity can be made, in that without study replication, it is impossible to draw conclusions about the application of results to software development environments that are different to the study context described here.

Study Reliability

Details of data specification and collection protocol are provided for use by other researchers.

6.9 Conclusion and Implications for Research

This case study set out to derive a practical measure for volatility and determine if there are identifiable attributes of requirements that render them prone to change. Representatives of our industrial partner identified requirement attributes because they expected that they would give rise to changes. Those attributes were requirement dependency, complexity (business and technical) and novelty (business and technical). The differences in patterns of volatility in the change domains identified in a previous study were also investigated.

A metric for volatility based upon relative change cost proved equally able to convey change amplitude, and better communicate requirements changeability by comparison to a simple change count. Results of the investigation of patterns of volatility in change domains indicate support for the observation that change domains have a specific qualities and characteristics. A requirement that is susceptible to change in one domain is not necessarily similarly change-prone in other domains, meaning that requirements are predisposed to change in a certain way. While correlation between volatility and dependency was positive in all domains, for the remainder of the requirements attributes, there was a less clear relationship. There was no correlation with business complexity and requirements with higher technical complexity tended to change less than those with lower levels of technical complexity, particularly in the domain of *requirement specification.* Correlation between volatility and business novelty was positive in the domain of *organisation*, negative in the domain of *specification*, and no overall significant relationship. Technical novelty is weakly positively correlated with volatility but only in the domain of *specification.* These results imply that there is a complex causal relationship between requirements and volatility that cannot rely solely upon the attributes under investigation in this study.

By assembling the results and implications of studies that so far make progress towards the goal of software requirements prediction, we can induce the layered causal account of software requirements change illustrated in Figure 6.12. This captures the salient implications of the results of case studied described in Chapters 4, 5 and 6, studies combined with conclusions drawn by other researchers.

Clearly evident in the diagram is the two-sided causal nature of requirements change which is reflective of the two types of constructs in the change source taxonomy (see section 4.7). These are 'uncertainties', which are situations that affect the disposition of a requirement to change, and 'triggers' which are events that promote change discovery. In the case study that set out to evaluate the informative efficacy of the change source taxonomy four change discovery activities were identified which are contained here within the box labelled "Trigger". The curve over the arrows from the Change Triggers to the 'Changes Found' node indicate that only one trigger is required in order to discover change. As discovered in the case study described in this chapter, apart from requirements dependency, no single attribute correlates with requirements volatility in all change domains. Previous studies have noted an influence of environmental factors, software development process and techniques, and effective communications upon requirements change [8][9]. Such factors were also identified as cause constructs in the first empirical study. Therefore, all factors are causally relevant and must work in combination to affect requirements uncertainty, as illustrated in the box labelled "Uncertainty".

In a fully developed causal model each of the factors would consist of a number of attributes, or variables, and there may be requirements attributes in addition to dependency, novelty and complexity that affect requirements uncertainty. Factors influencing process and technique quality, such as process maturity and staff effort and skill, would affect both the uncertainty of a requirement and the likelihood of change discovery. This is because both the uncertainty of a requirement, and the change trigger activities are both brought to fruition by tasks within the software development lifecycle. Since a requirement continues to change in the maintenance phase subsequent to software delivery, there is an amount of uncertainty remaining even after changes have been discovered during development. This is included here, alongside cost and value, as a consequence of changes found.

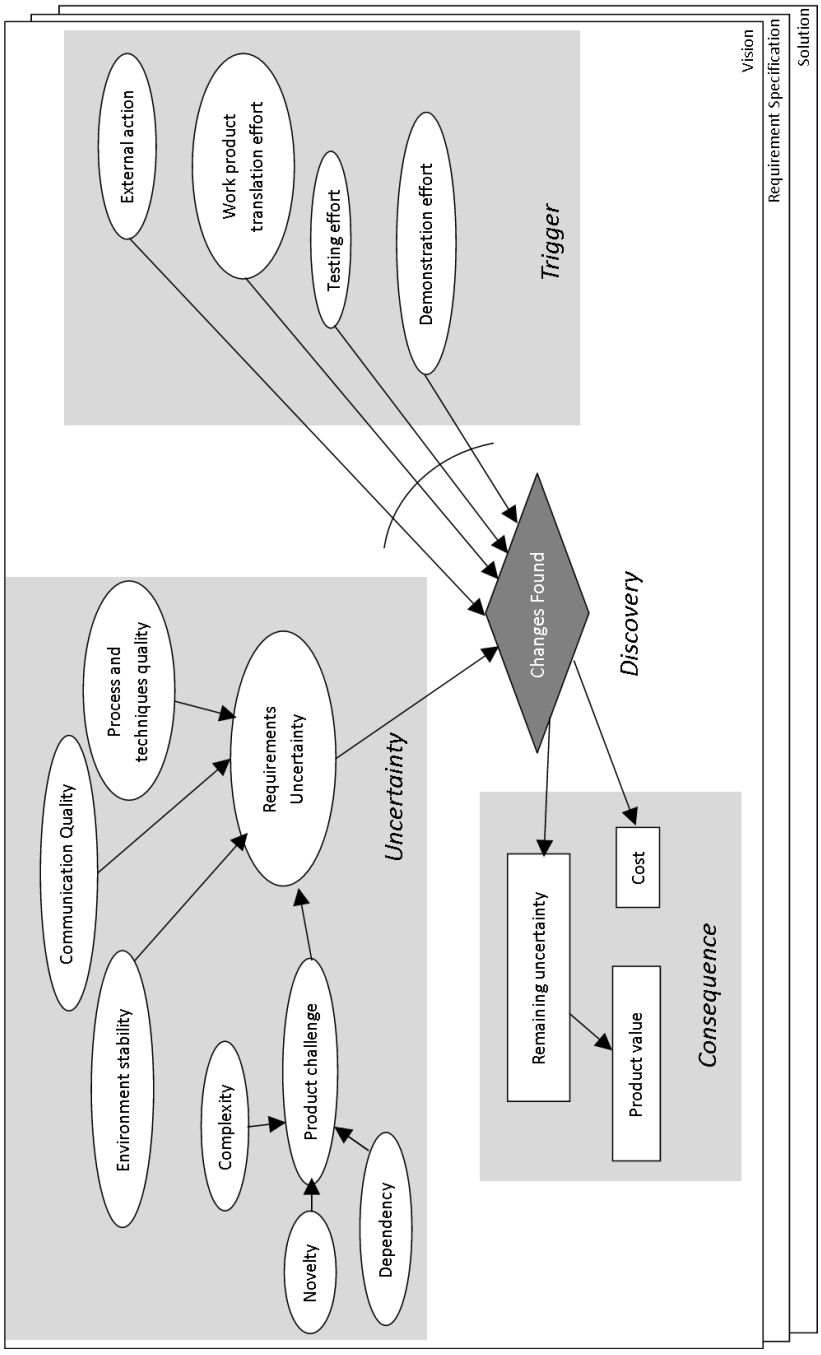

Figure 6.12 LAYERED CAUSAL ACCOUNT OF SOFTWARE REQUIREMENTS CHANGE

Findings from the previous case study illustrate that the change domains within the taxonomy are characterized not only by differing cost and value, but also by stakeholder involvement and change controllability. In addition, results from the case study in this chapter showed that there were different relationships between requirements attributes and volatility depending upon the change domain. Further, environment stability attributes will differ according to change domain. For example, *solution* domain changes are not directly

brought about by a change to customer's business process. Instead this would result in project *vision* volatility. This distinction of character and variable influence is captured here by layering seen in Figure 6.12. Only three domains are included, since our knowledge of the uncertainties and triggers in the *organisation* domain is limited, as discussed in section 0.

Having met the study objectives of determining a metric for volatility, and investigating attributes of requirements for correlation with volatility, a layered causal model has been derived from the results of all studies and will provide the foundation upon which to build models of prediction. The next chapter will therefore address the research question '**RQ4** Can we model requirements change in such a way that we can predict change?'

Chapter 7 Volatility Prediction

The findings from the case studies described in Chapters 4, 5 and 6 have provided a foundational theory of the causes and effects of software requirements change, upon which predictive models may be based. This chapter describes the structuring, calibration and verification of such models which is undertaken with industrial partners. Three Bayesian Network models are constructed, each containing variables and relationships pertaining to the domains of *vision*, *requirement specification* and *solution*. Methods and rationale for net construction are described as well as the variables and models themselves. Initial verification procedures by way of walkthroughs and sensitivity analysis are undertaken to ensure that the networks operate in accordance with the intended design. This study addresses the following research question:-

RQ4 Can we model requirements change in such a way that we can predict change?'

Chapter Contributions to Academic Knowledge

a) An exploration of the feasibility of volatility prediction by formal methods, through the construction and evaluation of Bayesian networks designed to identify change prone requirements and levels of volatility early in the development life-cycle.

7.1 Introduction

Compelled by the results of the preceding empirical enquiries, the next objective is to design causal models of requirements change in order to realise the goal of requirements change prediction. The approach described here was undertaken in conjunction with industry, and utilizes a theoretical foundation drawn from previous empirical studies to design, build and test a number of Bayesian Networks that provide predictions of both the type of change and the level of volatility that may be expected. A Bayesian Network is a probabilistic graphical model that represents a set of random variables and their dependencies. As applied by Fenton[103], Bayesian nets provide a means to architect predictive models based on causal rather than correlative relationships, and have a number of benefits, not least the capability of being formulated using expert opinion in addition to data. They also facilitate forward and backward reasoning, allowing for diagnosis as well as prediction. In a well calibrated causal model, should levels of volatility be observed, the models can provide revised beliefs in, for example, the quality of communication between the stakeholders. Bayesian Networks are described more fully in section 3.5. The models presented in this study are designed to predict levels of volatility for each requirement early in the project lifecycle, based upon causal factors of the product, process and environment. These factors are drawn from previous studies, re-examined by industrial practitioners and net construction is realised through a defined process involving a series of workshops.

An important distinction is made between the types of changes that may be seen. The classification used is based upon the results of three previous empirical studies examining the causes and effects of software requirements change. A taxonomy of change types was derived comprising five change domains representing changes coming from *market* sources, customer *organization*, the product *vision*, *requirements specification* and *solution*. Of these, models are built for the three change domains about which our industrial partners can reason and evaluate. These are *vision*, *specification* and *solution*. Although this places an immediate constraint upon the models' ability to predict requirements change in entirety, the domains of market and organisation are considered less predictable, and may need an alternative predictive approach. At the same time, anticipation of changes coming from the domains of *vision, specification* and *solution*, are considered sufficient to inform the process and technique decisions that are within the remit and control of the software engineering environment.

A second important distinction is made between the change-proneness of requirements, and the ability to uncover the change. While a requirement may be prone to change, qualities of the development process and team capability will encourage change to be drawn out. It is well known, for example, that prototyping with user involvement promotes the discovery of change early. Undiscovered change remains as uncertainty, perhaps until user acceptance testing, or beyond into application usage.

The three models have between them, more than 150 variables. It was noticed that the models had elements in common. For example, all models had variables corresponding to capacity of understanding. However, in the *vision* domain these would include 'stakeholder understanding and involvement', while in the *solution* domain they would relate to the development team. It was possible therefore to draw a core causal model from an abstraction of the three domain models. This core model communicates the salient design decisions of all models, and is introduced before more detailed descriptions of each domain model.

Since there was no data available to calibrate the models, the quantitative relationships were expressed by industrial partners, who used techniques based upon recommendations from other researchers [172] that considerably reduce the time needed. Model walkthroughs and sensitivity analysis complete the construction process and verify that the models perform in accordance with their intended design. The models were implemented using the Netica software package from Norsys corp [173].

The objectives of this study are:-

1. Construct and calibrate a series of Bayesian networks to predict software requirements change in the domains of *vision, requirements specification,* and *solution.*
2. Verify that the models operate in accordance with their intended design.

7.2 Theoretical Foundation

Together, the results of previous studies described in earlier chapters inform model construction. The derived theoretical foundation consists of the following elements described here in summary form:-

1. Measurement of Volatility.

Software Requirements volatility is measured in way that reflects both the changeability and cost of changes to requirements. For a requirement with original cost C, a revised cost of C', and used portion of previous estimate UC, the volatility over a given time for requirement r with N changes is:-

$$V_r = 100\ x\ \frac{\sum_{n=1}^{n=N} C'_n - (C - UC)_n}{C}$$

Equation 7.1 RELATIVE VOLATILITY

2. Taxonomy of Change Sources

Requirements changes arise from sources that may be considered to be distinct. These are referred to as change domains and are described in Table 7.1.

Table 7.1 REQUIREMENTS CHANGE SOURCE TAXONOMY

Change Domain	Description
Market	Differing needs of many customers, changes due to market forces, competitor activity, or government regulations.
Organisation	Changing strategic direction of a single customer organisation in which the software will be used.
Vision	Change arising due to the changes in the problem to be solved, product direction or priorities, stakeholder involvement, or process change.
Requirement Specification	Changes to the way in which the problem is to be answered by the software, resolution of ambiguity, inconsistency, or increased understanding.
Solution	Change accommodating new technical requirements, design improvement, solution elegance, COTs employment, software tiering, deployment needs etc.

Requirements may admit to changes from any and/or all of these domains during the product development lifecycle.

3. Differing Characteristics of Change Domains

Change domains differ in terms of their average change cost, value, stakeholder involvement, project management control, and whether they are considered to be defects or functional enhancements. This is illustrated in Figure 7.1. In addition to differences in cost and value, changes in each domain were discovered using different activities, and were found to exhibit different relationships between volatility and attributes of requirements. For example, while requirements highly dependent upon other requirements had higher levels of volatility in all domains, those requirements considered to be more novel to the business changed more from *organisation* changes than *requirement specification* domain sources.

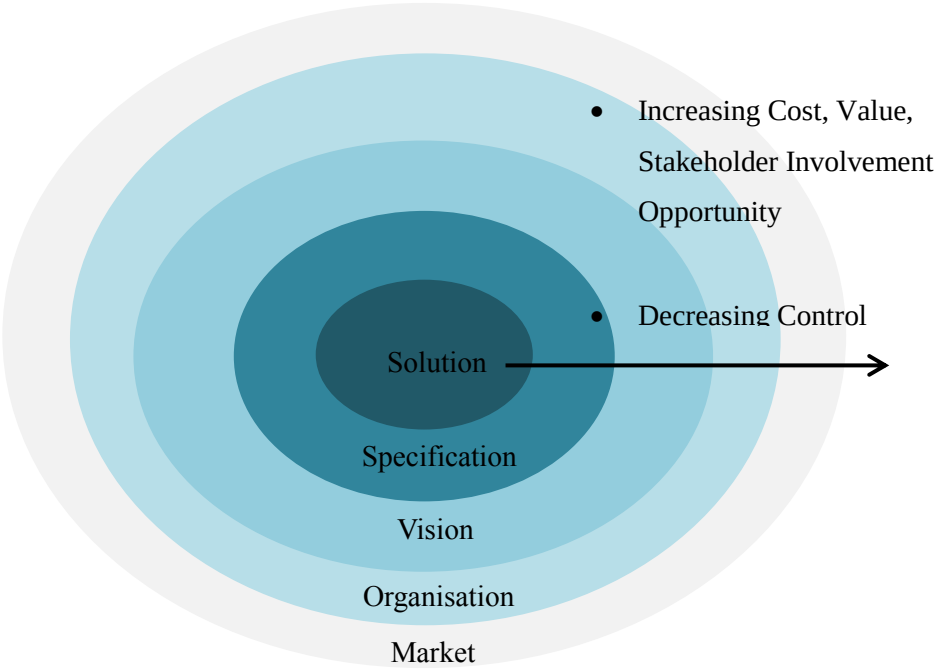

Figure 7.1 CHARACTERISTICS OF CHANGE DOMAINS

4. Two-sided nature of causal influence (uncertainty and trigger).

Change source constructs fall into two categories, identifiable as being change 'triggers' (events) or an 'uncertainties' (situations). A change trigger, such as 'Change to business case', could be considered an event which may drive requirements change. By contrast, an 'uncertainty' such as 'quality of communication', is a situation or state that is on-going and may be causative to volatility. Therefore, changes are 'caused' by a combination of both uncertainty and trigger. Uncertainty will predispose requirements to change, but they will only do so, if a trigger event discovers the need to make a change. Uncertainty will remain if changes are not discovered.

7.3 Bayesian Network Construction Process

Beginning with initial set of 53 change constructs elicited in a previous study (see section 4.5), a series of 19 1-2 hour focus groups, interviews and meetings with one or two project managers over a time period of 16 months, using paper and white board, resulted in a set of change domain specific predictive models. This empirical study was concurrent with the case study described in Chapter 6, and influential to the illustration of the causes and consequences of requirements change described in section 6.8. The models evolved significantly during that time, and consideration was given to the needs of data collection that would support model validation and usage, reflecting the observation by other researchers, that a compromise was needed between complexity and ease of use [163].

The process of model construction was defined for the purposes of this study, and is illustrated in Figure 7.2. It was derived from the knowledge engineering approach advocated by Korb [174], and Druzdel's [163] recommendations for network structuring to minimise the probability elicitation effort. Korb recommends a spiral approach to network development, which begins by testing an initial model, and then re-structuring the network based upon the results in order to generate a new set of results. He identifies five stages of model development, which include initial structuring, evaluation, field testing, industrial use, and refinement. These stages would be repeated in an iterative fashion until results are acceptable. The process used in this study, and illustrated in Figure 7.2, is concerned with the first two stages of Korb's model – construction and evaluation. These stages are broken down into their constituent tasks, given the need to structure models in accordance with

practical considerations such as data collection needs, and probability elicitation effort. As such, this study consists of a single iteration of Bayesian network development with a single point validation using industrial data. This chapter explains the execution of each stage of construction, and describes the resulting models. Model validation is the subject of the next chapter. Each stage is summarised in the following sections.

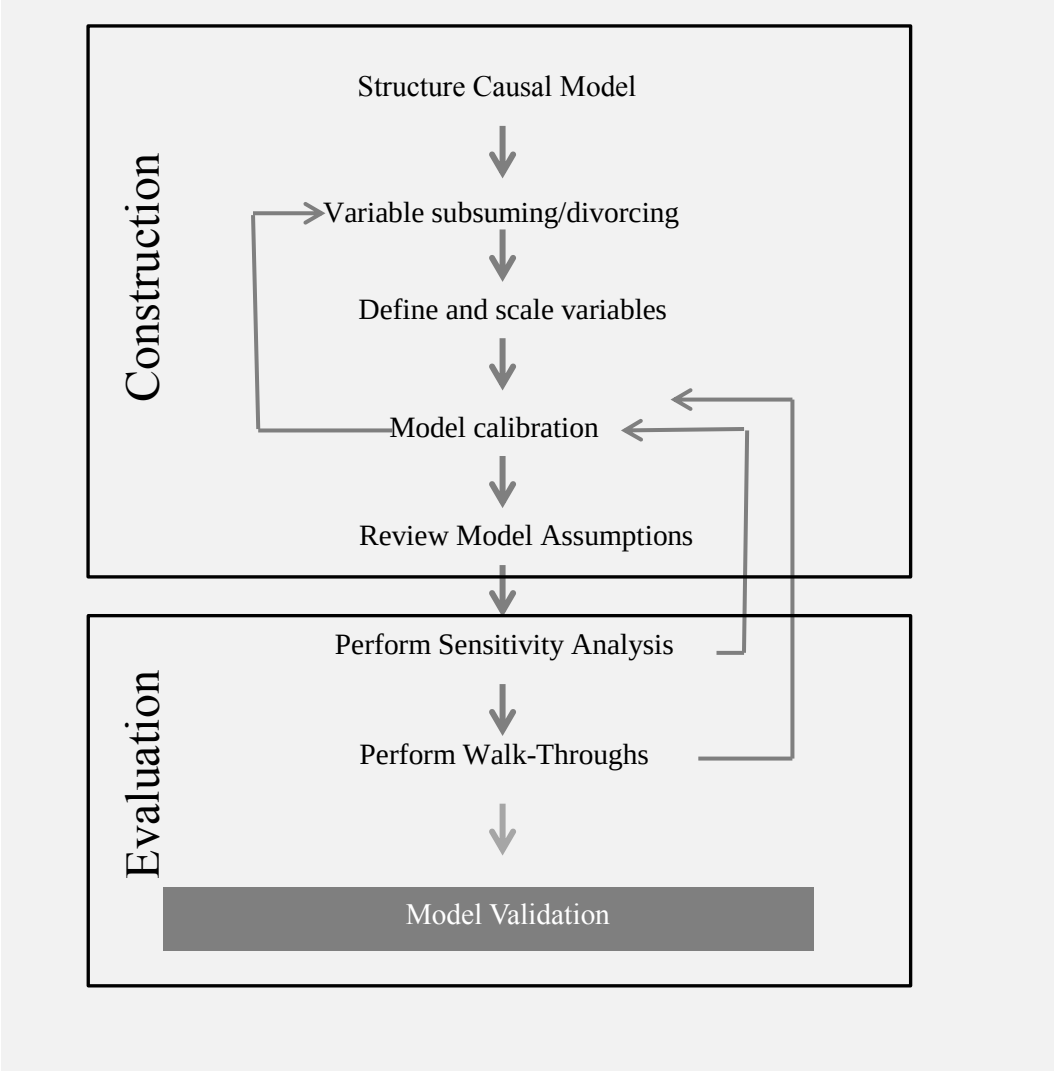

Figure 7.2 BAYESISN NETWORK DEVELOPMENT PROCESS

7.3.1 Structure Causal Model

It has been noted that Bayesian networks are more robust to calibration inaccuracies than structural deficiencies [175], so therefore it is important to ensure that the structure of the model reflects real world relationships as closely as possible. This can take considerable time and effort, requiring a great deal of skill and creativity and close communication with domain experts [159]. Druzdel [163] adds that model construction is a trade-off between the desire for a large rich model and the cost of construction, maintenance and complexity of probabilistic inference. Kjaerulff [159] recommends firstly identifying the variables and then adding the links between them, whereas Twardy [176] suggests that model construction begins by identifying the root cause and moving from there. From experience, Nadkani [175] reports that unstructured methods are best for causal modelling because of their exploratory nature. With one or two industrial practitioners an unstructured approach was taken, using paper and white board. Although the industrial practitioners had no experience of causal modelling, they were very familiar with the concept of domain and process modelling, having at least 15 years of UML experience. During each modelling session, photographs of the models were taken and notes kept regarding decision making. An example of modelling early in the construction process is illustrated in Figure 7.3.

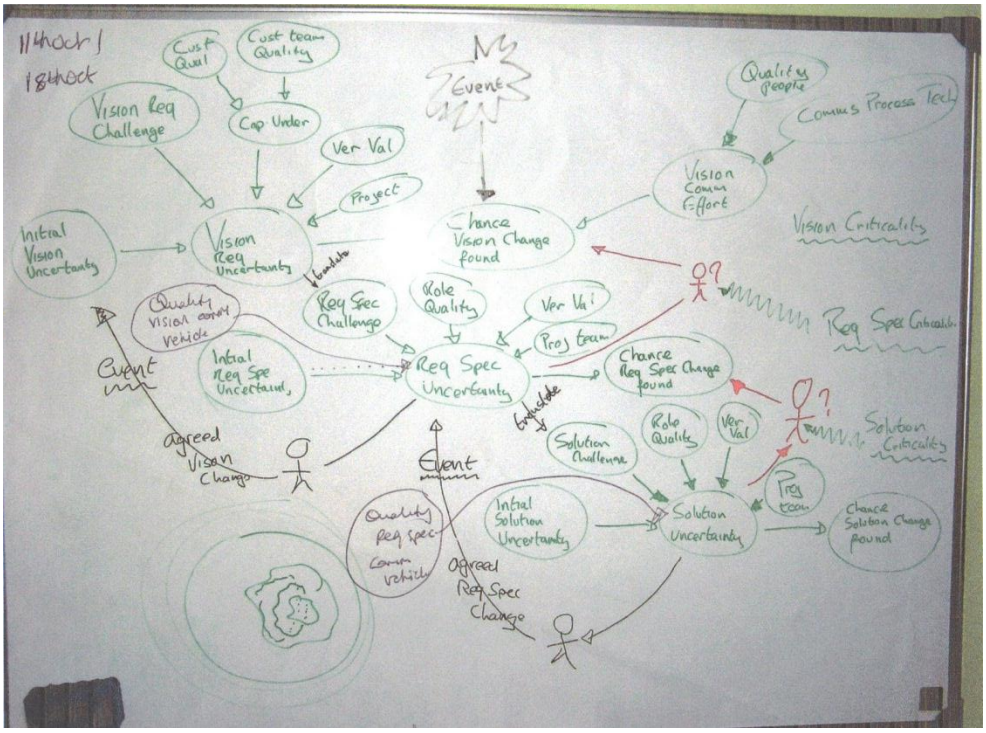

Figure 7.3 EARLY CAUSAL MODELLING

Model structuring began with a list of 53 causal variables identified in the study described in chapter 4. Initial attempts were made to causally link the uncertainties to the triggers in the change source taxonomy (Figure 4.3), but this was deemed unfeasible since many of the uncertainties affected many of the triggers. Instead, questions were posed concerning each trigger such as "What are the factors that may lead to increased customer understanding". Causal modelling focused upon the answers to those questions, and the uncertainties contained within the requirements change source taxonomy. In so doing, the uncertainties were verified, and also additional causal influences were identified. One such influence is the notion of a requirement group. The hierarchical nature of requirements means that certain groups of requirements may be more complex in terms of their entanglement than other groups. Therefore attributes of a 'requirement group' were added to the networks. In between meetings with practitioners, networks were updated on the software package Netica by the researcher, and further consideration given to model structure which often resulted in the formulation of questions regarding both the domain being modelled, and the causal relationships. These questions were brought to the next meeting. To further illustrate the development process, a small sample of those questions and answers are included in Table 7.2

Table 7.2 EXAMPLE NETWORK STRUCTURING QUESTIONS

1. Does the clarity of the shared vision depend upon the quality of communication or vice versa?
   The clarity of the shared vision depends upon quality communication
2. How does the synergy of the agenda differ from the clarity of the shared vision?
   Agenda is the goal of the stakeholder, shared vision is a (compromised) agreement.
3. If there is evidence that the quality of communication is poor, how does that effect belief in parent nodes. In other words, would this change the likelihood of the number stakeholders involved?
   Yes
4. How is change of stakeholder representative different a to new stakeholder role?
   One is a new person; the other is a new role
5. What is causal to a business process change, and a change to the business case?
   Don't know – these causes are not usually visible to a software provider.
6. Does the logical complexity of the problem affect communication?
   Level of communication must exceed the vision challenge – ie. low level communication quality is ok if complex, but needs to be better as vision challenge is higher.
7. Is workflow change represented by requirement novelty?
   Novelty to the customer is captured by change to workflow, but novelty to software provider is something else.

Initially, all domains were considered in the same model, though this was changed as results from the studies described in chapters 5 and 6 became available. At that stage, consideration was given to each change domain separately, and a distinction was made between factors affecting requirement uncertainty, and those affecting the likelihood of change discovery.

Through this process of model refinement, a set of three domain models emerged containing over 250 variables between them. It was noted that there were many similarities between the domain networks. It was possible, therefore to define a core model based upon an abstraction of the three domain models. The core model contains nodes that correspond to network fragments in each of the domain networks. This mirrors the modular construction approach taken by Laskey [165], in which large networks are designed through the use of single nodes that are place-holders for larger network fragments. Although the Bayesian networks described here were not implemented in this way, a core network consisting of fragment place-holders is a useful mechanism by which to communicate the salient network design features.

7.3.2 Variable Subsuming and divorcing.

Variable subsuming and divorcing are two methods used to simplify the models in order to minimise the data collection effort required to test and use the models, and also reduce the time needed for probability elicitation [163]. Both methods lead to changes in model structure, which will have a knock-on effect to calibration. Therefore it is important to make some effort to ensure that most restructuring is done prior to probability elicitation for model calibration [159]. However, often it is necessary to revisit this stage during the stage of specifying the conditional probabilities, as illustrated in Figure 7.2.

All models firstly underwent a simplification by the removal of variables considered to have the least influence upon requirements change. Often these variables were subsumed into others, meaning that they were considered to be part of another variable, giving the second variable a wider meaning. Notes were taken in order that variable definitions would include the revised variable meaning. Examples of three such variable subsumptions are provided in Table 7.3.

Table 7.3 EXAMPLES OF VARIABLE REMOVAL AND SUBSUMPTION

Date	Variable(s)	
4th Oct '10	Time Pressure and Commercial Pressure	Now included in 'Project Constraint'.
11th Oct '10	Use of Design patterns	Now included in 'Framework Usage'
7th April '11	Technique	Now included in 'Process Quality'

The probability elicitation exercise can take a considerable length of time, and is one of the known disadvantages of manual Bayesian network construction [163]. Since the number of probabilities needed to describe a relationship between two variables grows exponentially with the number of parent variables, one way to reduce the number needed is to disconnect a parent variable, and introduce an intermediate variable. This intermediate variable is referred to as a latent variable since it does not necessarily represent an entity in the real world [174]. This method is called parental divorcing, and as a rule of thumb was used in cases where there were more than 4 parent variables, and only where the parent variables had causal relationships to the child variable that were independent [174].

7.3.3 Define and Scale Variables.

All variables were defined by the industrial practitioners, and kept in a spread-sheet by the researcher. Variable scaling was affected using a discrete 5 point Likert scale between Very low and Very high. However, for purposes of data collection, and to allow more than one practitioner to collect variable data, each point on the Likert scale (1-5) was given a short working description. This helped ensure that there was some consistency in data collection. Two examples are provided in Table 7.4 and further details are provided in section 7.4 when the networks are introduced.

Table 7.4 VARIABLE DEFINITION AND SCALING EXAMPLES

Variable	Description
All Stakeholders Involved	Stakeholders can be broken down into major and minor stakeholders - Major Stakeholders are the drivers behind the need for the new solution and would likely include the business sponsor and business process owners. Minor stakeholders are not driving the core requirements for the system, but have an interest either in an interface system or in the role of compliance, IT, etc. Actively involved means the stakeholder roles are proactive in validating, verifying the requirements and providing the appropriate level of resource to do so. Informed stakeholders are kept in the loop but are not forward coming with owning their particular requirements. How actively are stakeholders involved? 1 Less than 10% 2 80% Key and 30% minor Stakeholders involved 3 All Key Stakeholders involved, all minor Stakeholder informed 4 All known stakeholders informed, 80% minor stakeholders engaged 5 All known stakeholders involved fully involved and informed
Stakeholder Understanding Of IT	Does the customer understand that IT is not a golden bullet, will need to work with the project team to articulate the requirements, may need to modify the requirements for a cost effective solution and that the development is often iterative and does not have unreasonable expectations? 1 No knowledge of IT 2 Use IT but never been involved with specification 3 Involved with IT specification in a minor role 4 Involved with a small number of projects in a major role 5 Involved with a large number of projects in a major role

7.3.4 Specify Conditional Probability Tables

Sometimes referred to as the parameters of a Bayesian network, the conditional probability tables (CPT's) denote the uncertain relationship between a node and its parent nodes [159]. See section 3.5.2 for an illustration of a conditional probability table. Algorithms are available that can create these tables from domain data [162]. However, in the case, such as this one, where no data is available, CPT's must be specified using expert judgement. To specify all probabilities for each combination of parent values can take considerable time [174]. Qualitative approaches can provide visualisation of probabilities and aid the elicitation effort. One such example is the probability wheel where each parental value combination is represented visually as a circle which is subdivided into 'wedges' whose size corresponds to the likelihood that the child variable takes a particular value [159]. Another

technique that can considerably reduce the effort required is the use of mathematical expressions to calculate the conditional probabilities. These may be built using standard statistical distributions, logical operators (eg, if.. then.. else) and relations (less than, greater than) [159]. In this way, rather than specify each and every probability, a number of rules are specified by experts that allow the automatic creation of the CPT's.

Such an approach is taken in this study, that of 'ranked nodes', which was introduced by Fenton [172], and re-used by other researchers such as Wagner [125]. Ranked nodes uses rules and mathematical expressions in situations where the causal influence of the parents combine to affect the value of the child node. For example, using the variables defined in Table 7.4, 'all stakeholders involved' and 'stakeholder knowledge of IT' together affect the mutual understanding of all stakeholders. However, it may be the case that having all stakeholders involved is more important that their knowledge of IT. In other words, parent variables have different levels of influence upon child nodes. Therefore, each parent variable is assigned a weight that reflects their significance relative to the other parent variables.

The likely value of the child node is then calculated based upon a normal probability distribution whose mean is the weighted mean of the parent values. To do this requires that all nodes have values that are continuous rather than discrete. Variables that are expressed on a discrete Likert scale, are mapped onto a bounded numerical scale. Fenton [172] assigns each point on a five point likert scale an interval width of 0.2. Thus 'Very low' is associated with the interval 0-2, 'Low' with 0.2-0.4 etc. These intervals are transparent to a user. Therefore, should 'all stakeholders involved' be twice as significant as 'stakeholder knowledge of IT' then weights of 2 and 1 would be assigned respectively. Given the values 'all stakeholders involved' = 0.1 (Very low) and 'stakeholder knowledge of IT' = 0.3 (Medium), the value of the child node will be represented as a normal distribution whose central tendency is 0.16 [((0.1*2) + (0.3*1))/(2+1)] (Very low).

Since all probabilities are between 0 and 1, the normal distribution is bounded by that interval. A normal probability distribution takes two parameters, the first being the mean, and the second the variance which can be said to denote the distance between the mean and the actual values. This is used to express a level of uncertainty; the higher the variance, the more likely will be the values that are further from the mean. In this way the rules that are defined by experts are the weights for each parent variable, and an uncertainty value for the relationship. CPT's are then specified using an equation based upon a truncated normal distribution whose parameters are a mean (calculated from provided weights) and a variance provided by uncertainty value. Fenton [172], and Wagner [125] found these technique to be

cost effective, easy to implement and effective in producing an initial set of CPT's which were then reviewed by experts. Refer to [170] for an explanation of probabilistic distributions and to [172] for a fuller description of ranked nodes for CPT calculation.

More details concerning the execution of this technique, and modifications in certain circumstances are provided in section 7.4.7.

7.3.5 Review Model Assumptions.

Any model of a real world scenario has an associated set of implicit or explicit assumptions. Although not included in papers concerned with Bayesian network construction methods, explicating the assumptions that underline a model is a useful mechanism to evaluate the limitations of the model, and is especially relevant to empirical research, where study replication may be undertaken by other researchers. This also provides an opportunity to review the models from a critical perspective to ensure that assumptions are acceptable at this juncture. Models were reviewed with practitioners, and underlying assumptions identified.

7.3.6 Sensitivity Analysis

Once Bayesian networks are constructed, prior to validation which will investigate model predictive accuracy, networks can be verified to ensure that they operate in accordance with their intended design. The first means by which that can be achieved is through sensitivity analysis, which can pinpoint problematic aspects of the network [159]. Sensitivity analysis measures the degree to which findings at any node can influence the beliefs at another node, given findings currently entered [131]. Thus an examination of the relative influence of all nodes can help verify the correctness of model structure and parameterization. Standard sensitivity analysis uses calculations of variance reduction when dealing with continuous variables. This is the expected shift in value of the target node, as values are obtained for other nodes. Since the measures are designed to indicate which nodes are most useful in reducing the uncertainty of the target node, the results can aid in network design by supporting decisions concerning the elimination, or subsumption of variables [165]. The results of sensitivity analysis can also be compared with results of other empirical studies to determine if models are designed in accordance with current understanding.

Sensitivity analysis was performed in an iterative fashion. The results were generated by the researcher and brought to the next meeting for discussion, whereupon changes to CPT specification (though not structure) were made. This continued until agreement was reached and the results of the analysis were deemed appropriate. The results of the sensitivity analysis were also used by the practitioners in the consideration of variable selection for model walkthroughs.

7.3.7 Model Walkthroughs

The second means by which networks can be verified are through model walkthroughs. Sometimes referred to as 'What if analysis' [159], walkthroughs allow a visual examination of the working of the models. Scenarios may be specified in advance, or done on an ad-hoc basis. Results are reviewed to ensure that they are in line with the expectations designers and experts involved in Bayesian network construction.

Three practitioners provided a series of 15 use cases upon which to determine a set of posterior probabilities. This process took approximately 2 hours and was attended by a researcher, and three practitioners who provided pre-prepared scenarios. Each practitioner had hard copies of the three complete Bayesian nets, a set of variable descriptions, and the results from the sensitivity analysis. They were asked to provide a short requirement description and at least 6 of the network variables, including a value for total requirement volatility. If possible the network variables were to be selected from amongst those with greater influence upon volatility. Practitioners provided data from a project that they had recently managed, and all measures were subjective, apart from the value for volatility which was the actual volatility observed. It was reported that the time taken to gather data for each use case was between 5 and 10 minutes.

7.4 Bayesian Networks to Predict Requirement Change

The approach to causal variable identification is to make use of the cause constructs contained within the requirements change source taxonomy (section 4.7) combined with the requirements attributes considered by our industrial partners to be influential to change (section 6.4). During modelling, particular emphasis was placed on generic variables rather than business or project specific attributes of requirements, software, or the environment.

This was in order to facilitate the use of the model in various contexts for predictive and analysis purposes. The models began as a single model incorporating all change domains, and evolved through a defined process described in section 7.3, to arrive at the models presented below.

7.4.1 Core Model

Through the process described in section 7.3, and in particular the stage of variable subsuming and divorcing, the variables became organized such that related variables were grouped together as a sub-net. Across change domains, common patterns of variable influence arose, and these patterns were encouraged to foster standard design features which promoted ease of understanding of net structure. In so doing, a common 'core' model emerged which can be considered an abstraction of the common elements of each of the independent domain models. This core model captures the salient predictive model design concepts and is illustrated in Figure 7.4. For the purposes of network discussion node names are referred to in in single quotation marks.

.

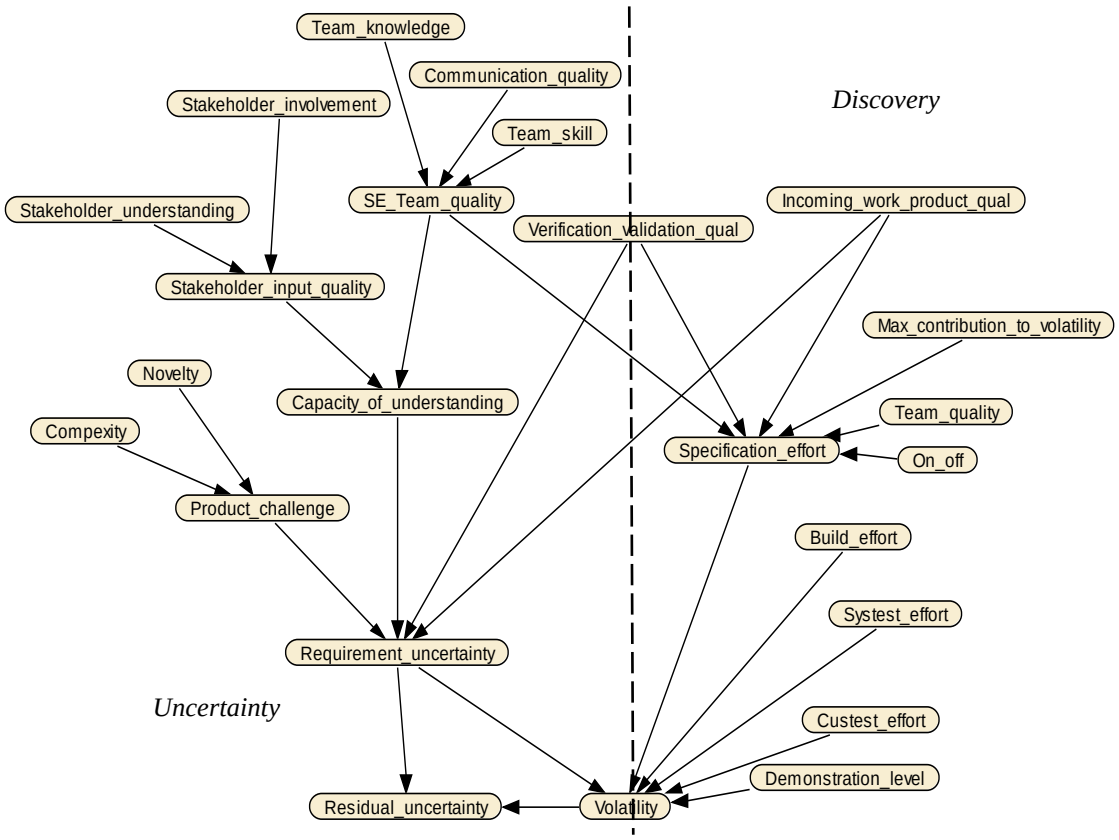

Figure 7.4 REQUIREMENTS CHANGE PREDICTION: CORE MODEL.

The variables are represented by nodes, and the arrows indicate a causal influence, through which the state of a variable raises or lowers the probability of the states of its children, as described in Section 3.5.2. Each node in the core model can be considered as a place-holder for a network fragment in each of the domain models.

As can be seen, there are two sides to the model, separated by dashed line. The situations on the left give rise to requirement uncertainty (the change-proneness of a requirement), and activities on the right lead to change discovery. This is reflective of the fact that a requirement may be uncertain for the duration of the application development (and beyond), yet changes are never discovered. Both sides are necessary to predict requirements volatility, that is, the measure of the change that will be seen in a particular development project phase. The uncertainty side of the model would predict the overall uncertainty of a requirement, and the discovery side would evaluate the likelihood that a proportion of that uncertainty would be realised by a particular activity. The difference between the requirement uncertainty and the realised volatility is 'residual uncertainty' which would remain until the next phase. However, it is possible to predict only the requirements uncertainty as a measure of the potential volatility of a requirement during the lifetime of the requirement, using only the uncertainty side of the network. This can take place near to the beginning of the project life-cycle, as soon as an initial set of requirements has been defined.

The core model is an abstraction of the domain models, and as such 'requirement uncertainty' is representative of each of the individual change domains. Hence, there are three uncertainty models, each predicting a particular aspect of requirements uncertainty defined by the change domains of *vision, requirement specification* and *solution.* Therefore, change 'discovery' refers to the types of changes that can be revealed such as inconsistency (in the domain of *requirements specification*), and also the types of changes that relate to business processes or an idea that has yet to be conceived (in the domain of *vision)*. Also, there are changes that will arise due to non-functional needs such as maintainability (in the domain of *solution*). Some of the nodes in this core model can be considered latent nodes, such as 'SE team quality', and 'stakeholder input quality', since they do not correspond to real world entities. While (subjective) data could be collected for these variables, modelled thus, these variables are not regarded as real world entities, but instead a convenient organisation of their parents [174].

The activity triggers on the right hand side of the model relate to elements of the software development process – specification, build and unit test, system test, and customer testing. They bear comparison to the change discovery activities identified in the previous case study

described in chapter 5. However, the discovery activities of translate and test discussed in section 5.6.6 are now subdivided into more specific development related tasks. Specifically, in this case, specification and build and unit test correspond to the previously used translate activity while system test and customer test correspond to the previous test activity. This illustrates an example of the evolving nature of model development. There is also an activity referred to as demonstration, which equates to demo in the previously defined change discovery activities. This captures situations such as product walkthroughs or presentations which are also opportunities to discover change. Table 7.5 contains a description of each activity. Although similarly named, the activities are not directly related to life-cycle phase, and may be undertaken concurrently. Each activity sub-net has an associated 'on_off' node reflective of the reality that not all activities will be on-going at all times. This allows the model to be agnostic to a particular development methodology since either waterfall or agile-based methods have activities corresponding to those in the model, though they may be on at different times.

Table 7.5 VOLATILITY PREDICTION: ACTIVITY DESCRIPTIONS.

Build	The tasks involved in designing system architecture, coding and unit testing
System Test	The specification of system test plans, the coding or writing of test procedures and the activities involved in system wide testing
Customer Test	The specification of customer test scenarios, the writing of test procedures and the activities involved in customer testing
Demonstration	The communication of requirements as they currently stand to an audience consisting of minor, or major stakeholders such as prototyping, demonstrations, presentations etc.

The demonstration activity sub-net has different influences, but the factors influencing 'specification_effort' are replicated in each activity relating to the software development life-cycle. They are not shown in Figure 7.4 for simplicity.

Each activity sub-net also has a 'max_contribution_to_volatility' node that captures the notion that any activity, even at the highest levels of quality and effort, cannot be relied upon to discover all change. For example, as observed in the previous case study, very little *vision* volatility was discovered through the activity of build and unit test. Therefore, regardless of the level and quality of task effort, there is a ceiling (though this is uncertain) on the amount of volatility that can be discovered in any activity for a particular change domain

uncertainty. The contribution variable is expressed as a percentage of the total potential volatility, and captures this constraint upon volatility discovery by any particular activity.

Since activity volatility contributions are expressed independently from one-another it is possible that when two activities are on simultaneously, the sum of the contributions may be in excess of 100%. For example, should it be the case that it is considered possible to discover all (100%) *requirement specification* volatility through the 'build' activity and also (100%) through the 'customer testing' activity, then in the situation where both 'build' and 'customer test' are 'on', the total contribution to volatility will be 200%. This may lead to double counting, but is reflective of the real world where it is entirely conceivable that the same change may be discovered at the same time through different activities. However, should it be the case that the same change is discovered more than once, the volatility for that requirement will only be increased by one occurrence of the change. Hence predictive networks are designed such that total volatility may not exceed the requirement uncertainty in order to provide a ceiling for volatility. This is reflective of reality, in that should the levels of discovered volatility be in excess of uncertainty, this would imply that the level of uncertainty is incorrect, and the requirement is more uncertain than originally believed. Activity sub-nets are discussed in more detail in section 7.4.6.

The networks evolved such that causal factors influential to requirements uncertainty (parent nodes of the node 'Requirement uncertainty' in Figure 7.4) can be broadly defined as:-

a) Product challenge ('Product_challenge') – the difficulty of the application.
b) Capacity of understanding ('Capacity_of _understanding') - the ability of the team to understand the product to be developed, including the Software Engineers and the stakeholders.
c) Quality of the processes and techniques used ('verification_validation_qual') methods of elicitation, process maturity etc.
d) Quality of the incoming work product ('Incoming_work_product_qual') – for example, this could refer to a set of predefined requirements as a result of a procurement process, or a set of business scenarios and rules from which test plans are developed. This could be a document, a set of models, or a verbal communication.

There are three variables that span both sides of the network. The ability of an activity to uncover change depends upon factors related to the team, the process, and the quality of the product they are given to work with. Therefore the team quality ('SE_team-quality') affects

both the uncertainty of a requirement, and the likelihood that change will be discovered. However, this can refer to different sets of people with differing skills and experience. Similarly, techniques and processes ('Verification_validation_qual') used will affect both the uncertainty and the likelihood that change is discovered. The incoming work product ('Incoming_work_product_qual') can also span both sides of the network since, for example, a low quality requirements specification document will increase requirement uncertainty, but also challenge the build effort, which will affect the likelihood of change discovery.

The variables may have different scope within a project. Those that affect the product challenge ('Product_challenge') relate to a specific requirement, whereas the team qualities ('SE_team_quality') may have less granularity, and be applicable to more than one requirement. The scope of such variables will depend upon the size of the software application. Larger developments may have teams of software engineers and customer stakeholders that are involved with a particular group of application requirements. Similarly, certain requirements may undergo more rigorous process and different analysis techniques may be used depending upon the nature of the requirement. In smaller projects it is possible that these variables tend more to constancy, and thus have the effect of influencing all requirements in the same way. In such situations, it is a combination of only the variables of the requirements themselves that influence relative uncertainty. Factors which would affect every requirement equally in all situations, such as the size of the project, overall time constraint, or project duration were removed, since they would affect all levels of requirement volatility equally.

Each node in the uncertainty side of the core model has corresponding variables in each of the three change-domain uncertainty models. For example, 'Product challenge' is illustrated here as having 'complexity' and 'novelty'. However, in the domain of vision, 'product challenge' is a combination of complexity, novelty, dependency and software framework (COTs) usage as these elements pertain to the business. By contrast, in the domain of requirements specification, product challenge combines not only the business aspects of these factors, but also their technical bearing. As discovered in a previous case study, (section 6.5) 'Novelty' can mean novelty to the business, so therefore it may be more difficult to articulate, or novelty to the software engineer, in which case it may be more technically innovative. Similarly, each of the other nodes in this network has change-domain specific sub-nets. The same activity nets are added to the domain specific uncertainty nets to predict volatility, though they have different strengths of influence upon change discovery reflective of the observation that changes in each domain are discovered through different

activities. For example, a large portion of *vision* uncertainty was discovered through 'customer testing', whereas the 'build' activity uncovered a higher proportion of *solution* uncertainty.

As already discussed, the models were simplified from their initial specification. One example of variable reduction concerned the sub-nets corresponding to 'verification_validation_qual'. Originally, this comprised a set of 6 nodes, but these were reduced to 3 when it became evident that they were replicated many times to capture the quality of different aspects of the software development process. Finally, in all models, the three variables used were 'procedures and techniques quality', 'process constraints' and 'tools'. Procedures and techniques would include process maturity and requirements engineering practices such as elicitation techniques and reviews. Process constraints includes both time and resource constraints, and tools refer to requirements management tools, risk registers etc.

A working definition of all network nodes is provided in section 7.4.2. The domain models are illustrated in Figure 7.5, Figure 7.6, Figure 7.7, and the activity models follow in Figure 7.8 and Figure 7.9.

7.4.2 Variable Definitions.

Table 7.6 contains a list of (non latent) variable definitions, which were agreed following network structuring. They are provided here for look up purposes, and describe the nodes in the networks presented in the following sections. It should be noted that these explanations were derived by our industrial partner to aid model (and later) data specification, and are not presented here as semantically correct definitions of the terms used. All variables are scaled on a 5 point Likert scale from Very low to Very high, which is a common approach to the scaling of variables that relate to quality issues [164]. However, industrial practitioners provided scaling heuristics which were used to ensure common understanding for data collection purposes during model validation. Table 7.6 is replicated in Appendix 2 where scaling heuristics is added. These definitions were held centrally on a spread-sheet by the researcher.

Table 7.6 BAYESIAN NETWORK VARIABLE DEFINITIONS

Variable	Description
All Stakeholders Involved	The level of involvement of all stakeholders. Stakeholders can be broken down into Major and Minor stakeholders. Major Stakeholders are the drivers behind the need for the new solution and would likely include the business sponsor and business process owners. Minor Stakeholders are not driving the core requirements for the system, but have an interest either as an interface system or in the role of compliance, IT, etc.
Synergy Stakeholder Agenda	The alignment of stakeholder agendas including agreement and understanding of explicit compromise where natural conflict occurs (e.g. compliance vs. cost of running)
Stakeholder Communication	Ability of stakeholders to effectively communicate with each other. Barriers include large numbers of stakeholders, separate locations, separate cultures, conflicting business drivers, poor communication tools (word of mouth, email, paper, electronic docs, video conferencing, collaboration tools, web cams).
All Stakeholders Identified	The likelihood that all stakeholders are identified.
Stakeholder Understanding of Problem	How well the customer really understands just what problems they are experiencing, what the solution should look like and how they are going to measure that they got there.
Stakeholder Understanding IT	How much experience and knowledge the customer has of the process of developing computer applications.
Business Dependency	The pervasiveness, or interrelatedness of a requirement.
Business Novelty	The novelty of the requirement to existing business practices within the organisation or to other common, in use, practices.
Business Complexity	The complexity of this requirement - how hard it is to understand, how many facets to it, how big is it.
Technical Dependency	The extent that the coded requirement relies upon, or is used by other coded requirements.
Technical Novelty	The novelty of the technical requirement to the design, build and test teams.
Technical Complexity	The complexity – how hard is the technical part of this requirement to understand
Framework Usage	Out of the box usage compared to design. Reliance upon, and use of design patterns.
Team Morale	The attitude of the team reflected in rotation of project members and customers' motivation levels
Senior Management	The support, involvement and commitment of the senior management to help and remove obstacles to progress.
Tools Quality [Business, Specification, Systest, Custest, Build]	The quality of the tools used in validating, verifying and managing individual and groups of requirements (missing, duplicate, conflicting requirements). There is one for each life-cycle activity, and one for the tools used by the business to arrive at a product vision.

Variable	Description
Process and Technique Quality [Business, Specification, Systest, Custest, Build]	Level of process definition with feedback and appropriate level of governance; appropriate techniques used for verifying that we have captured the right thing, validating that it is what is wanted, checked to make sure nothing is missed for both individual and groups of requirements. There is one for each life-cycle activity, and one for the tools used by the business to arrive at a product vision.
Process Constraints [Business, Specification, Systest, Custest, Build]	The constraints on the process that may lead to sub-standard performance, including time pressure and commercial pressure that may disincentives the discovery of change / clarification; obstacles and barriers to adherence to defined processes. There is one for each life-cycle activity, and one for the tools used by the business to arrive at a product vision.
Inter-team Communication	The effectiveness of the communication of requirements between project team members and project team to customer. Includes geographic separation, number of different locations, cultural gaps, number of people involved, tools available to mitigate cultural or geographical disparity.
No of Requirements	The number of requirements in the group. The more requirements in a group the harder it is to understand the big picture, their interrelatedness and possible overlaps/conflicts and therefore the greater effort and tool support needed.
Requirement Group Complexity	The interconnectedness of the requirements in the group, can they be thought of individual and atomic or is there a high degree of interrelatedness and 1 change can ripple through many others and therefore reducing the chance of understanding both of the requirement and the impact of change.
Incoming work product Quality (Vis, Spec, Build)	The clarity and unambiguity of the work product that is being used to further application development during a particular activity. There are three such work products – Vision, Specification, and Build.
Team Skill [Analyst, Solution, Systest, Custest]	The ability of an individual (or group of individuals) to translate higher level work product into a dependent work product (or provide verification, validation, Quality Control activities).
Team Knowledge [Analyst, Solution, Systest, Custest]	The ability of an individual (or group of individuals) to understand their respective domains (e.g. business knowledge, COTS/Framework knowledge, Software Language).
Level Ongoing Comms	The techniques and frequency of communication between major and minor stakeholders in the form of walk-throughs, prototypes or demonstrations in order to keep stakeholders and the wider audience informed of the current shape of the solution.

Variable	Description
Contribution to volatility (from spec, demo, build, systest, custest)	Expressed as a maximum percentage of the total discovery effort for each activity.
Uncertainty (Vision, Req Spec, Solution)	The disposition of the requirement to change.
Volatility (Vision, Req Spec, Solution)	The measure of changes.

7.4.3 Vision Uncertainty

Figure 7.5 illustrates the Uncertainty side of the *vision* domain model. Since the domain of *vision* describes sources of change that arise due to a change in the business functionality of the product to be developed, the focus here is upon stakeholder and senior management involvement and communication issues, as well as qualities of individual requirements attributes that convey a business content. Late engagement of a particular stakeholder may change the direction of the product to be developed, and the quality and quantity of interaction between all stakeholders and the engineering team is of prime importance to the domain of vision.

It is important that the stakeholders are identified (‘All_stakeholders_identified’), involved (‘All_stakeholders_involved’), and understand and agree upon the ‘problem’ to be captured (‘Stake_understanding_prob’, ‘Synergy_stakeholder_agenda’). It is advantageous if the stakeholders comprehend the process and constraints involved in software development (‘Stake_understanding_IT’). The capacity to collectively understand the product vision (‘Capacity_understanding’) is then a combination of the joint capability, involvement and agreement between the stakeholders (‘Stakeholder_input_quality’) and the quality of the team (‘SE_team_quality’). The roles involved in the team that might affect the product *vision* are stakeholders and senior management. Team efficacy is influenced by the quality of communication between stakeholders (‘Stake_communication’), team morale (‘Team_morale’), and the level of senior management support (‘Senior_management’). The variable ‘Bus_ver_val_quality’ captures the influence of the process, techniques, and tools (‘Bus_proc_tech_qual’, ‘Bus_tools_quality’) used in the derivation of the agreed product

vision and any constraints on the process ('Bus_process_constraints'). Requirement attributes that may challenge the clarity of understanding are the business faculties of novelty ('Bus_novelty'), complexity ('Bus_complexity'), and dependency ('Bus_dependency'). However, usage of off-the-shelf software ('Framework_usage') may assuage the effect of such attributes. There is no 'Incoming_workproduct_quality', since the product *vision* arises from a combination of sources not yet agreed, documented or formal.

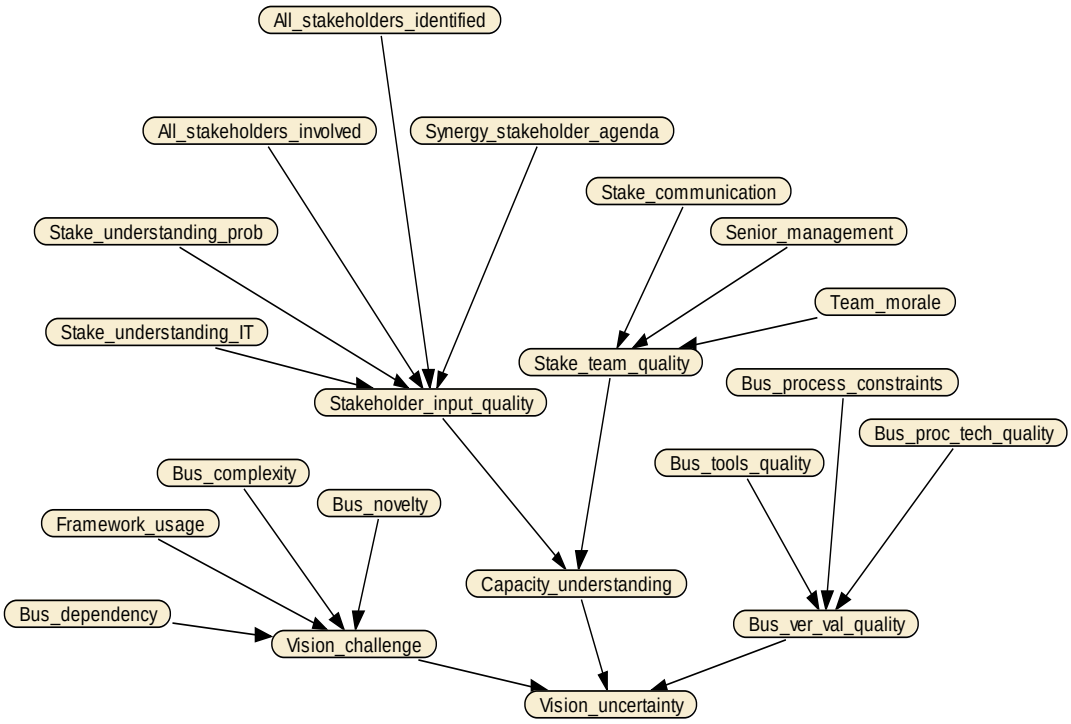

Figure 7.5 REQUIREMENTS CHANGE PREDICTION: VISION UNCERTAINTY

7.4.4 Requirements Specification Uncertainty

The domain of *requirements specification* is the most elaborate of all the domains in terms of the number of variables involved and their relationships. Since it is concerned with inconsistencies and ambiguity, there is more of an emphasis upon the qualities relating to the task of understanding and capturing requirements. The uncertainty side of the *requirements specification* domain model is illustrated in Figure 7.6.

The involvement of the stakeholders (‘Stakeholder_input_quality’) is still important, and as a network fragment, consists of the same nodes present in the vision uncertainty network described in Figure 7.5. However, the skill (‘Analysis_skill’) and knowledge (‘Analysis_ knowledge’) of the involved analysts has significant impact upon specification accuracy. The quality of communication (‘Interteam_communication’) and team morale (‘Team_morale’) of the wider team which includes the efforts of senior management (‘Senior_management’) is also influential to the efficacy of the team as a whole. The capacity of understanding (‘Capacity_understanding’) is now broadened to include the analyst team (‘Analyst_team_quality’) which is defined by the skill and knowledge of all analysts as a whole.

In the domain of *requirement specification* the product challenge (‘Product_challenge’) is affected by the joint influences of the business related requirements (‘Business_challenge’) qualities and the technical qualities (‘Technical_challenge’). The business related qualities (‘Business_challenge’) correspond directly to the net fragment ‘Vision_challenge’ in the domain of *vision* and shares the nodes in that network fragment, with the exception of ‘Framework_usage’ which now affects the challenge to the requirement specification (‘Reqspec_challenge’). The technical challenge (‘Technical_challenge’) is influenced by the technical aspects of novelty (‘Technical_novelty’), complexity (‘Technical_complexity’) and dependency (Technical_dependency’). Of importance is the concept of a requirements group (‘Req_group_challenge’), where entanglement (‘Req_group_complexity’, ‘Number_of_requirements’) and hence the potential for inconsistency is considered an influence as well as the qualities of the individual requirements involved. This is reflective of the truism that the whole is more than the sum of the parts. The requirement group may be at any level of granularity, and together with the technical and business related requirements attributes affect the product challenge (‘Product_challenge’). The verification and validation quality attributes (‘Spec_ver_val_quality’) are the same as those in the vision domain network fragment, but they may pertain to different techniques and processes. The incoming work product (‘Incoming_work_product’) in this case is the vision work product (output from the process of defining the product vision) and refers to any documentation or communication which is used during requirements specification.

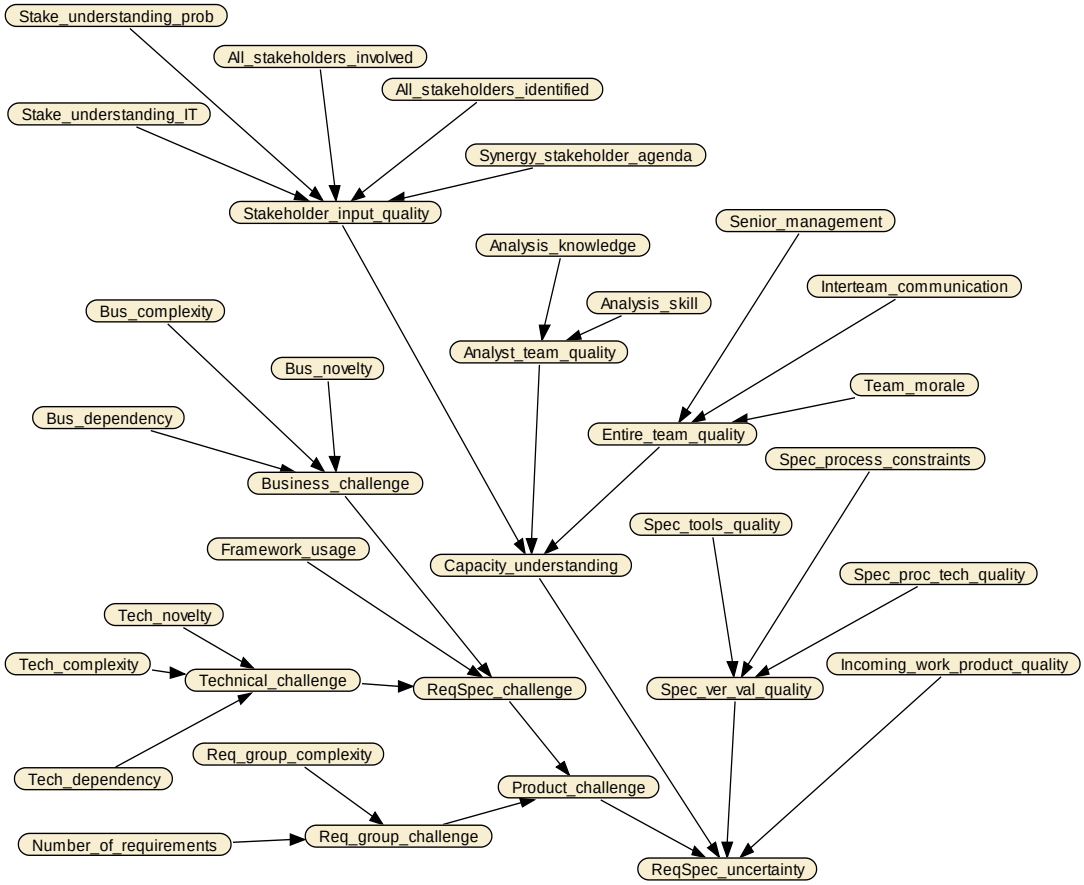

Figure 7.6 REQUIREMENTS CHANGE PREDICTION: REQUIREMENTS SPECIFICATION UNCERTAINTY

7.4.5 Solution Uncertainty

The *solution* uncertainty, illustrated in Figure 7.7, is concerned with the technical components of the solution architecture and supporting toolset, and is captured in a simpler model, mostly due to the lack of significant influence of the wider group of project stakeholders. Similarly to the requirement specification domain, the aspects of the involved team include those with direct responsibility for the build effort.

In the *solution* domain, the product challenge relates only to the technical aspects of the requirement, and the group to which they belong. As such the network fragments corresponding to the technical challenge ('Technical_challenge'), and challenge presented by the requirement group ('Req_group_challenge') contain the same nodes as the network fragments of the same name in the requirement specification domain described in section 7.4.4.

The capacity of understanding (‘Capacity_understanding’) is influenced by the skill (‘Solution_skill’) and knowledge (‘Solution_knowledge’) of the programmers and technical architects, which together is referred to as the ‘solution team’ (‘Solution_team_quality’) . The *solution* uncertainty is also influenced by the support of senior management (‘Senior_management’) and the team morale (‘Team_morale’) and communication (‘Interteam_communication’). The verification and validation quality (‘Build_ver_val_quality’) is the same as the corresponding network fragments in the domains of *vision* and *requirement specification*. However, the tools, techniques and processes will be concerned with application architecting and coding such as class patterns and standards compliance. The incoming work product (‘Incoming_work_product_qual’) in this case is the detailed specification of the requirements against which coding is defined and measured.

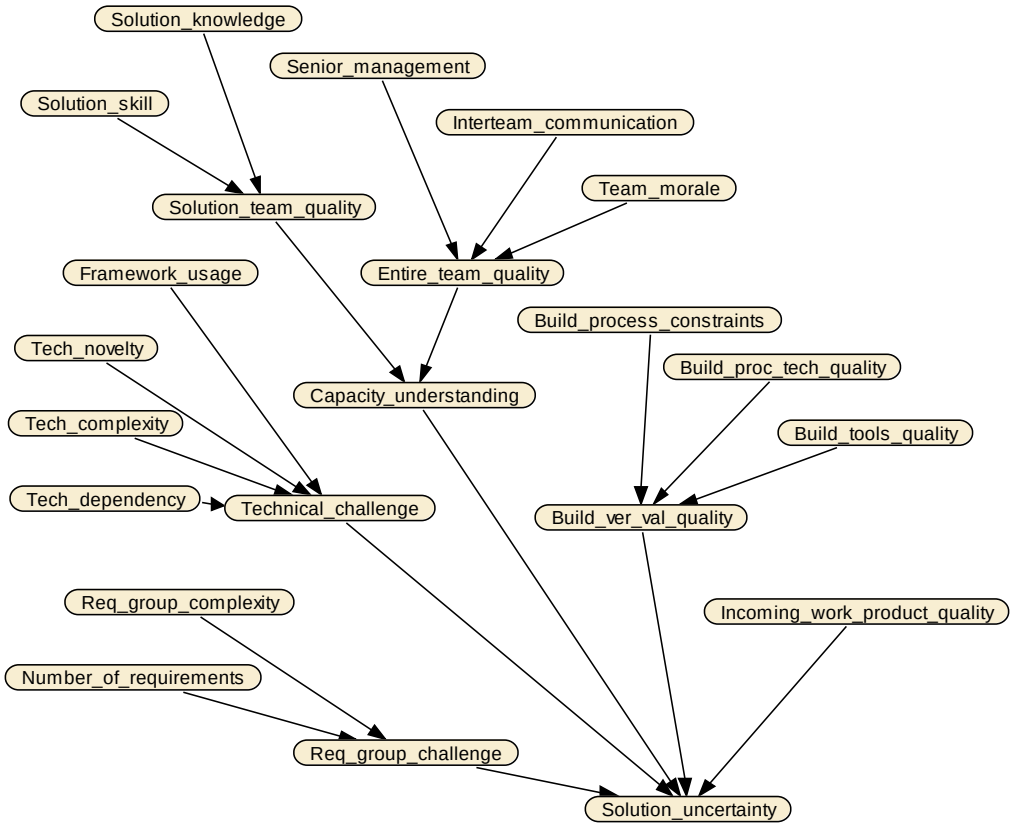

Figure 7.7 REQUIREMENTS CHANGE PREDICTION: SOLUTION UNCERTAINTY

7.4.6 Activity Sub-nets to Predict Volatility

There are five activity subnets which are used to determine the likelihood that requirements change will be discovered during product development. Thus the activities confine themselves to those that pertain to software development (rather than maintenance). Four of those activities relate to development process activities (specification, build, system test, customer test). There is an additional activity referred to as demonstration which accounts for presentations, prototypes, reviews and general communication of the requirements to stakeholders. All five activity subnets are attached to each of the uncertainty networks described in the previous sections, apart from the demonstration activity which is not attached to the *solution* domain. The development process activity subnet relating to the activity of specification is illustrated in Figure 7.8.

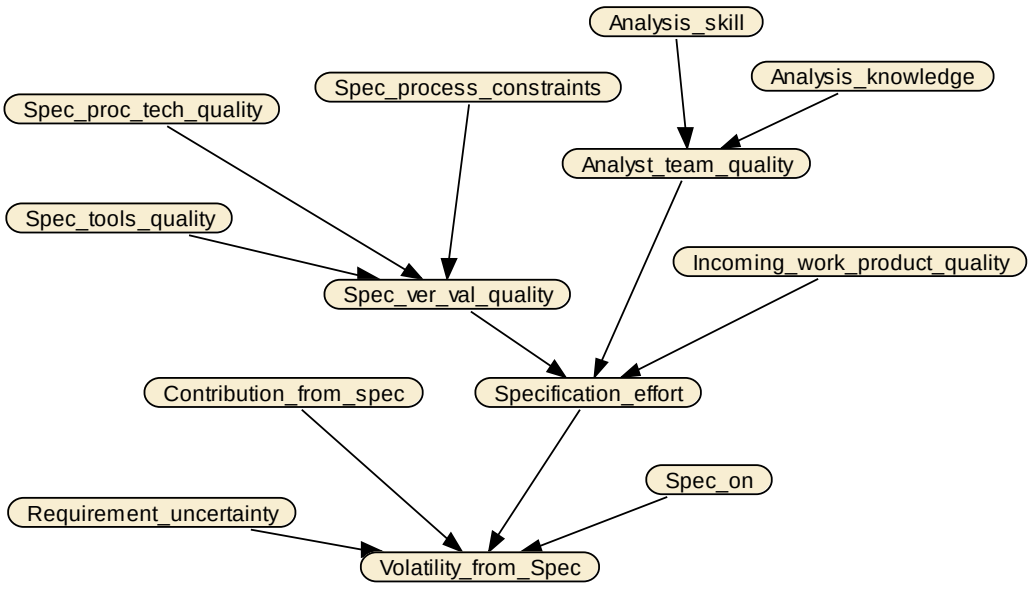

Figure 7.8 EXAMPLE ACTIVITY SUBNET: VOLATILITY DISCOVERED THROUGH THE SPECIFICATION PROCESS

The node 'Requirement_uncertainty' refers to the uncertainty predicted by the models illustrated in Figure 7.5, Figure 7.6, and Figure 7.7 for the domains of *vision*, *requirement specification* and *solution* respectively. While there is some similarity between uncertainty and activity networks, given that they are both influenced by the skill of the team members ('Analyst_team_quality') and the methods and processes used (Spec_ver_val_quality'), the focus of each activity subnet is upon the effort given to perform a particular task involving a particular requirement. This is in contrast to the emphasis upon the communication and

product challenge issues that may give rise to product uncertainty. The verification and validation variables ('Spec_tools_quality', 'Spec_proc_tech_quality', 'Spec_process_constraints') in the activity subnets reflect that sufficient time and attention is given to each individual requirement. In some circumstances the scope of these variables are at a higher level of granularity than the same variables in the uncertainty models. Since activity subnets are attached to each domain uncertainty network, each activity subnet will be used three times for a particular product phase, one for each of the domains of *vision*, *requirements specification*, and *solution*. Importantly, there is a node called 'contribution_from_spec'. This is used to control the level of potential volatility that can be discovered through each activity, as described in section 7.4.1. There is also a node called 'spec_on' which allows this activity to be on or off for predictions in a given project phase.

The subnet illustrated in Figure 7.8 is replicated for the activities of build, system test and customer test. Each development process activity model involves a particular team, a specific work product, and verification and validation nodes relating to that activity. These variables are causal to change discovery but also span both sides of the network and affect requirements uncertainty. However, this causal influence is not evident in all domains. For example, the team defined in 'analyst_team_quality' affects the specification activity effort, and also the *requirement specification* domain uncertainty. Similarly the incoming work product is specific to each activity, and can also affect uncertainty. Descriptions of teams, incoming work products and verification-validation measures that affect requirements uncertainty are described in the previous section and Figure 7.5, Figure 7.6, and Figure 7.7 for the domains of *vision, requirement specification* and *solution* respectively. Table 7.7 describes the incoming work product and the team that is relevant to each activity subnet. The verification-validation measure bears the same name as the activity.

Table 7.7 INCOMING WORK PRODUCTS AND TEAMS FOR EACH ACTIVITY SUBNET

Activity	**Incoming Work Product**	**Team**
Specification	Vision	Analyst
Build	Specification	Solution
System Test	Build	System Test
Customer Test	Specification	Customer Test
Demo	N/A	Stake/Entire Team

The team effort that affects the likelihood that a change will be discovered during an activity is confined to those people that have direct responsibility to carry out the task. This is by contrast to the teams influence upon requirements uncertainty which draws upon attributes of the wider team in addition to qualities of those with direct responsibility. While there is some causal influence, it was not thought that senior management or team morale had a significant impact upon the ability of the team to carry out their tasks effectively. The build team was referred to as the 'solution team' due to particular preferences of our industrial partners who considered that the term 'build team' was confusing since it may be seen to relate only to programmers. The solution team can include technicians, architects and technical team leaders. There are two teams that influence the 'demo' activity. In the domain of *vision*, is the 'Stake_team_quality' as illustrated in Figure 7.5 and in the domain of *requirement specification* it is the 'Entire_team' shown in Figure 7.6.

The incoming work products are straightforward in that each life-cycle stage culminates in a work product which is then used in the next phase. Thus, the specification activity uses the work product (if there is one) derived from an understanding of the product vision. Similarly, the build activity uses any work products from specification, and the system test effort is based upon the results of the build activity. Customer testing uses the work products from the specification to construct test scenarios. Such scenarios may be defined concurrently to the build phase.

A similar sub-net for demonstration is shown in Figure 7.9. In common with the development process activities, the demonstration sub-net has variables for 'on/off', 'Contribution_from_demo' and 'Requirement_uncertainty'. There are no variables describing qualities of techniques or process, but instead a variable describing the level of on-going communication ('Level_ongoing_comms'). This captures the regularity of product demonstrations, prototypes, or presentations that are undertaken by any member of the development team or wider team of stakeholders talk about different incoming work product. The level of demonstration is affected by the entire team quality ('Entire_team_quality'), and attitude.

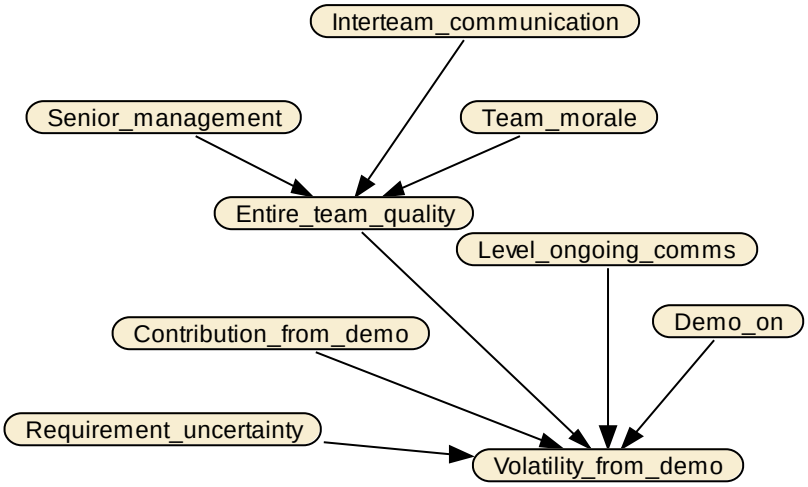

Figure 7.9 EXAMPLE ACTIVITY SUBNET: VOLATILITY DISCOVERED THROUGH DEMONSTRATION

7.4.7 CPT Specification

Since the networks were organized to reflect combinations of related variables, in most cases a technique called ranked nodes (see section 7.3.4) was used to provide a set of initial CPT's which would then be reviewed and reconsidered following sensitivity analysis and walkthroughs. This technique required the specification of parameters for an expression that would generate the CPT's; weights were provided by industrial practitioners for each parent node, as well as an overall uncertainty value for the relationship. This was achieved within 5 hours, and represented a significant time saving compared with seeking values for all CPT's. The weights and relationship uncertainties were written onto paper copies of the networks by the industrial practitioners, and transferred to the spread-sheet containing the variable descriptions and scales by the researcher. In all cases the uncertainty given was 0.001, which became the variance used in the calculation (see section 159).

Figure 7.10 illustrates such a net fragment which is part of the network 'Requirements Specification Uncertainty' illustrated in Figure 7.6, and Figure 7.11 shows the corresponding net fragment from the network 'Vision_uncertainty' illustrated in Figure 7.5. The weights, expressed as numbers summing to 10, that were assigned to each parent are shown beside the nodes. Each of the variables senior management, inter-team communication and team morale combine to give an overall value for the quality of the team.

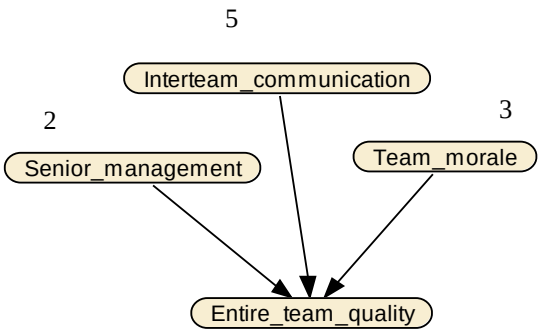

Figure 7.10 TEAM QUALITY NET FRAGMENT FROM REQUIREMENTS SPECIFICATION UNCERTAINTY

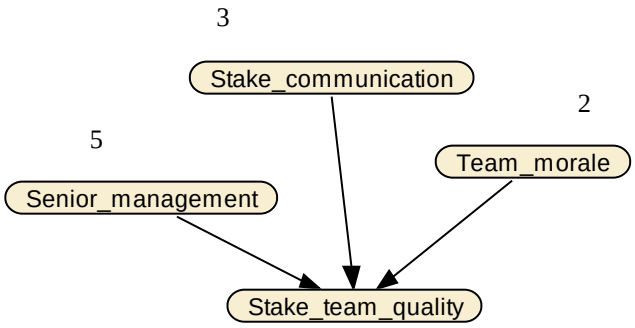

Figure 7.11 TEAM QUALITY NET FRAGMENT FROM VISION UNCERTAINTY

Since the weights indicate the relative influential significance, it can be seen that the industrial practitioners reasoned that in the domain of vision, where the focus is upon product direction, the influence of the quality and commitment of senior management upon the team quality is more significant than that of the communication between the various stakeholders. This situation is reversed in the requirements specification domain, where communication between all team members is considered to make more of a substantial contribution to the quality of the team as a whole. Similarly, between domains, there are differences in weights of the variables in corresponding net fragments, reflecting the observation that requirement attributes have different correlations with volatility in different change domains discussed in Section 7.2.

Sometimes, it is the case that a variable has a negative influence upon a child. For example, Figure 7.12 illustrates that tools, procedures and techniques as well as process constraints have an effect upon the total quality of verification and validation. However, while higher quality tools, procedures and techniques increase the likelihood of higher quality verification, a higher level of process constraints will have the opposite effect of lowering

the verification quality. In this instance, the contribution of process constraints was subtracted, rather than added to the central tendency in the normal distribution CPT equation. The total causal influence, which is a summation of all weights, is calculated by adding all numbers as thought they were positive. As a modification of the ranked nodes technique described in section 7.3.4 this somewhat crude approach was pragmatic and sufficient to produce initial CPT's in these cases.

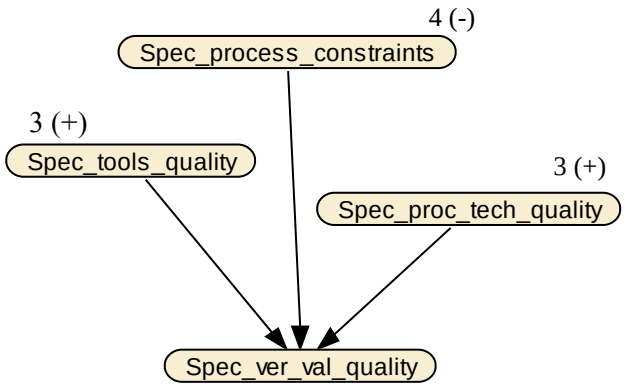

Figure 7.12 VERIFICATION VALIDATION NET FRAGMENT FROM REQUIREMENT SPECIFICATION UNCERTAINTY

This technique works well when the influence of the parents combines such that their effects can be added or subtracted, and provides a means by which a CPT table can be initially created for review. Fenton denotes this kind of relationship a 'synergy'.

However, there are situations where the relationships between nodes that aren't as straight-forward. One such scenario is the combined effect of the capacity of understanding, the product challenge and the validation and verification quality shown in Figure 7.14. The vision challenge will raise the requirements uncertainty, while both the capacity of understanding and the verification and validation quality lower the uncertainty. However, this is not in a linear fashion. Instead, the capacity of understanding, which reflects the ability of the team members, can compensate for lower quality techniques and processes, especially when the product challenge is low. This compensation was affected by weighting the contribution of the 'Bus_ver_val_quality' node according the combination of the other parent nodes. The table in Table 7.8 illustrates a partial conditional weighting for 'Bus_ver_val_quality' based upon the states of the other parent nodes. As can be seen, when the 'Vision_challenge' is Very Low and the 'Capacity_understanding' is High, the quality of the techniques and processes ('Bus_ver_val_quality') has less of an impact upon the 'Vision_uncertainty'. By comparison, when the 'Vision_challenge' is High and the

'Capacity_understanding' is low, then the 'Bus_ver_val_quality' has more of a significant influence upon the 'Vision_uncertainty'.

Each uncertainty node in all three domains was represented by such compensatory ranked nodes. Since the total number of weight contribution values to provide was 25 for each node (given 2 variables each having 5 states), giving a total of 75, this process took only ½ an hour.

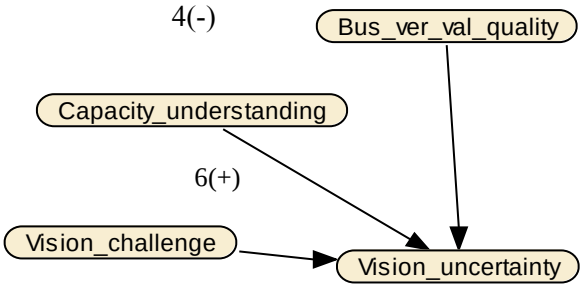

Figure 7.13 VISION UNCERTAINTY NET FRAGMENT

TABLE 7.8 EXAMPLE 'BUS _VER_VAL_QUALITY' WEIGHTS

State of 'Vision_Challenge'	State of 'Capacity_Understanding'	'Bus_ver_val' weight
Very Low	High	0.1
Med	Med	2
High	Low	4

The only situation not covered by ranked nodes or compensatory ranked nodes was the CPT for the nodes corresponding to volatility levels for each activity. Rather than a synergy of causal contribution from the parents, the volatility was a product of the uncertainty, the effort, and an estimated maximum contribution made by that activity to the total volatility. See Figure 7.8 for an example of a volatility node (volatility_from_spec) whose CPT's are generated from the product of parental values ('Contribution_from_spec', 'Requirement_uncertainty', 'Specification_effort'). There is also an on/off Boolean parent which, if off, yields a zero volatility. A value for total volatility was calculated by adding the volatility discovered through each of the five activities. However, this total volatility had an upper limit of the Requirement uncertainty in each domain to avoid the potential inflation of volatility due to double counting as described in section 7.4.1.

Once the CPT's had been generated, any assumptions inherent in the networks were reviewed. Subsequently networks were verified by means of sensitivity analysis and walk-throughs to ensure that they behaved in accordance with their intended design.

7.4.8 Model Assumptions/Limitations

Any model of a real world scenario has an associated set of implicit or explicit assumptions. These may compromise the ability of the model to handle certain complex situations, and are listed here to illuminate potential usage limitations, and further elucidate the rationale regarding model structure.

I. It is possible to discover all requirement uncertainty during development. This is unrealistic since requirements continue to change subsequent to delivery. However, the scope of this study is software development, and model validation is only possible for the time-frame during which validation data is available. Arguably, this limitation is shared with all predictive models that are tested within a bounded time period which is not fully inclusive of the entire product lifecycle.

II. Requirements naturally have a hierarchical structure. The model can predict changes at any single hierarchy level, but it cannot predict additions. The models will work with any level of requirement available, and assume that each requirement has the potential to change from a vision, specification or solution perspective. Requirement additions can be thought of as changes to a higher level requirement.

III. Since volatility is capped by the level of predicted requirements uncertainty, this means that all requirements uncertainty must be able to be predicted. There is some confidence that this is the case in the change domains included in this study.

IV. Changes may cause attributes of the requirements to change, but this reverse causal influence is not modelled. For example, a change may reduce the complexity of a requirement. This would affect phase-to-phase predictions, and the user will need to identify and change the requirements attributes manually before making the next set of predictions.

V. Use of COTS has an inhibiting effect on a change to an individual requirement, but a promoting effect on change to *other* requirements. This is not currently modelled. g

VI. There is a connection between domain uncertainties in that, for example, vision uncertainty will have an impact upon requirements specification uncertainty. This is not explicitly modelled, but is achieved through the notion of incoming work product quality.

7.5 Verification

Networks were firstly tested to ensure that there were no errors in equation definition and CPT formulation. This was achieved by running combinations of evidence for each net fragment and checking that the posterior probabilities were in line with the weights and direction (positive or negative) of causal influence. In addition to error checking to ensure that there are no erroneously specified expressions for CPT calculation, network verification is a process that involves ensuring that a developed system performs in accordance with the design initiatives [165]. In Bayesian Networks this includes sensitivity analysis and model walkthroughs. As described in section 7.3, Sensitivity analysis ensures that the relative influence of nodes upon volatility is faithful to the opinions of practitioners, and any available empirical evidence. Model walk-throughs utilise use cases which provide evidence for a subset of nodes. Posterior probabilities given by the networks are viewed to assessed for alignment with practitioner expectation [174].

7.5.1 Sensitivity Analysis

Two sets of sensitivity analysis are reported here, based upon the models in their final state. The first graphically compares the relative influence of the variables upon requirements uncertainty in each of the three change domains. The second lists in order of influential magnitude, each of the variables in the complete networks combining nodes affecting requirements uncertainty and the activities that discover change. The objective of this analysis is to assess whether the structure and CPT definition results in networks in which the nodes have relative causal strength in accordance with practitioner belief and prior empirical investigation. This process was iterative, and changes to CPT specification (though not structure) were made until agreement was reached. The results reported here are the final analyse when structure and CPT's were agreed.

Firstly, the nodes on the uncertainty side of each network were examined for relative influence upon domain uncertainty. In these analyses, any node that was a combination of

other nodes (latent nodes) were not examined, since these will not be collected in the process of network validation.

Figure 7.14 illustrates the relative influence of variables upon *vision* uncertainty. Business dependency has by far the most significant causal influence reflective of the findings from previous studies (see section Figure 6.3). The stakeholders understanding of the problem and the agreement of all stakeholders regarding the product direction (synergy stake agenda) are prominent causal contributors. Senior management is also considered more significant than requirements attributes such as novelty and complexity. The least important indicators for the likelihood of vision changes are that all stakeholders have been identified, are involved, and their knowledge of the process of software development. Business tools and procedures are equally insignificant, though business constraints are considered to be as influential as the quality of stakeholder communication.

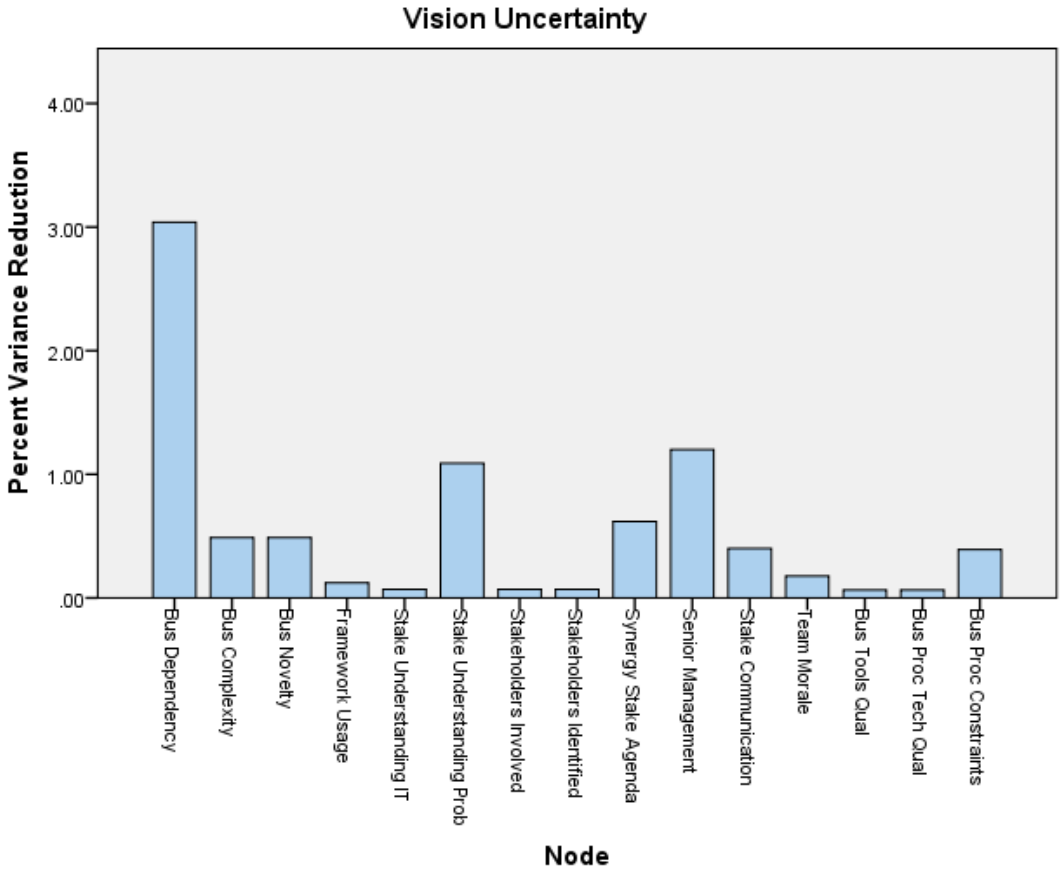

Figure 7.14 SENSITIVITY OF UNCERTAINTY VARIABLES IN THE DOMAIN OF VISION

These observations can be compared with causal influences in the domain of *requirement specification* as illustrated in Figure 7.15. Here we see that the incoming work product, that

is, the output from any previous definition of product vision, is considered to have the most influence upon *requirement specification* uncertainty. In this domain, tools techniques and constraints are of more significance than the knowledge and skill of the team. Although framework usage and business dependency are prominent, the technical attributes of a requirement are considered more influential to *requirement specification* changes than business complexity or novelty. This reflects the results from the previous study wherein the business complexity bore no correlation with volatility (section Figure 6.5), but disagrees with previous findings that technical complexity was negatively correlated with volatility in the domain of *requirement specification*. In this instance, practitioner intuition held out over empirical results. The requirement group complexity is the most significant of the requirement attributes, an attribute not even considered in the previous study. By comparison to the domain of *vision,* the stakeholder attributes are considered least influential to change.

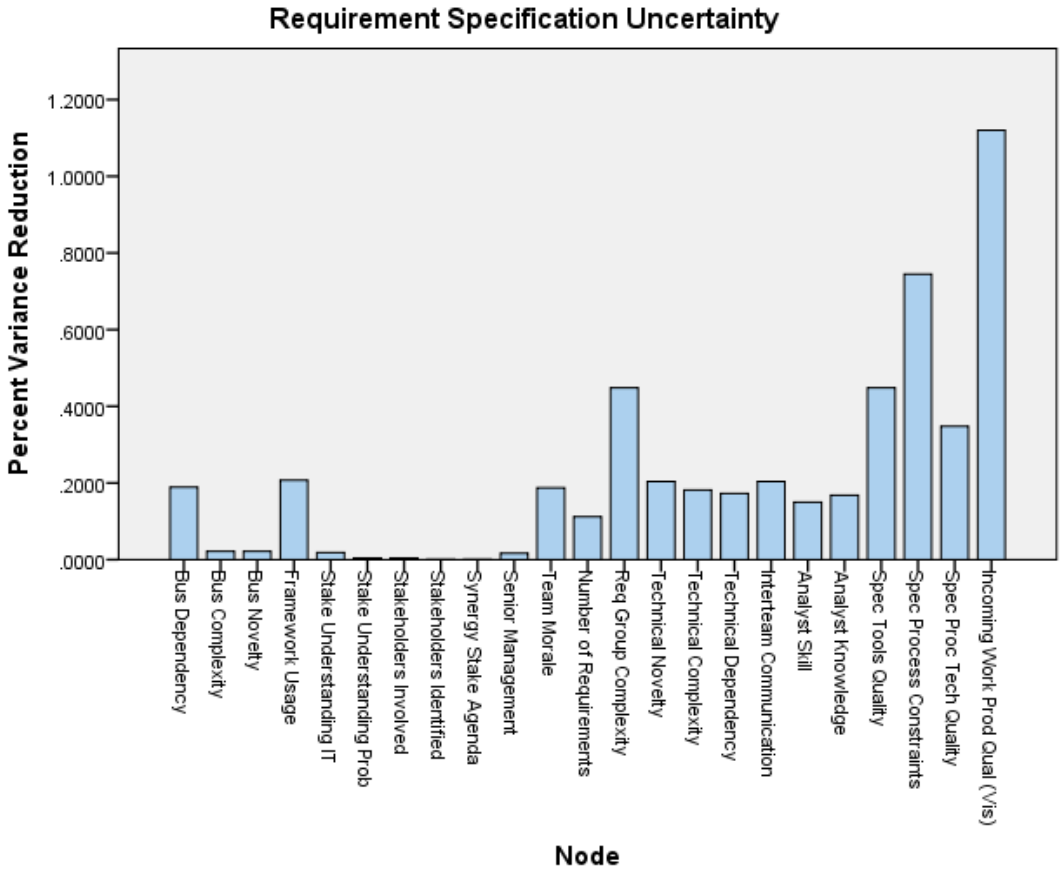

Figure 7.15 SENSITIVITY OF UNCERTAINTY VARIABLES IN THE DOMAIN OF REQUIREMENT SPECIFICATION

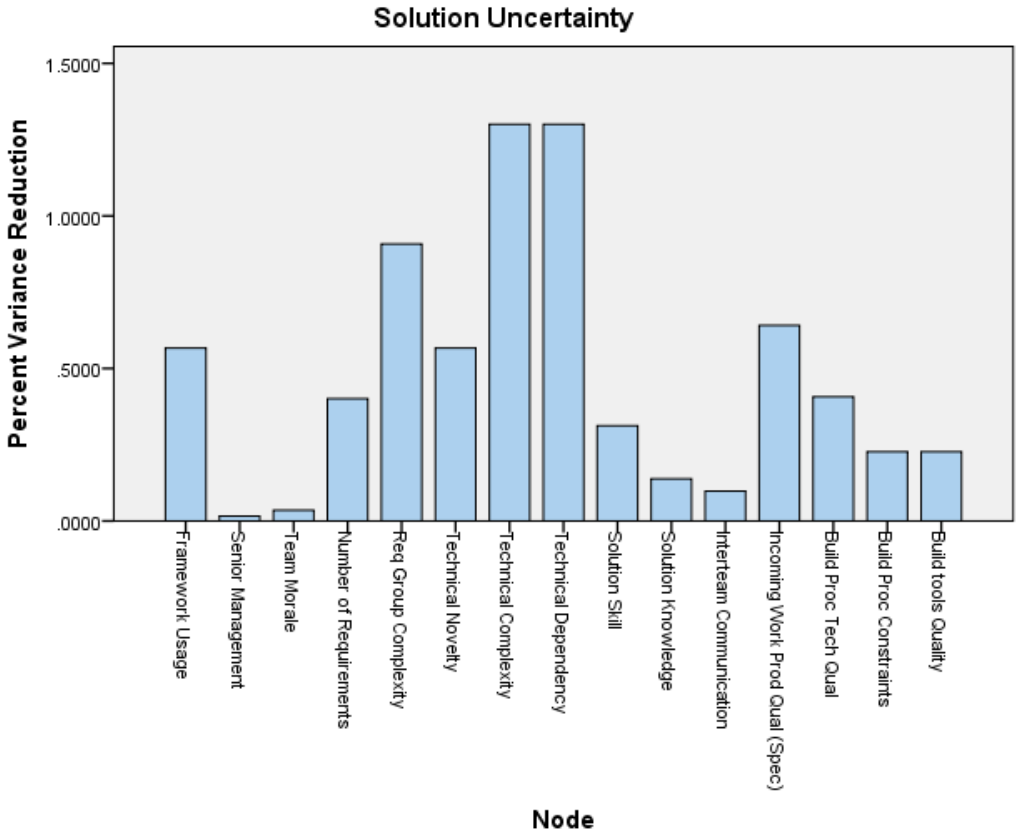

Figure 7.16 SENSITIVITY OF UNCERTAINTY VARIABLES IN THE DOMAIN OF SOLUTION

As can be seen in Figure 7.16, sensitivity analysis of the variables causal to *solution* uncertainty, reveals that the technical attributes of requirements, including the group to which they belong, are typically more influential to change than the tools and techniques or the skill of the team. This is not reflective of previous results when no correlation was discovered between technical novelty or complexity and volatility in the domain of solution. (see section 6.7.8 and 6.7.6 respectively). Nevertheless, it was considered that when all is known in the 'bigger picture' comprising more variables, these attributes are part of the causal picture. Similarly to the domain of requirements specification, framework usage is also considered important, alongside the incoming work product, which in this case is material related to the requirement specification. Interestingly, the influence of senior management is lower in both the domains of *solution* and *requirements specification*, than in the domain of *vision*.

The sensitivity analysis of the complete set of models for all three domains including activity sub-nets is presented in Table 7.9. There were too many variables to display this information graphically, so the domains variables are presented side by side for comparison and ordered by relative influence upon total volatility from most significant to least. The variables making up the uncertainty side of the networks are highlighted in blue, thus illustrating the interplay between variables on both sides of the networks. Nodes that have influence on both requirements uncertainty and change discovery are highlighted in blue and have an asterisk beside them. Volatility contribution nodes and 'on off' nodes have been omitted as well as latent nodes. The list of variables in each domain is separated into two, since the activity volatility nodes and the domain uncertainty nodes have most influence due to their proximal position to total volatility. These are listed first and all other nodes follow in the subsequent row.

It can be seen that the ordering of the relative influence of the activities is reflective of previous empirical results (section 5.6.6), such that, for example, in the domain of *vision*, the activity of 'customer test' has more of an influence upon total volatility, than 'system test'. The opposite scenario is the case for *solution* volatility.

While the same uncertainty variables are significant for overall volatility, the quality of the incoming work-product to the activities is also highly influential to overall volatility. In the domain of *vision,* the work-product used by the 'specification' activity features prominently. Similarly, the work-product used by the 'build' activity for the domain *of requirements specification*, and the work-product used by 'system test' in the domain of *solution*, are also highly significant to overall volatility. It can also be seen that in all domains, the skill of the teams involved in the activities that are most likely to discover change are more influential to change than the attributes of the requirements.

With the exception of the influence of process constraints in the domain of *vision*, in all other cases the verification and validation tools, techniques and constraints applicable to the relevant discovery activities are more influential than those affecting requirements uncertainty. In all domains, the least influential of the variables were those associated with activities felt not to be conducive to change discovery in that change domain.

Practitioners were satisfied that this network analysis met with their understanding of the causes and effects of software requirements change.

Table 7.9 SENSITIVITY ANALYSIS FOR VOLATILITY IN ALL DOMAINS

Vision	Requirements Specification	Solution
Volatility from custest Volatility from demo Volatility from spec Volatility from build Volatility from systest Vision uncertainty	Volatility from build Volatility from spec Volatility from systest Volatility from custest ReqSpec uncertainty Volatility from demo	Volatility from systest Volatility from build Volatility from spec Volatility from custest Solution uncertainty
Level on-going comms Bus dependency Analysis skill Analysis knowledge Incoming work prod (Vis) *Senior management Spec process constraints Custest skill Stake understanding prob Bus process constraints Bus complexity Bus novelty Synergy stakeholder agenda *Stake communication Custest knowledge Spec input comms quality *Team morale Framework usage Spec proc tech quality Spec tools quality Bus proc tech quality Bus tools quality Custest proc tech All stakeholders identified All stakeholders involved Stake understanding IT Custest tools quality Custest process constraints Solution knowledge Solution skill Systest knowledge Systest Skill Build input comms qual Systest process constraints Build proc constraints Build tools quality Build proc tech quality Systest proc tech quality Systest tools quality	*Vis input comms qual *Analysis skill Level ongoing comms Incoming work prod(spec) Solution skill *Analysis knowledge Solution knowledge *Spec process constraints Req group complexity Tech novelty *Spec proc tech qual *Spec tools quality *Interteam communication Custest knowledge Framework usage Number of requirements Tech dependency Build proc tech qual *Team morale Bus dependency Build proc constraints Build tools quality Tech complexity Custest proc tech qual Bus complexity Bus novelty Custest skill *Senior management Stake understanding prob Custest tools qual Custest process con Stake understanding IT All stakeholders involved All stakeholders identified Synergy stakeholder agenda Spec proc tech qual Build input comms Custest knowledge Systest proc tech qual Systest process qual Systest tools qual Systest skill qual	*Solution skill Systest skill Req group complexity *Solution knowledge Incoming work prod (build) Systest knowledge Number of requirements *Spec input comms qual Tech complexity Tech dependency Framework usage Tech novelty Systest proc tech qual Interteam communication Systest proc constraints Systest tools qual *Build proc tech qual Team morale *Build tools quality *Build proc constraints Senior management Analysis skill Analysis knowledge Spec process constraints Vis input comms qual Spec proc tech quality Spec input comms quality Spec tools quality Custest proc tech Custest tools quality Custest process Custest knowledge Custest skill

7.5.2 Walk-throughs

Three practitioners provided a series of 15 use cases upon which to determine a set of posterior probabilities, and were present during the process. Since volatility values represented total volatility over the entire project life-cycle, all activity nodes were set to 'on' (with the exception of one case, when no testing had yet taken place). Practitioners were unable to provide volatility values for domain volatility, and instead provided a single value for each requirement. While this means that direct comparison between predicted and actual volatility is not possible, nevertheless this process allowed examination of model behaviour. In some circumstances, when the predicted volatility was not in agreement with the actual, the volatility was entered as evidence and the posterior probabilities for non-evidence variables observed. Three of the 15 cases, one from each practitioner and each reflecting a typical use case scenario, are illustrated below. Since the objective of model walkthroughs are to verify that the models are behaving in accordance with design, predictive accuracy is of interest but not a priority concern. Networks cannot be assessed for accuracy due to the limited data provided.

Table 7.10 NETWORK WALK THROUGH. CASE 1

Basic template home page (empty frame to hold future webforms) **Volatility M**				
Bus Novelty	L	Vision Vol	**M** 40.9 %	**L** 28 %
Bus Dependency	L	Req Spec Vol	**M** 57.6%	**L** 25.9%
Bus Complexity	VL	Solution Vol	N/A	
Stake Involvement	M			
Synergy Agenda	M			
Analyst Skill	H			
Analyst Knowledge	H			

Table 7.10 describes the first use case from practitioner one. In this case, only 7 of the network variables were provided, and the volatility in both the *vision* and *requirements specification* domain agreed with the practitioner's observation of Medium. In the case of network predictions, there are two percentage values supplied for each domain which are probabilities of the two most likely outcomes. In this case, given the evidence in the left most table cell, there is a 40.9% likelihood of having a Medium level of volatility in the domain of *vision*, and a 28% chance that the volatility will be low. In this case, no *solution*

volatility was predicted, since there was insufficient evidence as only the variables 'analyst skill' and 'analyst experience' are present in the *solution* domain network. Although accurate in prediction, given that the network tends to predict a higher likelihood of average volatility when no evidence is entered, this result is not that informative. A second use case, described in the same manner, is presented in table Table 7.11.

Table 7.11 NETWORK WALK THROUGH. CASE 2

Create a Web service to except an XML file with rating information and make a call to Database **Volatility L**				
Bus Novelty	H	Vision Vol	**M** 44.2 %	**L** 21%
Bus Dep	L	Req Spec Vol	**M** 51.1%	**L** 33.8%
Bus Com	M	Solution Vol	**L**79%	**M** 12.1%
Framework usage	VL			
Tech Novelty	M			
Tech Depedency	L			
No of Requirements	M	*Posterior Probabilities following evidence of Vol* ***L***		
Analyst Skill	H	***Vision***		
Analyst Knowledge	H	Stakeholder understanding problem		VH
Spec Tools	H	Stakeholder understanding IT		VH
Spec process& technique	H	Stakeholder involvement		VH
Spec constraints	L	All Stakeholders identified		VH
Solution Skill	H	Senior management involvement		VH
Solution Knowledge	VH	Team morale		VH
System Test	Off	Stakeholder communication		VH
Customer Test	Off	***Requirement Specification***		
		Incoming work Product (Vis)		VH
		Req group complexity		VL

In the domains of *vision* and *requirement specification*, the most likely outcome is that volatility is Medium, though in both cases the second highest likelihood is a Low volatility. A Low volatility, in line with practitioner observation was predicted in the domain of *solution*, where, by comparison, there was a greater coverage of variable evidence within the network. To further understand the workings of the network, a volatility of Low was entered for both the domains of *vision* and *requirement specification*. The most significant changes to posterior probabilities following back propagation, are also summarised in Table 7.11.

The practitioner confirmed that in this case, the posterior probabilities for both the stakeholder concerns and the work product were in agreement with net likelihoods. However, the 'req group complexity' was more likely to be M rather than VL. Beginning again, and this time entering the posterior probabilities, (with the exception of req group complexity) as well as the original variable values as evidence yields the prediction of L for both *requirement specification* and *vision* volatility.

Table 7.12 NETWORK WALK THROUGH. CASE 3

Navigate the application, validate it meets the oracle - raising a step error for each failure **Volatility H**				
Bus Novelty	VH	Vision Vol	**H** 38.5%	**M** 30.8%
Bus Dependency	VH	Req Spec Vol	**M** 51.6%	**H** 36.8%
Bus Complexity	H	Solution Vol	**M** 31.1%	**H**26.8%
Tech Novelty	VH			
Tech Dependency	VH			
Tech Complexity	VH			
No of Requirements	VH	*Posterior Probabilities following evidence of Vol **H***		
Req Group Complexity	H	***Requirement Specification***		
Analyst Skill	M	Incoming work Product (Vis)		VL
Analyst Knowledge	H	Specification Constraints		VH
Build Constraints	VL	Specification Tools		L
Build Tools	H	Specification Process & Techniques		L
Build Process & Techniques	H	Stakeholder understanding problem		VL
Solution Skill	H	Stakeholder understanding IT		VL
Solution Knowledge	H	Stakeholder involvement		VL
Stake Communication	M	All Stakeholders identified		VL
All Stake identified	H	Stakeholder synergy of agenda		VL

In case 3, described in Table 7.12, a predicted volatility of High in the domain of *vision* agreed with actual volatility. However, the highest likelihood was that both *requirement specification* and *solution* volatility were Medium. In both cases, the second most likely outcome was a High volatility. Back propagation given evidence for High volatility in the domain of *requirements specification* resulted in significant updates to the beliefs regarding the quality of the incoming work product, process tools, techniques and constraints, and stakeholder concerns. The practitioner agreed with the reviewed beliefs of the variables

relating to the stakeholders, except that all stakeholders were involved. The process and techniques qualities were not in agreement with posterior probabilities, and were instead considered to be High. Similarly, specification process constraints were thought to be Low. Re-entering evidence, including the reviewed posterior probabilities, yielded a most likely outcome of High volatility in the domain of *requirements specification*.

All fifteen cases were processed in the same manner, and all practitioners were satisfied that the network was behaving in accordance with design.

7.6 Discussion

The Bayesian network models to predict requirements uncertainty and volatility were constructed with the aid of industrial practitioners. Due to the lack of available data, both model structure and calibration were determined using expert judgement. This is in line with most BN design in the SE environment [36]. Notable exceptions are Okutan [124], who derived structure from data in an open source repository, and Bibi [166], who used data from the available COCOMO dataset. However, these models have yet to be tested for accuracy in an industrial setting so it is not possible to compare between judgement-based and data-based models.

Although there is no comparable research that predicts requirements change using Bayesian networks, there are other studies aimed to predict software quality or effort. From the many factors in the domain of SE relating to process, product and people, there are alternative approaches to variable selection. The approach taken here is to assemble an initial set of potential causal variables relating to the product, project and environment identified in previous studies. In addition to product factors, other researchers similarly use facts pertaining to process factors such as testing regularity [125], and personnel [166] [127] to predict quality and effort respectively. Source code metrics are used when available [123] [126], but limits prediction to later in the developmental life-cycle. Though not yet validated, Bibi [128] bases Bayesian models to predict productivity upon the Rational Unified Process (RUP) with net fragments corresponding to phases of software development inception construction, elaboration, and transition.

Evidently both the notion of a fact as it pertains to an element of the software development environment, and the activities involved in producing software are both considered causal to

effort and quality. Radlinsky [122] uses 100 variables related to the developmental activities implementation, testing, and requirement specification in order to predict quality factors such as recoverability, effectiveness and fault tolerance etc. Just as the approach taken for calibration described in section 7.4.7, Radlinsky [122] also used Fentons [172] ranked nodes approach to determine CPT's from equations based on weighted means, and reported a similar ease of use.

The two-sided approach incorporating requirements uncertainty and discoverability is similar notionally to the approach taken by Fenton when predicting coding errors [37]. That Bayesian network includes variables for processes and staff quality, management effectiveness, and documentation. There is also a node referred to as 'errors in' which captures the notion that errors are created during a phase in the software development life-cycle. Each of three phases, specification, build, and test has a corresponding 'errors in' node. These are summed cumulatively as the project progresses. During the testing phase, there is a node depicting the probability that defects will be found and fixed. There is no distinction made between types of errors, which is a marked difference to the approach taken here which involving change domains that represent different types of requirement change. Indeed, given the disparity between causal influence noticed in the analysis of network sensitivity, this strengthens previous findings regarding the clear differentiation of the character of change sources as defined in the requirements change taxonomy.

Interestingly, Fenton [37] includes a network fragment consisting of three nodes to capture the probability of 'requirements creep', given that changing requirements poses a risk to product quality. In this way, the networks described here can be considered complementary to Fenton's efforts. Indeed since many of the variables are common (in idea if not exact replication) providing predictive capacity for both errors and volatility within one single network would provide practitioners with more valuable return from data collection investment. Currently, however the design approach taken differs in that Fenton's [37] networks walk through a set of phases consecutively, much like a waterfall process, whereas the approach taken here is agnostic to development methodology. Development activities can be turned on or off at any stage of prediction. Other differences include the inclusion of novelty and stakeholder concerns, and the separation between the business faculties of novelty, complexity and dependency. A significant advantage is the ability to provide an assessment of requirement uncertainty at the beginning of the software development life-cycle.

Model sensitivity analysis and walkthrough's confirmed confidence that the networks were behaving in the manner in which they were designed. However, they also revealed in insensitivity of networks to small amounts of data.

7.7 Conclusion and Implications for Research

A series of networks to predict requirements uncertainty and volatility were constructed during a series of 19 1-2 hour focus groups spanning a period of 16 months. Expert judgement was used to structure and calibrate the models, which were based upon results of previous research. Important design features unique to SE predictive models are:-

i. The two sided nature of the models which allow prediction of both requirement uncertainty and volatility (discovered change).
ii. The ability to turn development activities off and on, allowing models to be usable for any developmental methodology.
iii. The differentiation between the cause and effect mechanisms of different types of changes which correspond to the change domains in previous studies.

Although independent, the three domain models share a common core including factors pertaining to:-

i. The product itself, such as novelty
ii. The capacity of the team to understand the product, such as quality of communication.
iii. Process, technique and tools quality, including process constraints
iv. The quality of the incoming work product (if there is one)

However, there is a different causative focus in each domain. Changes coming from the domain of Vision focus upon factors such as synergy of stakeholder needs, while causative factors in the domain of Requirements engineering relate to all aspects of the task of specification including analyst skill that may give rise to inconsistency or ambiguity. Those in the domain of Solution place more significance upon the technical needs and complexity

of the product to be developed. The activity side of the networks contain subnets for each activity in the software development lifecycle: requirements specification, build&test, system test, customer test, and an activity called 'demonstration' which allows for opportunities to discover change, perhaps through presentation to stakeholders, which are not related to any particular development activity.

The results of sensitivity analysis of final models approved by industrial practitioners provide insight to the workings of the models, and thence the beliefs of experts regarding the causes and effects of software requirements change. In all domains, reflective of the previous study, the dependency of a requirement is considered greatly influential to requirements uncertainty. There are also notable differences between domains. Senior management and stakeholder concerns are most significant to *vision* uncertainty, whereas the incoming work product, team abilities and constraints are most influential in the domain of *requirement specification*. The technical attributes of requirements contribute the most to *solution* uncertainty. Although different activities are more influential to change, in all domains the skill of the involved team was very important. These results support the findings from previous studies that changes coming from the domains of *vision*, *requirements specification* and *solution* have different characteristics.

These reports of sensitivity were discussed with practitioners and compared favourably in the main with previous empirical findings. Model walkthroughs satisfied the verification objective, and therefore the Bayesian nets were deemed viable and reflective of reality. Thus we have answered the question '**RQ4** Can we model requirements change in such a way that we can predict change?'

Bearing in mind model assumptions, and the need for large numbers of variables in validation data, the next step is to test the validity of these models in an industrial setting in order to answer the question **'RQ5** Are model predictions more accurate than an available alternative?' This is addressed in the next chapter.

Chapter 8 Model Validation

Having structured and calibrated a series of Bayesian Networks to predict volatility, this chapter describes the validation of these models using industrial data, and an evaluation of model usability. Two tranches of validation are presented. The first examines the capability of the models to predict the total level of requirements volatility in four industrial projects and compares the results with that of expert estimation. In this exercise, three cases were considered: i) models that have been calibrated using only expert opinion, that is, those models arising from the construction process in the previous chapter; ii) those utilizing the learning capability of Bayesian Nets to parameterize the models from a project sharing industrial context; iii) models parameterized with data from a project with an alternative industrial context. The second validation tranche examines the potential of the models to predict changes to requirements for a project in-flight; prediction validation occurred at the end of each project phase, whereupon the models were recalibrated and used to predict changes during the subsequent project phase. Subsequent analysis of usability issues and limitations provides an evaluation of the benefits to industry of the models in their current state. This study answers the following research questions:-

RQ5 Are Bayesian models to predict software requirements change better than expert estimation

RQ6 How usable are these models?

Chapter Contributions to Academic Knowledge

a) Prediction of change-prone requirements using Bayesian networks with industrial data
b) Investigation of the efficacy of formal models for prediction by comparison to expert estimation
c) Comments on the importance of context in model calibration

8.1 Introduction

The usefulness of predictive models in SE has been debated [108][120], and therefore the need for validation and evaluation of such models in an industrial setting is essential in order to meaningfully assess model efficacy. Results of model prediction should be compared with available alternatives in order to determine their relative value. In practice, if requirements volatility is predicted at all, this is subjectively assessed by a member of the project team. However, research has shown that practitioner expectation does not correlate with actual measures of volatility [27]. In this study, the accuracy of practitioner estimation is compared with predictions made by the Bayesian models described in the previous chapter. Data from four industrial projects of varying sizes, and within two separate organisations supports this validation effort, and facilitates comparison of results in alternative empirical contexts.

Two tranches of validity assessment are undertaken:-

a) In the first tranche, the total (domain specific) volatility observed for each requirement during software development is compared with the estimate made by the practitioner, and also the prediction of requirements uncertainty generated by the Bayesian models described in the previous chapter. Further, the networks having been originally structured and calibrated using expert judgement, are re-calibrated using project data, and the accuracy of results of the re-calibrated models are compared with the both the original expert defined models, and practitioner estimation. Since two organisations each provided data for two projects, models were trained from project data with shared industrial context, and also from project data from an alternative context. Thus, a comparison of predictive accuracy is made between four predictive methods: models defined with expert judgement; practitioner estimation; models calibrated from a project with shared context; and models calibrated from a project with alternative context.

b) In the second tranche, the complete set of models, which includes the requirements uncertainty networks and the activity sub-nets, are tested for a single project in-flight. Phase by phase volatility data from one of the projects in the first validation tranche was compared with the predictions of each of the domain models. Predictions were made on a phase by phase basis by adjusting the values of the activity nodes. Between project phases models were re-trained and calibrated using the actual figures for volatility observed during the

previous phase. This facilitated an exploration of the potential of the models to learn during software development.

Having assessed the overall accuracy and learning potential of the networks, sensitivity analysis of networks trained with data from individual projects allow identification and comparison of the more influential factors on each project. Limitations to results applicability are considered before evaluating the networks' capability to support project management decision making during software development. This evaluation includes usability considerations, practical limitations, and model assumptions. Feedback from involved practitioners highlights some of the strengths and weaknesses of such formal predictions of requirements volatility.

The objectives of this study are:-

1. Compare model predictive accuracy with that of project manager estimation.
2. Investigate the performance of models trained with data from projects in similar and alternative empirical contexts.
3. Assess the learning potential of the networks to predict volatility phase by phase during software development.
4. Review and evaluate practical issues regarding model usability.

8.2 Industrial Context and Execution

Two industrial organisations, both in the private sector, provided validation data for this study. The first organisation also partnered the research presented in Chapters 4-7.

8.2.1 Organizations

The first organization employs 300 staff, has offices in England and Ireland, and delivers software systems to clients across both the public and private sectors. Most of their contracts involve a single customer and roughly 80% of these relate to governmental work. The second organization employs 380 staff, and has an office in Northern Ireland, It is a subsidiary of a larger organization employing around 60,000 staff, with offices in India and USA. Some software development is geographically distributed in some or all of these offices. All of the application development is to support the business of the larger organisation.

8.2.2 Projects

Data for four software development projects was collected. Table 8.1 describes the context of the projects for which data was collected. From each of the two organisations one project followed a waterfall process, and the other a hybrid process with both waterfall and more agile methods employed. Attempts were made in these cases to fully capture requirements at the beginning but delivery was iterative. The changes collected represent any change to the original requirement that occurred at any time from project inception to final delivery. Development of all projects took place between Feb 2008 and April 2013, and the number of requirements varied between 48 and 240. Staff were geographically dispersed in only one project (D). A stakeholder group is defined here as a business area such as finance, IT, customer etc. that has a vested interest in the software development and whose input is necessary to complete requirements gathering. The number of stakeholder groups involved varied between 3 and 5.

Table 8.1 VOLATILITY PREDICTION VALIDATION PROJECT CONTEXT

	Organisation 1		Organisation 2	
Project	**A**	**B**	**C**	**D**
Industrial Context	Government	Government	Internal	Insurance
Process	Waterfall, co-located	hybrid process, co-located	waterfall, co-located	hybrid process, geographically dispersed in three countries
Project start/end	Apr 2009 – Aug 2010	Feb 2008 – July2008	June 2012 – Mar 2013	July 2012 – Apr 2013
Number of Requirements	240	91	48	85
Total Cost (Days)	4222	358	369	2991
Number of Software Engineers	15	5	12	15
Number of Stakeholder groups	4	3	5	3

8.3 Data Collection

For all projects, data collection was done in three stages by the same project manager/analyst team. The volatility expert estimates were collected first, along-side project requirements details, including effort estimates, near the beginning of the project. The project manager estimated the expected severity of volatility for each requirement in each change domain on a 5 point Likert scale between Very low and Very high. Estimation by software project managers is frequently achieved by making comparative assessments [177] so in this case the measure of expected volatility was in relation to the other requirements. The change domains were familiar to both project manager and analyst since they had been two of the six participants in the first study to derive the change source taxonomy described in chapter 4. These data items were not passed to the researcher until the third stage was complete.

The second stage involved the collection of change data. The change domains of *market, organisation, requirement specification*, and *solution* were attributed to each change in addition to the involved requirement id(s). In all cases these were collected as they occurred by the project manager, alongside the change data used by the company in their change control databases which included effort estimates for each change. The change data for Project A was received in October 2010 and used for analysis in the studies presented in chapter 5 and chapter 6. (See section 5.2 for details of data specification, review process, data collection protocol, and change domain taxonomy evaluation).

The third stage entailed the collection of all network variables. Since the networks needed to be constructed prior to specification of the variables, and this was not completed until May 2012, this data was collected by the project manager or analyst at various stages during each projects' lifecycle. For example, it was collected well after the completion of project A and B, and near the beginning of projects C and D. The industrial participants used the model topology, and a list of agreed definitions of the variables to determine the data needs for each requirement. The variable definitions had been agreed during model structuring and these were augmented by the practitioners with heuristics to aid data collection. A list of variables, their definitions, and agreed scaling heuristics are provided for interest in Appendix 2. These lists and network illustrations were the basis upon which spread-sheets were designed to collect data.

Apart from the review process which took place for project A data, there was no involvement of the researcher in the data collection process, the most onerous of which was

stage three which was reported to have taken between 8 and 38 hours, with the longest being project D. Requirement data from Project A were available for research purposes in Jan 2011, and have previously been analysed to examine correlation between requirements attributes and volatility (see Chapter 6). The remaining network variable data for Project A (pertaining to project, process and people qualities) were added to the requirement in June 2012. Phase by phase data items were collected for each variable in the activity subnets for project A, to facilitate the second tranche of model testing, volatility prediction during software development.

For projects B-D, data arrived with the researcher on two excel spread-sheets per project between April and May 2013. The first contained the requirement/project details from stage three and the expert estimations form stage one. The second contained the change data and related requirements. Once the data was received, calculations of volatility for each requirement were made according to the formula in section 7.2. Where more than one requirement was involved in a change, the change cost was divided equally amongst the involved requirements.

8.4 Test Plans

A distinction can be made between model validation, wherein models are tested with industrial data within a research laboratory to ensure that they perform as they were designed, and field testing which refers to the testing of models under actual conditions of their use [42]. The testing in this study is of the first kind; all tests were performed by a researcher, having acquired the data through the process described in section 8.3. There are two broad tranches of model validation, each of which has a number of subtests:-

8.4.1 Comparison of Volatility Prediction between Models and Project Management Estimations.

These tests examine the ability of the Bayesian models to identify change prone requirements, by comparison to project manager estimation. All four projects were involved in the models used were the Uncertainty nets described in Section 7.4.3, 7.4.4 and 7.4.5, for the domains of *vision*, *requirement specification*, and *solution* respectively. The activity sub-nets were not used in this case, since we are examining the capability for the models to predict overall volatility during development. This carries the assumption that all volatility is

discovered, leaving zero residual uncertainty upon development completion as discussed in section 7.4.8. Therefore, total volatility is equivalent to requirement uncertainty.

Three model scenarios are considered and compared with project manager estimation: those having the CPT's defined by expert judgment as described in section 7.4.7; those having been calibrated from a project with a shared industrial context; those calibrated from a project with alternative industrial context. The arrows in Figure 8.1 illustrate model training. Projects sharing industrial context are those that were undertaken by the same organisation, and training is illustrated by black arrows. Models were trained using data from both projects that were undertaken by the alternative organisation, and this training is illustrated by the blue arrow. The means by which models are trained is discussed in section 8.4.4.

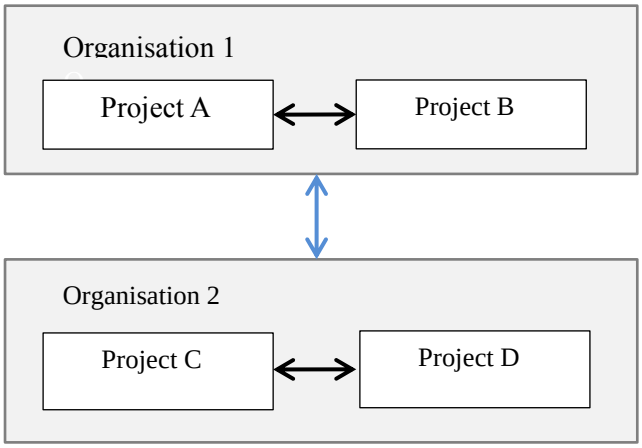

Figure 8.1 MODEL TRAINING - SHARED CONTEXT AND ALTERNATIVE CONTEXT

8.4.2 Requirements Uncertainty Discretisation.

The models used in the first validation tranche were designed to predict levels of uncertainty, whose node has discrete states from Very low to Very high expressed on an ordinal scale that can be mapped unto an underlying continuous interval between 0-1. (see section 7.3.4 for an explanation of node states). Since the project volatility data was expressed as a ratio of cost, this needed to be discretized to allow comparison. There was no data available which would allow the calculation of generic interval points for project volatility, and it had already been observed in a previous study that volatility ranges vary considerably between change domains. Therefore it was impossible to take the approach taken by some such as Shultz [121] to assign percentage intervals such as 0-10%, 10-20% etc. since in some cases,

that would result in a significant proportion of volatility falling into a single interval. Instead, intervals were calculated particular to each project. Firstly, a Very Low volatility was assigned the percentage interval [0,2] to account for the fact that some requirements have zero volatility. Then the 25th, 50th and 75th percentile points were used for interval boundaries, giving a total of 4 interval boundaries and 5 states. This means that the resulting scale for volatility is project-relative, which matches that of the project manager estimation. The resulting interval boundaries are shown in Table 8.2. For interest, the volatility range is included, and the percentage of requirements that had relevant changes is presented in brackets. The intervals for discretisation are the four coma-separated numbers in each table cell. As can be seen, there is great variety in the range of volatility in the different projects, particularly in the domains of *requirements specification* and *solution*.

Table 8.2 PROJECT VOLATILITY RANGES AND DISCRETISATION.

	A	B	C	D
Vision	0-600 (31%) 2, 12, 56, 279	0-400 (23%) 2,83, 125,216	0-200 (37.5%) 2, 22, 45, 114	0-300 (29.4%) 2, 18, 32, 92
Req Spec	0-501(75%) 2, 33, 88, 239	0-404 (53%) 2, 51, 117, 223	0-100 (52%) 2, 27, 44, 81	0-300 (51%) 2, 19, 35, 71
Solution	0-791 (23%) 0.5, 3.7, 11, 25	0-400 (59%) 2, 59, 139, 260	0-50 (42%) 2, 11, 21, 48	0-100 (21%) 2, 13, 27, 45

To further illustrate project context, Figure 8.2 shows the overall project volatility for all four projects. The three axes measure total volatility in each of the domains of *vision, requirement specification*, and *solution*. Interestingly, three of the projects have a similar shape, despite the alternative context, with significantly lower levels of *solution* volatility than either *vision* or *requirements specification*. In all three projects, A, C and D, the changes coming from the domain of *requirements specification* represented the most significant contribution to requirement volatility. Project B was subject to the highest level of volatility in all three change domains, but is remarkable given the considerably higher levels of *solution* volatility.

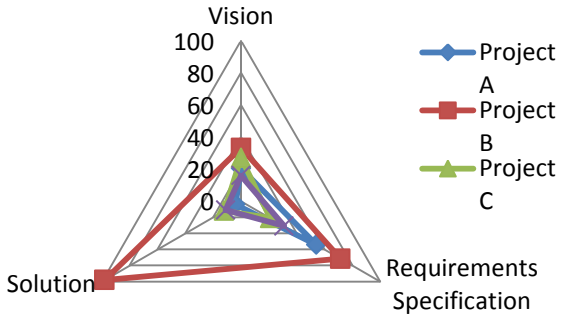

Figure 8.2 PROJECT VOLATILITY COMPARISON

8.4.3 Phase by Phase Prediction of Volatility with Interphase Learning

The second tranche of validation involved only project A, and examined the capability of the models to predict volatility during the course of software development. Since development followed a waterfall process which contained 4 phases, it was possible to make volatility predictions for each phase.

The process of phase to phase validation and re-training is illustrated in Figure 8.3. which can be considered to represent a single change domain. It should be noted that this is not a Bayesian network, though the arrows can be understood to represent causal relationships; detailed descriptions of the Uncertainty networks can be found in section 7.4.3, 7.4.4, and 7.4.5, and the Activity subnets in section 7.4.6. An overall prediction for requirements uncertainty was yielded from the Requirements Uncertainty Bayesian networks for each requirement in each domain (Initial_Uncertainty). The domain Uncertainty networks were initially trained from a project with shared context (project B).

The activity networks for Specification, Build, System test, Customer test, and Demonstration were then applied and levels of volatility predicted for the each activity for each requirement in first phase. As described in section 7.4.1, each activity had an associated maximum contribution to volatility, that is, each activity was considered to be able to uncover a certain percentage of overall uncertainty. The volatilities for each activity were summed to give a total volatility for each requirement in phase 1 (Volatility_dicovered_P1). This could not exceed the overall requirements uncertainty, as described in section 7.4.1.

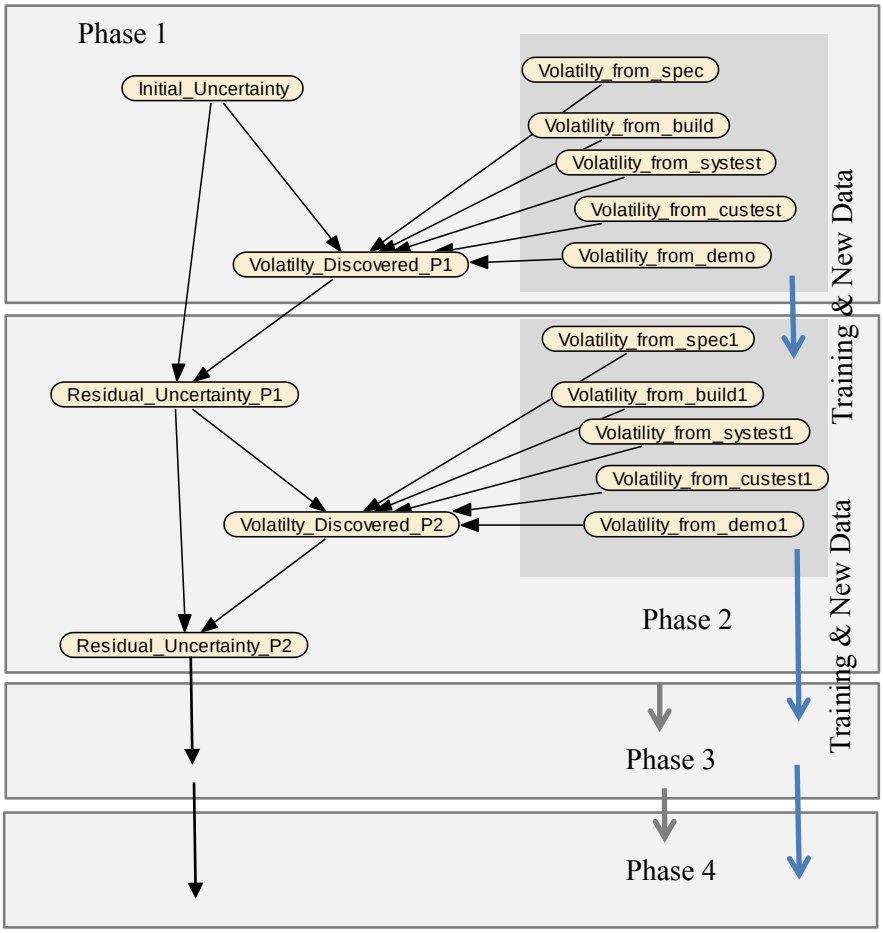

Figure 8.3 PHASE TO PHASE VOLATILITY PREDICTION

The predicted volatility was subtracted from the overall uncertainty (Initial_Uncertainty), to give a residual uncertainty that was used in the next phase (Residual_Uncertainty_P1). Since volatility is predicted on an ordinal scale from Very low to Very High, the calculation of residual volatility was not exact, and instead the underlying continuous scale was appropriated in order to make the calculation (see section 7.4.7). The activity sub-nets were then trained using actual volatility figures, and the trained models used in the next phase. New data for the activity sub-nets was used, since project factors such as process and team attributes may change as development proceeds. This phase by phase activity data had been provided alongside all other network variables as described in section 8.3.

Since not all activities were 'on' in each phase, only those that had been 'on' were trained for the next phase. This process was replicated for phase 2 and 3. Table 8.3 shows which activities were 'on in each phase of this waterfall-based project. For further information regarding these activities, a description can be found in Table 7.5, and details of the tasks involved in each phase can be found in Table 5.2.

Table 8.3 ACTIVITIES ON DURING PHASES.

	Phase			
	1	2	3	4
Specification	On	On		
Build		On		
System Test			On	
Customer Test	On	On	On	On
Demonstration	On	On	On	On

Levels of actual volatility were extracted from project data (see sections 5.4 and 6.5. for details of data specification). In this case, discretisation was affected by ranking all requirements by volatility, assigning any with a volatility <=2 a volatility ranking of V Low, and then dividing the remaining requirements into four equal divisions, each assigned a Likert scale designation from Low to V High. This gave five levels of volatility.

8.4.4 Model Learning

Model learning in all cases was achieved using an Expectation-Maximization (EM) algorithm with project data. Briefly, EM learning repeatedly takes a Bayesian network and uses it to find a better one by doing an expectation (E) step followed by a maximization (M) step. In the expectation step, it uses regular Bayesian network inference with the existing Bayesian network to compute the expected value of all the missing data, (in this case latent grouping variables) and then the maximization step finds the maximum likelihood Bayesian network given the now extended data (i.e. original data plus expected value of missing data). This results in a new set of CPT tables with values that best accommodate the data. For a full description of the EM algorithm and other learning techniques, refer to [174]. The CPT's were not removed prior to learning, since on doing so, there were many rows within CPTs with a normalized distribution, that is, an equal probability for each value of the child node, given the combination of values for the parents. For example, normalized rows in the example CPT shown in Table 3.4, would have 20% for all values of 'Requirement uncertainty', meaning that future use of the model would predict an equal chance of any outcome from Very low to Very high regardless of the values of specification accuracy and review quality. This implies that project data was not sufficiently varied to cover all possible

combinations of data values for some nodes. The expert defined CPT's provided a sample case for each combination of variable values not found in the data.

8.4.5 Results Reporting

The results of each test are presented by three error levels:-

E1. The general prediction error (Error). This illustrates the percentage of time the model (or expert) provided a wrong prediction.

E2. Non recognition of high uncertainty (High Error). This measures the percentage of times the model/expert did not accurately identify a change-prone requirement ie. those that have High or V High levels of volatility.

E3. False prediction of high uncertainty (False High). This measures the percentage of times a requirement was erroneously identified as being highly uncertain (High or Very High levels of volatility).

Though the models predict outcomes with varying levels of probability, it is the prediction with the highest level of volatility which is used for error calculation. This means that the analysis presents the most pessimistic view of model accuracy. For example, should a requirement have a V High level of volatility, and the Bayesian model predicted a 55% High level of volatility and a 45% chance of a V High level of volatility, this would still be calculated as an error. In practice though, a prediction of 45% chance of V High volatility would be sufficient to alert project managers of the high risk of a V high level of volatility.

Data for each scenario for all three errors were tested for normality using the Shapiro-Wilk test, and homogeneity of variance between groups using the Levene's test. Though Error E1 and Error E2 satisfied the assumption of normality, the Levene's test was significant in all cases, therefore a one-tailed Mann-Whitney test was used to compare prediction accuracy produced by the models with those generated by the project manager. This statistical test is not based upon the assumption of homogeneity of variance, and is based upon score ranking rather than mean values. Examining the one-tailed significance allows us to test for the assumption that one set of errors is lower, and therefore more accurate, than another, and not just different. See [170] for a full description of these tests.

These results were presented to industrial practitioners during a final meeting consisting of 2 practitioners and a researcher. This meeting lasted 2 hours.

8.5 Results

The results of the first tranche of model validation is firstly presented on a project by project basis, with error levels illustrated as percentages in tables and graphs for ease of comparison between prediction methods. Since there are 5 intervals, a series of random predictions would tend towards 20% requirements in each interval, yielding a random error rate of 80% for General error, 60% for High Error (since two categories, V High and High were combined), and 40% for False High (since in this three categories - V low, Low, or Med – are considered as accurate). As a generalization, error percentages below these numbers can be considered better than random guessing.

The error levels for all projects are then collectively examined for all projects to assess if there is any correlation between predictive method accuracy, that is, if any particular method was consistently more accurate than others. Since project managers estimated volatility consecutively on a project by project basis, the error rates produced by estimation are illustrated as they change over time. Thus, some comment can be made regarding the capability of the project managers to improve volatility assessment through experience.

The results of the second tranche of testing are presented in tabular form and also graphs showing trends in error levels as the project progresses through each project phase. Sensitivity analysis, which compares the relative strength of variable influence upon requirements uncertainty, is presented for networks trained by individual projects. This provides some insight into the qualities that are affecting volatility on the different projects.

8.5.1 Model Predictions of Volatility v's Project Management Estimations.

Percentage error rates for predictions of requirement uncertainty for all four projects can be found in the tables to follow. These indicate the percentages of times the prediction of requirement uncertainty did not match with the level of volatility observed during project development. Results for all three model scenarios (original CPT defined, learned within context, learned from alternative context) are presented alongside those for expert estimation for ease of comparison.

8.5.1.1 Project A

Results for project A are presented in Table 8.4, and illustrated graphically in Figure 8.4. Overall, the scenario where predictions are closest to reality are those of the models trained from project data with a shared context. Both the overall Error, and the High Error are lower than all other scenarios in all three change domains. A comparison of the overall Error indicates that the next best predictions came from expert estimation. However, when we consider the High Error, we can see that in all cases, both sets of models performed better than expert estimation. This would imply that in this case, the project manager was optimistic concerning the expected levels of volatility.

Table 8.4 PREDICTION ERROR RATES FOR PROJECT A

	Model with Expert Defined CPT's			Expert Estimation			Model learned – Shared Context			Model learned – Alternative Context		
	E1	**E2**	**E3**	**E1**	**E2**	**E3**	**E1**	**E2**	**E3**	**E1**	**E2**	**E3**
Vis	58.7	42.8	6.6	23.7	100	0	18.8	35.7	6.6	36.7	57.7	2.8
Rec	53.3	52.8	0	50.4	69.6	1.3	37.0	40.4	2.6	60.4	60	42.7
Sol	53.3	56.7	0	39.2	86.7	2.8	19.2	26.6	1.4	44.2	5	0.9

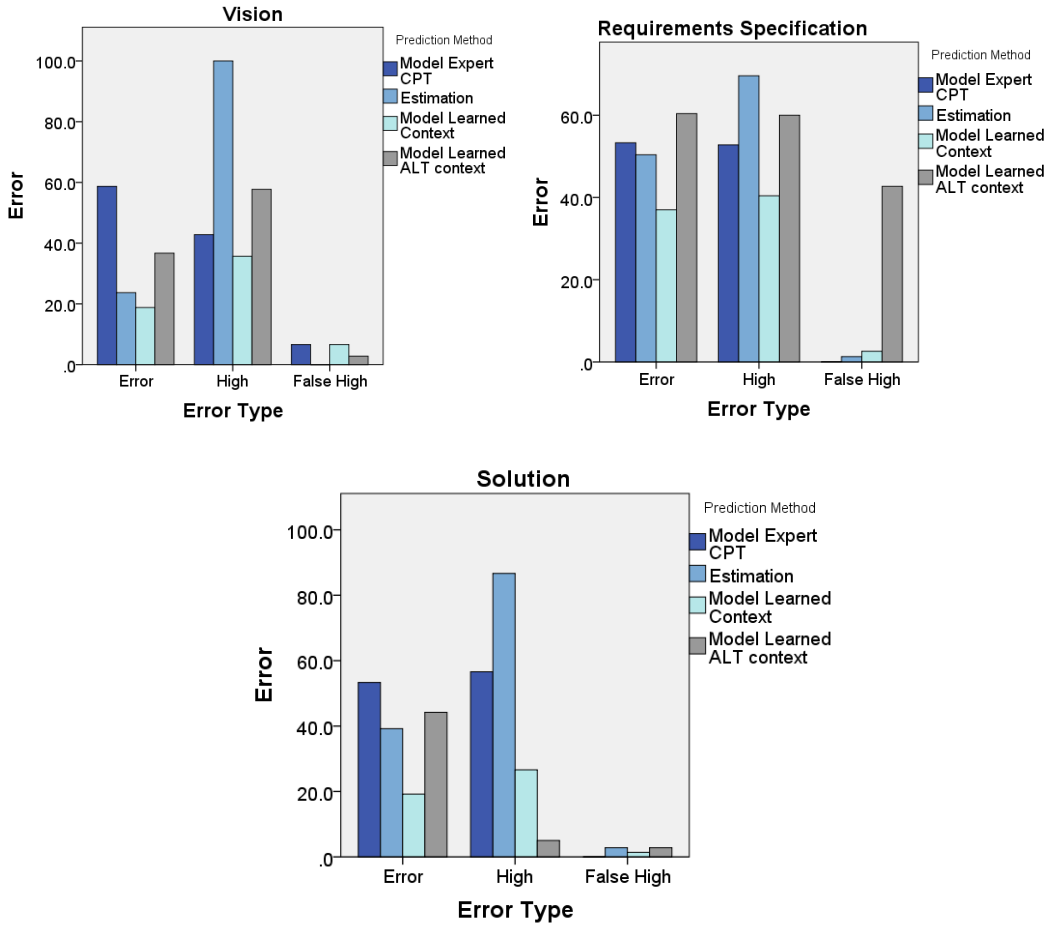

Figure 8.4 PREDICTION ERROR RATES FOR PROJECT A IN EACH CHANGE DOMAIN.

This can be further explored by examination of expert estimations presented in Table 8.5, where it can be seen that all expert estimations of vision uncertainty were V low or Low, and the overall Error rate was therefore low due to the correct prediction of low levels of volatility, rather than correct anticipation of change-prone requirements. Interestingly, the False High error was very low for both the expert estimation, and the models calibrated by expert judgment. In both scenarios, on only a few occasions were requirements erroneously predicted to be highly change-prone. This may mean that subjective estimates for project qualities captured by the project manager and used in the prediction of uncertainty were similarly optimistic. Model learning with shared context also produces low levels of False High predictions, but models learned from alternative context have a much higher level of False High error in the domain of *requirements specification*. Models calibrated from data with alternative context had the poorest overall performance. Somewhat surprisingly, there is some similarity of overall Error rates from the different change domain models, which is especially apparent in Error percentages in the original expert defined CPT models. There is

also no clear pattern of High Error rates observed in the domains; sometimes prediction is more accurate for one domain than others, but this is not consistent.

Table 8.5 EXPERT ESTIMATION OF VISION UNCERTAINTY FOR PROJECT A

Predicted					***Total***	***Actual***
V. Low	**Low**	**Med**	**High**	**V High**		
182	2	0	0	0	184	**V. Low**
13	1	0	0	0	14	**Low**
12	2	0	0	0	14	**Med**
13	1	0	0	0	14	**High**
13	1	0	0	0	14	**V High**
233	7	0	0	0	240	

8.5.1.2 Project B

The results for the predictions of requirement uncertainty for project B are presented in Table 8.6 and Figure 8.5. Once again, in general, the most accurate predictions come from models that have been trained with data from a project with shared context. The False High rate for the domain of *requirement specification* are slightly higher than that of models with expert defined CPT's and those of expert estimation, but the other two errors percentages are generally lower than all other predictive scenarios. There is a less defined difference between predictions from the models with Expert defined CPT's than those of expert estimation, with similar levels of High Error. Only the overall Error for *vision* has a marked difference, and again there is considerably less overall Error made by the expert. In this case, it is difficult to assess which of the model scenarios performed worst with most error rates between 40% and 65%. Even the results of the models trained from shared context are not as good as those of project A. Once again, False High error rates are very low for all scenarios, with the exception of the domain of *requirements specification* in the models trained from data in an alternative context. Project managers' predictions are better than those made in project A, though the High Error remains between 48% and 58% by comparison to a High Error in learned within context models of between 33% and 50%.

Table 8.6 PREDICTION ERROR RATES FOR PROJECT B

	Model with Expert Defined CPT's			Expert Estimation			Model learned – Shared Context			Model learned – Alternative Context		
	E1	**E2**	**E3**	**E1**	**E2**	**E3**	**E1**	**E2**	**E3**	**E1**	**E2**	**E3**
Vis	61.6	50	6.2	22	58	0	36.3	50	1.2	40.	80	0
Rec	61.6	66.7	4.5	59.3	54.1	5.9	34.1	29.2	13	56.	64.1	40.2
Sol	50.6	44	0	51.7	48.1	10.9	29.7	33.3	0	58.2	67.8	0

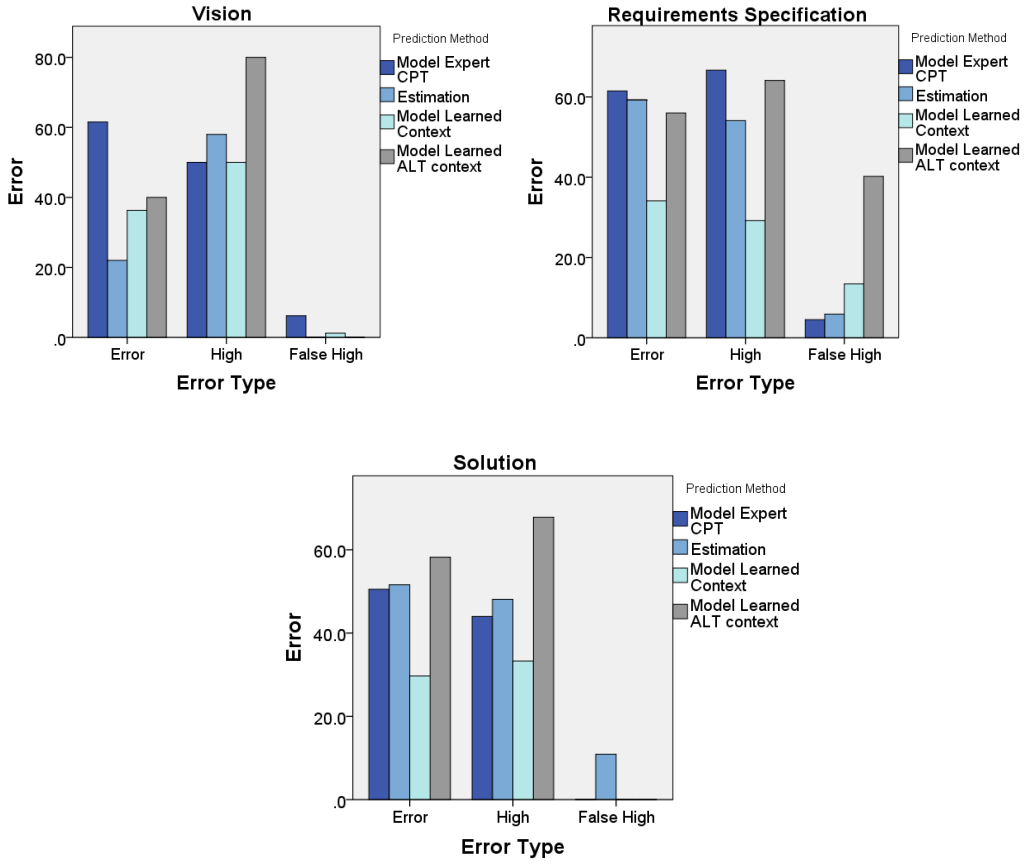

Figure 8.5 PREDICTION ERROR RATES FOR PROJECT B IN EACH CHANGE DOMAIN.

8.5.1.3 Project C

Table 8.7 and Figure 8.6 contain the error rates for project C. In general, the model learned from shared context performs best overall, though the False High rates for the domain of *requirement specification* (28.6%) are considerably higher than all other scenarios, meaning that although change-prone requirements are being identified, the model is pessimistic for over a quarter of the projects requirements. This result was also observed for project B, though less marked, and may imply that the project data from which the CPT's were learned had quality factors that, having been previously optimistic are now producing refined predictions based upon a previously optimistic outlook. For example, if an optimistic value for analyst skill of V High had resulted in a high rate of volatility, a trained model will reason that V high quality analyst skill may result in High volatility in the new project. However, it may also be the case that the requirements continue to be uncertain, but that changes have not been discovered. Like project A and B, the levels of False High resulting from expert estimation are very low, while the High Error remains fairly constant at between 38% and 44%. Interestingly, in this project High Error rates vary significantly in the models with expert defined CPT's, with *vision* high error being 0% and solution high error being 70%. This may imply that more accurate subjective estimates were made for project quality factors related to vision domain change, or that an unaccounted for factor was affecting the level of *solution* volatility. The *solution* High Error rate was improved with training both from alternative context and shared context projects.

Table 8.7 PREDICTION ERROR RATES FOR PROJECT C

	Model with Expert Defined CPT's			Expert Estimation			Model learned – Shared Context			Model learned – Alternative Context		
	E1	**E2**	**E3**	**E1**	**E2**	**E3**	**E1**	**E2**	**E3**	**E1**	**E2**	**E3**
Vis	56.2	0	5.1	45.8	44.4	2	20.8	0	0	37.5	0	5.1
Rec	56.2	30.7	0	70.1	38	2	31.2	0	28	47.9	30.8	8.6
Sol	58.3	70	18	56	40	15.8	22.9	50	7.9	56.3	50	26.3

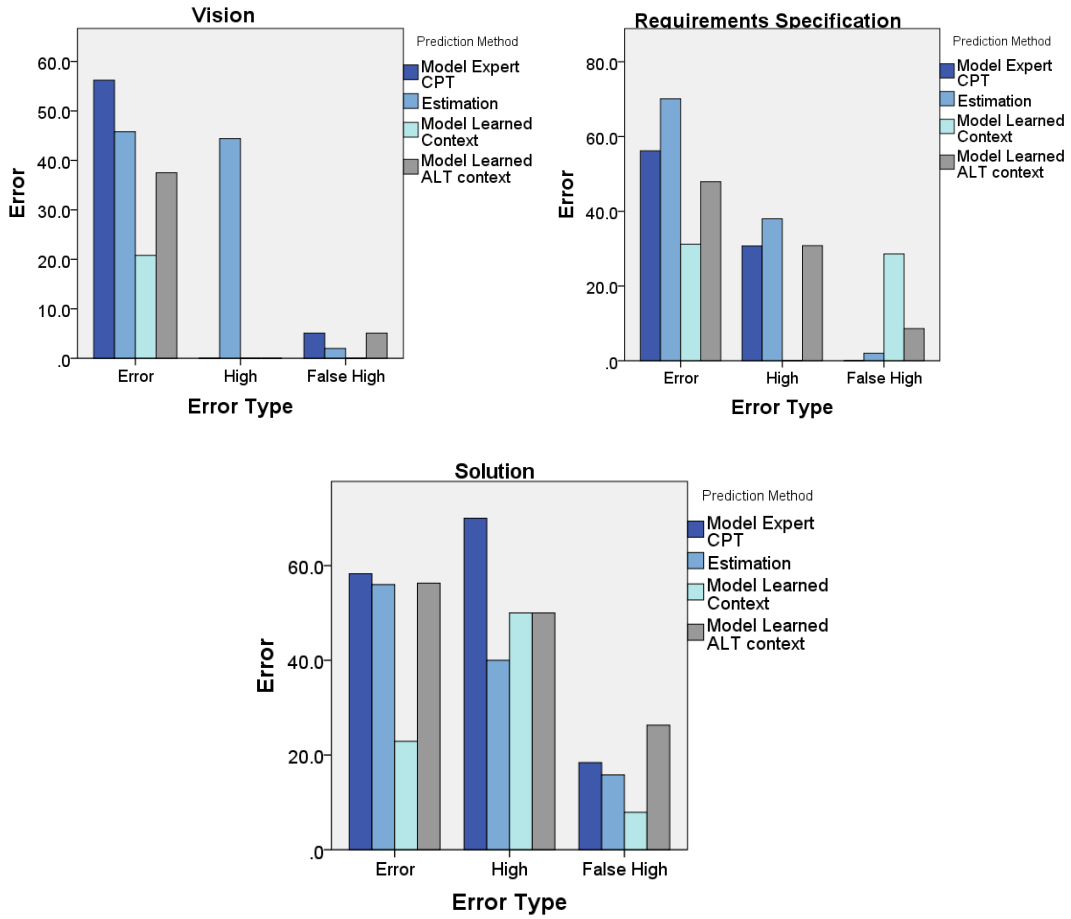

Figure 8.6 PREDICTION ERROR RATES FOR PROJECT C IN EACH CHANGE DOMAIN.

8.5.1.4 Project D

The results from project D are presented in Table 8.8. and Figure 8.7. Reflecting the error rates in projects A, B, and C, the best overall predictions of requirements uncertainty in all three change domains come from models that have been trained from data in a project with shared context. On this occasion, however, the High Error rate for the domains of *requirements specification* and *vision* were disappointing relative to those observed in the other three projects. Indeed we observe that in project C, these rates are both 0 while the False High rates are much higher than in Project D. Since project D's learned predictions were based upon project C data, and vice versa, this would support the notion that optimistic (in the case of project D), or pessimistic (in the case of project C) evaluations of quality factors were affecting predictions. With the exception of *solution* uncertainty, expert estimation was much better than both the original models and those learned from alternative context.

Table 8.8 PREDICTION ERROR RATES FOR PROJECT D

	Model with Expert Defined CPT's			Expert Estimation			Model learned – Shared Context			Model learned – Alternative Context		
	E1	**E2**	**E3**	**E1**	**E2**	**E3**	**E1**	**E2**	**E3**	**E1**	**E2**	**E3**
Vis	57.7	58.3	0	22	50	0	22.3	33.3	5.5	55.3	58.3	0
Rec	57.7	68.1	0	42	40.9	2	31.8	40.9	2.5	62.4	27.2	55.6
Sol	57.7	11	0	36.5	44.4	2.6	21.2	11.1	0	36.5	88.9	10.5

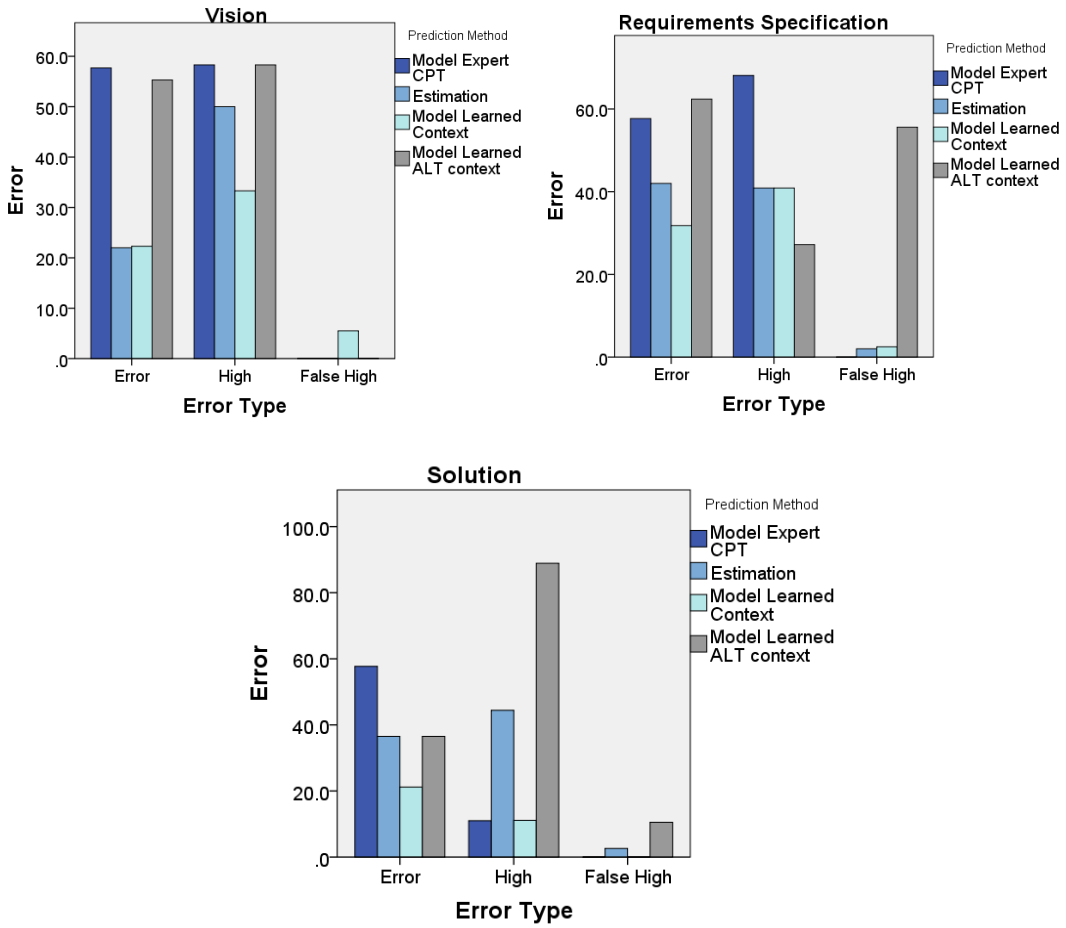

Figure 8.7 PREDICTION ERROR RATES FOR PROJECT C IN EACH CHANGE DOMAIN.

8.5.2 Comparison of Methods

In order to make clearer comparison between scenarios, the range and median values of error rates for models and expert estimation are shown in Table 8.9. and a comparison of median error rates illustrated in Figure 8.8. Error values include all three domains in all four projects, so therefore there are 12 values for each prediction method.

Table 8.9 RANGE AND AVERAGE ERROR RATES

	Expert Defined CPT	Estimated	Shared Context	Alternative Context
Error	50.6-61.5 (57.6)	22-70.1 (43.9)	18.8-37 (26.3)	36.5-62.4 (51.6)
High Error	0-70 (47).3	38-100 (59)	0-50 (33.3)	0-88.9 (58)
False High	0-18.4 (0)	0-15.8 (2)	0-28.6 (2.5)	0-55.6 (6.9)

It can be seen that the average general Error and High Error rates are much lower from models that have been learned from shared industrial context. However, expert estimation is often as good, if not better than both the original models, and those calibrated from an alternative context. Further exploration by use of the Mann-Whitney test, indicates that the level of Error made by the project manager in estimating volatility is significantly lower than that of the model with CPT's defined by expert judgment ($U = 26$, $p = 0.007$). However the models trained from a project sharing context performed significantly better than expert estimation in terms of general prediction accuracy ($U = 23$, $p=0.004$), and identification of uncertain requirements ($U= 17.5$, $p=0.001$). There was no significant difference between expert estimation and models trained with alternative context using any measurement of prediction error. Models built using expert defined CPT's, as well as those trained with data from an alternative context, identify uncertain requirements no more often than a project manager. There was also no significant difference between the efficacy of any model of prediction when considering false identification of uncertain requirements (False High).

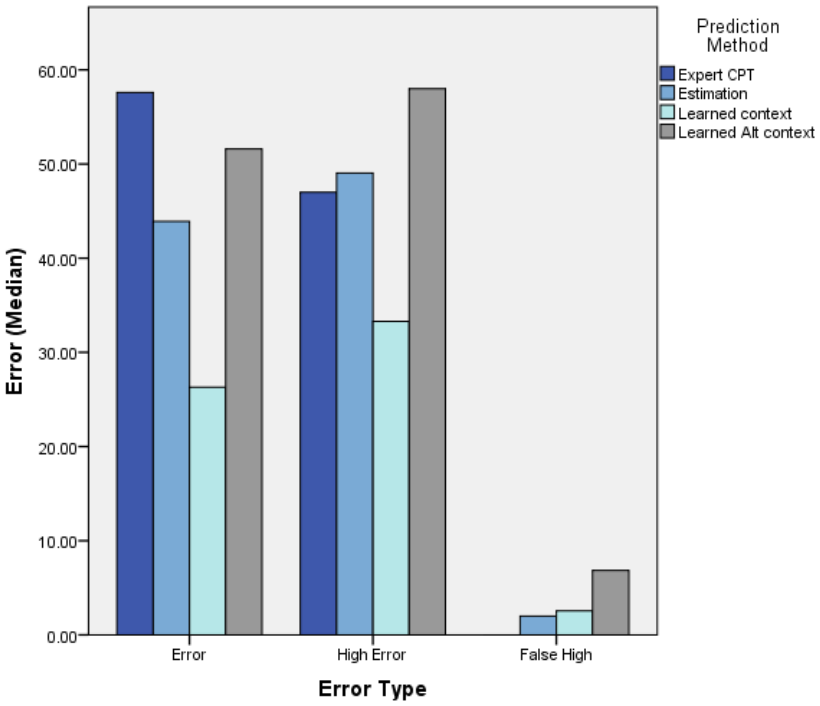

Figure 8.8 COMPARISON OF MEDIAN ERROR RATES

For completeness it is worth considering the learning capability of the project manager. Since the data was collected consecutively, and volatility observed in each project before commencement of the next, it is possible to examine expert estimations as a sequence. Figure 8.9 illustrates the trend in estimation error levels during the four projects. Each graph shows one of the three error levels for all three domains. It could be argued that the most important error is that of non-identification of highly uncertain requirements, since these requirements pose most risk to timely and costly software delivery. As can be seen, identification of change-prone requirements (High Error) shows signs of overall improvement. However, there is no obvious general improvement in the estimation of volatility, since both Error and False High rates do not show indication of significant improvement. These results are included here for interest; no conclusion can be drawn from such a small number of projects. When asked to comment on the difficulty of volatility estimation, the project manager said that while used to estimating effort, no explicit consideration had been given to the prediction of volatility. Partaking in model structuring increased awareness of the factors involved – "brought them to conscious thought".

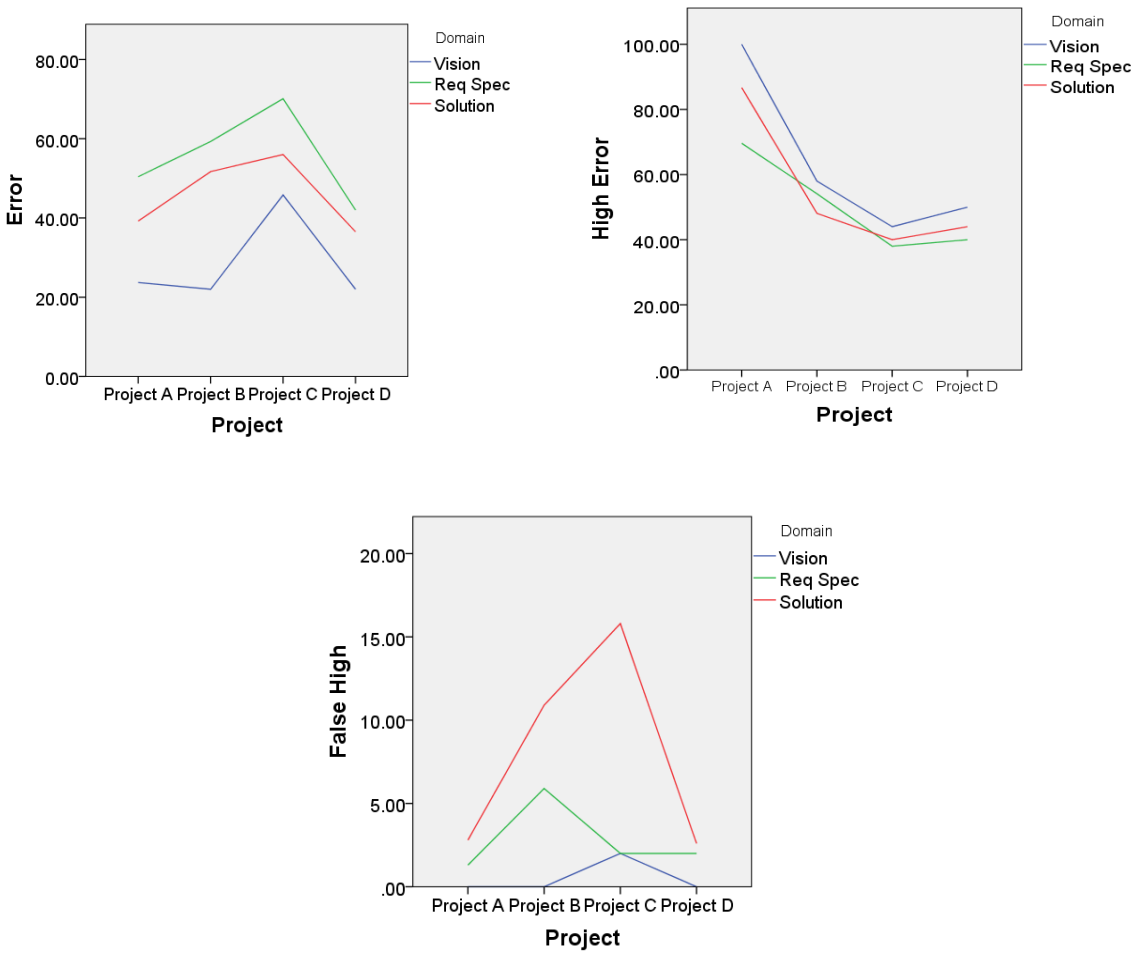

Figure 8.9 ERROR RATES FOR EXPERT ESTIMATION OF VOLATILITY DURING CONSECUTIVE PROJECTS

8.5.3 Phase by Phase Prediction with Interphase Learning.

Initial uncertainty for Project A was predicted from the Requirements Uncertainty networks that were trained from project B. While this means that the prediction of requirements uncertainty is not totally accurate, it is known from tranche 1 validation that prediction accuracy is, on average, around 70%. Volatility was predicted at the beginning of each of the four phases (Requirements Analysis, Design and Code, System Test, and User Acceptance Test, see section 5.4). At the end the predictions were compared with the actual levels of volatility recorded, and the accuracy of those predictions recorded in terms of the three errors described in section 8.4.5. Phase to phase Error rate results are shown in Table 8.10. Where the table cells are empty this indicates that there was no volatility predicted for that phase/change domain combination since the activities that would lead to change

discovery were not ‘on’. See Table 8.3 for details regarding which activities were ‘on’ in each phase.

Table 8.10 PREDICTION ERROR RATES FOR PROJECT A VOLATILITY WITH INTERPHASE LEARNING

	Phase 1			Phase 2			Phase 3			Phase 4		
	E1	**E2**	**E3**	**E1**	**E2**	**E3**	**E1**	**E2**	**E3**	**E1**	**E2**	**E3**
Vis	58.2	66	38.9	55	0	38	48.2	0	21	33.1	24.5	18.9
Rec	61.9	60	40.8	34	41.6	30	29.3	11.1	39	32.1	27.1	1.5
Sol				52	65.2	8.3	20.5	17.3	15	2.98	0	0

The error levels for the first prediction in each change domain were similar to those of the uncertainty predictions in the first tranche of testing. In general the errors for total volatility for each requirement fell by 50%, and by phase 4, over 70% of predictions were within 20% volatility. The error levels through progressive project phases are illustrated in Figure 8.10.

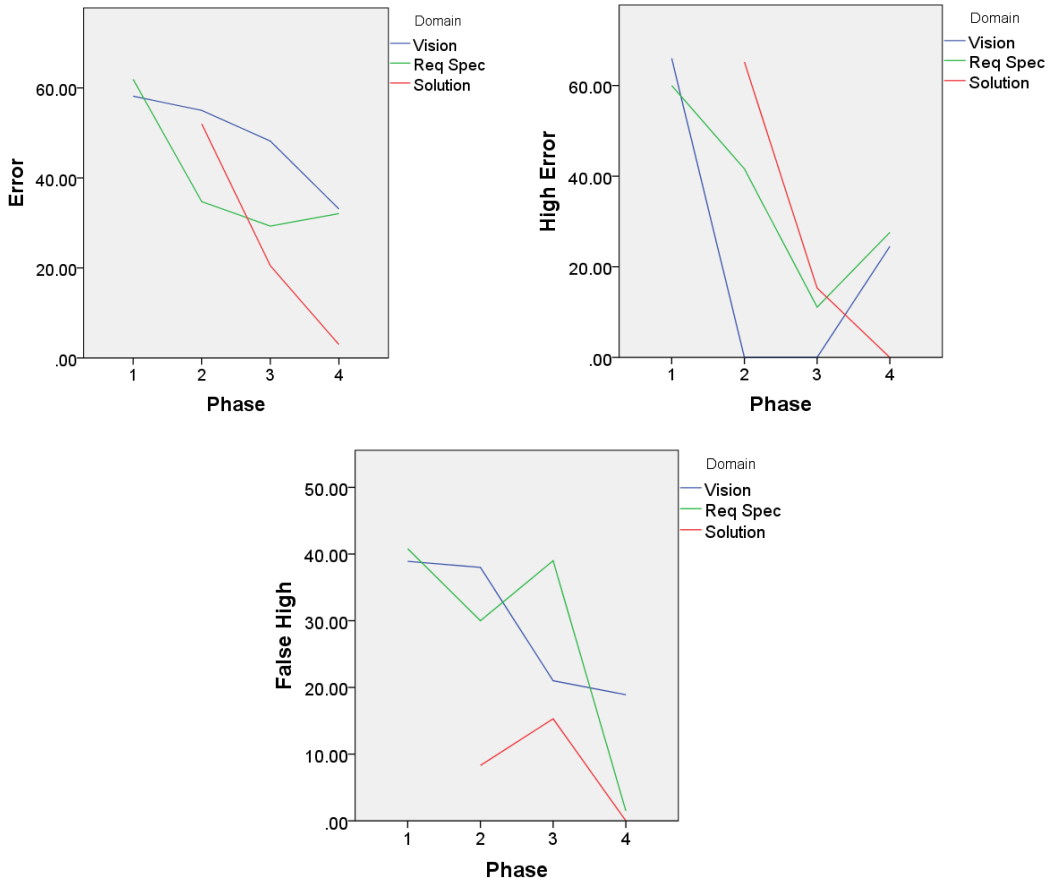

Figure 8.10 PHASE TO PHASE ERROR RATES

It is most interesting to note the levels of High Error and False High seen in the domain of *vision*. High Error is zero in phase 2, indicating that there were no requirements with a high level of volatility that were not identified as such. However, the level of False High remains almost the same as phase 1. This would indicate that more *vision* volatility was predicted for phase 2 than was discovered. These results can be further illuminated by the actual change costs that we observed in the previous case study described in Chapter 5. For convenience, the change costs from that study are presented again in Table 8.11.

Table 8.11 CHANGE COSTS FOR PROJECT A BY PHASE

	1	**2**	**3**	**4**
Vision	266.0	5.0	2.0	163.0
Requirement Specification	193.9	222.0	4.5	737.0
Solution	0.0	78.0	2.5	0.0

As can be seen little *vision* change was discovered in phase 2 or phase 3, though some requirements remained uncertain, as evidenced by the changes found in phase 4. While the level of High Error for *vision* volatility has remained zero in phase 3, the False High error has fallen, indicating the ability of the models to learn the discoverability of volatility, despite remaining uncertainty. In this case, less volatility was predicted, though levels of uncertainty would have remained almost constant, since little volatility was discovered. In phase 4, the *vision* High Error is about 25% meaning that about a quarter of highly volatile requirements are not being identified, while fewer requirements are being falsely assigned a High level of volatility.

In the domain of *requirement specification*, it can be seen that ability of the models to learn between phase 2 and phase 3 means that the rate of non-identification of highly volatile requirements has fallen sharply. However, there has been a rise in False High error indicating the models were erroneously identifying requirements as being highly volatile. This may also be explained by the change costs in Table 8.11, in that little change was actually discovered in phase 3, though a considerable amount of volatility was evidenced in phase 4.

Although the error levels in the domain of solution are falling, this cannot be attributed to model learning since, as can be seen from Table 8.11, nearly all solution volatility was discovered during phase 2. Some High Error and False High error was seen in phase 3, which may be explained by inaccuracies in uncertainty prediction.

While we have no estimation data with which to make comparison, this study illustrates the potential of the models to learn from their environment and produce more accurate predictions. Most importantly, volatility predictions will be most useful in practice when considered alongside the predicted uncertainty. The actual volatility experienced during

phase 4 of this project represented 38% of the entire project volatility, and comprised changes from the domains of Vision and Requirements Specification. Predicting those levels with a degree of accuracy will support the planning and management of late change which may otherwise be unexpected.

8.5.4 Sensitivity Analysis

As discussed in section 7.5.1, sensitivity analysis measures the degree to which findings at any node can influence the beliefs at another node [131]. Therefore, by examining sensitivity of variables on networks that have been trained using project data, an assessment can be made regarding the important factors that have had an effect upon volatility in each project. Such a comparison deepens understanding regarding the differences between projects, and also allows the identification of significant influences that are shared in all project scenarios.

Figure 8.11, Figure 8.12, and Figure 8.13 provide an illustration of all variable influence on all four projects for the domains of *vision, requirement specification* and *solution* respectively. Since data was only available for nodes relevant to the uncertainty of requirements, variables are confined to those in the uncertainty networks illustrated in Figure 7.5, Figure 7.6, and Figure 7.7. All sensitivity graphs show the level of contribution (by percentage) that an individual node makes to affect a change in the target node of 'requirements uncertainty'. Although each percentage is not connected to another by time or trend, each project is represented by a line, rather than a bar so that comparison can more easily be made. Project A and B share context, and are coloured dark blue and light blue respectively. Similarly, project C and D share industrial context and are coloured orange and red. A list of variables and their descriptions can be found in Appendix 2.

Sensitivity analysis in the domain of *vision* is illustrated in Figure 8.11. What is striking about this graph is the mirrored shape of the influence of variables in projects C and D (orange and red lines). Although the absolute values are different (lower in project D), the relative influence is similar for many nodes. The two projects diverge on variables related to tools, techniques and quality of processes, with project D seeing a greater change to volatility given lower quality verification and validation procedures by comparison to project C. Project constraints are similarly influential in these two projects. The blue coloured lines that represent projects A and B are also similar in shape, though perhaps not as precise a match as those in projects C and D. Notable differences between A and B by

comparison to C and D is the higher influence of requirements complexity and novelty. Overall, the most significant influences to vision uncertainty in all projects is the stakeholder understanding of the problem, stakeholder agenda and senior management involvement.

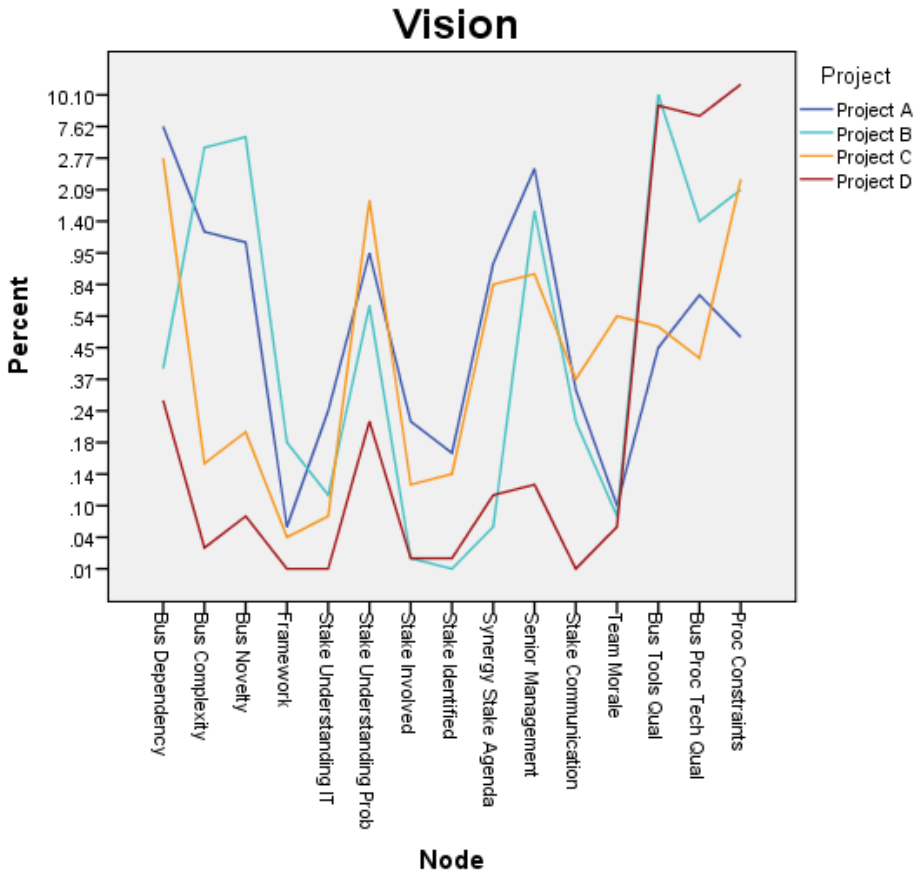

Figure 8.11 SENSITIVITY ANALYSIS OF ALL PROJECTS IN THE DOMAIN OF VISION

Figure 8.12 illustrates the relative influence of variables in the *requirement specification* domain. Much similarity can be seen between projects with a shared context. Project C and D are closely aligned with only minor deviations in tools and techniques quality and technical novelty. Projects A and B have also similar levels of variable influence. The volatility on all four projects is most significantly affected by the quality of the incoming work product, with process factors and analysis skill and knowledge also highly influential. A difference can be seen between the influence of variables between the two industries. Requirements volatility in industry 1 (projects A&B) are more affected by framework usage and stakeholder understanding, than those in industry 2 (projects C&D) . Also, volatility in industry one is likely to be higher with lowered levels of inter-team communication and morale by comparison to industry 2.

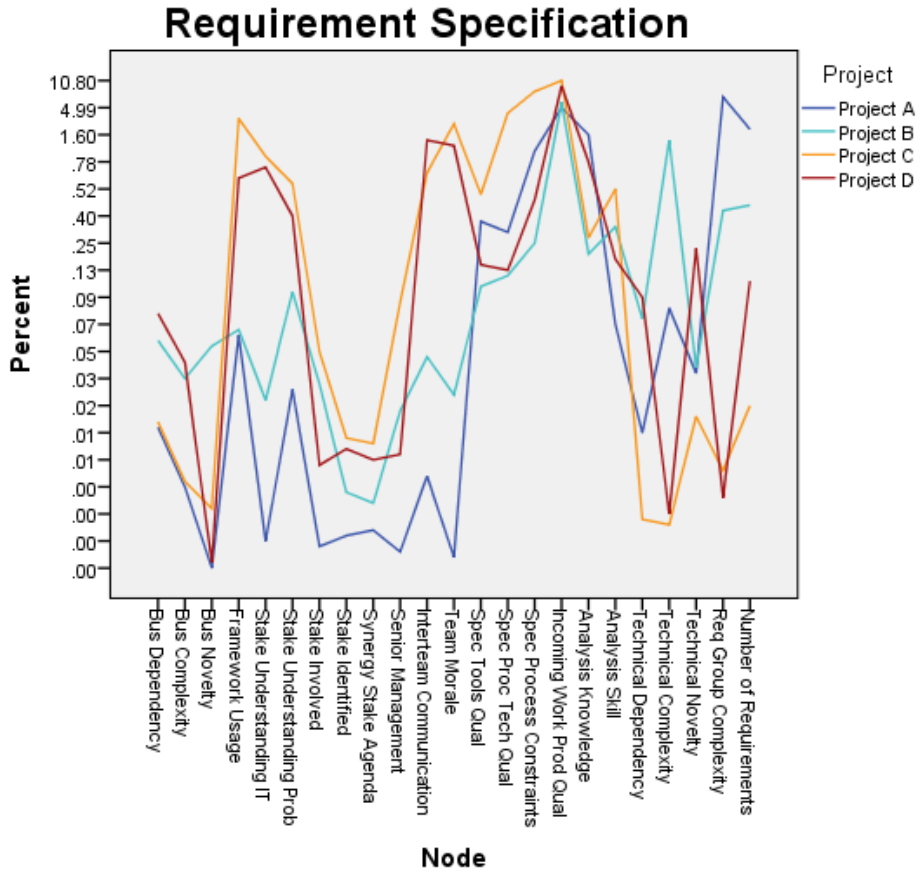

Figure 8.12 SENSITIVITY ANALYSIS OF ALL PROJECTS IN THE DOMAIN OF REQUIREMENT SPECIFICATION

The sensitivity analysis for the domain of *solution* is illustrated in Figure 8.13. Once again similarity of variable influence can be observed between projects sharing industrial context. Projects A and B are closely mirrored for very many variables, and deviate only very slightly on the influence of senior management and solution skill and knowledge. Reflecting the observation in the domain of *requirement specification*, volatility is highly influenced in project A and B from the quality of the incoming work product. Projects C and D are less closely aligned, and vary quite significantly in terms of the influence of technical complexity and novelty. Interestingly, project D is the only project whose volatility was influenced to such a small extent by these two factors. Projects C and D are more affected by inter-team communication by comparison to projects A and B. By contrast, the volatility on projects A and B is much more influenced by constraints upon the build process. It can be seen that in the *solution* domain, the main influencing factors for these two projects is the number of requirements in the requirement group, and the group complexity, though the number of requirements is less influential on project C.

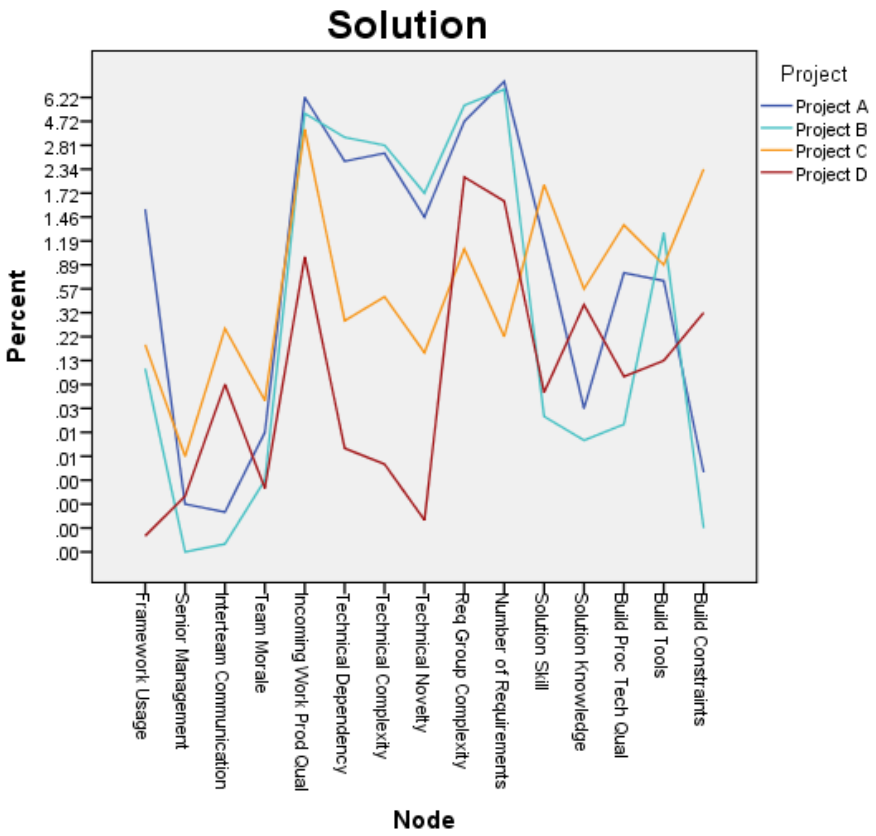

Figure 8.13 SENSITIVITY ANALYSIS OF ALL PROJECTS IN THE DOMAIN OF SOLUTION

8.6 Discussion of Results

As far as is known, there are no other validated models that predict levels of requirement volatility early in the project life-cycle with which to compare the results of this study. However, validation of these models using industrial project data facilitated comparison between model predictions of requirements uncertainty and project management estimation, and also allowed exploration of the efficacy of models when used in alternative industrial context. In this way, we not only assess model predictive accuracy, but also make comparison with the available alternative.

Overall, the results are in keeping with other Bayesian net models used in Software Engineering and validated in an industrial context. Results from models with CPT's defined using expert judgment are comparable with those of Stamalos yielding an accuracy of the prediction of change-prone requirements of around 50% [38]. Model learning from project

data with a shared industrial context, improved results in all cases, with average accuracy in excess of 70% of requirements, which mirrors the results seen by Hearty [129], and supports the argument by Menzies that prediction models, albeit using static code attributes, have the ability to predict defect-prone models 71% of the time. However, this result is only seen when models were trained using data from a project with shared industrial context, which was the only scenario to out- perform that of project management estimation. Indeed, since project management estimation was seen to improve somewhat during consecutive projects, the results seen here corroborate Jørgensen's efforts to focus research upon better project management estimation [108]. However, there need not be an either/or approach, but instead methods and techniques that strengthen both means of prediction so that triangulation between a project managers belief and that of a formal model may encourage confidence and commitment to appropriate management strategies.

The results of the expert defined models don't match those of Fenton who predicts software defects to an accuracy level of 93%. Defect prediction may be an application area about which more is known, or it may be possible to improve upon the results presented here. For example, it also may be the case that including a variable corresponding to requirement type such as that presented by Lim [13] would improve general prediction. Fenton also reported an ease of transition between context that isn't reflected here. However, while the CPT's did not 'hold' between organizations, the structure of the models was robust since all cases yielded good results with appropriate training. Moreover, the observation made by Zimmerman that cross project defect prediction is not always bi-directional was not borne out [111], though the level of optimism of project managers when estimating subjective quality factors would seem to affect even accurate cross project results. One of the most significant observations to derive from the results of this study is that formal models using subjective metrics not only rely upon robustness of model and calibration but also upon a similarly 'optimistic' outlook by those collecting the data. Regardless of the efficacy of the model, the consistency between subjective estimate 'attitude' is also necessary for cross project prediction accuracy and may be difficult to assess.

Sensitivity analysis provided some explanation for the observation in this study that intra-context training yielded more accurate results. The relative contribution of project factors to overall uncertainty was mirrored in the projects sharing industrial context. This is especially interesting, given that in each company one project followed a waterfall process, and one a hybrid methodology containing aspects of both waterfall and agile method. It would seem therefore, that there are aspects of industrial context other than development methodology that determine the influence of project factors upon requirements volatility. This carries

implications for empirical research and usage of all formal methods of prediction in software engineering. In all projects, the influence of requirement variables such as complexity and novelty differed between domains echoing the results seen in the earlier study described in chapter 6. This is further confirmation regarding the differing nature of changes from each of the domains in the requirements change source taxonomy described in chapter 4, and evaluated in chapter 5. Through the identification of the most influential factors, these results may inform future research in requirements volatility concerning requirements management and formal methods for prediction.

Though the use subjective estimates may have an impact upon predictive accuracy, freeing the prediction process from reliance upon source code metrics allows prediction to happen early in the software development lifecycle, and also supports the use of causal mechanisms. Induced from the results of the second tranche of testing, the causal approach taken would allow projects managers to avail of 'what if' analysis and explanation to inform project management decisions during development as well as during project retrospectives. For example, should predicted levels of uncertainty greatly exceed that of predicted discovered volatility, the networks can be used to determine which project factors are contributing towards a low level of change discovery. Though not undertaken in this validation study, performing 'what if' analysis using these Bayesian networks would have allowed project managers to ascertain the affects upon volatility of changing levels of activity effort and process factors, thus supporting decisions relating to late change discovery.

While thus addressing usability issues discussed by Penta [118], the time taken to collect and maintain the data necessary must also be taken into account. Collection of the model variable data took some considerable time, and further testing is necessary to ascertain the benefits of phase by phase learning since data must also be kept up-to-date throughout project development. While model design is agnostic to the development process, such data maintenance requirements may be prohibitive to agile development, and more suited to large projects with process constraints requiring more detailed reporting and tighter control. That said, one of the main benefits of the models presented in this study is the distinction between types of requirements change. This is by contrast to studies that identify change-prone requirements but make no distinction between types of changes, be they defects or functionality improvements [99] [66].

Providing not only an estimate of the volatility of requirements, but also a prediction of the type of volatility seen, will allow project managers to tailor process and techniques to best accommodate the expected changes, thus answering to Moynihan's observation that project

managers manage risk and uncertainty in different ways depending upon the type of uncertainty faced [178]. In fact, techniques such as partial modelling [30] require that analysts are aware of the uncertainty of the requirements, which is a dubious assumption given the results of this study, and those of Loconsole [27]. The change domains in this study were confined to those about which project managers could determine likely causes, and excluded the change domains of *customer organisation*, and *marke*t. Changes arising from these sources are outside of the remit of most software development and additional studies, such as those of Maxwell [14] who predicts changes to government regulation, complement the prediction efforts of this study.

8.6.1 Review of Results Validity

In addition to the model assumptions and limitations detailed in section 7.4.8, there are aspects of this study that may limit the validity of the results. The following is a discussion of results validity and makes reference to the types of validity described in section 3.4.

Construct Validity has been addressed through extensive variable definition and scaling which has been agreed by all involved parties. Since both the model variables, and the data needs were discussed and specified through meetings with researcher and practitioners, there is a shared understanding of the meaning of the data items. However, many of the measures are subjective, and collected by only one project manager at different times during development. Therefore there is potential for a difference of opinion about the values of the variables, and there is also the possibility that subjective data collected earlier would be less accurate in that more would be known later in project development. This means that data may be less certain, but reflects how the models would be used in practice. Increasingly SE research is recognizing the need to reflect this reality, and 'road-map' papers in areas of empirical research [34][4][134], cost estimation[150] [20] and quality estimation[103] call for a balance between scientific rigor and relevance. Rather than ignoring expert knowledge in favour of direct objective measurement, the benefits of subjective measurement are being recognised, meaning that metrics that are more closely aligned with process and 'causative' factors. Use of the same project manager for all four projects allows for some consistency, and facilitates comparison between models and estimates.

Internal validity is of concern when causal relations are examined, given that a third invisible (confounding) factor which is not included in the study may affect the results. The results of this study claim to rely upon a causal mechanism, and although every attempt was

made to include all influential factors through literature review and expert opinion, it is possible that something has been omitted. However, this limitation refers to studies of correlation erroneously claiming causality which is not the case here. It is believed that the factors included are sufficiently causal for requirements volatility.

External validity refers to the extent that the results of this study would apply to other projects. This is, in part, the intention of this study – to examine if a predictive model calibrated in one industrial context can be used in other. Our results suggest that only shared context calibration is efficacious. However, given the small number of projects involved, we cannot claim that this would be true in any other situation. Rather, these results contribute to the debate. By comparison to open source projects for which data is available, the context of these studies more closely reflects the target context within which the models may actually be used. Also, industrial participation allows consideration of the complexities of the software engineering environment, and facilitates study specific data collection. In this way, this study supports the prediction of different types of volatility, a novel idea that is informative to project management, but prohibitive to large investigative study due to lack of publicly available data. The results of the comparison between project manager estimation and model prediction can similarly make no claim to external validity but support the conclusion that project managers have difficulty predicting volatility drawn by Loconsole [27].

Attempt has been made to ensure *study reliability* which is defined as "the extent to which the data and analysis are dependent upon specific researchers" [32]. Threats to this aspect of validity are, for example, if it is not clear how data was specified and collected. Details of model structure, learning algorithms and statistical tests are provided for use by other researchers. While the results rely upon subjective measures, the participants in this study can be considered experts since their knowledge of both the business and the software development domain is extensive. All participants have at least 10 years of experience in their respective software development roles and at least 5 years of experience of software application development within the government sector.

In recognizing that conflicting results have been observed in studies comparing formal models and human estimation and also in studies comparing the use of formal models using local data with those using cross company data, Menzies explores ways to reduce this 'conclusion instability' [115]. This study contributes to both discussions, and attempts have been made to address the issues raised by Menzies. There are two main causes of conclusion instability which relate to variance of sampling, and model bias. The issue of subjective

estimation giving rise to model bias has already been discussed, and there may also be sampling related bias in that the projects for which data was available may not be typical of the organization, therefore invalidating the claim regarding shared context. Of particular importance is the discretization of volatility. The use of project specific interval 'bins' allow volatility to be assessed relative to the project is aligned project manager estimation, but it is clear that use of an alternative means of discretization would have offered different results. The ranges presented in this paper will inform volatility discretisation for field study when project specific measures are still not known. Having a common scale with which to measure volatility will be advantageous since it will then be possible to predict absolute volatility (rather than relative) for project requirements. It is hoped that results have been presented fairly and that no experimenter bias or 'preference' is evident when considering the compared methods of prediction.

8.7 Evaluation

Setting aside limitations to results validity which would require empirical studies to explore further, this section reflects upon the usability of the networks as they stand, given model assumptions, structure design and necessary data collection effort. Feedback was gathered on an informal basis, and notes taken by the researcher, both during model walkthroughs with industrial practitioners, described in section 7.5.2, and the presentation of the results of model validation described above. Both modes of model usage are considered; the prediction of requirements uncertainty as an indication of total potential volatility for each requirement, and the prediction of phase to phase volatility.

8.7.1 Practitioner Feedback

The practitioners involved in this study had no prior experience of formal methods for effort or quality estimation, preferring instead to use comparison techniques (from project to project and requirement to requirement) for effort. Neither had they made any previous attempt to estimate requirements volatility by any means. They affirmed the potential value of the prediction of volatility, especially in scenarios where a large number of stakeholders were involved, or the project was being managed by a junior management team. A particular benefit was to "justify contingency budget" which is often reduced due to pressures involved with contract winning, and/or customer delivery constraints.

The practitioners reported a number of ancillary benefits incurred during the process of data collection and volatility prediction review. The first was that a much greater focus was placed upon the issue of requirements volatility, which would not ordinarily be the case. It was felt that using the variables in the networks as a guide to identifying the factors involved with requirements change would highlight potential risk. Continued observance of such factors, especially those related to requirements uncertainty, would support increased awareness of volatility and improved estimation. In addition, keeping a record of the measures of the variables involved would lead to an accumulation of company knowledge, and also result in keener subjective estimation of team capabilities and communication etc. While these ancillary benefits are not directly related to the objectives of this study, these comments reflect similar observations made by practitioners involved with other empirical research [27] [16] .

Of particular interest to practitioners were the results of the sensitivity analysis that compared the relative influence of project factors upon volatility. This, they felt, could be used to foster improvements in important influences highlighted, such as the involvement of senior management. There is also opportunity for this technique to underline the importance of appropriate tools and techniques. They regarded this as an opportunity to gain empirical evidence of the effect of tools, techniques and constraints upon project outcome and quality. They expressed interest to further understand the implication that within particular industrial contexts, different project factors had significantly more influence upon volatility.

While data was useful, the time taken to collect it and maintain it was seen as a potential limitation since, as an unrequired unit of work, it would be culled if time or budget was short. Also, the practitioners felt that there were situations that occurred during projects that were not encapsulated in the volatility model. One such scenario that can often occur is that a single person, either a customer, or a stakeholder on the provider side, can initiate an undue amount of change, either through particular attention to detail or inflexibility of expectation. As the models currently stand, measures of attributes of individuals can only be included as they are absorbed into team qualities. Therefore, while the model structure reflects a good approximation of a generic software development project, some complexity of individual situations will mean that accuracy of results will always be potentially compromised, regardless of the success of model training and accuracy of model calibration.

Taken together, practitioners saw value in maintaining the data for learning purposes, value from sensitivity analysis for project retrospectives and process improvement, and value in

the prediction of requirements volatility in situations where there were many stakeholders or an immature project team. The data collection and maintenance effort would require support and budget allocation from senior company management in order for any benefit to be accrued. Such investment would be more likely in the event that the process was further validated and captured within a commercial tool.

8.7.2 Structure, assumptions, and limitations

Without examination of alternative structures, it is impossible to make an assessment of the efficacy of these Bayesian models over alternative designs. Further examination of the accuracy of different structures, especially those with smaller amounts of variables could attend to the data collection requirement. Since this study was limited to the change domains that were confined within the software engineering environment, the extension of models to include organisation and market changes would more fully reflect the real world, and provide more exhaustive volatility prediction.

Currently, there is insufficient empirical data to support the stratification of volatility to make possible the prediction of absolute levels, as opposed to the relative (Very High to Very Low) volatility levels predicted in this study. Until such discretisation is available, there is a limit to the value of phase to phase prediction since, as was encountered during this study, there is difficulty in performing mathematical operations on data defined using an ordinal Likert scale. Use of scalar variables with actual percentages would allow ‘residual uncertainty’ to be calculated with more accuracy at the end of each phase. This would greatly increase the usability of the complete set of models, and enable the prediction of volatility in addition to the change-proneness of individual requirements. Importantly, volatility for each requirement could be summed, and thus provide a project level volatility for a specified period of time. However, there is justification for furthering research into the use of models with a two-sided design, as presented here, since both sides are necessary to identify change discovery over and above requirements uncertainty.

During validation, the means by which phase to phase prediction was affected was to predict a single over-all value for requirements uncertainty, and use that as the basis for change discovery prediction. In practice, since the prediction of requirements uncertainty may not be accurate, implementation of models such that re-prediction of requirements uncertainty is possible at each phase would be beneficial. This would also allow models to incorporate new knowledge as development progresses and subjective estimates are better known.

Given the positive response from the practitioners regarding the sensitivity, the inclusion on the network of variables pertaining to particular software engineering practices, techniques and tools would support analysis of their affects upon volatility. Rather than a generic variable denoting quality of techniques, the networks could be instantiated with nodes for techniques such as 'Inspections', 'Test driven development' etc. It would therefore be possible to determine which techniques best address different types of volatility. Thus, empirical evidence may be gained to support the intuition that, for example, prototyping is useful to identify *vision* change, while requirements inspections may be better suited to uncover volatility from the *requirements specification* domain. This may also illuminate the somewhat puzzling relationship between requirements attributes such as complexity and volatility as evidenced in the results of this study and the previous one presented in section 6.7. In a similar vein, the type of requirement such as screen, or function may also be influential not only to volatility, but to the type of volatility seen as defined in the change source taxonomy. Inclusion of a variable denoting the type of requirement may increase accuracy of volatility prediction, and also, through retrospective sensitivity analysis, provide information regarding the inherent uncertainty, and the timing of change discovery in relation to different types of requirements.

Of the other assumptions listed in section 7.4.8, none significantly compromised the results applicability, nor rendered models less usable during the validation exercise. However, they may provide interesting avenues for further research. This is discussed more fully in the following chapter.

8.8 Conclusion

A set of Bayesian net models to predict requirements volatility built in accordance with results from previous empirical studies, were validated to investigate prediction accuracy, and were also evaluated for model usability.

There were two tranches of empirical validation involving four projects in two organizations. The first tranche involved all four projects and compared the ability of the models to predict levels of requirements uncertainty with that of expert estimation. Three model scenarios were considered; models which had been calibrated using expert judgment, those which had been trained using data from a project in the same company, and those having learned from a project in an alternative company. The second tranche of validation involved assessing the potential of the networks to predict levels of volatility phase by phase

through application development. Predictions were made at the beginning for the first phase, whereupon results were validated, and the models (re) trained before making predictions for the next phase.

Results indicate that there is no significant difference between project manager estimation and models built using expert judgment. The models trained from data in an alternative context did not perform any better than expert estimation either. On average, the rate of predicting the correct level of requirement volatility in all of these scenarios was roughly 50% (20% would be considered random). However, there was a significant improvement with models calibrated from projects sharing industrial context, generating accuracy of around 70%. Project managers also exhibited increased accuracy of prediction during the four consecutive projects, from on average 15% accuracy to 55% accuracy. Similarly, the phase by phase validation accuracy improved from 40% to over 70%. These results are in line with Bayesian net predictive accuracy reported in other research and contribute to the debate concerning the use of formal methods. In particular, they carry implications regarding the calibration, and validity of models in alternative industrial contexts.

Sensitivity analysis of the variables contributing to requirements uncertainty facilitated the comparison of the relative influential significance of project and environment factors in all four projects. This helped to explain the results of the first tranche of validation given that projects with a shared industrial context, regardless of development methodology, exhibited similar patterns of variable influence. Different variables had more influence in certain domains and this was observed in all four projects. In the domain of vision, stakeholder and senior management involvement were of most significance, while the quality of analyst skill and knowledge were more influential in the domain of requirement specification. In the domain of solution, the complexity and number of requirements in the requirement group contributed the most to volatility. These results are particularly interesting given the study presented in chapter 6 wherein six requirements attributes were examined for correlation with volatility, none of which have been seen as major causal contributors in this study. These results further confirm the differing nature of changes in the domains of the change source taxonomy.

Model usability evaluation revealed that practitioners considered that there was benefit in maintaining the model data for analysis purposes, and value in the sensitivity analysis for project retrospectives and decisions regarding process improvement. The prediction of requirements volatility would be particularly useful in situations where there were many stakeholders or an immature project team.

There were also a number of limitations relating to results validity, structure assumptions and the cost of data maintenance that would need consideration prior to industrial usage. While no claim can be made that the results of this study are generalizable, the four industrial projects used for validation have a close fit with the type of context within which they may be used. Before commencement of field testing, further validation is necessary to investigate validity in alternative industrial contexts and determine generalized ranges of volatility to allow prediction of absolute (rather than relative) volatility.

The following chapter also reviews all work undertaken, highlights significant results, and reviews progress towards the goal of software requirements volatility prediction.

Chapter 9 Conclusion

Towards the goal of requirements change prediction, a series of studies has been undertaken, culminating with the construction and validation of a set of Bayesian networks to predict levels of volatility. This chapter provides a summary of the empirical research and highlights the most significant results and contributions to academic knowledge. A review of progress towards the goal of volatility prediction reveals potential areas for further research. These are introduced briefly.

9.1 Introduction

Changes to requirements during software development have the potential to add value to a system of software but also have an impact upon schedule and cost, and thus represent a risk to timely and cost-effective software delivery. The management of changing requirements falls into three broad categories. The first intends to constrain future change through requirements engineering techniques, creative or futuristic, to determine a more thorough and accurate set of requirement specifications [5][10][11]. The second uses risk management or process methods such as agile techniques, to engender a more evolutionary approach to requirements gathering and delivery [23][24]. The third attempts identify change prone requirements in order to support decision making concerning techniques and processes [14][10][33]. However, current research in this area is confined to the use of source code metrics as predictors, which precludes identification of more volatile requirements early in the development life-cycle. Thus decision support for project budgeting and planning is limited.

The approach taken in this research was to provide a means for constrained flexibility in the planning process by reducing some of the uncertainty inherent in requirements change discovery. The overall goal was to improve decision support for requirements handling, architectural planning, and schedule and cost estimation through the prediction of volatility. The objective was to identify both change prone requirements and levels of volatility early in the project lifecycle. Architectural flexibility and creative techniques can thereby be focussed upon change-prone areas, while scope, schedule and costing decisions can accommodate anticipated volatility. In so doing, realistic planning is facilitated.

Since relatively few empirical studies of software requirements change exist, some investigation of the causes and effects was necessary before the derivation of formal models of prediction. A review of the literature revealed a number of research challenges:-

RC1. Clearly define context and terminology
RC2. Investigate the causes of requirements change.
RC3. Derive an informative measurement of requirements change.
RC4. Examine feasibility of change prediction by formal model.
RC5. Build models relating causes of change to volatility.
RC6. Validate models for accuracy and evaluate models for usability.

The results of a series of empirical studies undertaken in collaboration with industry addressed RC1, RC2 RC3, and RC4, and their results formed the theoretical basis upon which a set of Bayesian Networks were structured. These were validated for predictive accuracy with industrial data from four projects in two separate organisations, and evaluated for usability, thus answering the needs of RC4, RC5, and RC6.

This chapter concludes this research effort, and provides a review of all studies undertaken, summarising the most significant results. These are considered in light of the challenges listed above, highlighting the contribution made to academic knowledge, and the benefit to industrial practitioners. Further work that may build upon these results is considered, as well as other potential opportunities for research that have been revealed through the course of these studies. Finally, progress towards the ultimate goal of software requirements change is reviewed, and future direction proposed to move this work towards practical usage in industry.

9.2 Summary of Empirical Studies and Significant Results

Prior to the construction of formal models to predict requirements change, a number of studies were necessary in order to better understand the causes and consequences of change. Models of prediction would need to accommodate causal factors that could be identified and measured, and produce predictions that were practical and informative. To that end, three empirical studies were undertaken to identify the causes of change, classify them in a way that was informative to project management, and investigate if there were attributes of requirements that made them more susceptible to change. These three studies formed the theoretical foundation of a series of Bayesian networks which were structured in accordance with expert judgement and validated with data from 4 industrial projects.

9.2.1 Software Change Classification

Given that various change classification systems have been proposed in order to answer to specific needs, the objective of the first study was to categorise requirements changes that occur during software development such that the system of classification reflected the source of the change. In order to do so, it was first necessary to refine current understanding of the terms ‘development’ and ‘maintenance’. Nomenclature used by other researchers was

considered, and a somewhat narrower view of maintenance was defined in this study which included product upkeep and servicing, but excluded evolutionary development. This is in line with the naming convention used by our industrial partners.

Drawing upon the literature, a set of change source constructs, such as ‘new stakeholder’, or ‘novelty of application’, were identified, each of which was considered to be a cause of requirements change. Through workshops and card sorting, six project managers and maintenance engineers derived a taxonomy of change source constructs comprising the five change domains illustrated in Table 9.1.

Table 9.1 CHANGE SOURCE DOMAINS

Change Domain	**Description**
Market	Differing needs of many customers, government regulations, external to project.
Customer Organisation	Strategic direction of a single customer, customer organisation considerations, external to project.
Project Vision	Problem to be solved, product direction and priorities.
Requirements Specification	Specifying the requirements of the established problem.
Solution	Technical answer to problem.

An initial study based on sources of change relevant to software development resulted in a classification which made the important distinction between uncertainty (situation) and trigger (event) giving rise to change. A second study incorporating significantly fewer sources of requirements change during software product maintenance classified the constructs according to the change source domains previously derived. The sources of maintenance requirements change could easily be attributed to the change domains defined in the initial study, and many of the constructs had been included within the original taxonomy. Those constructs relevant solely to development related to requirements and domain understanding, while those pertaining only to maintenance were concerned with system age. The extended requirements change source taxonomy is illustrated in Figure 9.1. Those constructs pertaining to software development only are followed by (D), while those applying only to maintenance or iterative delivery are shown as (M/I).

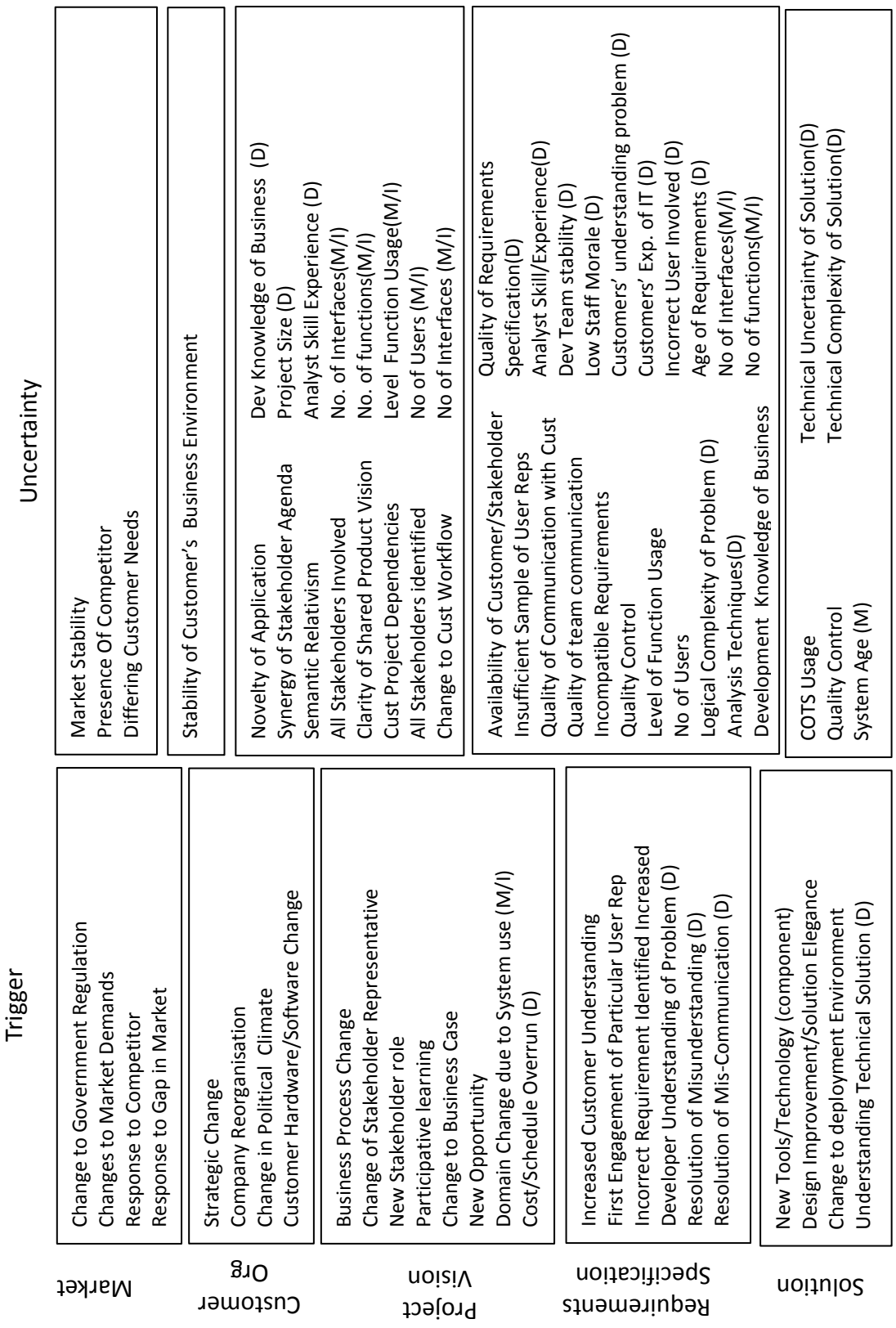

Figure 9.1 SOFTWARE REQUIREMENTS CHANGE SOURCE TAXONOMY

9.2.2 Evaluation of the Software Change Source Taxonomy

Before using the previously derived classification as a basis for volatility measurement and thence prediction, it is necessary to determine if the domains of *market, organisation, requirement specification*, and *solution* can be said to represent different types of changes requiring particular management considerations.

To this end, a case study was undertaken in an industrial context wherein 269 requirements changes were examined in a project costing a total of 4222 days effort over a period of 16 months. Data specification followed a Goal Question Metric approach that involved practitioners to define a list of data items pertinent to the management of changes. These were cost, value, opportunity v's defect, number of involved stakeholders, discovery activity and level of project management control. Statistical methods were employed to determine if there were significant differences in these attributes between change domains. The results are summarised in Figure 9.2.

While no results are available for the domain of *Market*, findings indicate the following:

- There are significant differences in cost, value, project manager control and stakeholder involvement between changes arising from each of the non-market sources.
- Generally changes from the *organisation* domain are more costly, have a higher value, more often represent an opportunity rather than a defect, but also have increased stakeholder group involvement, considered less easy to control.
- From *Vision* to *Specification* to *Solution*, change costs fall, stakeholder involvement decreases, and there in an increased level of control.
- Changes coming from activity considered external to the project are all from the *Organisation* and *Vision* domains.

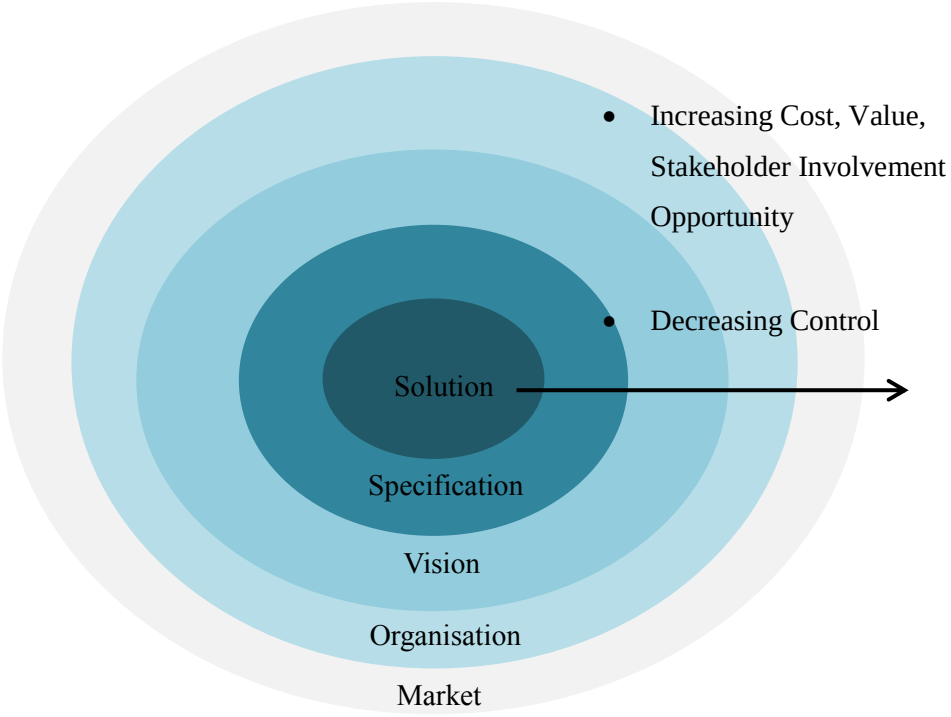

Figure 9.2 EVALUATION OF THE CHANGE SOURCE TAXONOMY

This would imply that assessment of risk and management of changes should be tailored according to the characteristics of these change domains. These results also have significant implications for the feasibility of change anticipation. Not only does requirements volatility arise in response to changes in the immediate 'small world' of the development environment, but more challengingly, the 'larger world' of the *organisation* and *market* whose needs must be met by the software. Changes discovered through 'external action' are going to be very difficult if not impossible to predict.

9.2.3 Identification of Change-prone Requirements

The next case study further examined the causes of requirements change, but this time the focus was upon the requirements themselves. Attributes felt by industrial participants to be influential to requirements change were identified and collected for the 240 requirements delivered in the project for which change data had already been analysed. The attributes under investigation were business and technical novelty, business and technical complexity, and requirements dependency. These attributes were examined for correlation with volatility, for which a metric was first specified that accommodated a perspective of volatility

reflective of cost. This study also investigated if requirements exhibited the same pattern of volatility in the change domains in the change source taxonomy.

Results were not straightforward. Apart from requirement dependency, which positively correlated with all measures of volatility, the results were somewhat counterintuitive. In summary, the results were as follows:

- Some requirements are more change-prone than others given that 80% changes involved 31% requirements.
- In the main, changes coming from the domains sources effected different groups of requirements.
- The only clear correlation with requirements volatility was dependence, which was positively correlated in all domains.
- Requirements business complexity showed no correlation with volatility, while technical complexity was negatively correlated but only in the domain of *requirement specification.*
- Requirement business novelty exhibited different relationships with volatility in each of the domains. Notable was the positive correlation between business novelty and volatility in the domain of *organisation* and the contrasting negative correlation in the domain of *requirements specification.*

That business complexity bore no correlation to requirements changes raises the possibility of a confounding factor, which is further supported by the surprising result that technical complexity is negatively correlated with volatility. However, these results confirm the differing nature of the domains of the change source taxonomy. The implication from the results is that changes coming from sources of *organisation*, *vision*, *specification* and *solution* are affecting different groups of requirements. Also, since business novelty was positively correlated with volatility in the domain of organisation, and negatively correlated in the domain of requirements specification, they would support the case for different relationships between requirement attributes and volatility in the different domains. Taken together these results indicate that a prediction of changing requirements will not be achieved based solely upon the requirement attributes examined in this study, and rely upon further consideration of more complex causal factors.

9.2.4 Bayesian Nets to predict Requirements Change

The results from the previous studies provided a clarification of terminology, a list of potential causes of requirements change, a classification of change that is informative to requirements management, and a metric to measure the size of the change. Bayesian network construction was achieved through a series of workshops and interviews with practitioners, and was founded upon the following theoretical basis:

1. Measurement of Volatility

The metric for volatility that would be the unit of measurement for prediction was based upon the relative amplitude of the change, which was measured by cost in days. Thus, the measure expressed the changeability of requirements in a way that was practical to collect. The metric is defined in Equation 6.1.

2. Taxonomy of Change Sources

Since it has been shown that changes coming from the domains of *customer organisation*, *vision, requirements specification* and *solution* can be considered as distinct groups of requirements the Bayesian network was designed to predict levels of each type of requirement. These domains are described in Table 9.1. However, to predict changes coming from the domain of *customer organisation* would require knowledge concerning organisational causes about which current research can tell us little. Therefore prediction was confined to the domains of *vision, requirements specification* and *solution*.

3. Differing characteristics of change domains

Changes coming from each domain differ in terms of their cost, value and management considerations as illustrated in Figure 9.2. In addition requirements attributes have different relationships with volatility as discussed in section 9.2.3. Therefore a different network was required for each change domain.

4. Two-sided nature of causal influence (uncertainty and trigger).

Change source constructs fall into two categories, identifiable as being change ‘triggers’ (events) or an ‘uncertainties’ (situations). A change trigger, such as ‘Change to business case’, could be considered an event which may drive requirements change. By contrast, an ‘uncertainty’ such as ‘quality of communication’, is a situation or state that is on-going and may be causative to volatility. This is discussed in section 9.2.1 and illustrated in Figure 9.1. Therefore a series of networks were defined reflective of this two-sidedness. Requirements uncertainty was predicted through three networks, for the domains of *vision, requirements specification* and *solution*, respectively. Volatility was predicted through a combination of requirements uncertainty, and the activity sub-networks of specification, build, system test, customer test, and demonstration. A description of these activities can be found in Table 7.5. It was possible to abstract a network core from the completed models, as illustrated in Figure 9.3.

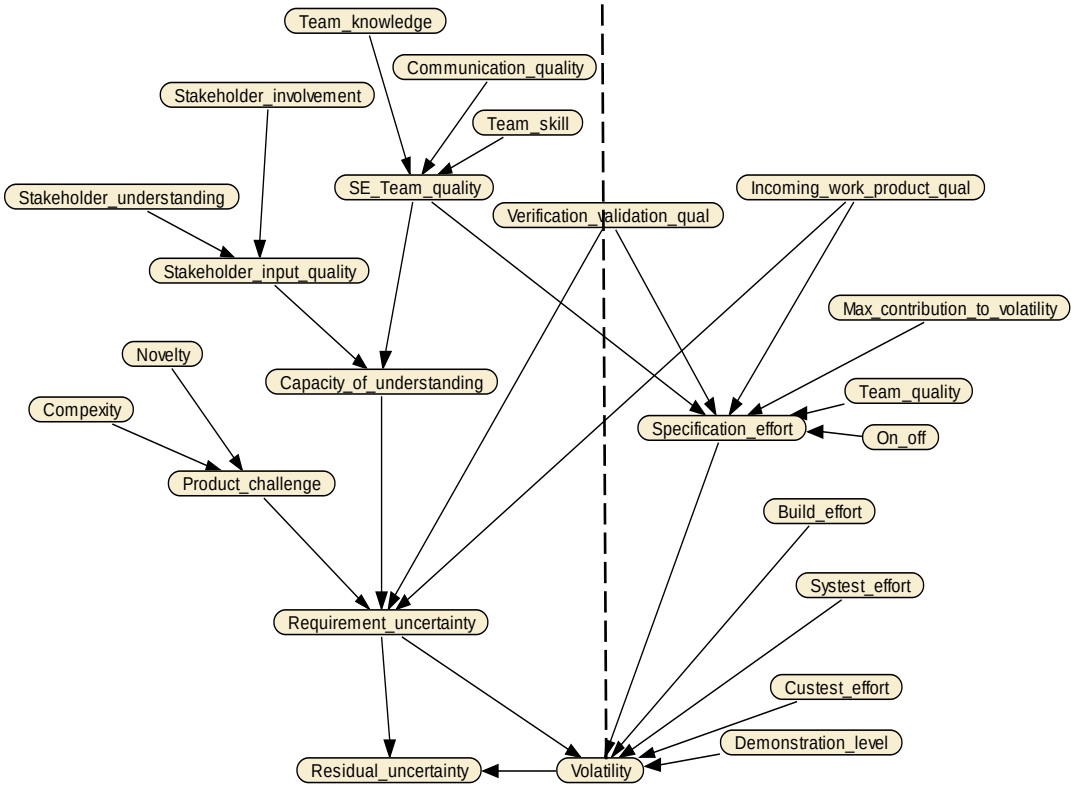

Figure 9.3 VOLATILITY PREDICTION: CORE MODEL

As a summary, factors influential to requirements uncertainty (parent nodes of the node 'requirement uncertainty' in Figure 9.3) can be broadly defined as:-

a) Product challenge – the difficulty of the application.
b) Capacity of understanding - the ability of the team to understand the product to be developed.
c) Quality of the processes and techniques used - methods of elicitation, process maturity etc.
d) Quality of the incoming work product – for example, this could refer to a set of predefined requirements as a result of a procurement process, or a set of business scenarios and rules from which test plans are developed.

In each domain, the variables contributing to each of these factors may be different. Activity subnets for specification, build, system test and customer test are all the same, though with different values for variables such as team quality. Each activity subnet has variables relating to:-

a) Quality of the incoming work product – this variable is defined above, and spans both sides of the network since it both influences the uncertainty of a requirement and the likelihood that change will be discovered through an activity that uses it.
b) Team Quality – the skill and experience of the involved team.
c) Quality of the processes and techniques used – this is defined above and spans both sides of the network since it affects both the requirement uncertainty and change discoverability.
d) Max contribution to volatility – this variable sets a ceiling for the amount of volatility that can be discovered by a particular activity.

Conditional Probabilities were specified using equations to build distributions based upon information provided by practitioners. Networks were verified through sensitivity analysis and model walkthroughs to ensure that they behaved in accordance with previous empirical results as they had been designed.

9.2.5 Predictive Accuracy and Evaluation

Model validation examined the capability of the models to accurately predict volatility using industrial data from four projects in two organisations. Two tranches of validation were undertaken. The first compared model predictions of requirements uncertainty with project management estimation. Requirements uncertainty networks were used for each of the domains of *vision, requirements specification* and *solution*. Three cases were considered: i) models that have been calibrated using only expert opinion; ii) those utilizing the learning capability of Bayesian Nets to parameterize the models from a project sharing industrial context; iii) models parameterized with data from a project with alternative industrial context.

The second validation tranche examined the potential of the models to predict changes to requirements for a project in-flight. In this case the requirements uncertainty networks were trained using data from a project sharing industrial context, and the activity sub-networks used to predict volatility. Comparison between model prediction and actual levels of volatility occurred at the end of each project phase, whereupon the models were recalibrated from actual data, and used to predict changes for the subsequent project phase.

Sensitivity analysis of the requirements uncertainty sub-networks that had been trained with project data facilitated examination of the influence of network variables upon the changeability of requirements and the comparison Subsequent analysis of usability issues and limitations provides an evaluation of the benefits to industry of the models in their current state.

Results of validation indicated the importance of appropriate context for model calibration. Models with CPT's defined by experts and those calibrated from a project with alternative yielded accuracy levels of around 50%. However, all domain models trained from a project sharing industrial context performed considerably better with accuracy levels of roughly 70%. By comparison, expert estimation was on average slightly better than models with CPT's provided by experts and those trained from alternative context, and there was some indication, particularly through comments from practitioners, that estimation accuracy was improving through subsequent projects. The phase by phase validation accuracy improved from 40% to over 70% illustrating the ability of the models to learn as project development progresses.

Sensitivity analysis helped to explain the results of the first tranche of validation in that projects with a shared industrial context exhibited similar patterns of variable influence. Different variables had more influence in certain domains than others, and this same pattern was observed in all four projects. In the domain of *vision*, stakeholder and senior management involvement were of most significance, while the quality of analyst skill and knowledge were more influential in the domain of *requirement specification*. In the domain of *solution*, the complexity and number of requirements in the requirement group contributed the most to volatility.

Model usability revealed that there was an associated cost incurred through data collection and maintenance, which may be prohibitive to formal models usage that are based upon these Bayesian networks. Widening scope may address that limitation by re-using the same data to provide more functionality such as quality and cost estimation. On the whole, practitioners considered that there was benefit in maintaining the model data for analysis purposes, and value in the sensitivity analysis for project retrospectives and decisions regarding process improvement. The prediction of requirements volatility would be particularly useful in situations where there were many stakeholders or an immature project team.

These results further confirm that there are differences in the characteristics of changes coming from the domains in the change source taxonomy, and that it is possible to predict software requirements change in certain circumstances by the Bayesian networks constructed in this research.

9.3 Progress towards Research Goal, Contributions and Further Work

Progress towards the overall goal of requirements change prediction was advanced through attendance to a number of research challenges which had been identified as a result of a literature review. A series of empirical studies, summarised in the previous section, addressed questions arising from these research challenges. In what follows, results are considered from the perspective of meeting the needs of the research challenges, contributing to academic knowledge, and providing benefit to industry. Future studies that may progress this research are also considered.

RC1. Clearly define context and terminology, especially software development, maintenance and evolution.

This challenge responded to the disparity of terminology and meaning attributed to terms associated with volatility, and in particular software development, evolution and maintenance. Beginning with a simple definition and clarification of such terms as change and volatility, a literature review was complemented with practical guidelines from practitioners to arrive at a contractually based definition of development and maintenance.

Academic Contribution

Until terminology is specified by a common board such as IEEE such that it is reflective of practice and not ambiguous, researchers who discern that clarity is needed will continue to define their own terms. However, the terms used here serve to clarify the context of this research and enable easier comparison with other studies.

Industrial Benefit

This facilitates common understanding for future work with practitioners involved with research, and communicates the applicability of models.

RC2. Investigate the causes of requirements change.

Many causes of change were identified in the literature, including those in the software engineering environment and those pertaining to the requirements themselves. Causes were extracted from the literature, consolidated by experts and an important distinction was made between causes arising from situations such as low level of analyst knowledge, and those coming directly from events such as business process change. Attributes of requirements such as novelty and complexity that were also considered causative to requirements were investigated for correlation with volatility. Bayesian network sensitivity analysis provided some understanding of the relative influence of some of these causes upon volatility.

Academic Contribution

In bringing together literature on the subject, researchers can find a single resource containing all published causes of requirements change. The investigation of requirement attributes was largely inconclusive, but highlights the complexity of the causal picture, and has significance for any correlative or causative study concerning volatility. The sensitivity

analysis will deepen understanding of the phenomena of software change as well as inform any future predictive models.

Industrial Benefit

Such a list allows a more comprehensive view of the potential causes, thus supporting risk analysis. Sensitivity analysis may help focus contingency efforts upon the most significant factors.

Further Work

The list provided attempts to be exhaustive, but there are other potential causes to be considered such as the type of requirement, screen, function, or type-classification based upon particular aspects of domain specific software, such as security or financial systems. There would also be benefit in examining factors that limit the impact upon change; techniques, or methods that reduce volatility. As was discovered during model evaluation, some of the complexity of requirements change is not yet fully understood. Examination of the 'human factor', when considering change causality in terms of personality and political climate, would also be of benefit to a complete understanding of the causes and effects of volatility.

RC3. Derive an informative measurement of requirements change.

Requirement changes are not equal since a single change can have a great variance in cost and value. Therefore to be informative for decision making, change measurement must capture the differing nature of change. A profile construct for change measurement was derived in three stages. The first stage resulted in a proposed a taxonomy of changes based upon the source of the change. The second determined that the change domains comprising the taxonomy represented different groups of requirements according to cost, value and management considerations. Subsequent studies further confirmed that each of these change domains had particular characteristics. The third stage added a quantitative component that together with the taxonomy provides an informative means to measure change.

Academic Contribution

These studies provided a novel taxonomy of changes built upon previous classifications and containing the change causes previously identified. Importantly, this is the only change type classification to have been evaluated empirically. The evaluation study also contributed to knowledge concerning the cost, value and controllability of requirements changes as they

occur in practice. There is potential for this classification to be used in many ways such as a basis for project comparison which would further promote understanding of different types of software projects.

Industrial Benefit

As it stands, there is concrete and valuable benefit to industry from the classification of requirements changes in particular. It has been shown that the character and management of changes differs according to the domains in the taxonomy. Practitioners have a practicable means to utilise the taxonomy within change control databases to analyse project performance, compare project change profiles, and consider changes to process, staff, training, planning or customer communication in order to attend to some of the problems associated with volatility. This classification also confers some notion of the idea of 'good change' and 'bad change' that is, that some change is to be encouraged since it adds value, but attempts can be made to the prevent the type of changes that add limited value and can therefore be avoided.

Further Work

Further validation of the informative value of the taxonomy through study replications in differing industrial contexts will ensure widened applicability. In line with further work investigating causes of change, the taxonomy could be extended to cover techniques that specifically address changes in different domains. In addition, studies that examine volatility in different projects will progress towards an understanding of levels of volatility that are tolerable in successful projects, so that the health of a project can be reviewed during software development. Another interesting direction would be to examine patterns of change domain volatility during the entire software lifecycle to discern if different types of changes typically follow a predictable frequency pattern in any given circumstance. Circumstances such as particular methodology, project size, or application domain could be considered.

During the presentation of the software change taxonomy at a conference, a lively discussion ensued regarding its applicability to areas outside of Software Engineering. Once such area was the building industry and in particular domestic housing, where changes to architectural planning, customer desire, and governmental decree would be causes for changes to plans. This would make an interesting inter-discipline study.

One of the limitations inherent in empirical studies in software engineering is the assessment of results applicability. Without a means to identify a project as a certain 'sort', validation of theories or tools is limited to the exact context(s) in which validation takes place. Industrial

project classification would allow some discernment of the more general applicability of study results, and foster the transfer of academic knowledge and method research to industry. This would be a major undertaking since there are many ways to view a Software Engineering project, and therefore any dependable classification would be very complex. It is possible that this taxonomy can contribute to such a classification, since projects may be identified as having particular change domain sensitivity.

RC4. Examine feasibility of change prediction by formal model.

Since there were no validated studies of formal models to predict software requirements volatility, it was necessary to determine if one of the reasons for the lack of such studies was due to infeasibility. The taxonomy of changes revealed that some sources of change were external to the project environment and would therefore be more difficult to predict unless a causal model was drawn from the wider world containing *market* and *customer organisation* factors. Subsequent prediction studies focussed upon the change domains about which our industrial partners could reason, placing a limitation upon the extent of prediction possible in this study. Thereafter, feasibility was examined through trial of model building and validation. This process revealed difficulties of model calibration and applicability, and also the lack of available volatility figures for use in generic models. This further limited the prediction of volatility to a relative ordinal scale rather than an absolute quantitative scale.

Academic Contribution

Given that this is the first known attempt to build and validate models of volatility prediction, communicating the limitations encountered will inform future research attempting the same. An important contribution has been made to the debate concerning the calibration of models by experts.

Further Work

The obvious related work is to address the limitations encountered and thus to discover the factors and processes that give rise to changes from external sources. Future studies in levels of project volatility would address the issue of generic volatility scaling.

RC5. Build models relating causes of change to volatility.

A series of Bayesian net models were built to predict some of the volatility seen during project development. A two-sided model was constructed during a series of meetings with practitioners which combined predictions of requirements uncertainty in each domain, with activity networks to predict discovered change.

Academic Contribution

The two-sided nature of the Bayesian networks is a novel approach for predictive models in software engineering reflective of the fact that although requirements may be uncertain, change does not necessarily follow unless effort is given to the discovery of changes. The models themselves represent the first attempt to build formal models of volatility prediction and should be useful for other researchers investigating volatility or quality prediction. Lessons may also be learned regarding the time and effort required in order to structure and calibrate such models.

Further Work

Since construction can be lengthy and require several hours of expert time, a worthwhile study would be the examination of the efficacy of different model structures. It may be possible that smaller models requiring less data collection effort would perform equally as well. Current understanding is that the structure of a model is more important than the calibration, though this has not been tested empirically. Comparing the accuracy of model structures may signify that such investment in model structure is not necessary. Implementation of volatility prediction through alternative techniques would support a more thorough assessment of the suitability of Bayesian networks by comparison to other methods.

Extension of models to accommodate changes during maintenance, and also external changes would provide a more exhaustive approach to volatility prediction and thus be more useful to industry. In line with further research into the causes of requirements change, models may benefit from the addition of variables pertaining to particular methodology, techniques and requirement types.

RC6. Validate models for accuracy and evaluate models for usability.

Since the usefulness of formal models is a subject of debate, the models were tested with data from two industrial contexts, and where possible the results compared with project manager estimation.

Academic Contribution

This validation study makes a significant contribution to two debated areas. The first concerns the value of formal models over project manager estimation, and the second concerns the need for similar projects from which to calibrate models. Although the number of projects involved was low, the context is exactly that for which the models may be used, meaning that results have more validity than studies using contexts such as student projects, or open source development. This study also makes an important contribution to knowledge regarding the potential for volatility to be predicted using formal methods.

Industrial Benefit

Though the models are not yet incorporated within a standalone off the shelf tool, the potential value of volatility prediction has been illustrated. Ancillary benefits relating to raised awareness of the influences upon change have also been reported, with the results of sensitivity analysis particularly useful for project retrospectives and analysis.

Further Work

While the results of this study are insufficient to make any conclusions about the ability of project managers to improve upon estimation accuracy during consecutive projects, it would be beneficial to investigate the competence of project managers to make subjective estimates, not just of volatility but of other project related factors. Research attempting to provide support for subjective estimation would be value to industry and also to academia, especially in regards to formal model usage and necessity. Along this vein, it would be interesting to explore whether managers are consistently optimistic in their estimation across all project factors.

The results of this study support the hypothesis that training models with data from a project with shared industrial context gave more accurate results than both those trained with data from a project in another company, and project manager estimation. This is worthy of further investigation within varied industrial contexts to inform research in formal models for

software engineering, where data is often limited and there is a reliance upon experts for model calibration.

The viability of a commercial tool based upon the Bayesian models herein depends upon a number of issues that may be addressed with future research. Foremostly, further testing within a wider empirical context is necessary to secure trust in model prediction accuracy. This could be achieved alongside model expansion to include requirement types and process techniques that may increase accuracy levels, and also help ensure consistency of results between software engineering contexts. Such research would provide project volatility statistics that could support the derivation of stratum of volatility to be used in models as absolute values. Field testing with live projects would also help determine the best use of functionality offered by Bayesian networks to support project and environmental analysis.

Once results are dependable, consideration can be given to reducing the data collection effort required and/or expanding the functionality of networks in order to provide more value return for effort investment in data maintenance. Companies that already make use of requirements management tools, such as Rational Doors or Serena, would benefit from prediction systems built in to those tools. This would allow requirements information and volatility data to be fully integrated, and may lessen data collection effort while at the same time providing inclusion of volatility awareness within developmental processes. Similarly, integration with test automation and/or maintenance tools would serve to combine requirements quality with volatility, and begin to broaden the scope of requirements prediction to include software maintenance. Since many of the variables that predict requirements volatility are also used to predict effort and quality, re-using data to provide forecasts of effort and quality in addition to volatility would make models inherently more valuable.

Alternatively, once results accuracy is established, consideration of changes arising from sources external to the project environment is another potential research direction about which little is known. Fusing external change prediction with the models presented here will provide more exhaustive requirements change anticipation. Decisions concerning research direction may proceed from studies that investigate the relative accuracy and learning capability of both formal models and human experts.

9.4 Final Comment

Progress towards the overall goal of the prediction of software requirements change has been evidenced through these series of studies. It possible to predict change coming from within the software engineering environment with Bayesian models to an acceptable degree of accuracy, provided that local data is used for model training. Bayesian models also provide functionality, such as sensitivity analysis, that is useful for practitioners. However, there is some way to go before the prediction of volatility is commercially viable.

In the meantime, these studies have made contributions to the fields of software engineering and Bayesian networks, and in particular to the measurement and classification of volatility. The results further the progress made by Fenton [35] who pioneered efforts to use Bayesian networks as a means of prediction in Software Engineering, and contributes to the investigation by Menzies [115] and Jørgensen [177] concerning the efficacy of formal methods compared with that of expert estimation. Given the need for investment in data collection required by Bayesian networks, the method of prediction introduced here may be more suited to large projects adhering to more a waterfall methodology. However, Boehm [26] recommended that process selection be dependent upon a number of factors, among them an estimate of volatility. The results of these studies imply that it is not only the quantity of volatility that is significant, but the quality of requirements changes that may be expected. The classification of changes derived in this work illuminates the differing characteristics of changing requirements. In so doing, process selection can be tailored within a project to address the particular type of risk posed by potential changes, as recommended by Moynihan [25] and Mac Cormac [24]. Study findings also complement recent efforts by Pimentel [11] and Lim [13] to anticipate change, by offering an alternative approach to the identification of change-prone requirements which can support decisions concerning architectural stability.

As a whole, exploring the feasibility of requirements change prediction has clarified that changes come from a wide range of sources, both within the project environment, and in the wider arena of the organisation and market place. An more exhaustive approach to the prediction of volatility would combine the efforts of this study with those of Maxwell [14], who examines ways to understand and anticipate changes to government regulations. Together with research by Nurmuliani [83], Zowghi [69], Ferriera [68], and Thakurta [20] the results of these studies lead to improved understanding of the risks and benefits of software requirements change in order to better respond to the needs of industry.

References

[1] C. Jones, “Patterns of large software systems: failure and success,” Computer, vol. 28(3), pp. 86–87, 1995.

[2] B. Boehm, “Requirements that Handle IKIWISI, COTS, and Rapid Change” Computer, vol. 33(7), pp. 99–102, 2000.

[3] S. Chapman, “Local Demands drove Fujitsu from HNH IT programme, MP’s told” , Computerworld, Jun 2008.

[4] S. Pfleeger, “Software Metrics: Progress after 25 Years?” IEEE Software, vol. 25, (6), pp. 32–34, 2008.

[5] N. Maiden, G. Rugg, “ACRE: selecting methods for requirements acquisition”, Software Engineering journal vol. 11(3), pp 183-192, 1996.

[6] H. Breivold, I. Crnkovic M. Larsson, “A systematic review of software architecture evolution research” Information and Software. Technology, vol. 54(1), pp. 16–40, 2012.

[7] L. Mathiassen, T. Saarinen, T. Tuunanen, M. Rossi, “Managing Requirements Engineering Risks: An analysis and Synthesis of the Literature", in Working Papers on Information Systems, 2004.

[8] D. Turk, R. France, B. Rumpe, “Limitations of agile software processes” Third International Conference on eXtreme Programming and Agile Processes in Software Engineering, 2002.

[9] R. Thakurta, F. Ahlemann, “Requirements Volatility Awareness and Management - A Qualitative Study in German Context” 6th Annual Conference on Information Science Technology and Management, 2008.

[10] D. Bush, A. Finkelstein, “Requirements stability assessment using scenarios,” Requirements Engineering Conference, pp 23-32, 2003.

[11] J. Pimentel, E. Santos, J. Castro, X. Franch, “Anticipating Requirements Changes - Using Futurology in Requirements Elicitation”, International Joural of Information. System Modeling and Design, vol. 3(2), pp 89–111, 2012.

[12] A. Loconsole, J. Börstler, “An Industrial Case Study on Requirements Volatility Measures”, 12th Asia-Pacific Software Engineering Conference (APSEC’05), pp 249–256, 2005

[13] S. Lim, A. Finkelstein, “Anticipating Change in Requirements Engineering” in Relating Software Requiirements to Architectures, P. Avgeriou, J. Grundy, J. G. Hall, P. Lago, I. Mistrík, Springer Berlin Heidelberg, pp 17–34, 2011,.

[14] J. Maxwell, A. Anton, P. Swire, “Managing changing compliance requirements by predicting regulatory evolution” 20th International Requirements Engineering Conference, pp 101–110, 2012

[15] C. Jones, “Strategies for managing requirements creep” Computer, vol. 29(6), pp 92–94, 1996.

[16] A. Nolan, S. Abrahao, P. Clements, A. Pickard, “Requirements Uncertainty in a Software Product Line” 15th International Software Product Line Conference, pp. 223–231, 2011

[17] Y. Fu, M. Li, F. Chen, “Impact propagation and risk assessment of requirement changes for software development projects based on design structure matrix,” Int. Journal Project Management, vol. 30(3), pp. 363–373, 2012.

[18] B. Sharif, S. Khan, M. Bhatti, “Measuring the Impact of Changing Requirements on Software Project Cost : An Empirical Investigation” Int. Journal Computer. Science, vol. 9(3), pp. 170–174, 2012.

[19] R. Robbes, D. Pollet, M. Lanza, “Replaying IDE interactions to evaluate and improve change prediction approaches” 7th IEEE Working Conference on Mining Software Repositories, pp 161–170, 2010

[20] R. Thakurta F. Ahlemann, “Understanding Requirements Volatility in Software Projects - An Empirical Investigation of Volatility Awareness, Management Approaches and their Applicability” 43rd Hawaii International Conference on System Sciences, pp 1–10, 2010

[21] D. Zowghi, “The Three Cs of Requirements: Consistency, Completeness, and Correctness”, International Workshop on Requirements Engineering, 2002.

[22] M. Bano, S. Imtiaz, N. Ikram, M. Niazi, M. Usman, “Causes of requirement change - a systematic literature review” International Conference on Evaluation & Assessment in Software Engineering, pp 22–31, 2012

[23] F. Massacci, D. Nagaraj, F. Paci, L. M. S. Tran, A. Tedeschi, “Assessing a requirements evolution approach: Empirical studies in the Air Traffic Management domain,” Second IEEE International Workshop on Empirical Requirements Engineering, pp 49–56, 2012

[24] A. Maccormack, R. Verganti, “Managing the Sources of Uncertainty: Matching Process and Context in Software Development” Journal Productive Innovative Management, vol. 20(3), pp. 217–232, 2003.

[25] T. Moynihan, “‘ Requirements-Uncertainty ’: Should it be a latent , aggregate or profile construct ?” in Australian Software Engineering Conference, pp 181–188, 2000

[26] B. Boehm, R. Turner, “Balancing agility and discipline: evaluating and integrating agile and plan-driven methods,” 26th Int. Conference Software Engineering, pp 718–719, 2004.

[27] A. Loconsole, J. Borstler, “Are size measures better than expert judgment? An industrial case study on requirements volatility” Software Engineering Conference: 14th Asia-Pacific, pp 238–245, 2007

[28] D. Parnas, “Designing Software for Ease of Extension and Contraction” IEEE Transactions on Software Engineering, vol.5(2), pp. 128–138, 1979.

[29] B. Nuseibeh, “Weaving Together Requirements and Architectures,” IEEE Computer, vol. 34 (3), pp 115–117, 2001.

[30] R. Salay, M. Chechik, J. Horkoff, A. Di Sandro, “Managing requirements uncertainty with partial models,” Journal of Requirements Engineerimg, vol. 18 (2), pp 107–128, 2013.

[31] A. Loconsole, “Definition and Validation of Requirements Volatility Measures,” SERPS’, 2006.

[32] A. Loconsole, “Empirical studies on requirement management measures” 26th International Conference on Software Engineering, pp 42–44, 2004

[33] A. Sharafat, L. Tahvildari, “Change Prediction in Object-Oriented Software Systems : A Probabilistic Approach” 11th European Conference on Software Maintenance and Reengineering, vol. 3(5), pp 27–38, 2008

[34] N. Fenton, Q. Mary, W. College, M. Neil, “Software Metrics : Roadmap” Conference on Future of Software Engineering, pp 359–370, 2000

[35] N. Fenton, M. Neil, D. Marquez, “Using Bayesian networks to predict software defects and reliability,” Journal Risk and Reliability, vol 222 (4), p. 701–712, 2008.

[36] L. Radlinski, “A Survey of Bayesian Net Models for Software Development Effort Prediction” International Journal Software Engineering and Computing, vol. 2(2), pp 95–109, 2010.

[37] N. Fenton, M. Neil, W. Marsh, P. Hearty, Ł. Radliński, P. Krause, “On the effectiveness of early life cycle defect prediction with Bayesian Nets" Empirical Software Engineering, vol 13 (5), pp 499–537, 2008.

[38] I. Stamelos, L. Angelis, P. Dimou, E. Sakellaris, “On the use of Bayesian belief networks for the prediction of software productivity” Information and Software Technology, vol. 45 (1), pp 51–60, 2003.

[39] N. Fenton, M. Neil, W. Marsh, P. Hearty, D. Marquez, P. Krause, R. Mishra, “Predicting software defects in varying development lifecycles using Bayesian nets” Information and Software Technology, vol. 49 (1), pp 32–43, 2007.

[40] S. Pfleeger, “Soup or Art? The role of evidential force in empirical software engineering,” IEEE Software, vol (1), pp 66–73, 2005.
[41] P. Runeson, M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” Empiral Software Engineering, vol. 14 (2), pp 131–164, 2008.
[42] C. Wohlin, M. Host, K. Henningsson, “Empirical Research Methods in Software Engineering,” Empirical Methods and Studies in Software Engineering, R. Conradi and A. Wang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp 7–23, 2003
[43] M. Lehman, J. C. Fern, “Software evolution—Background, theory, practice” Information Processing Letters, vol. 88 (1) pp 33–34, 2003.
[44] M. Lehman, J. Ramil, P. Wernick, D. Perry, W. Tursky, “Metrics and Laws of Software Evolution - The Nineties View” International Software Metrics Symposium, pp. 20–32,1997
[45] L. Belady, M. Lehman, “A model of large program development,” IBM Systems Journal, vol. 15 (3), pp. 225–252, 1976.
[46] M. Lehman, “Software ’ s Future : Managing Evolution,” IEEE Software., vol. 15(1), pp 40–44, 1998.
[47] K. Villela, J. Doerr, A. Gross, “Proactively Managing the Evolution of Embedded System Requirements” 16th IEEE Int. Requirements Engineering Conference, pp. 13–22, 2008.
[48] E. Barry, “Software evolution, volatility and lifecycle maintenance patterns: a longitudinal analysis synopsis” International Conference on Software Maintenance, pp 474–477, 2002
[49] C. Kemerer, S. Slaughter, “An Empirical Approach to Studying Software Evolution,” IEEE Transactions on Software Engineering, vol. 25 (4), pp 493–509, 1999.
[50] K. Bennett,V. Rajlich, “Software Maintenance and Evolution : a Roadmap,” Conference on the Future of Software Engineering, pp 73-87, 2000.
[51] “IEEE Standard Glossary of Software Engineering Terminology,” pp. 1–84, 1990 (2002 edition)
[52] M. Godfrey, D. German, “The past, present, and future of software evolution,” Frontiers of Software Maintenance, pp. 129–138, 2008
[53] B. Kitchenham, G. Travassos, A. von Mayrhauser, F. Niessink, N. Schneidewind, J. Singer, S. Takada, R. Vehvilainen, and H. Yang, “Towards an ontology of software maintenance,” Journal Software Maintenance vol 11 (6), pp 365–389, 1999.
[54] V. Basili, “Viewing maintenance as reuse-oriented software development,” IEEE Software, vol. 7(1), pp. 19–25, 1990.
[55] N. Chapin, J. Hale, K. Khan, J. Ramil, W. Tan, “Types of software evolution and software maintenance” Journal of Software Maintenance and Evolution: Research and Practice, vol. 13 (1), pp. 3–30, 2001.
[56] R. Fablo, C. Menezes, A. Rocha, “Using ontologies to improve knowledge integration in software engineering environments,” 2nd World Multiconference on Systemics, Cybernetics and Informatics, 1998.
[57] N. Chapin, “Software Maintenance Life-cycle" IEEE Conference on Software Maintenace, pp. 6–13, 1988.
[58] C. Ebert , J. De Man, “Requirements Uncertainty : Influencing Factors and Concrete Improvements,” 27th International Conference on Software Engineering, pp. 553–560, 2005
[59] J. Li, H. Zhang, R. Jeffery, Q. Wang, M. Li, “Preliminary results of a systematic review on requirements evolution,” in 16th International Conference on Evaluation & Assessment in Software Engineering, pp 12-21, 2012
[60] N. Ernst, A. Borgida, I. Jureta, “Finding incremental solutions for evolving requirements,” 19th International Requirements Engineering Conference, pp 15-24, 2011

[61] G. Stark, P. Oman, A. Skillicorn, C. Ameele, "An Examination of the Effects of Requirements Changes on Software Maintenance Releases," Journal of Software Maintenance, vol. 11 (5), pp. 293–310, 1999.

[62] N. Nurmuliani, D. Zowghi, S. Williams, "Using Card Sorting Technique to Classify Requirements Change," in 12th IEEE International Conference on Requirements Engineering, pp 24-248, 2004,

[63] S. Anderson, M. Felici, "Quantitative aspects of requirements evolution," in 26th Annual International Computer Software and Applications, pp 27-32, 2002

[64] P. Bhattacharya, M. Iliofotou, I. Neamtiu, M. Faloutsos, "Graph-based analysis and prediction for software evolution," International Conference on Software Engineering, pp 419-429, 2012

[65] B. Braunschweig, N. Dhage, M. Viera, C. Seaman, S. Sampath, G. Koru,"Studying volatility predictors in open source software," International symposium on Empirical software engineering and measurement, pp 181-190, 2012

[66] A. Sharafat, L. Tahvildari, "A Probabilistic Approach to Predict Changes in Object-Oriented Software Systems" 11th Conference Software Maintenance and Reengineering, pp. 27–38, 2007.

[67] H. Benestad, B. Anda, E. Arisholm, "Understanding software maintenance and evolution by analyzing individual changes : a literature review," Jounal of Software Maintenance and Evolution Research and Practice, vol. 21(6), pp. 349–378, 2009.

[68] S. Ferreira, D. Shunk, J. Collofello, G. Mackulak, A. Dueck, "Reducing the risk of requirements volatility : findings from an empirical survey," Jounal of Software Maintenance and Evolution Research and Practice, vol. 23 (5) pp 375–393, 2011.

[69] D. Zowghi, N. Nurmuliani, "A Study of the Impact of Requirements Volatility on Software Project Performance," in Ninth Asia-Pacific Software Engineering Conference, pp 3-11, 2002.

[70] J. Chudge, D. Fulton, "Trust and co-operation in system development: applying responsibility modelling to the problem of changing requirements," Software Engineering Journal vol. 11 (3). pp. 193–204, 1996.

[71] N. Nurmuliani, D. Zowghi, S. P. Williams, "Requirements Volatility and Its Impact on Change Effort: Evidence-based Research in Software Development Projects," Eleventh Australian Workshop on Requirements Engineering, 2006.

[72] D. Weiss, V. R. Basili, "Evaluating Software Development by Analysis of Changes : Some Data from the Software Engineering Laboratory," IEEE Transactions on Software Engineering, voll 11(2), pp 157–168, 1985.

[73] B. Boehm, V. Basili, "Top 10 list [software development]," Computer, vol. 34 (1), pp 135–137, 2001.

[74] K. Beck, Extreme Programming Explained: Embrace Change. Addison-Wesley Professional, 2004

[75] M. Poppendieck, T. Poppendieck, Lean Software Development: An Agile Toolkit, First Edition. Addison Wesley, 2003.

[76] P. Van Cauwenberghe, "Refactoring or Upfront design ?" 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering, pp. 50–53, 2003.

[77] T. Nakatani, S. Hori, M. Tsuda, M. Inoki, M Keiichi Katamine, "Towards a strategic Requirements Elicitation", International Conference on Software and Data Technologies, pp. 145–150, 2009

[78] T. Javed, M. Maqsood, Q. Durrani, "A Study to Investigate the Impact of Requirements Instability on Software Defects," ACM Sigsoft Software Engineering Notes, vol. 29 (4), pp 1–7, 2004.

[79] S. Ferreira, J. Collofello, D. Shunk, G. Mackulak, P. Wolfe, "Utilization of Process Modeling and Simulation in Understanding the Effects of Requirements Volatility" International Conference on Software Process Simulation and Modeling, 2003.

[80] W. Lam, V. Shankararaman, "Managing Change in Software Development Using a Process Improvement Approach," in 24th EUROMICRO Conference, pp. 779–786.

1998,
[81] R. Costello, "Metrics for Requirements," Journal of Systems and Software, vol. 29 (1), pp. 39–63, 1995.
[82] A. Gupta, O. P. N. Slyngstad, R. Conradi, "An Emprical Study of Software Changes in Statoil ASA – Origin , Prority Level and Relation to Component Size," International Conference on Software Engineering Advances, 2006
[83] N. Nurmuliani, D. Zowghi, S. Powell, "Analysis of Requirements Volatility during Software Development Life Cycle," Australian Software Engineering Conference, pp. 28–37, 2004
[84] N. Nurmuliani, "Understanding The Effects of Requirements Volatility," Journal of Systems Software, vol. 82(10) , pp. 1568–1577, 2008.
[85] G. Stark, "Measurements for managing software maintenance," International Conference on Software Maintenance, pp. 152–161, 1996.
[86] G. Kulk, C. Verhoef, "Quantifying requirements volatility effects," Science o f Computer Programming, vol. 72 (3), pp. 136–175, 2008.
[87] S.Nidumolu, "Standardization, requirements uncertainty and software project performance," Information Managment, vol. 31 (3), pp. 135–150,1996.
[88] J. Heales, "Factors affecting information system volatility," International Conference on Information Systems, pp. 70–83. 2000
[89] D. Christopher, "Prediction of Software Requirements Stability Based on Complexity Point Measurement Using Multi-Criteria Fuzzy Approach," International Journal Software Engineering Applications, vol. 3(6), pp 101–115, 2012.
[90] A. Davis, S. Park, "Requirements Change : What's the Alternative ?," Computer Software and Applications, pp. 635–638, 2008
[91] S. Harker, K. Eason, J. Dobson, "The change and evolution of requirements as a challenge to the practice of software engineering," IEEE International Symposium on Requirements Engineering, pp. 266–272, 1993
[92] A. Loconsole, Construction and Validation of Prediction Models for Number of Changes to Requirements. Department of Computing Science, Umeå University, 2008.
[93] C. Kemerer, S. Slaughter, "Determinants of Software Maintenance Profiles : An Empirical Investigation," Journal of Software Maintenance and Evolution: Research and Practice, vol. 9 (4) , pp. 235–251, 1997.
[94] E. Swanson, "The dimensions of maintenance," International Conference Software Engineering, pp. 492–497, 1976
[95] T. Mens, J. Buckley, "Towards a Taxonomy of Software Evolution," Workshop on Unanticipated Software Evolution, pp. 1–18, 2003
[96] J. Buckley, T. Mens, M. Zenger, A. Rashid, "Towards a taxonomy of software change," Journal of Software Maintenance and Evolution: Research and Practice, vol. 175 (5) pp 309–332, 2005.
[97] I. Sommerville, Software Engineering, 7th Edition, 9th ed. Addison Wesley, 2010
[98] D. Perry, "Dimensions of software evolution," International Conference on Software Maintenance pp. 296–303 1994
[99] S. Eski, F. Buzluca, "An Empirical Study on Object-Oriented Metrics and Software Evolution in Order to Reduce Testing Costs by Predicting Change-Prone Classes," Fourth International Conference on Software Testing, Verification and Validation Workshops, pp. 566–571, 2011
[100] C. Mao, Y. Lu, X. Wang, A study on the distribution and cost prediction of requirements changes in the software life-cycle, vol. 3840. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 136–150, 2006,
[101] D. Card, "Myths and Strategies of Defect Causal Analysis," Pacific Northwest Software Quality Conference, 2006

[102] A. Höfer, W. Tichy, "Status of Empirical Research in Software Engineering," in Experimental Software Engineering Issues: Assessment and Future Directions, vol. 1(1), V. R. Basili, Ed. Springer_Verlag, pp. 10–19, 2005
[103] N. Fenton, M. Neil, "A Critique of Software Defect Prediction Models," IEEE Transactions on Software Engineering, vol. 25(5), pp. 675–689, 1999.
[104] N. Fenton, M. Neil, W. Marsh, P. Hearty, P. Krause, "Project Data Incorporating Qualitative Factors for Improved Software Defect Prediction and Institute of Information Technology in Management," Third International Workshop on Predictor Models in Software Engineering (PROMISE'07), 2007.
[105] M. Shepperd, S. MacDonell, "Evaluating prediction systems in software project estimation," Information Software Technology, vol. 54 (8), pp 820–827, 2012.
[106] M. D'Ambros, M. Lanza, R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison" Empirical Software Engineering, vol. 17 (4–5), pp 531–577, 2011.
[107] M. Kasunic, "The State of Software Measurement Practice : Results of 2006 Survey", No. CMU/SEI-2006-TR-009, Carnegie-Mellon University, Pittsburgh, 2006.
[108] M. Jørgensen, B. Boehm, S. Rifkin, "Software Development Effort Estimation: Formal Models or Expert Judgment?," IEEE Software, vol. 26 (2), pp 14–19, 2009.
[109] S. Grimstad, M. Jørgensen, "A framework for the analysis of software cost estimation accuracy," International symposium on empirical software engineering, p. 58, 2006
[110] B. Kitchenham, E. Mendes, G. H. Travassos, "Cross versus Within-Company Cost Estimation Studies : A Systematic Review" IEEE Transactions on Software Engineering, vol. 33 (5), pp 316–329, 2007.
[111] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, B. Murphy, "Cross-project defect prediction: a large scale experiment on data vs. domain vs. process ," 7th Joint Meeting of the European Software Engineering Conferece and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp. 91–100, 2009.
[112] T. Menzies, A. Greenwald, A. Frank, "Data Mining Static Code Attributes to Learn Defect Predictorstle," IEEE Transactions on Software Engineering, vol. 33 (1) , pp. 2–13, 2007.
[113] T. Mende, "Replication of defect prediction studies," in Proceedings of the 6th International Conference on Predictive Models in Software Engineering - PROMISE p 1, 2010
[114] T. Schulz, Ł. Radliński, T. Gorges, W. Rosenstiel, "PROMISE Repository of empirical software engineering data," [Online]. Available: http://promisedata.org/?p=224, 2010
[115] T. Menzies, M. Shepperd, "Special issue on repeatable results in software engineering prediction," Empirical Software Engineering, vol. 17 (1–2), pp. 1–17, 2012.
[116] C. Catal, B. Diri, "A systematic review of software fault prediction studies," Expert Systems and Applications, vol. 36 (4), pp. 7346–7354, 2009.
[117] N.Fenton, P. Hearty, M. Neil, Ł. Radliński, "Software project and quality modelling using Bayesian networks," Artificial Intelligence Applications for Improved Software Engineering Development: New Prospect, pp 1-25, 2010.
[118] M. Di Penta, "Nothing else matters," in Proceedings of the 7th International Conference on Predictive Models in Software Engineering - Promise , pp. 1–3, 2011
[119] P. Pendharkar, G. Subramanian, J. Rodger, "A probabilistic model for predicting software development effort," IEEE Transactions on Software Engineering, vol. 31 (7), pp. 615–624, 2005.
[120] E. Mendes, "A Comparison of Techniques for Web Effort Estimation," First Internationsal Symposium on Empirical Software Engineering Measures pp. 334–343, 2007.

[121] T. Schulz, Ł. Radliński, T. Gorges, W. Rosenstiel, "Defect cost flow model," 6th International Conference on Predictive Models in Software Engineering - PROMISE, 2010,
[122] Ł. Radliński, "A conceptual Bayesian net model for integrated software quality prediction," Annales. UMCS, Informatica., vol. 11 (4), pp 49–60, 2011.
[123] Ł. Radliński, "Enhancing Bayesian Network Model for Integrated Software Quality Prediction," Fourth International Conference on Information, Process, and Knowledge Management, pp 144–149, 2012.
[124] A. Okutan, O.Yıldız, "Software defect prediction using Bayesian networks," Empirical Software Engineering, p. 1028, 2012.
[125] S. Wagner, "A Bayesian network approach to assess and predict software quality using activity-based quality models," Fifth International Conference on Predictive Models for Software Engineering - PROMISE '2009.
[126] S. Mirarab, A. Hassouna, L. Tahvildari, "Using Bayesian Belief Networks to Predict Change Propagation in Software Systems," IEEE International Conference Program Comprehension, pp. 177–188, 2007.
[127] X. Wang, C. Wu, L. Ma, "Software project schedule variance prediction using Bayesian Network," IEEE International Conference Advances Management Science, pp. 26–30, 2010.
[128] S. Bibi, I. Stamelos, "Software Process Modeling with Bayesian Belief Networks," 10th International Software Metrics Symposium, pp. 14–16, 2004
[129] P. Hearty, N. Fenton, D. Marquez, M. Neil, "Predicting Project Velocity in XP Using a Learning Dynamic Bayesian Network Model," IEEE Transactions on Software Engineering, vol. 35 (1), pp. 124–137, 2009.
[130] M. Neil, N. Fenton, Q. Mary, "Improved Software Defect Prediction," in 10th European SEPG, London, 2005.
[131] F. Jensen, Bayesian Networks and Decision Graphs. Springer, 2007
[132] D. Perry, A. Porter, L. Votta, "Empirical studies of software engineering: A roadmap," in Future of Software engineering - pp. 345–355, 2000
[133] P. Sawyer, I. Sommerville, S. Viller, "Capturing the Benefits of Requirements Engineering," IEEE Software 16(2) pp78-85, 1999.
[134] D. Sjoberg, T. Dyba, M. Jorgensen, "The Future of Empirical Methods in Software Engineering Research," in Future of Software Engineering, pp. 358–378, 2007
[135] V. Basili, "The Role of Experimentation in Software Engineering :past, present and future," 18th IEEE international conference on Software engineering pp. 442–449, 1996.
[136] B. Kitchenham, L. Pickard, S. Pfleeger, "Case studies for method and tool evaluation," IEEE Software, vol. 12(4), pp. 52–62, 1995.
[137] N. Fenton, S.Pfleeger, and R. Glass, "Science and substance: a challenge to software engineers," IEEE Software, vol. 11 (4), pp. 86–95, 1994.
[138] T. Lethbridge S. Sim, J. Singer, "Studying Software Engineers : Data Collection Techniques for Software Field Studies," Empirical Software Engineering, Vol 10 (3) pp. 311–341, 2005.
[139] B. Curtis, "Measurement and experimentation in software engineering," Proc. IEEE, vol. 68 (9), pp. 1144–1157, 1980.
[140] C. Seaman, "Qualitative methods in empirical studies of software engineering," IEEE Transactions on Software Engineering, vol. 25 (4) pp. 557–572, 1999.
[141] B. Nuseibeh, S. Easterbrook, "Requirements engineering: A roadmap," Conference on The future of Software engineering, pp. 35–46, 2000
[142] R. Yin, Case Study Research, 3rd ed. London, Sage, 2003.
[143] M. Host, P. Runeson, "Checklists for Software Engineering Case Study Research," First International Symposium Empirical Software Enginering Measurement, pp. 479–481, 2007.

[144] E. Feigenbaum, The Fifth Generation: Artificial Intelligence and Japan's Computer Challenge to the World. Addison Wesley Longman Publishing Co, 1983.
[145] G. Rugg, P. McGeorge, "The sorting techniques: a tutorial paper on card sorts, picture sorts and item sorts," Expert Systems vol. 22 (3), pp. 94–107, 2005.
[146] S. Fincher, J. Tenenberg, "Making sense of card sorting data," Expert Systems, vol. 22 (3), pp. 89–93, 2005.
[147] C. Paul, "No A Modified Delphi Approach to a New Card Sorting Methodology" Journal Usability Studies, vol. 4 (1), pp. 7–30, 2008.
[148] V. Basili, G. Caldiera, H. Rombach, "The goal question metric approach," Encyclopedia of software engineering, vol. 2, pp 528-532, 1994
[149] M. Fowler, UML Distilled: A Brief Guide to the Standard Object Modeling Language (Object Technology Series). Addison Wesley, 2003
[150] B. Boehm, C. Abts, S. Chulani, "Software development cost estimation approaches – A survey," Annals Software Engineering, vol. 10 (1–4), pp. 177–205, 2000.
[151] A. Gray, S. MacDonell, "A comparison of techniques for developing predictive models of software metrics," Information and Software Technology, vol. 39 (6), pp. 425–437, 1997.
[152] K. Choi, D. Bae, "Dynamic project performance estimation by combining static estimation models with system dynamics," Information and Software Technology, vol. 51 (1), pp. 162–172, 2009.
[153] A. Idri, T. Khoshgoftaar, A Abran, "Can Neural Networks be easily Interpreted in Software Cost Estimation?" IEEE International Conference on Fuzzy Systems, vol 2, pp 1162-1167, 2002
[154] S Wooldridge "Bayesian Belief Networks", Australian Institute of Marine Science, 2003.
[155] S. Morphet, J. Fawcett, K. Bolazar, M. Gungor, "Neural Net Analysis of the Propensity for Change in Large Software Systems," IEEE International Joint Conference on Neural Network Proceedings, pp. 2606–2610, 2006
[156] M. Kellner, R. Madachy, D Raffo "Software process simulation modeling: Why? What? How?" Journal of Systems and Software 46 (2) pp 91-105, 1999
[157] R. Ahmed, T. Hall, P. Wernick, "A Proposed Framework for Evaluating Software Process Simulation Models," pp. 1–10.
[158] N. Singpurwalla, S. Wilson, Statistical Methods in Software Engineering: Reliability and Risk (Springer Series in Statistics). Springer, 1999
[159] B. Kjaerulff, A. Madsen, "Probabilistic Networks for Practitioners- A Guide to Construction and Analysis of Bayesian Networks and Influence Diagrams," Denmark, Aalborg University, Aalborg, Denmark, 2006.
[160] N. Fenton, P. Krause, M. Neil, C. Lane "A Probabilistic Model for Software Defect Prediction", 2001.
[161] J. Pearl, Causality: Models, Reasoning, and Inference. Cambridge University Press, 2000,
[162] D. Heckerman, A Tutorial on Learning With Bayesian Networks, Springer Berlin Heidelberg, 2008
[163] M. Druzdzel, L van der Gaag, "Building probabilistic networks: 'Where do the numbers come from?' guest editors' introduction," IEEE Transactions Knowledge and Data Engineering, vol. 12 (4), pp. 481–486, 2000.
[164] M. Neil, N. Fenton, L. Nielsen, "Building Large-Scale Bayesian Networks," Knowledge Engineering Review, vol. 15 (3), pp. 257–284, 2000.
[165] K. Laskey, S.Mahoney, "Network engineering for agile belief network models," IEEE Transactions on Knowledge and Data Engineering, vol. 12 (4), pp. 487–498, 2000.
[166] S. Bibi, I. Stamelos, L. Angelis, "Bayesian Belief Networks as a Software Productivity Estimation Tool," 1st Balkan Conference on Informatics, 2003.

[167] B. N. Fenton, M. Neil, "Managing Risk in the Modern World." Application of Bayesian Networks, 2007.
[168] M. Christel, K.Kang, "Issues in Requirements Elicitation," Software Engineering Institute CMU/SEI-92-TR-012, 1992.
[169] B. Curtis, H. Krasner, N. Iscoe, "A field study of the software design process for large systems," Communications of the ACM, vol. 31 (11), pp. 1268–1287, 1988.
[170] A. Field, Discovering Statistics Using SPSS (Introducing Statistical Methods series), 3rd ed. Sage Publications Ltd, 2005
[171] E. Barry, S.Slaughter, "Measuring software volatility: a multi-dimensional approach," Twenty first international conference on Information systems, pp. 412–413. 2000,
[172] N. Fenton, M. Neil, J.Caballero, "Using Ranked Nodes to Model Qualitative Judgments in Bayesian Networks," IEEE Transactions on Knowledge and Data Engineeing, vol. 19 (10), pp. 1420–1432, 2007
[173] Norsys Software Corp., "Netica." [Online]. Available: https://www.norsys.com/.
[174] K. Korb, A. Nicholson, Bayesian Artificial Intelligence, Second Edition (Chapman & Hall/CRC Computer Science & Data Analysis). CRC Press, 2011
[175] S. Nadkarni, P. Shenoy, "A causal mapping approach to constructing Bayesian networks," Decision Support Systems, vol. 38 (2), pp. 259–281, 2004.
[176] C. Twardy, K. Korb, "Causal Interaction in Bayesian Networks.", No. 2002/118. Tech. Rep, 2002
[177] M. Jorgensen, "Relative Estimation of Software Development Effort: It Matters with What and How You Compare," IEEE Software, vol. 30(2) , pp. 74–79, 2013.
[178] T. Moynihan, "How experienced project managers assess risk," IEEE Software, vol. 14 (3), pp. 35–41, 1997.
[179] A. van Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications. John Wiley & Sons, 2009
[180] R. S. Pressman, Software Engineering: A Practitioner's Approach. McGraw-Hill Higher Education, 2009,
[181] K. Wiegers, Software Requirements 2. Microsoft Press, 2003
[182] A. Davis and K. Nori, "Requirements, Plato's Cave, and Perceptions of Reality," 31st Annual International Computer Software and Applications Conference, vol. 2, (2), pp. 487–492, 2007
[183] M. Lehman and J. Ramil, "Towards a Theory of Software Evolution - And its Practical Impact," in International Symposium on the Principles of Software Evolution, pp. 2–11, 2000

Appendices

Appendix 1 Requirements Change Source Constructs

Development Trigger Constructs

ID	Development Trigger Construct	Source	Removed	Applicable to Maintenance
45	Use of Prototype	[2], [91]	Technique covered by 21,44	
68	New Stakeholder (role)	[2]		Y
96	Customer Company Reorganisation	[179], [83], [91], [2], [180]		Y
51	New Solution tools/technology	[2], [179], [169], [91], [181]		Y
54	Change to government policy or regulation	[179], [91], [181]		Y
20	Participatory Learning	[91]		Y
28	Local Customization	[91]	New Stakeholder	
50	Customer Migration to new solution	[91]	Type of Requirement	
39	Customer Need Change	[168],[169], [83], [180]	Too woolly, covered by 47,96,54,82,21,45 ,42,90	
44	Developers increased understanding of problem	[169], [83], [7] ,		Y
56	Scope Reduction	[62], [83]	By-product of 88,66	
34	Changes to packaging/licensing/brand ing	[62]	Covered by 61	

65	Solution Elegance (design improvement)	[169], [62]		Y
67	Resolution of miscommunication	[62]		
49	Testability	[62]	Type of Requirement	
82	Business Process Change (continuous improvement)	[97], [181]		Y
42	Response to Competitor	[169], [181]		Y
16	Functionality Enhancement	[83]	Covered by 78,65	
11	Defect Fixing	[83]	Doesn't result in requirement change	
69	Redundant Functionality	[83]	Covered by 66,44,20,90,67,23 ,51,65,21	
8	Missing Requirement Identified	[83]	Not a reason/cause/ source	
86	Clarification of Requirement	[83]	Covered by 67,23	
21	Increased customer understanding	[179], [97], [181], [7]		
72	New Class of User	[179]	Result of other changes covered by 82,68	
74	New usage condition	[179]	Covered by 78,85	
15	New way of doing things	[179]	Covered by 82,96	
77	Correction to requirement specification	[179]	Covered by 23,67	
78	New Opportunity	[179]		Y
1	Change in the use of the information	[7]	Covered by 82	
88	Cost or schedule overrun	[180], [179]		
49	Testability	[83]	Type of	

			requirement	
85	Change to customers hardware/software	[62]		Y
58	System usage(after installation, not prototype)	[2], [97], [91]	Out of scope of development	
90	Changes to market demands	[181], [91], [83], [180]		Y
62	Resolution of conflicting requirement	[83]	Covered by 83	
55	New functional feature	[179]	New requirement	
3	Improved quality feature	[179]	Change to requirement	
14	Change in political climate (needs of particular group emphasized)	[168],[181], [169]		Y
93	Change to customers environment	[97]	Covered by 96,68,85,47,66	
18	Changes in underlying technologies	[169]	Covered by 85,51	
83	Incorrect Requirement Identified	[83]		Y
23	Resolution of misunderstanding	[169]		Y
92	First or re-engagement of representative	Added		
66	Change to business case (Return on Investment, Total cost of ownership)	Added		Y
61	Customer organization strategic change (new marketing/sales direction, change to organization goals)	Added		Y
12	Change of stakeholder representative	Added		Y
4	Understanding technical	Added		

	solution			

Development Uncertainty Constructs

ID	Development Uncertainty Construct	Source(s)	Removed	Applicable to maintenance
31	Analyst Skill or experience	[79], [168] ,[25], [181]		
95	Development team knowledge of business area	[168],[178]		
79	Quality of Analysis techniques employed (workshops, interviews, modeling etc)	[79],[181],[97] [180], [179]		
59	Project size	[168],[7] ,[97]		Y
30	Novelty of product (business novelty)	[178],[181]		Y
41	Logical complexity of problem	[7],[178],[168]		
19	Availability of communication with customer	[7] ,[181]		Y – added words ‘stakeholder’
22	Involved customer knowledge/understanding/clarity of requirements	[168], [182], [178], [181]		
64	Quality of communication between analyst and customer	[168], [7],[182],[181]		
33	Involved customer experience with working alongside IT	[178]		
9	Diverse user community	[178],[97], [181]	Summary of 29,27,40, 41,2,60	
32	Incompatibility between requirements	[179]		Y
24	Lack of well understood model of utilizing system	[7]	Unclear	
6	Lack of Structure for activity or	[7]	Covered	

	decision being supported		by 22	
52	Stability of Customers business environment	[168]		Y
76	COTS usage	[179],[7]		Y
2	All stakeholders identified	[2], [58]		Y
29	All stakeholders involved	[168],[58], [2] ,[181]		Y
40	Clarity/unity of shared product vision	[25],[91], [58],[181]		Y
27	Synergy of stakeholder agenda	[179], [178],[91]		Y
43	Unknown customer project dependencies	[58]		Y
46	Market stability	[169],[182], [91]		Y
13	Differing customer needs	[169]		Y
38	Type of user doing specification (incorrect user involved)	[181], [25],[7]		
89	Change in utilizing system	[7]	Unclear	
80	Low staff morale	[79]		
10	Large number of users	[7]	Not a risk if correct user involved 38	Y
87	Level of participation of users in specification	[7]	Covered by 19, 64	
81	Lack of user experience in utilizing system	[7]	Covered by 22	
63	Degree of change to customers workflow	[178], [181]		Y
48	Quality of requirements specification	Added		
71	Technical uncertainty of solution	Added		
84	Technical complexity of solution	Added		

53	Quality of development team communication	Added		Y- removed word ‘development’
73	Age of requirements (elapsed time since completion of requirements documentation)	Added		
60	Insufficient sample of user representatives	Added		
35	Development team (pm and analysts) stability	Added		

Maintenance Trigger Constructs

ID	Maintenance Trigger Construct	Source	Removed	Equivalent development construct
161	Use of Case Tools	[48]	Technique	
141	Maintenance activities	[48]	Technique	
146	Changes to deployment environment	[52]		
153	New Opportunity	[52], [183],	Covered by 137	
128	Domain changes due to system usage	[183], [98], [91]	Activity	Domain changes due to system development – iterative development only
125	Evolution of surrounding environment	[52]	Unclear	
113	Deferred requirement during development	[88]	A planned enhancement	
109	System usage	[183], [98], [91]	Activity	
142	Changing Customer needs	[183]	Unclear	Variation of development

				constructs
104	Changing technology (for solution)	[183], [55]		New Tools/Technology
116	Increased Customer Understanding	[98]		Increased Customer Understanding
151	New technological methods	[98]	Covered by 104	
119	Organisation Changes	[98]		Company reorganization
138	Change to operational domain	[183]	Covered by 146,104,151	
132	Increased user sophistication	[183]	Covered by 116	
136	Response to competition	[183]		Response to competitor
165	Ambition	[183]	Covered by 143,137	
162	Business Process improvement	[183]		Business process change
101	Migration to other technology environments	[91]	Covered by 146	
144	Function added, replaced or deleted	[55]	Not a cause	
103	Adapt to new technological environment	[55]	Covered by 146	
123	Alter system performance	[55]		Design improvement/solution elegance
147	Alter maintainability	[55]		Design improvement/solution elegance
137	Response to gap in market	Added		Response to gap in market (added)

Maintenance Uncertainty Constructs

ID	Maintenance Uncertainty Construct	Source	Removed	Equivalent Development Construct
166	Business Size	[48]	Covered by 157,163	
121	Maintenance team instability	[48]	Not a cause of req change	
135	System complexity	[48]	Defects, but not req change	
143	In-house software	[79]	Increase change capability but not a cause for change	
158	Maintenance team knowledge	[50]	Not a cause	
117	Cohesive Architecture	[50]	Effects defects but not req change	
148	Presence of competitor	[52]		Presence of competitor (added)
118	Market Environment	[52]		Market stability
154	Semantic relativism	[88]		Semantic relativism (added)
115	System age	[88]		
102	Period	[88]	Unknown	
167	Economic climate	[183]	Effects ability for change but not cause	
164	COTS usage	Added		COTS usage
157	No of interfaces	Added		Project Size
139	Diversity of user needs	Added		Differing

				customer needs
156	Stakeholder agreement	Added		Synergy of stakeholder agenda
163	No of functions	Added		Project size
221	Quality control during development	Added		Quality control during development (added)
102	Function Usage	Added		Function Usage (iterative development only)
126	No of Users	Added		No of Users (iterative development only)

Appendix 2 Variables in Predictive Models

Variable	Description
All Stakeholders Involved	Stakeholders can be broken down into Major and Minor stakeholders - Major Stakeholders are the drivers behind the need for the new solution and would likely include the business sponsor and business process owners. Minor Stakeholders are not driving the core requirements for the system, but have an interest either as an interface system and in the role of compliance, IT, etc. Actively involved means the stakeholder roles are proactive in validating, verifying the requirements and providing the appropriate level of resource to do so. Informed stakeholders are kept in the loop but are not forward coming with owning their particular requirements. How involved are stakeholders? 1 Less than 10% 2 80% Key and 30% minor Stakeholders involved 3 All Key Stakeholders involved, all minor Stakeholder informed 4 All known stakeholders informed, 80% minor stakeholders engaged 5 All known stakeholders involved fully involved and informed
Synergy Stakeholder Agenda	Have the Stakeholders aligned agendas? and where natural conflict occurs (e.g. compliance vs. cost of running) have compromises been made that are understood and agreed by everyone? 1 Clash of key stakeholder agendas 2 Major Stakeholder Agendas mostly aligned with agreed compromises 3 Major Stakeholder Agendas aligned with no compromises 4 Major and minor Stakeholder agenda's aligned although compromises made 5 Major and minor Stakeholder agendas fully aligned
Stakeholder Communication	Ability of Stakeholders to effectively communicate with each other, barriers include separate locations, separate cultures, conflicting business drivers, poor communication tools (word of mouth, email, paper, electronic docs, video conferencing, collaboration tools, web cams), No. Stakeholders. How well do stakeholders communicate? 1 All Stakeholders, geographically, culturally separated and don't share business objectives 2 All key stakeholders , geographically, culturally separated and don't share business objectives 3 All key stakeholders , geographically and share business objectives 4 All Stakeholders co-located, 4 or more, talk on a day to day basis and understand each other's business requirements 5 All Stakeholders co-located, less than 4, talk on a day to day basis and understand each-others business requirements
All Stakeholders Identified	What is the likelihood that additional Stakeholders may be identified after the project has started and therefore change needed to accommodate their requirements? 1 Very Low

	2 Low 3 Medium 4 High 5 Very High
Stakeholder Understanding of Problem	Does the customer really understand empirically just what problems they are experiencing, what the solution should look like and how they are going to measure that they got there? 1 Very Low 2 Low 3 Medium 4 High 5 Very High
Stakeholder Understanding IT	Does the customer understand that IT is not a golden bullet, will need to work with the project team to articulate the requirements, may need to modify the requirements for a cost effective solution and that the development is often iterative and does not have unreasonable expectations? 1 No knowledge of IT 2 Use IT but never been involved with specification 3 Involved with IT specification in a minor role 4 Involved with a small number of projects in a major role 5 Involved with a large number of projects in a major role
Business Dependency	What is the persuasiveness of this requirement, what is the chance if another requirement changes that there will be a knock on effect to this requirement? 1 Requirement has no dependency to any others. 2 Requirement has dependencies with non-functional requirements, for example, security, performance. 3 Requirement may have dependencies upon non-k functional requiren and also dependencies to other requirements within a single business functional area, e.g.. Maintain address, Manage messages. 4 Requirement had dependencies to other requirements within a single business function and consumes services from generic components (those without business rules). 5 Requirement is a consumer of many other requirements throughout t solution.
Business Novelty	How novel is the requirement to existing business practices within the organisation or to other common, in use, practices? 1 Part of Day Job. 2 Occasionally used. 3 Occasionally used but not in this job 4 Familiar with concept. 5 Entirely new idea.

Business Complexity	How complex is this requirement to understand - how many facets to it. How niche to the business is it? 1 Simple 2 Some thought needed 3 Can be done whilst multi-tasking 4 Challenging 5 Very Challenging
Technical Dependency	Does the coded requirements rely upon, or is used by other coded requirements? 1 Coded Requirement has no dependency to any others. 2 Coded Requirement has dependencies with non-functional requirements, for example, security, performance. 3 Coded Requirement my have dependencies upon non- functional requirements and also dependencies to other requirements within a single business functional area, e.g.. Maintain address 4 Coded Requirement had dependencies to other requirements within a single business function and consumes services from generic components (those without business rules). 5 Coded Requirement is a consumer of many other requirements throughout the solution.
Technical Novelty	How novel is the technical requirement to the design, build and test teams? 1 Very similar to something done before by many of the team 2 Some of the team have done something similar 3 We have examples but no direct experience 4 We have no examples and no experience but it has been done by oth 5 We are going to attempt something entirely unknown
Technical Complexity	How complex is the technical part of this requirement to understand? 1 Simple 2 Some thought needed 3 Can be done whilst multi-tasking 4 Challenging 5 Very Challenging
Framework Usage	How much does this requirement need unique thought against it vs. how much out of the box usage? 1 100% custom code - bespoke 2 mostly custom code 3 mix of custom code and in-built features 4 Mostly parameterisation - will use a package and change settings to more closely match with our requirement 5 Out of the Box, no changes needed - Will adapt to package features
Team Morale	How much rotation of project members through the team / customers through the team, Staff motivation, and attitude 1 This is a doomed project, people want to leave and High level of staff turnover 2 Project would be okay if they would just stop rotating people out of it 3 Project is looking good and just normal temporary poaching of junior members 4 Good project , just need to cope with getting the

	new guys on board 5 We have been one team since the beginning and are really excited about the positive difference it will make
Senior Management	How much support are senior management showing the team and helping remove obstacles? 1 I am convinced that senior management want to see this project die, continually being poached of resources for other project and none of our requests are ever answered 2 We get sporadic project support from management 3 Management can sometimes be supportive of our problems 4 Generally management are quite quick to remove any impediments 5 CEO is 100% behind us, gets us everything we ask for and is often around smiling and finding out how we are doing
Tools Quality	What is the quality of the tool use in validating, verifying and managing individual and groups of requirements (missing, duplicate, conflicting requirements)? Applies to Business, Specification, Systest, Custest, Build 1 We have to do everything manually 2 We are mostly manual but have to maintain many separate cross reference spread sheets 3 Kept centrally, electronically and we maintain electronic pointers to master list 4 For individual requirements, we enter a requirement once, it is checked syntactically, for appropriate adjectives and validated against the separate object model 5 For all individual and for group requirements we confirm that all parts of the life history can be accounted for, there is no overlap or conflicts.
Process and Technique Quality	Processes need to be refined with feedback and appropriate level of governance, techniques used for verifying that we have captured the right thing, validating that it is what is wanted, checked to make sure nothing is missed for both individual and groups of requirements. Is this the case? Applies to Business, Specification, Systest, Custest, Build 1 Left up to each individual on how and what to do 2 Rough guide as to what needs to be done 3 Checklists for all core activities 4 Fairly mature process 5 Right sized and detailed REQM, RD process, easy to use and Has been continually improved upon over many projects. Supported by quick reference checklists

Process Constraints	What constraints are on the process that may lead to sub -standard performance, this includes time pressure (do things fast and to the 80/20) and commercial pressure that may disincentives the discovery of change / clarification (e.g. fixed price, wanting change to increase revenue vs. fixed price not wanting to incur any more cost with investigation….) What is the chance that the process will be followed? Applies to Business, Specification, Systest, Custest, Build 1 I know the estimate is showing 100 days, but it must go live in 60. 2 We are tight on time and we don't want to add any changes into this version of the solution because we will incur penalties 3 We can afford some change and there is scope in time and budget for this, but would prefer to capture large changes for a later release. 4 We can afford some change and there is scope in time and budget for this. The process must be followed, it will be audited and if cost or time overruns occur due to the process there will be no problem. 5 The customers wants the right solution, even if it costs more or takes longer - they have budget to cover this.
Inter-team Communication	How effective is the communication of requirements between project team members and project team to customer? Includes geographic separation, number of different locations, cultural gaps, number of people involved, tools available to mitigate gap 1 Very Low 2 Low 3 Medium 4 High 5 Very High
No of Requirements	How many requirements in this group? The more requirements in a group the harder it is to understand the big picture, their interrelatedness and possible overlaps/conflicts and therefore the greater effort and tool support needed 1 Very Low 2 Low 3 Medium 4 High 5 Very High
Requirement Group Complexity	How interconnected are the requirements in the group, can they be thought of individual and atomic or is there a high degree of interrelatedness and 1 change can ripple through many others and therefore reducing the chance of understanding both of the requirement and the impact of change? 1 Each Requirement Stands Alone 2 Very loosely coupled 3 There are several interrelated groups of requirements 4 Requirements are all linked together but can be understood in isolation 5 Very complicated relationships in this group

Incoming work product Quality (Vis, Spec, Build)	How clear and unambiguous is the way the Vision requirements are communicated to the Req Spec team? 1 It is all word of mouth 2 Documented briefly but sometimes conflicting and poor readability scores 3 Documented briefly with good readability scores, some models and some examples 4 Documented in more detail with good readability scores, some models and some examples 5 Documented in great detail, supported by cross referenced models, Prototypes, and pointers to similar systems
Team (analyst/ solution) Skill	Roles: Analyst, Developer, System Tester Customer Tester How well can this individual (or group of individuals), translate higher level work product into a dependant work product (or provide verification, validation, QC activities)? 1 Rudimentary 2 Understanding but some supervision needed 3 Work unsupervised 4 Competent and can train others 5 Very experienced and can publish on the subject
Team (analyst/ solution) Knowledge	Roles: Analyst, Developer, System Tester Customer Tester How well does this individual (or group of individuals),understand their respective domains (e.g. business knowledge, COTS/Framework knowledge, Software Language)? 1 No knowledge of area 2 Some knowledge of similar area 3 Some knowledge of area 4 Worked in area for less than a year 5 Many years experience in area resulting in deep understanding
Level Ongoing Comms	What techniques are used to keep Stakeholders and the wider audience informed of the current requirements shape of the solution? 1 We only communicate progress to schedule and cost 2 At the monthly project board we do a very short presentation on what the system currently looks like. 3 We maintain a project site where people can logon and see the current base-lined requirements document. 4 We maintain an information radiator both in a main corridor and electronically. And maintain a FAQ and are available to answer questions. 5 The project team regularly have brown bag lunches, open demos where we show the mock-ups of the system and demo the system to date. The Stakeholders also have access to the demo system so they can experiment with the solution to date.