Sources of Software Requirements Change from the Perspectives of Development and Maintenance

Sharon McGee[1] and Des Greer[2]
School of Electronics, Electrical Engineering and Computer Science
Queens University
Belfast, United Kingdom
{[1]smcgee08|[2]des.greer}@qub.ac.uk

Abstract— **Changes to software requirements occur during initial development and subsequent to delivery, posing a risk to cost and quality while at the same time providing an opportunity to add value. Provision of a generic change source taxonomy will support requirements change risk visibility, and also facilitate richer recording of both pre- and post-delivery change data. In this paper we present a collaborative study to investigate and classify sources of requirements change, drawing comparison between those pertaining to software development and maintenance. We begin by combining evolution, maintenance and software lifecycle research to derive a definition of software maintenance, which provides the foundation for empirical context and comparison. Previously published change 'causes' pertaining to development are elicited from the literature, consolidated using expert knowledge and classified using card sorting. A second study incorporating causes of requirements change during software maintenance results in a taxonomy which accounts for the entire evolutionary progress of applications software. We conclude that the distinction between the terms maintenance and development is imprecise, and that changes to requirements in both scenarios arise due to a combination of factors contributing to requirements *uncertainty* and events that *trigger* change. The change trigger taxonomy constructs were initially validated using a small set of requirements change data, and deemed sufficient and practical as a means to collect common requirements change statistics across multiple projects.**

Keywords- Requirements change; requirements management; project management; card sorting; software evolution; development; maintenance.

I. INTRODUCTION

To some, effective management of changes to software during its lifetime is the key to the effective software project management [43]. While accepting that requirements changes are inevitable during software development, the increased cost of changes later in the development lifecycle [53][2], combined with the threat that volatility poses to project schedule, cost [3][4], and defect rates [5][4], means that requirements volatility constitutes one of the top ten risks to successful project development [6]. Continuing post-delivery, constant adaptation and change is necessary if software is to retain value and remain useful [38]. 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.

Working with an industrial partner, our shared objective is to design and conduct a series of studies that collectively address the challenge of requirements change anticipation. Our longer term aims are 1) To investigate the correlation between the source of change and requirement type, 2) To assess the impact of change source upon requirements volatility and 3) To examine the pattern of source-induced change during development and maintenance. The first step is an exploration of the causes of requirements change, both pre- and post delivery.

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]. An obvious starting point would therefore be to analyse existing change control databases. However, it has been observed that reasons for change are insufficiently recorded *for the purpose of analysis* [9]. While this statement cannot be said to apply generally, it has also been the experience of the authors. Standardizing data collection across multiple projects regardless of life-cycle phase will not only inform explorative research, but also provide a means by which industrial software providers can take ownership of empirical opportunities. In this study we set out to build a taxonomy of requirements change based on the source of the change, including and comparing sources of change during software development and maintenance. 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 within one organization, for the purpose of future analysis.

Thus, the following questions are addressed:-

1. What are the sources of requirements change during software development and maintenance?
2. Can they be similarly classified according to change source domain?

This paper combines a previous study [1] with new results and is organised as follows. Section 2 describes previous studies related to the classification and causes of requirements change. Section 3 outlines the research approach and methods used in this study. In Section 4 we establish the software project categorisation used in this study. Section 5 describes the research process, and illustrates the derived taxonomy. Section 6 discusses our findings with respect to previous work and outlines possible application limitations. Finally we end our paper with conclusions and plans for further work.

II. RELATED WORK

More abstract theories suggest that requirements change because our perceptions of reality differ from actual reality [32], or that the real world is unbounded yet our understanding of the world is both bounded and based upon assumptions which are often invalid [38].

Empiricists, seeking to complement these ideas with more practical support, explore the causes of requirements change by examining evidence during software development and maintenance. Studies designed to classify requirements changes fall into one of two camps. The first are those that advocate the need for a domain-specific taxonomy. Lam et al [10-12], who address the problem of managing volatility by process control, recommend that volatility classification should capture the domain-specific nature of change in order to facilitate change estimation and reuse. This is echoed by Stark [13] who analyses the impact of maintenance changes on release schedule. The following discussion focuses upon those studies such as Harker et al. [14] which propose a more generic re-usable requirements change classification.

A. Software Development Change Classifications.

Harker et al. [14] 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 et al.'s study, an appraisal by Sommerville [15] 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 et al [16] 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 changes [9]. 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.

As can be seen there is little agreement in the terminology used for classifying requirements change, and it would seem at first sight that studies to date have little commonality. 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 et al.'s change reason of 'missing requirement' is included within Harker el al.'s change source of 'increased understanding'.

A genre of studies related to requirements engineering risk and uncertainty is also of relevance. Mathiassen et al. [17] classify requirements engineering risks by reliability, complexity and availability, and relate these to appropriate techniques.

B. Software Maintenance Change Classifications

Much empirical and theoretical work re-uses or builds upon Swanson's classification [41] of maintenance changes [34][44][46], which includes corrective, adaptive and perfective changes. Chapin et al. [42] provide a thorough review of literature referring to maintenance change types, and propose a new classification which is an extension and clarification of previous work, and is based upon observed activities. These include changes to documentation, code, and business rules. Incorporating both errors and enhancements, this classification focuses not upon the reason, cause or source of the change, but instead upon the type of change being made. Both Kemerer & Slaughter [44] and Heales [37] take a different approach and classify changes according to what is being changed. From a theoretical view point, Perry [39] 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 are subsequently implemented upon a foundation of sometimes weak 'technical theory'.

Due to the divergence of change sources compiled in these studies, none of the classifications exclusively met the needs of the subsequent stage of this research. However, their findings, together with requirements change causes derived from other studies are used to provide a collection of change constructs upon which to base our classification effort. A full list of change source constructs elicited from the literature can be found in the appendix.

III. RESEARCH APPROACH

This study is the first of a family of studies [18] employing a collaborative research approach, in that it seeks to contribute to the body of knowledge in this area, whilst answering to the need of our industrial partner to better understand, manage and measure requirements changes. Collaboration with industry is generally recommended to ensure relevance and better transfer of research results [19]. In this instance the industrial partner gave of their time to provide expert knowledge of software project management and product maintenance.

A. Preliminary Studies

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.

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 experience in IT.

B. Organisational Context

Our industrial partner in this research employs 300 staff, 136 of whom are involved with software development. They have 6 offices around the UK and Ireland and deliver 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 governmental work. Nearly all project managers are Prince2 certified and work with a range of traditional and agile methodologies.

C. Workshops

In requirements engineering, group elicitation techniques such as workshops aim to foster stakeholder agreement and buy-in [20], and are a mechanism whereby individuals can make decisions through the consensus building leadership of a facilitator [21]. 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.

D. Card Sorting

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 [21] 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 [54] argue that card sorting overcomes one of the disadvantages of the repertory grid method of categorisation since this uses Likert-type measurements to capture participant responses and is not well suited to nominal scale data. However, the repertory grid approach does lend itself easily to statistical analysis, which is one of the challenges of card sorting [22]. 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) [55] 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 [22]. A special edition of the journal 'Expert Systems' in 2005 [23] 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.

IV. 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 [35], we firstly derive a character based project categorization founded upon existing studies. It is from this basis that we establish understanding between academic and industrial research team members and consider the validity of the results of this study.

A. Software Evolution

Lehman's influential and continuingly relevant work on software change [38][45] 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" [47], Belady & Lehman are deliberately inclusive of all stages of the software lifecycle, including initial development [38][43]. Subsequent to this work, authors have applied the term to development [48], used it as a substitute for maintenance [34][44], and proposed that it refers to a period of time between initial development and servicing [35]. Noting that the term lacked a standard definition, Bennett & Rajlich [35] sought to clarify its meaning by asking the question "What is maintenance?" and proposing a staged model for the software lifecycle [35]. This 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 context is specified, and secondly explore the best solution for each stage. Interestingly, in a retrospective examination, Lehman & Ramil observed that their empirical research supported the staged model [38].

B. Software Development and Maintenance

The term maintenance has been defined by the IEEE [49] 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 Godfry & German [36], 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 et al. [46] developed an ontology of maintenance in which two scenarios are outlined. The first scenario (A), 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 (B) 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 [50] 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 et al. [42] 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 et al. [46] is derived from and bears direct semblance to a development process ontology proposed by de Almeida et al. [52]. 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 et al 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 [51]. Kemerer & Slaughter 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 [44].

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, the observations made by Bennett & Rajlich [35] Chapin [51] and Kemerer & Slaughter [44], who advocate the benefit of differentiating between life-cycle phases, are of sufficient significance to warrant empirical investigation.

While an exhaustive account of the comparison between development and maintenance is out of the scope of this study, the categorisation illustrated in Table 1 was derived for the purposes of this and future empirical studies. It combines the staged model proposed by Bennett & Rajlich [35], Kitchenham et al.'s maintenance scenarios [46] and Chapin's classification of change types [42]. The division between development and maintenance was drawn to reflect the importance of the factors relating to team knowledge, stability and responsibilities [46][51], coupled with the distinct contractual governance prevalent during 'product upkeep' and 'servicing'. From Table 1 we derive 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 and enhancement to production software.

V. TAXONOMY DEVELOPMENT

This section describes the process of taxonomy development. Upon agreement of the proposed categorisation, a consideration of the sources of requirements changes observed during 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.

A. 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.

TABLE 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 et al. [46]

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 et al.'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' undertaken under the terms of the maintenance contract. The project categorisation as proposed was used in the remainder of the study.

B. 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' [24] would not have been included. The authors felt that this would be an oversight.

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' [14, 15, 25]. 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 can not 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).

C. 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 the Appendix 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.

D. Participant Card Sorting (development)

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 which could be seen clearly in the photographs. 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.

E. Second Workshop- Consesus 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. Therefore, the final taxonomy was built according to the variance of classifications made during the card sorting procedure. Consequently, a taxonomy comprising 5 groups was derived and agreed.

These groups comprised the change domains illustrated in Table 2. 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.

TABLE 2 CHANGE DOMAINS AND DESCRIPTIONS

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.

Nonetheless, as discovered by Curtis et al. [25] '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.

F. 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% [35]) 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 the argument of Kitchenham et al. [46] 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.

G. 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 36 constructs to 11 triggers and 12 uncertainties (see the appendix). 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. 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 added to reflect 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. 'Function Usage' was also added since system functions used more frequently are prone to higher levels of change.

H. Fouth 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 [22] within a workshop setting. Provided with the change domains derived previously and described in Table 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 [37] 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.

I. 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.

1) Maintenance and Development Construct Consolidation.

Seven of the 12 maintenance related triggers, and 6 of the 12 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, the difference lay in the effects of the uncertainty. For example, while 'project size' affected the capability of the development team to understand and model the problem, from the perspective of the maintenance team it increased the likelihood for change discovery during maintenance.

2) Development and Maintenance construct review

Only two sources of maintenance related requirements change - 'semantic relativism' and 'response to gap in market' were deemed applicable to initial software development. However, when taking iterative development into consideration, the constructs relating to system usage also became 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 within the realm of software maintenance related only to system age.

However, many of the constructs pertaining only to 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. However, it was also noted that many of these sources, particularly those in the domains of *market* and *project vision* would result in the initiation of a new product release. 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.

The resulting taxonomy is shown in figure 1. The reader is referred to the appendix 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. Those constructs marked '(D)' apply solely to development, while those marked '(M)' are relevant only to maintenance. 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'.

J. 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. This taxonomy will firstly be used within the context of development, assessed for informative capability and internal validity, before considering the broader scope of applicability. Further validation of this taxonomy is the subject of an on-going study using a current project.

VI. DISCUSSION

This section evaluates the taxonomy thus derived with respect to previously published change classifications, explores the implications of the study with respect to the comparison between development and maintenance and outlines some possible limitations of this work.

A. Comparison with Previous Classificatons

The classification proposed in this study bears little synergy with change reasons derived by Nurmuliani et al. [16] 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 et al. [14]. 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". 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 et al.'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.

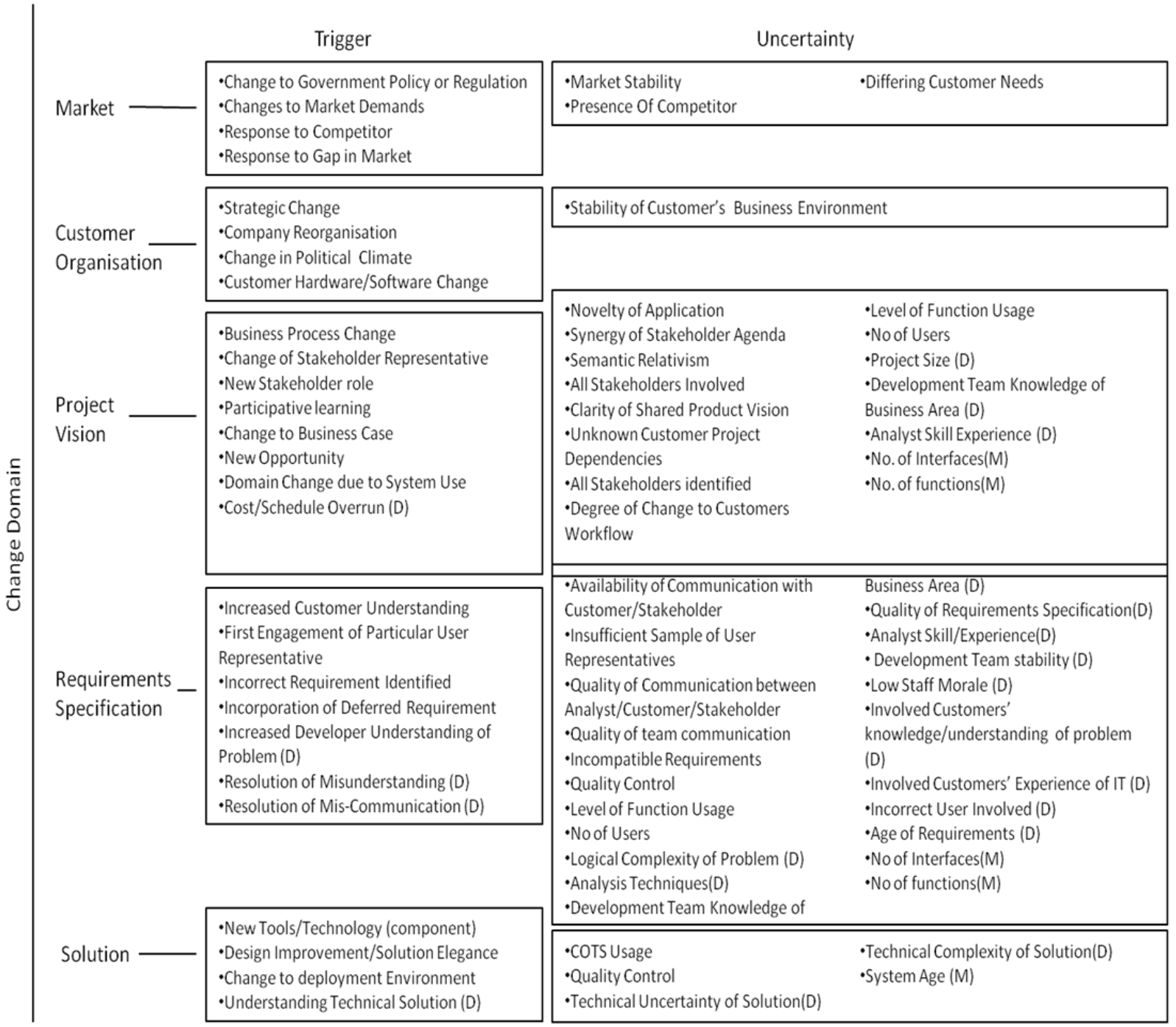

Figure 1 Requirements Change Source Taxonomy

There are no analogous domains within this taxonomy for the remainder of Harker et al.'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 [15], 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 et al. [17] 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 [42]. 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 [39]. 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.

Of particular significance for our on-going research is that by comparison to the descriptive or uncertainty based nature of previous work, the clearly defined 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.

B. Comparison between development and Maintenance

Having derived a project categorisation based upon the work of Kitchenham et al. [46], Bennett & Rajlich [35] and Chapin et al [42] (refer to Table 1), the taxonomy derived in this study verifies that many requirements change sources are similarly relevant to development and maintenance. Thus supporting Bennett & Rajlich's [35] observation that software evolves during both iterative development and maintenance, the differentiation presented by Kitchenham et al. [46] 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 & Rajlich's [35] model, not as a lifecycle stage, but as an iterative element of evolutionary product versioning. Refuting Kitchenham et al.'s contention that maintenance changes are event driven, while changes during software development are requirement driven [46], 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, 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.

C. Limitations

Generality of results are often sacrificed for richness and complexity, reflecting an inherent conflict between internal and external validity [19]. 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. Motivated, however, by the potential for improvement to requirements change visibility and management, modelling change sources is a worthy initiative. The collaborative approach taken here has led to an internally usable model and reflects Sjoberg's et al.'s recommendation [19] to "formulate scope relatively narrowly to begin with and then extend it gradually". Therefore no claims can be made with regard to external validity 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. The initial constructs are provided here, along with methods description such that it should be possible to replicate this study. Given the collaborative nature of this research, and its immediate applicability, it has a high level of relevance.

VII. CONCLUSION AND FURTHER WORK

This study set out to explore, classify and compare the causes of requirements change during software development and maintenance. A review of the terminology highlighted the fuzzy distinction between projects termed 'development' and those referred to as 'maintenance'. The disparity of terminology in the literature is complemented, and to some extent explained by the lack of distinction observed in industry. Project nomenclature is decreed dependent upon the size of the proposed change, and the supporting funding agreement. This carries the implication that research in the field of software development may apply to software maintenance and vice versa. Further, that establishing context in empirical research requires more than a reference

to development or maintenance in a research article. A somewhat narrower view of maintenance was defined in this study which excluded evolutionary development and reflected the naming convention used by our industrial partners.

Expert knowledge of experienced project managers and maintenance engineers was used to consolidate and classify change source constructs elicited from the literature. 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. In itself, this taxonomy supports project risk visibility and facilitates the collection of clearly defined change source data. In differentiating between source domains pertaining to market, customer and project vision a software provider using this taxonomy can assess the level of changes that are arising due to a *change in the direction or vision* of the problem, by contrast to those pertaining to an *increased understanding* of the problem to be solved. In so doing, project managers can make use of internal empirical data to support process and technique selection, and risk management.

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. A comparative exploration revealed that most of the change constructs applied to both development and maintenance, though it was observed that the effects of the uncertainties differed, and that some of the changes incurred during maintenance would necessitate a new product release requiring software development. Those constructs relevant solely to development related to requirements and domain understanding, while those pertaining only to maintenance were concerned with system age. By contrast to the contention that software maintenance changes are event driven while development changes are requirement driven, an implication of this study is that changes to requirements are driven by a combination of event and uncertainty during both development and maintenance. Further, opportunities for requirements change may be sought pro-actively in both situations.

This study was founded upon previously published requirements change taxonomies, thus evaluating and building upon their efforts. Therefore it addresses the problems of divergent change source constructs, and reasons that some of the classifications previously described as 'causes' were either consequences of other changes, types of requirements, or more abstract concepts less easy to evaluate.

Having answered the questions posed in this study, it is now possible to further our research and begin exploring what kinds of requirements are more susceptible to change arising within the change domains defined in this taxonomy. This is currently on-going with our industrial partner. A further study is envisaged which will explore patterns of requirements change throughout the evolutionary progress of software development and usage. The theoretical aspect of the work presented here may contribute to ontological studies, and open more issues in relation to emerging paradigms such as dynamic updates in Service Oriented Architecture and alternative delivery models such as cloud computing. In the meantime the derived taxonomy can assist practically in the identification and analysis of requirements volatility and has particular relevance to customer driven software development especially those working within the government sector.

ACKNOWLEDGMENT

We would like to thank the project managers and maintenance engineers whose valuable time was devoted to the organization of this taxonomy.

REFERENCES

[1] S. McGee, D. Greer, "A Software Requirements Change Source Taxonomy", proc. 4th Intl. Conf. Software Engineering Advances, Porto, 2009.

[2] B. Williams, J. Carver and R. Vaughn, "Change Risk Assessment: Understanding Risks Involved in Changing Software Requirements", Proc. International Conference on Software Engineering Research and Practice, Las Vegas, Nevada, 2006.

[3] D. Zowghi and N. Nurmuliani, "A study of the impact of requirements volatility on software project performance", Proc. Ninth Asia-Pacific Software Engineering Conference, 2002.

[4] S. Ferreira, F. Collofello, D. Shunk, G. Mackulac and P. Wolfe, "Utilization of Process Modeling and Simulation in Understanding the Effects of Requirements Volatility in Software Development", International Workshop on Software Process Simulation and Modeling, Portland, Oregon, 2003.

[5] T. Javed, M. Maqsood and Q. Durrani, "A study to investigate the impact of requirements instability on software defects", ACM Software Engineering Notes, 29, 3, 2004.

[6] B. Boehm," Industrial Software Metrics Top 10 List", IEEE Software, 4(5), 1987.

[7] S. L. Pfleeger, "Software Metrics: Progress after 25 Years? ", IEEE Software, 25(6), 2008.

[8] K. Wiegers, Software Requirements, Microsoft Press, 2003.

[9] N. Nurmuliani, D. Zowghi and S. P. Williams, "Using card sorting technique to classify requirements change", Proc. 12th IEEE International Conference on Requirements Engineering, Kyoto, Japan, 2004.

[10] W. Lam, M. Loomes and V. Shankararaman , "Managing requirements change using metrics and action planning", Proc. 3rd European conference on Software Maintenance and Reengineering, Amsterdam, Netherlands, 1999.

[11] W. Lam and V. Shankararaman, "Requirements change: a dissection of management issues", Proc 25th EUROMICRO Conference, Milan, Italy, 1999.

[12] W. Lam and V. Shankararaman, "Managing change in software development using a process improvement approach", Proc. 24th Euromicro Conference, vol 2, Vasteras Sweden, 1998.

[13] G. Stark, A. Skillicorn and R. Ameele, "An Examination of the Effects of Requirements Changes on Software Releases", CROSSTALK, The Journal of Defense Software Engineering, 1998.

[14] S. D. P. Harker, K. D. Eason and J. E. Dobson, "The change and evolution of requirements as a challenge to the practice of software engineering", Proc. IEEE International Symposium on Requirements Engineering, San Diego, CA, USA, 1993.
[15] I. Sommerville, Software Engineering, Personal Education Ltd, 2007.
[16] N. Nurmuliani, D. Zowghi and S. Powell, "Analysis of requirements volatility during software development lifecycle", Proc. Australian Software Engineering Conference, Melbourne, 2004.
[17] L. Mathiassen, T. Saarinen, T. Tuunanen and M. Rossi , "Managing Requirements Engineering Risks: Analysis and Synthesis of the Literature", Helsinki School of Economics Working Papers W-379, 2004.
[18] D. Perry, A. Porter and L. Votta, "Empirical Studies of Software Engineering: A Roadmap", Proc. 22nd International Conference on Software Engineering, Limerick, Ireland, 2000.
[19] D. I. K. Sjoberg, T. Dyba and M. Jorgensen, "The future of Empirical Methods in Software Engineering Research", Proc. Future of Software Engineering, Minneapolis, MN, 2007.
[20] B. Nuseibeh and S. Easterbrook, "Requirements Engineering: A Roadmap", Proc 22nd International Conference on Software Engineering, Limerick, Ireland, 2000.
[21]N. A. M. Maiden and G. Rugg. "ACRE: selecting methods for requirements acquisition", Software Engineering Journal, 11(3), 1996.
[22] S. Fincher and J. Tenenberg, "Making sense of card sorting data", Expert Systems, 22(3), 2005.
[23] Anonymous, Expert Systems Special Edition on Card Sorting, 22, 3, 2005.
[24] M. Christel and K. Kang, "Issues in Requirements Elicitation", Technical Report No. CMU/SEI-92-TR-012 Software Engineering Institute, 1992.
[25] B. Curtis, H. Krasner and N. Iscoe, "A field study of the software design process for large systems", Communications of the ACM, 31(11), 1988.
[26] H. Barki, S. Rivard and J. Talbot, "Toward an assessment of software development risk", Journal of Managment Information Systems, 10(2), 1993.
[27] B. Boehm, "Requirements that handle IKIWISI, COTS, and rapid change", Computer, 33(7), 2000.
[28] A. Lamsweerde, Requirements Engineering : From System goals to UML models to software specifications, John Wiley & Sons Ltd, 2009.
[29] R. Pressman, Software Engineering. A Practitioner's Approach, McGraw Hill, 2005.
[30] T. Moynihan, "'Requirements-uncertainty': should it be a latent, aggregate or profile construct?" Proc. Australian Software Engineering Conference, Canberra, ACT, Australia, 2000.
[31]T. Moynihan. "How experienced project managers assess risk", IEEE Software, 14(3), 1997.
[32] A. M. Davis and K. V. Nori, "Requirements, Plato's Cave, and Perceptions of Reality", Proc. International Conference on Computer Software and Applications, Beijing, China, 2007.
[33] C. Ebert and J. De Man, "Requirements uncertainty: influencing factors and concrete improvements", Proc. 27th International Conference on Software Engineering, ST Louis, Missouri, USA, 2005.
[34] E. Barry, "Software evolution, volatility and lifecycle maintenance patterns: a longitudinal analysis synopsis", Proc International Conference on Software Maintenance, 2002
[35] K. Bennett, V. Rajlich, "Software Maintenance and Evolution: a roadmap", Proc 22nd International Conference on Software Engineering, ACM Press, New York, 2000.
[36] M. Godfry, D. German, "The past, present, future of software evolution", Frontiers of Software Maintenance, 2008, FoSM 2008.
[37] J. Heales, "Factors Affecting Information System Volatility", Proc. 21st Intl. Conf. Information Systems, Brisbane, Australia, 2000.
[38] M. Lehman, J. Ramil, "Software Evolution", Information Processing Letters, 88(1-2), 2003
[39] D. Perry, "Dimensions of Software Evolution",n Proc. Intl. Conf. Software Maintenance, Victoria, Canada, 1994.
[40] M. Lehman, J. Ramil, "Towards a Theory of Software Evolution – And it's Practical Impact"
[41] E. Swanson, "The dimensions of Maintenance", Proc. 2nd Intl Conf Software Engineering, CA USA, 1976.
[42] 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, 13(1), 2001.
[43] M. Lehman, "Software's future: managing evolution, IEEE Software, 15(1), 1998.
[44] C. Kemerer, S. Slaughter, "An empirical approach to studying software evolution" IEEE Transaction on Software Engineering, 25(4), 1999.
[45] M. Lehman, J. Ramil, P. Wernick, D. Perry, W. Turski, "Metrics and Laws of Software Evolution - The Nineties View", Fourth International Software Metrics Symposium (METRICS'97), 1997.
[46] **.** Kitchenham, G. Travassos, A. von Mayrhauser, F. Niessink , N. Schneidewind, J. Singer, S. Takada, R. Vehvilainen, H. Yang, "Towards an ontology of software maintenance", Journal of Software Maintenance: Research and Practice 11(6), 1999.
[47] L. Belady, M. Lehman, "A model of large program development", IBM Systems Journal 15(3), 1976.
[48] K. Villela, J. Doerr, A. Gross, "Proactively managing the Evolution of Embedded System Requirements", Proc. 16th International IEEE Conf. Requirements Engineering, Caltunya, 2008.
[49] IEEE std. 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology, IEEE, New York, 1991
[50] V. Basili, "Viewing Maintenance as Reuse-Oriented Software Development**"**, IEEE Software 7(1), 1990.
[51] N. Chapin, "Productivity in Software Maintenance", Proc. National Computer Conference, Illinois, 1981.
[52] F. de Almeida, S. de Menezes, A. da Rocha, "Using ontologies to improve knowledge integration in software ebgineering environments", Proc. 2nd World Multiconference on Systemics, Cybernetics and informatics, 1998.
[53] N. Nurmuliani, D. Zowghi and S. P. Williams, "Requirements Volatility and Its Impact on Change Effort: Evidence-based Research in Software Development Projects", Australian Workshop on Requirements Engineering, Adelaide, 2006.
[54]G. Rugg, P. McGeorge, "The sorting techniques: a tutorial paper on card sorts, picture sorts and item sorts", Expert Systems, 22(3), 2005.
[55] C. Paul, "A modified Delphi approach to a new card sorting methodology", Journal of Usability Studies, 4(1), 2008.

Appendix

ID	Development Trigger Construct	Source	Removed	Applicable to Maintenance
45	Use of Prototype	[27] , [14]	Technique, covered by 21, 44	
68	New Stakeholder (role)	[27]		Y
96	Customer Company Reorganization	[28], [16], [14], [27], [29]		Y
51	New solution Tools/technology	[27], [28], [25], [14], [8]		Y
54	Change to government policy or regulation	[28], [14],[8]		Y
20	Participatory Learning	[14],		Y
28	Local Customization	{14]	New Stakeholder	
50	Customer migration to new solution	[14]	Type of requirement	
39	Customer need change	[24], [25], [16], [29], [29]	Too woolly, covered by 47, 96, 54, 82, 21, 45, 42, 90	
44	Developers Increased Understanding of problem	[25], [16],[17]		Y
56	Scope Reduction	[9][16]	By-product of 88, 66	
34	Changes to packaging/licensing/branding	[9]	Covered by 61	
65	Solution Elegance (Design Improvement)	[25], [9]		Y
67	Resolution of mis-communication	[9]		
49	Testability	[9]	Type of Requirement	
82	Business Process change (continuous improvement)	[15], [8]		Y
42	Response to competitor	[25], [8]		Y
16	Functionality Enhancement	[16]	Covered by 78, 65	
11	Defect Fixing	[16]	Doesn't result in requirement change	
69	Redundant Functionality	[16]	Covered by 66, 44, 20, 90, 67, 23, 51, 65, 21	
8	Missing Requirement Identified	[16]	Not a reason/cause/source	
86	Clarification of Requirement	[16]	Covered by 67, 23	
21	Increased customer understanding	[28], [15], [8], [17]		
72	New Class of User	[28]	Result of other changes, covered by 82, 68	
74	New Usage Condition	[28]	Covered by 78, 85	
15	New way of doing things	[28]	Covered by 82, 96	
77	Correction to Requirements specification	[28]	Covered by 23, 67	
78	New Opportunity	[28]		Y
1	Change in the use of the information	[17]	Covered by 82	
88	Cost or schedule overrun	[29], [28]		
49	Testability	[16]	Type of requirement	
85	Change to Customer's hardware/software	[9]		Y
58	System Usage (after installation, not prototype)	[27], [15], [14]	Out of scope of project development	
90	Changes to Market Demands	[8], [14], [16], [29]		Y
62	Resolution of Conflicting Requirement	[16]	Covered by 83	
55	New Functional Feature	[28]	New Requirement	
3	Improved Quality Feature	[28]	Change to requirement for another reason	
14	Result of Change in political climate (needs of particular group emphasized)	[24], [8], [25]		Y
93	Change to customer's environment	[15]	Covered by 96, 68, 85, 47, 66	
18	Changes in Underlying technologies	[25]	Covered by 85, 51	
83	Incorrect Requirement Identified	[16]		Y
23	Resolution of Misunderstanding	[25]		Y
92	First or re-engagement of user 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 Solution	Added		

	Development Uncertainty Construct	Source(s)	Removed	Applicable to Maintenance
31	Analyst skill or experience	[4], [24], [30], [8]		
95	Development team knowledge of business area	[24], [31]		
79	Quality of Analysis techniques employed (workshops, interviews, modeling etc)	[4], [8], [15], [29], [28]		
59	Project Size	[24], [17], [15]		Y
30	Novelty of product (business novelty)	[31], [8]		Y
41	Logical complexity of problem	[17], [31], [24]		
19	Availability of communication with customer	[17], [8]		Y – added word 'stakeholder'
22	Involved customer's knowledge/understanding/clarity of requirements	[24], [32], [31], [8]		
64	Quality of Communication between analyst and customer	[24], [17], [32], [8]		
33	Involved customers experience with working alongside IT to produce solutions	[31]		
9	Diverse User Community	[31], [15], [8]	Summary of 29, 27, 40, 41, 2, 60	
32	Incompatibility between requirements	[28]		Y
24	Lack of well understood model of utilizing system	[17]	unclear	
55	Lack of well-understood model of the utilizing system	[17]	unclear	
6	Lack of structure for activity or decision being supported	[17]	Covered by 22	
52	Stability of Customers Business Environment	[24]		Y
76	COTS usage	[28], [27]		Y
2	All stakeholders identified	[27], [33]		Y
29	All Stakeholders involved	[24], [33], [27], [8]		Y
40	Clarity/unity of shared product vision	[30], [14], [33], [8]		Y
27	Synergy of stakeholder agenda	[28], [31], [14]		Y
43	Unknown Customer Project Dependencies	[33]		Y
46	Market Stability	[25], [32], [14]		Y
13	Differing Customer Needs	[25]		Y
38	Type of user doing specification (incorrect user involved)	[8][30] [17]		
89	Change in the utilizing system	[17]	unclear	
80	Low Staff morale	[4]		
10	Large number of users	[17]	Not a risk if correct user involved - 38	Y
87	Level of participation of users in specification	[17]	Covered by 19, 64	
81	Lack of user experience of utilizing system	[17]	Covered by 22	
63	Degree of Change to customers workflow	[31], [8]		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 analyst) stability	Added		

ID	Maintenance Trigger Construct	Source	Removed	Equivalent Development Construct
161	Use Of Case Tools	[34]	Technique	
141	Maintenance activities	[34]	technique	
146	Changes to deployment Environment	[36]		
153	New Opportunity	[36], [38] , [40]	Covered by 137	
128	Domain changes due to system Use	[36], [38]		
125	Evolution of surrounding environment	[36]	Unclear	
113	Deferred Requirements during development	[37]		
109	System Usage	[38] , [39] , [14]	Activity	
142	Changing Customer Needs	[38]	Unclear	Variation of development constructs
104	Changing Technology (for solution)	[38] , [40], [42]		New Tools/Technology
116	Increased customer understanding	[39]		Increased Customer Understanding
151	New technological Methods	[39]	Covered by 104	
112	New Tools	[39]	Covered by 104	
119	Organisation Changes	[39]		Company re-organisation
138	Change to Operational Domain	[40]	Covered by 146,104,151	
132	Increased User Sophistication	[40]	Covered by 116	
136	Response to competition	[40]		Response to Competitor
165	Ambition	[40]	Covered by 153, 137	
162	Business Process Improvement	[40]		Business Process Change
101	Migration to other technology Environments	[14]	Covered by 146	
144	Function added, replaced, deleted	[42]	Not a cause	
103	Adapt to new technological environment	[42]	Covered by 146	
123	Alter system performance	[42]		Design improvement/.solution elegance or response to competitor/new opportunity
147	Alter Maintainability	[42]		Design Improvement/solution elegance
137	Response to Gap in Market	Added		Response to gap in market (added)

ID	Maintenance Uncertainty Construct	Source	Removed	Equivalent Development Construct
166	Business Size	[34]	Represented by 157,163	
121	Maintenance team Instability	[34]	Not a cause of req change	
135	System Complexity	[34]	Increase defects but not req change. Represented by 157, 163	
143	In-house software	[34]	Increase change capability but not cause	
158	Maintenance Team Knowledge	[35]	Not causing req change	
117	Cohesive Architecture	[35]	Effects defects but not req change	
148	Presence of Competitor	[36]		Presence of Competitor (added)
118	Market Environment	[36]		Market Stability
154	Semantic Relativism	[37]		Semantic Relativism (added)
115	System Age	[37]		
102	Period	[37]	Unknown	
167	Economic Climate	[38]	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 (added)
126	No. Of Users	Added		No Of Users (added)