Requirements Change Management:
Why are current change request forms inadequate?

Bee Bee Chua, June Verner

Abstract
Poor estimates of effort and schedule are frequently responsible for software project failure. Accordingly, there is a need to investigate and better understand the impact of requirement changes (RC) on software development projects in terms of effort. Requirements changes are unavoidable and unpredictable during the development of most software projects. Regardless of software development team size or the size of the project, every RC must be dealt carefully to ensure that every team member understands what the RC is, how the change is to be implemented, and the amount of time needed to correct or to modify the software.

To estimate accurately the effort required for rework for any RC is a difficult task for many experienced cost and schedule estimators. This is because there is normally not enough data available on which to base a model, and there is no useful categorisation of the various requirements types for estimating the amount of effort.

IT change request (CR) forms designed by industrial practitioners are not based on any theoretical framework that categorises RC in a logical, economical and structured manner. Practitioners often design CR forms based on the types of requirements outlined in the project without a real understanding of the requirement’s actual characteristics and attributes. Until now, there has been no standard method for categorising RC able to provide an understanding (both analytical and theoretical) that will enable IT practitioners to categorise RC at both the project level and the requirements level. We provide such a categorisation and use this as the basis of a rework effort estimation model.

This research presents a framework of attributes present on CR forms. These attributes must be understandable so that IT practitioners are able to categorise RC clearly. This will enable them to consider using this information as input into a cost estimation model. This framework will benefit IT practitioners by providing them with a method for better effort estimation. The added contribution is that it can help mitigate project risks early.

1. Introduction
Configuration Management (CM), is defined as a discipline for controlling the evolution of software systems [1]. A standard definition by IEEE [2] highlights, in particular, the operational aspects of CM, which is to identify, to control, to record status, to audit and review the structure of a product through a proper process of controlling the release of a product throughout the life cycle. The purpose of control is to ensure that software is consistent via the creation of a baseline product. Issues that are reported in CR forms must be concise and precise for IT practitioners to manage CR forms effectively.

The process of controlling, managing and determining the status of CR forms by the change management and software development teams is time-consuming and tedious. Every RC has its own unique characteristics and attributes that are not always easily understood by IT practitioners.
Managing RC is a difficult task for the following reasons: (1) every RC is not equal, (2) every RC type, size, and impact are uniquely categorised, (3) RC can appear at different points in the software development life cycle, (4) there is insufficient data available for estimating the rework effort required, and (5) there is no framework to help change management teams understand how to deal with RC effectively. Factors 1, 2 and 3 are not new; they have been widely discussed [3, 4, 5, 6, 7, 8, 9, 10, 11, 12]. Factors 4 and 5 have been broadly mentioned in the estimation area, but no one has yet developed an innovative, rigorous and targeted solution.

Most change management systems are manually-driven. This means that every paper-based CR form is recorded by staff. While automated change management systems seem efficient, their online CR forms often lack useful information for understanding the rationale for a RC. Some of the commercial change management systems do not have good, integrated, project management tools that provide support for estimating rework based on RC categorisation.

Estimating an accurate software project schedule is never an easy task. Not only must the estimator be knowledgeable about cost estimation but must also be technically-inclined, business-oriented, and able to provide an accurate schedule with appropriate justification to customers. The role of an IT estimator is not just planning for resources but rather to estimate a project schedule precisely and concisely. Hence, there are several challenges during estimation. The first challenge is the wide variety of cost estimation techniques developed for the purpose of estimating a project schedule based on past data rather than that from present projects. The second challenge is usually a lack of appropriate data for calculating a project implementation timeframe. The third challenge is that any change in requirements is driven by factors such as: socio-economic, technology, business, politics and legal, impact on the actual software development effort required.

These challenges increase an IT estimator’s difficulties in meeting the following conditions for projects: 1) no time and budget overruns, 2) no unacceptable quality and non-agreed functionality, and 3) no delivery of empty promises of benefits to the intended users.

2. Background

The objective of this study is to help IT practitioners understand the real causes of project failure, many of which are associated with poorly understood and constantly changing requirements. Poor requirements analysis in the early stages of software development can make the problem worse [24] and lead to project failure due to the number and cost of changes required to satisfy customers [11, 12, 13, 14, 15, 16, 17, 18, 19, 20].

Verner et al [21] note that, for the projects they investigated in North Eastern U.S.A., Australia and Chile, poorly managed change control, is a risk to project success. In addition, these researchers also reported that poor estimation is frequently based on poor requirements. Chua et al [22] note that failing to consider requirements characteristics and attributes when making changes during software development also is a risk.

Sommerville [23] noted that RCs are unavoidable, unpredictable and unanticipated because of changes in software project circumstances. Early RC risk mitigation in a software project will help ensure a project is completed on time and within budget. Hence, having a good understanding of RC is necessary for estimating accurately the effort required to rework them. In addition, a categorisation of RC must be incorporated into a cost estimation model to expedite estimating “person effort” more accurately.
To do so, it is important not only to understand the types of RC but also to be aware of the characteristics of each type of change, and the likely effort to make the change, to avoid inaccurate effort estimation.

RC may occur during: 1) software design, 2) programming, 3) testing, 4) implementation, and 5) documentation. Each RC is regarded as being either due to a defect in the original requirements or caused by a change in the requirements [4, 5, 6, 7, 8, 9, 10, 11, 12]. Therefore, there is a need to have a process in place to: 1) measure the number of RC in order to balance them against the cost and effort required for the entire software development, and 2) to ensure RC are appropriate given the project resources.

Analysing patterns of RC is difficult because we need to not only understand the types of defect or omissions we must deal with, but also how much rework must be done. Parametric models estimate overall budget, provide cost and schedule breakdown by component, stage and activity, and can show the cost and schedule sensitivity of software project decisions. Other research tools for managing requirements have been developed. These tools assist in detecting and correcting poor requirements and helping to identify requirements defects [26, 27]. However, such tools do not provide any support in determining the cost of making changes.

According to Yashwant et al [28], the cost of fixing a defect is expensive because there is a large amount of time and effort required to analyse the cause of a defect before actually fixing it. If a large amount of time and effort is spent fixing a defect, it is detrimental to overall project duration and can heavily affect project cost.

Most IT projects begin with users experiencing difficulty defining their real requirements. We do not necessarily see this as a major problem. It is the responsibility of the IT staff to extract and define the true nature of the requirements. However, users’ initial requirements are often not the main cause of project delay, but rather it is the difficulty in reworking RC. No project is developed in a vacuum and even projects with the best initial set of requirements in the beginning can expect changes as the project progresses. The larger the project and the longer it takes, the more RC can be expected. An understanding of requirements characteristics and their attributes can be confusing for IT practitioners and this makes it difficult to schedule rework accurately. In addition a great deal of time and effort may be invested in meetings between change management teams and software development teams in discussing every aspect of users’ change requests.

There is a substantial body of research into software cost estimation tools for software project development effort including those based on expert judgment, parametric models, and analogy [29, 30, 31, 32, 33, 34]. However, estimation results may not always be accurate because there is too little data for analysis and/or inappropriate data types are used to estimate effort for RC and integration.

Cost estimation models provide estimates for overall budget; provide cost and schedule breakdowns by component, stage and activity; and show the cost and schedule sensitivity of software project decisions. Parametric models used for estimating software development efforts have been developed for several reasons: (1) to estimate overall budgeting accurately, (2) to provide cost and schedule breakdowns by component, stage and activity, and (3) to illuminate the cost and schedule sensitivity of software project decisions. Only a few cost estimation models are used for estimating rework effort based on the number of RC during software development.
While projects are often not delivered on time to customers due to IT management and technology failures, another reason is that rework effort has not been included in the effort and schedule estimation. In other words, the number of expected RCs typically found in software development projects has not been taken into account.

Another challenge IT practitioners face is the difficulty of understanding and interpreting results derived from complicated scientific and mathematical equations [27, 28]. RC can be identified, and evaluated by considering the four types of software maintenance [36]; adaptive, perfective, corrective and preventive. However, even within these four maintenance types we do not really know what the impact of the change is, and how it relates to other requirements.

Each issue reported by users or IT practitioners is in the context of their awareness of the problem rather than how the problem can be tackled. It is important to understand RC attributes and characteristics and their relationships in order to estimate the rework effort that will be required. While the four classes of software maintenance are useful when quantifying the numbers of changes they are not suitable for qualifying how each change is to be resolved. The time and effort spent in discussion between a change management committee and a software development team to agree on rework requirements changes is difficult to estimate because a detailed breakdown of this activity is not normally considered.

To be more precise, rework effort should be distributed into (1) re-investigation, (2) re-analysis, (3) re-identification, and (4) re-estimation. This will allow the team to trace the cause of the change and predict its cost. The team needs answers to important questions, such as “Where do the changes lie?”, “Where does the correction or modification fit in the lifecycle?”

Effort estimation for rework is usually based on analogy. However, we believe that based on an appropriate requirements change effort multiplier, cost estimation models can help project managers make better estimates. A cost estimation model such as COCOMO 2.0 [30] is useful for estimating effort based on appropriate personnel effort and schedules.

Estimating costs accurately at various stages of the software life cycle is the goal of any estimator. There are two types of parametric models for estimating software cost and effort. The first is a parametric cost model and the second a constraint model. These models aim to estimate software cost accurately by using two main variables, effort and size. Effort is expressed in standard units of time, using person hours, person days, person weeks, person months or person years as the measurement units.

Many IT estimators calculate effort in fairly small units (person hours). Person hours may be expressed as a function of one or more variables (e.g., size of the product, capability of the developers and level of reuse). Size is usually defined as source of lines code or the number of function points that are derived from a product specification. Boehm’s COCOMO model [30] and Albrecht’s function point model [29] are examples of cost models. Putnam’s SLIM model [33], based on the Rayleigh curve, is an example of a constraint model. A cost model has one factor as its primary input, such as a size driver, and secondary factors such as cost drivers. It may also include other factors that contribute to overall cost. Constraint models, however, are based on demonstrating the relationship over time between two or more parameters of effort, duration or staffing level. Since both types of models rely on mathematical equations, they may not estimate a project accurately because the time and effort for RC has not been considered.
This explains why it is difficult to achieve project effort estimation accuracy. Before discussing our proposed framework, the information asked for on IT industry CR forms must be evaluated. This will consider why they are not sufficient to provide estimates of RC costs for software development teams.

3. Evaluating IT Industry Change Request Forms

The purpose of evaluating IT industry CR forms is to gain a better understanding of the information presented to IT practitioners with respect to the RC costs and how they affect schedule and rework effort.

Our approach is based on the evaluation of twenty different CR forms published by software development companies on Google. These forms are designed specifically for tracking RC during and after software development. Twenty CR forms provide an adequate number on which to report findings for a small pilot study; 80% of the forms are used for documenting changes as defects or non-defects during the software development stages and the remaining 20% are used to record defects and non-defects reported by the users and software development teams after the launch of an IT system in the production environment.

The aim of this paper is not to criticise industry CR forms, but rather to help IT practitioners understand RC in enough detail to be able to use empirical data in the context of IT change management. The objective is to discover: (1) which data on CR forms are informative enough to aid IT practitioners understand RC, (2) what is the philosophical view in terms of IT change management, (3) what is not explicit in the production of CR forms, and (4) what of the specific information presented in the CR forms is relevant to enable estimation of project duration effectively, specifically when the change impacts other requirements and the entire project as well.

Although the variety of CR forms may help us with respect to understanding RC, the information required is very similar from form to form. The only difference we found is that the structure of the change forms is based on a project perspective and not organisational perspective. Evaluating CR forms is not a simple task. First, all CR forms provide standard information and are similar to one another. As they are used during and after software development, we categorise them into two groups: non-production (during software development) and production (after development). We want to know if CR forms have enough information to address the requirements for estimating software development effort accurately based on an understanding of what a RC is, what impact analyses are required before and after implementation and how much rework effort is required on the basis of the change being either an error correction or a non-error correction.

Unfortunately, all of the CR forms we reviewed have two main areas: 1) one asking for a detailed explanation of the reason(s) for a RC, and 2) the change type. None asks for any information to identify other requirements characteristics and attributes.

An illustration of a RC is the changing of a Web site’s screen colour. Many CR forms specify change type as non-error correction, new, modify, update or screen design and reasons for the change may be specified as wrong colour, colour is not interesting or dull or to seek screen enhancement. This information cannot be seen as providing a value-added approach for explicitly defining a RC. An in-depth analysis would tell us to what extent the change of a requirement is simple and easy to implement.
Perhaps, adding another field, requirements characteristics and attributes, will provide a better definition of RC in a more structurally, meaningful and understandable manner.

The following explains what a RC is before and after implementation with respect to change type, process, the reason for change and requirement characteristics and attributes.

Before implementation:
1. During requirements analysis, a RC may be required as the project progresses while prototypes are developed [24].
2. RC occur during pre-functional specification, i.e., changes in requirements during the early phases, elicitation, elaboration, analysis, modelling and negotiation of software development schedules, before a functional specification has been completed and signed off [36].
3. RC will occur during the later phases of software development lifecycle i.e. design, coding, development and testing, after the pre-release functional specification has been formally signed off by the client [36]. In the context of software testing, a RC is regarded as an error correction or non-error correction. Both types of requirements changes are noted as urgent or non-urgent. They can also be trivial or non-trivial [22]. An example of a trivial requirement change is one that does not affect and disrupt overall system performance. For instance, a requirements change in relation to spelling correction. A non-trivial requirements change, it is one that disrupts and affects overall system performance. In this case, adding new functionality is related to business value.

After implementation:
4. In the context of software maintenance, a RC is categorised as one of the three maintenance types: corrective, adaptive, perfective or preventive [36].

\[ \text{Definition of requirements change} = \text{Reason for requirements change} + \text{Requirements change type} + \text{Requirements characteristics and attributes} \]

*Figure 1: First order change*

Figure 1 shows the data necessary to properly understand requirements characteristics and attributes as a complete structure, and provides a definition of a RC. Not only does this definition provide insight into the real requirements problem, but it also increases our knowledge of a RC. We believe that defining a RC is not just stating reasons for change and the change type.

Incorporating requirements characteristics and attributes help IT practitioners to understand the value of a RC. Clarifying the characteristics and attributes of a RC increases the probability that an estimator will accurately estimate and possibly help control the number of future RC. Characteristics and attributes may be inherited and either internal or external. Hence, the time involved in approving RC by change management teams, project managers and users may be reduced. Much of the complexity of RC arises from an unclear understanding of its meaning. Ambiguous, incomplete, inconsistent, inaccurate and missing information create uncertainty about how a change can be implemented.
Our next goal is to evaluate CR forms related to impact analysis for IT practitioners, looking at what risk has been identified for the overall project’s schedule. Under non-production CR forms, the change impacts effort, functionality, schedule, risk, cost, and quality. These are standardised concerns regarding impact on a project. For production CR forms, the impact and consequences were examined for no implementation change, impact on testing procedures, impact on project risk associated with the change, and business priority. These are non-standardised concerns at the business requirements level.

Our investigations into CR forms are quite distinct regarding impact analysis before and after software development. The implication is that the cost of a RC before implementation is based strictly on project schedule risk and effort, and after production requirements changes are based strictly on business value. Project schedule for risk and effort refers to a deadline critical for a RC. The issue for a requirements change in a production environment is that business value is placed as a higher priority, and every change is tied in to risk to the system environment in addition to business and technology, rather than simply meeting a deadline.

The cost of implementing a RC after production is higher [28, 32] than before production because of necessary testing procedures. Figure 2 shows the impact of a RC is equivalent to the analytical risk for effort and rework. The original RC is impacted by factors from the environment, technology, projects and business. Considering how each of these factors negatively or positively impacts the project before or after production helps justify the value of both project and business outcomes.

![Figure 2: Second order change](image)

We propose that impact analysis should extend to the level of considering requirements risk. Questions should be asked about the level of requirements risk caused by the RC with respect to its direct impact on project and business value.

Of the twenty CR forms examined, only two of the ten non-production CR forms dealt with “estimated effort in hours” and “estimated hours required to implement a change”. However, the production CR forms required a detailed breakdown of estimated effort and actual time required for its software development stages. Software development effort can be classified into two main types: effort for new requirements and effort for rework on existing requirements. Although both affect the original project estimates, there is a distinction based on whether the RC is caused by an error or a non-error. Error based RC requires rework effort driven by steps in some kind of procedure. Non-error RC are driven by innovative solutions based on new ideas. An example of an error for correction is related to logical and computational corrections which can affect the success or otherwise of the coding.

However, none of the CR forms addressed these differences and did not categorise effort based on new versus existing requirements. Moreover, the classification of RC effort may involve individuals or groups and the amount of effort may range from some minimum to a maximum.
Inaccurate estimated project schedules can be produced when the effort for new requirements and for rework on existing requirements have not been taken into account. Estimating hours of effort for new requirements during production should be added the original schedule because new requirements typically mean that additional functionality will be added. These should be considered as maintenance or else the changes should be saved for a future release.

![Figure 3: Estimating effort for requirements changes](image)

The proposed solution is to incorporate these two factors into CR forms so that IT practitioners can use this as a basis when revising project schedules to avoid over or under estimated projects. Figure 3 illustrates this.

4. Proposed Framework

Our approach to developing a framework (see Figure 4) for estimating person-effort for RC consists of four stages. The first is to identify the different kinds of RC listed on the CR forms. The second is to analyse the causes of the RC to better understand the reasons for the changes. The third distinguishes between the different types of changes and allows them to be classified both horizontally and vertically. RC, such as incomplete, inaccurate, missing and inconsistent information is shown at the horizontal level. The vertical level of RC refers to type. For example, error correction refers to language or environment, and non-error correction is categorised as (1) enhancements to improve performance or other system attributes, (2) addition of new requirements, and (3) adaptation of the system to a new environment like a new operating system. The final stage allows us to estimate the person effort required for making a change.

![Figure 4: Proposed framework for estimating person effort on requirements changes](image)

The rationale for the development of this framework is two fold. Firstly, we wish to improve the current process of reviewing and approving RC by change management committees. Secondly, we wish to assist project managers in better planning for RC through predicting more accurately the person effort required for these changes.
As our proposed framework for estimating person effort for RC is a stepwise approach (see Figure 4) it is important to outline each step in detail as follows:

**Step 1:** Categorise requirements changes into first order change and second order change.

**Step 2:** Note the reasons for the RC.

**Step 3:** Understand factors relating to, and impacting on the RC.

**Step 4:** Distinguish vertical and horizontal dimensions of requirements change relationships.

**Steps 5 and 6:** Identify the relationship between effort and various change types; and estimate the amount of person effort required. (To be incorporated with COCOMO 2.0).

The above figure illustrates our proposed framework developed to provide a basis for gaining greater insight and understanding into RC when they are present in CR forms. This framework should enable us to more accurately estimate the person-effort required to implement a proposed change. In addition, the framework may help to reduce rework through analysing patterns of RC and help in terms of assessing individual and group person-effort for each RC. We believe that estimating person effort through an understanding of the RC process using variables that have a direct effect on changes may be a more effective way of estimating RC effort rather than simply using effort estimation models developed mainly for initial schedule and effort estimation for a proposed software system.

The framework is to be calibrated with real world data (i.e. categorisation of RC) from IT practitioners who have dealt with software maintenance projects. One of the authors in this paper has several contacts with software industry partners in Australia as well as worldwide while the other is currently involved with a large outsourced software development project.

**5. Conclusions and Future Work**

The goal for the development of our framework, 1) is to improve the level of accuracy of estimation with respect to software development effort when there are RC, 2) changes are often based on information provided in CR forms without any specific criteria because a real understanding of RC has been neglected, 3) the categorisation of RC has not been integrated into parametric cost estimation models, 4) A lesson that we have learned from real world project failure case studies is that we need to understand why the project failed, rather than focus on RC or inadequate estimation; we need to explain specific reasons for the project failure and to use this information to prevent future failure, and, 5) If the categorisation of RC is to be incorporated into parametric cost estimation models, then this will add value to other cost drivers. We propose to extend the work on COCOMO 2.0 [30] by introducing a new cost driver, that is, the categorisation of RC that will contribute to a more accurate estimate for personnel effort spent on rework in software development projects.

Our next step is to conduct a pilot study to validate the proposed framework via a case study with a software development project. A pilot study is recommended for the initial validation of a framework such as ours. A case study will provide real-world data gathered from IT practitioners for effort and time spent on rework for various types of RC at different phase of software development lifecycle. Our pilot case study will be followed by further case studies that will provide us with further data for our RC cost driver. This data will be evaluated with the aim of a better understanding of the framework’s suitability, reliability, and usefulness.
6. References


