

A Metric of Confidence in Requirements Gathered from Legacy Systems: Two Industrial Case Studies

James Marchant, Christos Tjortjis, Michael Turega
School of Informatics, The University of Manchester
P.O. Box 88, Manchester, M60 1QD, UK.
james@marchant.com, {c.tjortjis, m.turega}@manchester.ac.uk

Abstract

It is known that well over 50% of replacement projects fail. Requirements gathering go some way to contributing to this statistic; if the requirements we gather for the new system do not match those of the system to be replaced then the project is bound to fail, at least in part.

This paper proposes an empirical metric that assists measuring the confidence in the requirements extracted from a legacy system. This metric capitalises on five techniques for gathering requirements from legacy systems and caters for a number of different types of project. The metric can be used to estimate the likelihood of a project's success or failure and is evaluated by two industrial case studies; conclusions are drawn from these and directions for further work are presented.

1. Introduction

It is well known that the majority of replacement projects fail [1]. The successful implementation of such a project is not the outcome of any single phase of the software maintenance lifecycle, but the collaboration of them all. As a result a replacement project can fail for any number of reasons, for instance, poor management, lack of resources or lack of knowledge.

Lack of knowledge is a major obstacle to overcome early on; if the system to be replaced is not fully understood, then how can one expect to implement a successful project that will replace that system [2].

Gathering the knowledge, or requirements, is one of the first milestones that must be attained in order to start a replacement project successfully [3]. Getting the requirements wrong has a 'snowball' effect on the rest of the project; time is wasted designing, implementing

and testing the unwanted features, then re-defining, re-designing, re-implementing and re-testing the application. This severely impacts on the likelihood of success for the project.

Success is a term that is subjective and relative to the project which it is being applied to. Success factors for one project are likely to be different from those of another. A measure of success may be whether the deadline of the project is met, the functionality of the application is complete or the cost of the project falls within the budget.

The importance of getting the requirements right and the authors' experience in commercial software replacement projects has motivated the work presented here. Establishing the risks as early in a project as possible has prompted this investigation into how a metric can be applied to the requirement gathering phase in order to determine the success likelihood of the project. This metric should be based solely on the requirements gathering phase [4].

Our approach to this problem is to analyse the common techniques for requirements gathering and calculate the contribution they make to defining all the requirements for the project. Values representing the confidence in these techniques will be proposed that together with the contribution factor for the technique will be used to calculate a confidence metric for all the requirements. This value can then be used to make subjective predictions as to whether the project is likely to succeed or fail, purely based on the requirements gathering technique employed.

This paper explores the requirements gathering techniques and proposes an empirical confidence metric which relates these techniques to a number of project types. Requirements gathering techniques contribute towards the confidence metric which in turn can be used to calculate a risk factor for the project, which can in turn be used to determine a likelihood of failure for a project. The proposed metric was used in two industrial case studies as a guide to reason about the success or failure of the project.

The remaining of the paper is organised as follows: Section 2 reviews a number of requirement gathering techniques and their suitability in contributing to the proposed confidence metric. Section 3 proposes the confidence metric and justifies the selected criteria. Section 4 presents the two case studies and section 5 discusses how the metric can be applied in such cases. The paper concludes and directions for further work are given in section 6.

2. Background

Gathering the correct requirements is essentially the fulcrum point in a software replacement project, this point determines whether the project tips towards success, or failure. It is estimated that 85% of defects in developed software originate in the requirements [5]. For a software project to be successful, 100% of the core requirements for the application need to be discovered, it is this discovery that we will explore.

In terms of legacy systems replacement software engineers are given the opportunity to use a number of requirement gathering techniques [5]. An objective for the replacement of a legacy system may solely be to reproduce the existing process or application in a more modern programming language. This type of software project is not that common [6], [7], it would be more common to replace the system, adding new functionality and redesigning the user interface [6], [7]. It is at this point where the requirements of the system creep away from the original and the task of replacing legacy systems becomes more complex.

Five techniques have been proposed that aid with the discovery of requirements from legacy systems. These techniques include the morphological, source code, functional, and use case view as well as documentation analysis; these are reviewed in the following sections.

2.1. Morphological View

This is the systematic exploration of the applications features [8]. It involves uncovering the applications operations by using the application as the end user would. A common method for this approach is to work through the menu structure or commands list entering each item into a table, recording the functionality of that feature. This approach is suitable for recording user known operations and the composition of the user interface. The morphological view is strong at discovering the legacy systems initial requirements for the application. The morphological view is poor at uncovering hidden procedures and operations the user is not familiar with, for instance, processing data.

2.2. Source Code View

The source code view provides a detailed view of how the original system was developed. It uncovers the methodology behind the internal data processing and how operations are performed. The source code view aims to find functions and methods that define the operation of the program. The source code view is strong at gathering requirements for algorithm re-engineering. The downside of this technique is the reliance on the skills of the original developer and their programming approach. Comments and structure of the original code are paramount to the performance of this technique.

2.3. Functional View

The functional view is a description of what the features do [8]. Essentially this involves the discovery of all the operations provided by the application. The functions are then categorised, linked and any relationship determined. It is the discovery of these operations that will be used to define the requirements. Discovering the operations is the hardest task. In the absence of any documentation or program specifications these operations are uncovered by traversing the morphological views, the source code views and observing the operations that the application performs. The functional view is very strong at providing low level core requirements. These generally form the basis of the application, and are some of the first requirements that should be defined. The functional view is weak in discovering requirements that are not user orientated, as it still relies on the operations being discovered by dynamically running or statically examining the application.

2.4. Use Case View

In the replacement of legacy systems, one source of information to determine the requirements of the legacy system is to inspect or question the user or consumer of the legacy system. The user of the legacy system may be a person, or another application. In either case, the primary concern is to find out the input and the output of the application. Essentially, one wants to produce a simple box, where data are poured, the application processes the data and the output is what the consumer expects [9].

This simplification provides substantial Use Case views that simply define the requirements of the legacy system. Use Case views are exceptionally strong at defining high level requirements for an application; they provide the requirements as long as the user

knows what they want. Use Cases provide an abstraction above any code details or physical operations. They are an exceptionally powerful tool when structured correctly [9].

Use Cases may be dangerous if wild assumptions are made about the functionality of an operation [10]. The level of abstraction they encompass may be too abstract to give any real meaning to the requirement set. Use Cases are often more complicated than first imagined, each level of abstraction requires different sets of use cases [7].

2.5. Documentation Analysis

Documentation analysis involves the study of all available documentation for a given project. The relevance of this documentation is at the discretion of

the analyst. If suitable requirements specification documentation exists, the software maintainer should seek to establish how up to date is the latest version of the document, and whether the documentation release corresponds to the application release. Project release notes may exist that identify all the operations of the application. Help files may exist that detail the entire user side features and operations. Design documentation may exist that detail the data structures, databases and design considerations for the original application. Documentation analysis is strong at sourcing the original requirements of the application which the application was intended to address. It is let down by the reliance of the documentation to be kept up to date, and the partial completeness of that documentation [11].

Table 1 – Review of requirements gathering techniques

Requirement gathering technique	Advantages	Disadvantages
Morphological View	Good at gathering user known operations Discover the initial application requirements Record the user interface requirements	Uncovering hidden requirements Internal processes
Source Code View	How the original system was developed Methodology behind internal data processing Algorithm reengineering	Reliance on how the code was originally written Need for structure and comments
Functional View	Providing low level requirements Gathering core requirements	Discovering non-user oriented requirements Internal processes Dynamically and statically examining the program
Use Case View	High level requirements Provide user known requirements Abstraction above code or implementation Can be structured	Dangerous if wild assumptions made May be too abstract More complicated than first imagined
Documentation Analysis	Source original application requirements If documentation complete, then accurate source of requirements	Documents must be kept up to date Need to be consistently maintained

Table 2 –The Confidence Metric Table

	Morpho-logical	Source Code	Functional	Use Case	Documentation	Confidence Metric
<i>Contribution Factor</i>	0.15	0.2	0.15	0.25	0.25	
Projects						
A - Internal Process (Replacement 1 for 1)	50	75	5	80	60	58.25
B - Internal Process + New Functionality	50	75	5	60	60	53.25
C - Replacement Desktop Application (Replacement 1 for 1)	80	30	80	70	50	60
D - Replacement Desktop Application + New Features	80	30	80	50	50	55
E - Replacement Desktop Application (No source code or documentation)	80	0	80	90	0	49.5

2.6. Summary of Background

The five techniques we have discussed are commonly used in software replacement projects, however, each of the methodologies are not equally weighted in terms of their usefulness for any given project. In the replacement of a legacy system, one technique, for instance source code analysis, may be favoured over defining use cases. This is the preference of the project team and the type of project undertaken. We have seen that each of the techniques has its advantages and disadvantages. These are outlined in Table 1.

3. Proposed Solution

Knowing the requirements gathering techniques is important to the success of any project, but their effect on different types of projects needs also to be known. A data driven process project will perform differently to a Windows based application; gathering requirements for one is much harder than the other, some projects have explicitly defined boundaries while others do not.

It is being proposed that each of the five requirements gathering techniques be given a *contribution factor*. This is a measure of how good that particular technique is at gathering reliable requirements in any type of project. The sum of these factors should add to up to one.

Each type of project is analysed against the types of requirement gathering techniques discussed, they are then rated out of 100. This figure represents the success of gathering the desired requirements via the technique used. This shall be known as the *Technique Confidence*. Typically, one would not make an assumption that any given technique will uncover 100% of the requirements. Combinations of techniques are more likely to gather all the requirements. The confidence metric is calculated by summing the products of the contribution factor and the Technique Confidence, as depicted in Table 2.

The values entered into this table for the contribution factors have been sourced from the analysis of previous projects. The authors experience with projects and the relative usefulness of the requirement gathering technique as discussed in Section 2 is how the figures were determined. We propose that Use Case view and Documentation analysis are assigned a contribution factor of 0.25, Source Code view 0.2 and Morphological and Functional views a contribution factor of 0.15.

The Morphological view has been assigned the value of 0.15. This figure, along with the functional

view is the lowest assigned to the view collection. The figure itself cannot be reasoned about in isolation, it must be considered in respect to the other views. When the other views are considered, it is noted that the morphological and functional view are slightly less useful at sourcing requirements than any of the other views. This is partly due to the fact that they cannot discover hidden processes and procedures, these may be essential requirements.

The source code view has been deemed the intermediate technique. Theoretically, source code analysis can be used to uncover the same requirements that Morphological and Functional analysis can, but

l aselnet(p)31.7(e)0.6(l n)-15. Muol atctio (s)16.1(iuo)31.rit g

g
F3 Tw[(lu)4-39.ue8(it)24.4048c-1.7857 -1.-
Ri rR60d8(it)2ETL00rg372 362088 21.22
MoEsDIUMedg
Cef-8.1(()-39.2(d))23.35mce8(it)24.9524=1.TJ0]-1.
tp(0al)-3.2(teo)23.8 projseotstnteo,8(it)2-e

metric applied to them so we can reason about the likelihood of success for each of the two projects given the requirements gathering techniques used and the confidence in those techniques. The figures displayed

in Table 4 are subjective, empirical views of how successful the authors felt these requirements gathering techniques were at generating useful requirements.

Table 4 –The Confidence Metric Applied to Case Studies

	Morpho-logical	Source Code	Functional	Use Case	Documentation	Confidence Metric
<i>Contribution Factor</i>	<i>0.15</i>	<i>0.2</i>	<i>0.15</i>	<i>0.25</i>	<i>0.25</i>	
<i>Projects</i>						
Case Study 1 – Replacement Desktop Application, with new functionality, Transaction Redevelopment, Database Maintenance.	40	30	40	50	30	38
Case Study 2 – Desktop application replacement with new features.	70	20	70	60	80	60

4.1. Case Study 1

This case study was the investigation into the large replacement project for a financial organisation. The project was to replace the existing call centre agent’s expert system. The existing system was used by call centre staff to handle telephone banking requests; these were anything from creating new accounts to setting up standing orders and direct debits. The system is a Microsoft Windows based desktop application; it relies on running transactions based on a central server to perform the financial transactions. The migration to replace the system was from a language based on PASCAL, to a new C++ orientated language.

The system contained approximately 500 windows style forms, each of the forms had approximately five supporting logic scripts that would perform the procedures on the information. The size of these logics ranged from 10 lines to no more than 1000 lines. The system contained approximately 1,250,000 lines of code. This related to approximately 2,500 procedures.

A major problem encountered during the requirements analysis phase was the compilation of all the requirement sources, i.e. gathering documentation and source code analysis. The haphazard approach to reengineering of the system over several years lead to inconsistent documentation, locations of the documents and the detail of the documents. The same applied to source code written in different styles by several different developers. The project, although considered essential to future productivity, was postponed during the requirements analysis phase due to the lack of understanding of requirement sources.

4.2. Case Study 2

This case study was the investigation into the smaller replacement project for the legacy network monitoring application. The previous application was written in C. Its primary function was simply to listen for SNMP traps and alerts arriving on a designated COM port. The alerts would be logged and displayed to the user in nothing more complicated than a list. The replacement project must essentially do the same task, but with the addition of a new modern user interface to allow more effective management of the network. The new development platform was C# in .Net.

The system to be replaced contained approximately 100 files, each with around 250 to 1000 lines of code. The approximate total for the number of lines of code would be around 60,000. This approximated to 150 functions.

In terms of use, the system was as critical to the productivity of the team who used it as the large project was to the call centre staff, but the major difference was the shear scale of the application. The purpose of the application was well known by a number of service engineers and documentation existed which would support requirements gathering. The analysts were not overwhelmed by requirements gathering and confidence was high that the project would be complete successfully within the time estimated.

5. Discussion

The results in the confidence metric Table 2 are derived as follows, the figures in bold indicate the usefulness of that approach in a given project. Further projects can be added to the table, providing they can

be reasoned about. Other requirement gathering techniques can be added to the table, providing the contribution factor is adjusted accordingly.

Essentially, the contribution factor is the key figure in the equation, the accuracy of this figure is vital to the end confidence metric for that project. The figures for the contribution factor in our table have been uncovered by analysis of past project experience, and the resultant requirements generated from each of those methodologies. These figures would be adjusted according to others relative success with the appropriate gathering techniques.

The Confidence Metric is a subjective, empirical measure of how likely all the requirements will be gathered by using only the techniques described and the resources available, given the contribution factor. Confidence metrics approaching 100 will be high confidence, those around 50 will be medium confidence, and those below 25 have very low confidence. As software should be engineered [12], [13] there should ideally be high confidence in any set of requirements; medium confidence suggests uncertainty and possible lack of direction for the project, whilst low, indicates that the confidence in the requirements is dangerously uncertain.

In projects where the failure risk falls into the medium band, it is expected that the projects experience a significant requirements creep, the introduction of requirements to meet the projects goals [14].

Two case studies were introduced in Table 4. The figures entered for these two case studies were sourced from the authors' involvement and opinion of how successful each of the techniques were in sourcing requirements.

Case study one was expected to have a low confidence metric, the lack of consistent documentation and the different tasks to be completed all added to the shear complexity of the project. The confidence metric was higher than the authors expected, falling into the Medium – High failure risk category, with so much uncertainty in the project, we believed that the risk would have fallen nearer to the High category.

This is partly due to the application of use cases in the requirements gathering techniques. This technique was heavily relied upon, mainly because the project was business driven and the majority of the requirements were defined by the business side of the financial organisation. This meant that the business knew exactly what they wanted it to do, but were not sure how they wanted it doing. Use cases are commonly used to define abstract requirements. As Table 1 indicates though, use cases can be dangerous if used to define abstract requirements, they can be

wildly used to generate endless requirements sets that need to be broken down and some form of structure applied to them.

The second case study fell more or less where the authors had expected it to, in the Medium – Low risk category. This was not a surprise, as the authors' experience with replacement projects were what led to the contribution factors and the requirement completeness figures in the first instance.

The figure that stands out from case study 2 in Table 4, is the lack of requirements gained from the source code. Considering that the new implementation language C#, was an evolution of the original implementation language C, it was initially hoped that the source code would have been more help. After considerable analysis of the source code, it was found to be of little help due to the style and lack of comments used. Poor choices for variable names made the code hard to follow and to understand its functionality. If anything, this technique of gathering had a negative impact on the project due to the time wasted not gathering useful information. It may be proposed that negative values could be entered into the table for counter productive techniques of gathering requirements.

The case studies highlight where requirements gathering techniques do have an impact on the likelihood of success of a project. The two case studies reflect their retrospective outcomes, where case study one failed and case study two was considered a success.

6. Conclusions and further work

Without suitable requirements, no project can be expected to reach its objective; even a well defined list of requirements is no guarantee that the requirements for the success of the project have been fulfilled. The requirements gathered are only as good as the source they have been gathered from [5].

In this work we have introduced the use of metrics which can be used to highlight the potential flaws in requirements gathering, we aimed to quantify the quality of the requirement gathering techniques for a project. A confidence metric was established which represented the sum of the products of the metrics, in order to determine a single metric which could be used to determine the risk factor for the project.

In order for this hypothesis to be applied to other replacement projects, some degree of understanding of requirements gathering within the designated project needs to be understood, i.e. the manner by which requirements are gathered and the skills of the team responsible for gathering the requirements. It could be argued that any project could be made successful given

enough time and resources, eventually all the requirements would be uncovered, but this was not the intended application of the metric. We set out to produce an empirical confidence metric that would assist in measuring the confidence of the requirements extracted from the legacy system in everyday projects with finite resources, skills and time.

The technique confidence is the value which needs to be calculated for a given project, 'X'. Depending on the type of project X, would depend on the values assigned. In safety critical systems, one would hope that the technique confidence values were high, approaching 100. For less critical software, such as a simple web application, the values may be acceptably lower.

In summary, in applying the confidence metric to any project, the likely hood of gathering requirements from the techniques outlined in Table 1 need to be established. Past project experience with the given resources should be used to come to the figure used for the confidence metric. The same project carried out by different teams with different mentalities would undoubtedly have different outcomes.

Future work leads to the validation and analysis of as many types of software project as possible, gathering data to generate contribution factors from multiple development projects. The model could then be applied to different projects with varying complexity.

References

- [1] H.M. Sneed, 'An Incremental Approach to System Replacement and Integration', *Proc. 9th European Conf. Software Maintenance Reengineering (CSMR 05)*, IEEE, Comp. Soc. Press, 2005, pp. 196-206.
- [2] C. Tjortjis and P.J. Layzell, 'Expert Maintainers' Strategies and Needs when Understanding Software: A Qualitative Empirical Study', *Proc. IEEE 8th Asia-Pacific Software Engineering Conf. (APSEC 2001)*, IEEE Comp. Soc. Press, 2001, pp. 281-287.
- [3] C.A. Dekkers, 'Creating Requirements Based Estimates before Requirements are Complete', *Cross Talk – The Journal of Defence Software Engineering*, April 2005, pp13-15.
- [4] H. Hofmann and F. Lehner, 'Requirements Engineering as a Success Factor in Software Projects', *IEEE Software*, July/Aug. 2001: 58-66.
- [5] R.R. Young, 'Effective Requirement Practices', Addison-Wesley, 2001.
- [6] R.C. Seacord, 'Modernising Legacy Systems', Vol. 5, No. 4, Fourth Quarter, *The Cots Spot*, Carnegie Mellon, Software Engineering Institute, 2002.
- [7] M.L. Brodie and M. Stonebraker, 'Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach', Morgan Kaufmann, 1995.
- [8] I. Hsi and C. Potts, 'Studying the Evolution and Enhancements of Software Features', *Int'l Conf. Software Maintenance (ICSM 98)*, IEEE Comp. Soc. Press, 2000, pp.143-151.
- [9] G. Schneider, J. Winters and I. Jacobson, 'Applying Use Cases: A Practical Guide', Addison-Wesley, 1998.
- [10] I.F. Hooks, and K.A. Farry, 'Customer-Centred Products: Creating Successful Products through Smart Requirements Management' AMACOM, 2001.
- [11] A. Florence, 'Reducing Risks Through Proper Specification of Software Requirements', *Cross Talk – The Journal of Defence Software Engineering*, April 2002, pp 13-15.
- [12] I. Sommerville, 'Software Engineering', Addison-Wesley, 1998.
- [13] I. Sommerville, and P. Sawyer, 'Requirements Engineering: A Good Practice Guide', John Wiley & Sons, 1997.
- [14] K. Wiegers, 'Karl Wiegers Describes 10 Requirements Traps to Avoid', *Software Testing & Quality Engineering*, vol. 2, no. 1 January/February 2000.