
http://dx.doi.org/10.1109/ICRE.2004.1335649
Use Case Estimation – The Devil is in the Detail

Abstract

Mission critical and complex software projects habitually exceed budget expectations significantly. Dependable cost estimates are often required by customers long before detailed analysis and design activities would produce this information during a project. A number of estimation methodologies have evolved to produce reliable cost information at an early stage in the software life-cycle, however estimation continues to be a contributor to budget blowouts.

Contemporary techniques for costing requirements described as use cases are increasingly challenged as the size and complexity of the system expands. In addition use case representations of requirements fail to directly map into structures used by project managers, leading to ongoing comparisons of individual costs that are subjective and often unrepresentative of final project expenditure.

A large and complex system development project is described to demonstrate some of these problems and a potential solution is proposed to improve use case estimation.

1. Introduction

Agreement between customer and supplier on the scope and price of a software development project is generally necessary before contractual arrangements can be finalised and work commenced. At this point it is likely that the parties have differing views on scope and an inadequate understanding of costs, frequently leading to dissention and animosity over final contract price.

Lederer and Prasad [1] report that only one in four systems development projects in a study group completed within cost estimates. The Standish Group’s CHAOS Report [2] also found that “32% of projects terminate before delivery and only 11% are completed on budget. Of the remaining 57% of projects that are completed, the average budget overrun is 87%.”

Contributors to cost blowouts in software development projects are many and varied, with requirements creep and poor project management being well highlighted. This paper examines one of the problems associated with producing an initial accurate cost estimate for a project with extensive and complex use case based requirements, namely managing the combinatorial complexity of such systems.

Use cases, formalized by Jacobson [3], provide a popular technique for capturing requirements. Use case steps are written in an easy-to-understand structured narrative using the vocabulary of the domain. This is engaging for the end users, who can easily follow and validate the use cases, and the accessibility encourages users to be actively involved in defining the requirements.

The authors suggest that there is a non-linear relationship between the summed complexity of the use cases and the complexity of the total system. Consider a large composite enterprise system that encompasses:

- Multifaceted rules for a multitude of business processes;
- Complex rules for user roles, controlling which users can perform specific business functions in particular elements of a business process;
- Business to business asynchronous messaging to a variety of third parties;
- Financial functions that process payments for many 100s of types of items controlled by legislation, and that collect many millions of dollars a week;
- Complex online and offline printing, and reporting to a variety of stakeholders.

Such a system would run to as many as 100+ use cases. Many use cases would be expected to “include” a common business/system function and a single use case may utilize many common functional elements. This many-to-many relationship is shown in Figure 1.
Cost estimation of small systems based on use cases is manageable if a structured approach is followed (see sections 3 and 4), but as the number of use cases increases the task becomes increasingly more difficult. Usually some form of model is developed for estimation, if possible based on past project experience and bringing in other information sources. All too often budgetary constraints on the estimation process mean that this model is not fully developed before the estimates are required, and information sources may not be available at estimation time. Also many small to medium companies will not have previous experience in a similar project, but market forces will cause them to bid for any work they have a chance of winning. In effect traditional cost estimation represents a rational approach often being applied in an irrational environment.

Figure 1: Relationship between Use Cases and Functions

This contracting problem is compounded by the technical predicament that multiple use cases may need to utilize a business/system function in a slightly differently manner, which to the author, as a domain expert, is obvious. Consider a system that must calculate fees based upon extensive legislative rule combinations in different use cases. In 90% of cases the rules are simple, as they apply to the average member of the public, but the remainder are extremely complex. Complexity could be caused by: legislation; political factors (discounts for pensioners, powerful lobby groups); an evolution of business practices; or exceptions for special classes of customer. At a high level the complexity of these use cases and their interactions can easily be overlooked and the costs underestimated, because the estimator is not the domain expert. For large or complex systems the “devil is in the detail”.

Jacobson [3], as the architect of the contemporary use case driven approach, would not recommend estimating the cost of a system at the use case or requirements phase. An accurate cost requires information that can only be obtained after detailed analysis and a significant amount of design is undertaken. It is not until the design phase that implementation issues are uncovered. However customers need firm estimates of system costs long before design can be undertaken and cost estimation techniques become very important in defining budgets.

2. Use Case Estimation

Use case based techniques for capturing requirements are beginning to supersede the early functional techniques [4], forcing an evolution of cost estimation methodologies. In many organizations the use case is often employed as the basic unit of work upon which a system is estimated. Estimation could be based upon: a form of function point analysis (FPA), such as COSMIC FFP; use case points (UCP); or simply expert opinion. These can work well for small projects, but can be increasingly inaccurate the larger the project.

The introduction of use cases has improved programming efficiency but has not generally led to improved cost estimation of complex systems. In fact it could be argued that since many estimation techniques are still based on functions, use case estimation has not yet fully adapted. ISO/IEC has approved four separate functional size counting methods standards because it cannot currently select the most reliable, and this demonstrates the lack of global agreement on the effectiveness of this type of estimation technique.

The use case is written in an easy to read style, making it easy for the business to understand what the system will do. Estimators are able to rapidly gain an overview of the requirements, which for a small system is usually sufficient for reasonably accurate cost estimates.

Typically a use case document would include statements such as:

*The system checks the data is correct as per section 19.1 of the regulations. The system then calculates a rebate as per section 19.2 of the regulations.*

The detailed functionality of the system is not specified in the use case, but in ancillary documents. This could amount to many thousands of pages of documentation including rules, conditions, exclusions and commentary. In fact, use case writers strive not to functionally decompose the system. They do this by checking for the following problems:

- Small use cases, in which the description of the flow of events is only a few sentences.
- Too many use cases, where the number of use cases is a multiple of many hundreds, rather than a multiple of tens.
- Use-case names that are like “do this operation on this particular data” or “do this function with this particular data”.

Figure 1: Relationship between Use Cases and Functions
Use cases do not make it easy for estimators of software systems to include costs associated with logical and infrastructure components. Consider the example of a system that must generate receipts. The use case documentation would read something like this:

Once the system accepts payment it will generate and forward a receipt to the postal address of the customer.

The use case diagram is shown in Figure 2. The description is innocuous, however the implications are complex. It would not be appropriate for the “offline printing” use case to be included into the “process application” use case (or any other use case that requires offline printing). The “offline printing” use case describes the process of printing the documents and mailing them, not the generation. A use case to pass the data to offline printing is not included or created, since that would be functional decomposition. The offline printing use case will require considerable supporting documentation on: document formatting, printing mechanisms, the format of the data required by a printing contractor or a bulk mailing machine, mailing methods, and mailing frequency.

Figure 2: Sample Use Case Model

The unsophisticated language of the use case contributes to the problem of identifying infrastructure elements for estimation purposes in large and complex systems. It also lures the estimator into a false sense of security. There are often implied requirements that the domain experts who represent the business understand, but which the estimator would not recognize. This could result in omissions of infrastructure elements and occasionally multiple inclusions of the same elements.

In the example given above it would be easy to underestimate the effort required for formatting the print job because it sounds simple, just “format the document”. Given the time pressures on the estimator no consideration is given to the type of formatting required and a number of issues are left open, including the size of paper and whether it is critical for the output to align with fields of pre-printed forms.

Another problem area in use case estimation relates to the complexity of the GUI. The use case should not describe the screens that are to be presented to the user. The use case writer would write something like:

The actor selects the customer and displays the account information.

Nothing is assumed about how the customer is selected or how the accounts are displayed. The use case writer would typically be thinking of the “average” customer who has a few hundred entries in the account, however large corporate customers could have many millions of account items. It is implied that these variants must be handled. It is the designer’s job to decide on the most effective way to display the information to the user, but the design is unlikely to have been undertaken at the stage where an estimate is required. A common approach is to introduce a multiplication factor to the estimate, such as a multiple of one for what appears to be a simple set of screens, a multiple of two for medium complexity, and a multiple of three for high complexity fat client. This has obvious parallels with FPA and UCP described in sections 3 and 4. If the estimator makes the wrong choice and estimates a GUI as simple when it is hard the estimate will be at least a factor of three out. It is also arguable whether a multiplication factor of three is sufficient for a complex fat client.

3. Function Point Analysis

FPA was first published in 1979. “In 1984, the International Function Point Users Group (IFPUG) was set up to clarify the rules, set standards, and promote their use and evolution. FPA provides a standardized methodology for measuring the various functions of a software application. FPA measures functionality from the user's point of view, that is on the basis of what the user requests and receives in return. The function points (FPs) are obtained by measuring the software application from two distinct perspectives:

1) The functional size, calculated by assigning weights to each individual function...

2) The value adjustment factor (VAF), calculated using predefined general systems characteristics (GSC) to assess the environment and processing capacity of the software application as a whole.”

ISO/IEC 14143-1 [6] “defines the fundamental concepts of Functional Size Measurement (FSM) and describes the general principles for applying an FSM Method”. It “does NOT provide detailed rules on how to measure Functional Size of software using a particular method, use the results obtained from a particular Method, or select a particular Method.”

IFPUG [7] defines Functional User Requirements as “a sub-set of the user requirements. The Functional User Requirements represent the user practices and procedures that the software must perform to fulfil the users' needs. They exclude Quality Requirements and any Technical Requirements”.

IFPUG 4.1 Unadjusted (ISO/IEC 20926:2003) and COSMIC FFP (ISO/IEC 19761:2003) from the
A WBS can be expressed down to any level of
definition. A WBS displays and defines the product, or
product-oriented family tree composed of
definitions. Defense [15] is:

evidenced by the Project Management Institute's [14]
component of project management methodologies, as
only small, in the order of 2000-3000 hours of effort.
estimates can be improved. The projects studied were
by tuning the technical and environment factors
underestimated the effort required. They conclude that
projects and found that these all consistently
development environment used. They studied three
value. The productivity factor is dependent on the
function point approach of Albrecht [10], variations
approach similar to FPA. While Karner [9] modified
the function point approach of Albrecht [10], variations
on UCPs have been proposed by Rational Software and
other software companies, although there seems to be
suggest that Karner's approach is similar to the function
point approach, using weighting of use cases and
Actors.

To calculate the UCPs, each actor and use case is
categorized according to its complexity and assigned a
weighting (the complexity of the use case is measured
by the number of transactions). The unadjusted UCPs
are calculated by summing the weights for each actor
and use case. This raw value is adjusted based on the
values for technical factors and environmental factors.
Finally the adjusted UCPs are multiplied with a
productivity factor.

Anda et al [13] describe 13 technical factors and
eight environmental factors to adjust the unadjusted
value. The productivity factor is dependent on the
resources used on the project and the language and
development environment used. They studied three
projects and found that these all consistently
underestimated the effort required. They conclude that
by tuning the technical and environment factors
estimates can be improved. The projects studied where
only small, in the order of 2000-3000 hours of effort.

5. Work Breakdown Structure

The Work Breakdown Structure (WBS) is a key
component of project management methodologies, as
evidenced by the Project Management Institute's [14]
production of a WBS Standard. The WBS concept, as
defined by its originator, the US Department of
Defense [15] is:
- A product-oriented family tree composed of
  hardware, software, services, data, and facilities…
- A WBS displays and defines the product, or
  products, to be developed and/or produced. It
  relates the elements of work to be accomplished to
each other and to the end product.
- A WBS can be expressed down to any level of
  interest...

The ability of a project manager to manage costs
versus budget is critical to the success of a project.
Project managers monitor costs on an ongoing basis at
the individual WBS level. The software “products” and
associated costs for a system have been traditionally
represented in a WBS on a functional basis,
corresponding well with functionally based estimation
techniques and the functional representations of
requirements. In the absence of alternative guidelines it
is likely that use cases and their estimated costs will be
directly mapped into the WBS as “functions”. The
many-to-many problem described in section 1 implies
that this approach is flawed for large, complex systems.
Project managers may not be supplied with initial use
case estimates that are accurate and in addition may
incorrectly allocate infrastructure components.

6. System Complexity in a Real Project

This section describes a large multi-million dollar
system development project for a government
department, which specified requirements as use cases
and which experienced budget and schedule overruns
partially due to complexity issues.

The system was specified in 134 use cases,
amounted to 900,000 source lines of code (SLOC) and
took five years to develop through three major releases
and numerous minor releases. It had interfaces with
five major systems run by independent government
departments, and 20 smaller interfaces.

The use case model and the actual use cases were
developed and written by business users, with a range
of experience within the business domain. The business
users found they quickly were able to grasp the
concepts of use case writing and were able to
determine a series of issues for the business to address
as part of the development of the use cases. The use
cases were then approved by middle management who
were also the domain experts.

The coding estimates were produced from these
approved use cases based on a limited functional
model, due to time constraints to meet deadlines. A
WBS was developed to allow tracking of the project,
but was based directly on the use cases. This meant that
the developers could not easily report earned value to
the project management during the course of the
project for any of the infrastructure components and
any of the use cases that suffered from the
combinatorial complexity issues.

Most of the personnel who developed the use cases
had a comprehensive understanding of the basic day-
to-day elements of the business processes, but could
not specify all of the detailed business requirements.
The detailed knowledge of the specific business
processes and requirements resided in the heads of
single individuals. In a number of cases the use case authors were not aware of the added complexity with a specific business domain.

It soon became apparent that there was hidden business complexity in a number of areas. An example was the interface to a large financial institution that received many tens of millions of dollars every month from the Department. This fiscal transfer was a legal requirement for insurance. The existing interface had been written in the early 1990's in COBOL. The use case stated:

*The system will produce a report of the day’s takings and EFT the monies to …*

The calculation of the monies was straight forward, as it was simply the summation of the various cost codes, but the report was considerably harder as the detailed knowledge of the data calculations had been lost. Eventually a document from 1994 was found and the COBOL was reverse engineered to produce the requirements. The original estimate was an order of magnitude out because of this hidden complexity in the reporting.

The system required an extensive list of batch jobs to be executed each night and process actions that occurred during the day. The use cases that generated the data described loosely what data was required and each batch job was represented as a use case. The detail of linkages between the sources of data for the batch jobs was not present in the use case. It was only when the use cases were being fully specified at the design phase that this information became available and the true complexity of the interface revealed, and again the task was significantly underestimated.

The system had two types of auditing: local auditing for the department and external auditing for another department. The local auditing was simple:

*All transactions will be audited.*

But the external auditing was more complex as the use case stated:

*All transactions of interest will be audited.*

Only when the system was being designed did the enormity of this task become apparent. The definition of “of interest” was extremely difficult as the external agency did not want all of a particular type, but subsets based on data within the transaction. Although the task was not difficult it was extremely time consuming and again was badly underestimated.

The financial reporting was extremely complex owing to the nature of the payments. A single payment would have components that would be distributed between as many as four government departments. This requirement was written in the use case, but each agency had a different requirement for the data format, ranging from modern asynchronous XML messaging to roll ups in Oracle Financials. This added complexity was underestimated.

### 7. A Possible Solution

Research undertaken for this paper shows little evidence that the academic community has scientifically investigated the link that should exist between the costs applied to use cases and those costs used to represent requirements in a complete WBS. The experience of the described project and subsequent research suggests that directly mapping use cases into the WBS is not necessarily valid for large or complex projects. This is particularly an issue when there are significant numbers of “include” and “extend” use cases and common functional elements that do not warrant a use case, but are key core business/system functional components. In these situations there is a many-to-many relationship between use cases and functionality.

FPA has a long history of success with functionally specified requirements and has now been applied to use case specifications. However the authors argue that where use cases describe large or complex systems there is no simple functional output from use cases at the estimation stage of the project. FPA has been applied successfully to small scale use case systems [16], but could encounter difficulties with large scale systems because not all functional requirements can be initially defined.

What is missing is a mapping layer that maps the use cases to the functional and infrastructure requirements. This second layer of decomposition could potentially map directly into the WBS and provide the basis of function point or UCP counting at the estimation stage. In the ideal world this would be a full and detailed decomposition of the use cases, but this is impractical during the estimation process, since estimates need to be produced within tight time frames. Thus the functional decomposition would need to be at a high level, but must be of sufficient detail to flush out as many of the implied requirements and hidden complexities as possible, and as early as possible. This would allow the model used for estimation to be formalized within the initial WBS of the project. Once a project was underway and detailed analysis undertaken, this initial WBS could be expanded to produce a final WBS that identified functionality and costs in exactly the same format used in the original use case description and estimation activities. This would give the project manager more accurate ongoing information on the delivered value of the requirements (earned value) against the expenditure.

Mapping the use cases to high level functional business and system requirements within the WBS at
the estimation stage would allow the complexity of a use case to be described, rather than implied, and map elements of disparate use cases to single WBS elements. Then each of the functional elements could be estimated as accurately as possible using established techniques such as FPA or UCP.

An important resulting benefit would be a cost breakdown common to the procurement, project management and software engineering lifecycles that could be consistently used to portray value.

Typically a budget allocation is set at an early phase of a procurement process and a decision on contract price made on these budget constraints and suppliers’ tender responses. This proposal could potentially allow procurers to more accurately determine the value for money associated with each supplier response, and more accurately evaluate:

- Whether the suppliers have effectively documented and costed the implied requirements;
- Which areas are anticipated to be most expensive;
- Whether the budget allocation is feasible for the scope of the project.

A consistent representation of requirements and costs from the early procurement stages through to project completion should lead to greater ongoing harmony between customer contract manager, supplier project manager and software developers.

Greater competency in the procurement process through a better understanding of the impact of complexity upon costs also significantly benefits capable suppliers. Costing errors contribute to problems in value for money judgements and can lead to the selection of an inappropriate supplier at the expense of the most capable. Additionally, regardless of the cause, suppliers often bear the political and financial implications of unsatisfactory contract outcomes. Hidden complexity is often a major contributing factor to poor outcomes.

8. Further Work

The authors have drawn upon experience and research in the fields of software engineering, project management and procurement to highlight problems in the cost estimation of large complex use case based software engineering projects. A simple modification is proposed that could contribute to increased estimation accuracy and improved project outcomes.

A multi-disciplinary research project is being developed to progress this research, including experts in use case requirements, function point analysis, project management and procurement. A number of industrial organizations that produce and/or use large complex use case based software systems have agreed to participate in the project to develop these ideas further. Funding has been sort from the Australian Government to fund the significant academic component of this project.

9. References