A PRICE Systems Thought Leadership Article

By Arlene F. Minkiewicz, Chief Scientist, PRICE Systems

Estimating Software from Use Cases

Abstract

As the software industry improves its processes, there is an increasing demand for more information earlier in a project. Project managers and project planners want to start estimating as soon as preliminary requirements are complete. Unfortunately, preliminary requirements don't generally cover the architecture or implementation details required by most software estimating methodologies to perform a successful estimate. What is needed is a methodology that helps translate information early on in the process into a credible estimate of effort and cost.

Use cases describe typical transactions between the user and the software. This paper describes a new sizing methodology based on Use Cases. This methodology enables product managers to use available information to perform estimates earlier in the software project life cycle before architecture and implementation decisions are made, as well as providing a framework for capturing historical data to facilitate better future estimates.

Introduction

In early versions of parametric software estimation models, software size was measured in source lines of code (SLOC). From a historical perspective, SLOC made sense as a metric to gauge productivity of software development, especially when projects have similar programming languages and functional characteristics. This held true when software development projects typically employed third generation programming languages and standard functional programming practices. Over time, advances in software development practices and processes have challenged the effectiveness of SLOC as the best metric for estimating software costs. Programming languages have become more efficient, packing more and more functionality into a single line of code; requirements have emerged for productivity metrics based on business functionality rather than the number of lines of code required implementing that functionality; and the object oriented methodology of software development has changed the value of lines of code.

PRICE S and now True S have addressed these emerging needs by adding Function Point and Predictive Object Point (POP) calculations that act independently of the more traditional SLOC calculations. Today, an additional sizing need has surfaced as product managers find that they need to perform estimates earlier in the software product lifecycles, before architecture and implementation decisions have been made. To address this PRICE is adding a Use Case Sizing methodology to the True S product which will make it possible to perform estimates earlier in the software project lifecycle using the information available at that time.

Use cases, first introduced by Ivar Jacobson in the mid 80's[1], provide a language for describing the requirements of a software system in a way that facilitates communication between the eventual users of the system and the developers of the system. Each use case describes a typical interaction that may occur between the user and the software, focusing on the functions that a user may want to perform or have performed rather than on how the software will actually perform those functions.

Use cases are a top level description of the functionality the software system is intended to deliver. By themselves, they do not contain details about system architecture and implementation of functionality necessary to perform a comprehensive software estimate suitable for detailed project planning. They do, however, provide an excellent mechanism for estimates early in the software development lifecycle supporting rough order of magnitude, feasibility study and early project planning estimates.

This paper discusses the Use Case Sizing methodology added to the PRICE True Software Estimating product. Section 2 presents the solution methodology. In Section 3 the problem is described and bounded. Section 4 gives a detailed description of what use cases are and what they are good for. In Section 5, Use Case Conversion Points are defined and the research behind them is discussed. Section 6 presents the limitations and caveats of an estimating methodology based on use cases and Section 7 discusses conclusions and further directions.

Solution Methodology

The first step in any Operations Research project is to identify the problem being solved. The problem we are attempting to solve is that of identifying a credible methodology for creating estimates early in a project when only use case information is available. The data set included use case and function point data for a variety of software systems developed primarily for the federal civilian market place. Literature reviews and expert knowledge supplemented the data collection process.

Once the problem is identified, the next step in constructing a parametric cost estimating solution is to study and understand what information is available at the use case stage of a project. Data was studied and a verifiable relationship was established between use case information and an accepted size metric, function points. Research led to the development of a mathematical model based on this relationship. This mathematical model was than exercised by software developers, project mangers and estimators to determine how it fared when applied to real life situations. Once satisfied that the model was useful for practitioners, data from various datasets containing both commercial and aerospace data was applied to determine where it worked well and where further work is required.

The Problem

Early estimators measured software size using Source Lines of Code (SLOC). From a historical perspective, SLOC made sense as a metric to gauge productivity of software development, especially when projects have similar programming languages and functional characteristics. This held true when software development projects typically employed third generation programming languages and standard functional programming practices. Additionally, SLOC was (and continues to be) the best metric for collecting actual software size data because SLOC counts can be automated to provide consistent counts across projects.

Over time, improvements in software development practices and processes have challenged the effectiveness of SLOC as the best metric for estimating software costs. Programming languages have become more efficient, packing more and more functionality into a single line of code; requirements have emerged for productivity metrics based on business functionality rather than the number of lines of code required to implement that functionality; and the object oriented methodology of software development has changed the value of lines of code. In many organizations, Function points and object-oriented metrics have replaced source lines of code since they represent better measures of the work being done or the functionality being delivered.

Today, project managers not only want size metrics in line with the practices and tools they employ, they want to be able to perform estimates earlier in the software lifecycle. The challenge here is that the information available early in a project lifecycle is not sufficient to estimate any of the traditional size measures. SLOC and Object point measures depend on decisions made about architecture, programming language, databases and technology. Even Function Points, intended to be a measure that transcends these differences in software projects, require information about data structures and data exchanges that are not available at this point in the project.

Early efforts attempted to draw relationships directly between effort and use case counts. These efforts led to drastically disparate metrics ranging for hours per use case ranging several orders of magnitude [2,3]. Clearly, additional information about the complexity of individual use cases was required to reign in these values. In 1993 Gustav Karner introduced such a measure, use case points, which used attributes of the use case to assign relative complexity differences between use cases [4]. This research uses Karner's work as a starting point but is unique in that it identifies a relationship between Use Cases and Function Points and then uses that relationship as a basis for effort estimation in the context of a methodology that addresses factors other than size that impact effort in software projects. The research is based on an analysis comparing top-level use cases with function points.

Use Cases

Ivar Jacobson first introduced use cases in the mid 1980's[1]. One motivation behind their inception is the difficulty experienced in communication of requirements between the developers of software systems and those who eventually consume those systems. They provide a language for software people to effectively communicate with users. One can think of a use case as an English translation (or whatever language you happen to speak) of requirements for software. They provide an extremely useful tool to facilitate the development of software that meets user expectations and does 'the right things'.

Each use case describes a typical interaction that may occur between the software system and a user of that system. It presents a specific and complete instance of behavior of that system. The user could be a human user interfacing with the software through a graphical user interface or another software system talking to the system through some predetermined protocol or an exposed Application Program Interface (API). The focus of the use case is on the behaviors of the software not on how those behaviors will be implemented.

A use case description in a simplest form contains:

- Name of the use case
- List of the actors involved in the transaction
- The steps the software is required to perform to make the behavior happen.

The use case name simply describes the goal of the use case and is typically in the form of 'action verb + object'. In a use case description, the actors represent the roles that depend on this behavior or that the software depends on to accomplish this behavior. An actor is either a person or another software system that is assuming a particular role in the context of the use case. The steps in the use case description are simply a list of all of the things the software has to do or facilitate to satisfy the goal of the use case. A good use case focuses at the top level on the successful satisfaction of the goal but more detailed use cases should follow to account for what should occur in the cases where the goal cannot be satisfied.

As way of a simplified example, let's consider the web site of a major department store. One of the things this software should be able to do is allow customers to place orders on line. The use case for this activity might look like this:

Use Case Name:	Place Order
Actors:	Customer
	Credit Check Service
Steps:	

1. Customer browses for items
2. Customer adds desired items to 'shopping cart'
3. Customer indicates desire to 'check out'
4. Customer enters shipping information
5. Customer enters billing information
6. Credit check is performed
7. Confirmation is sent to customer

A common criticism of use cases is the fact that there is no formal standard for developing use cases. Some use cases are very high-level and informal such as the example above while others go into a great deal more detail, defining at a very low level the steps and sub steps required to execute the desired behavior [5]. Some say that use cases need to be kept informal and simple or their entire reason for existence is negated. End users will generally lose focus if asked to slog through pages and pages of use case descriptions and the results of their review will be no better than if they were asked to read a formal specification or complex design document. For the purposes of this research, since it is intended as a tool for use early in a projects lifecycle, high-level, simple use cases were assumed.

Research Details – Use Case Conversion Points

The data collected for this project included the study of use cases for twenty software components. Function point counts were performed on each of these components by a Certified Function Point Specialist. (A Certified Function Point Specialist is a person who learns the counting practices standards for function points as published by the International Function Point Users Group (IFPUG), takes and passes the IFPUG Certification exam, and keeps this certification current through periodic retesting) The use case data that was collected from top-level use cases included

- Number of actors
- Types of actors
- Number of steps in the behavior description

For each use case, a complexity was assigned based on the number of steps or behaviors detailed in the description. Each complexity was assigned a weight. As a starting point, the weighting values were those from Karner's work [4], although this research extended that work to include more complex use cases. Figure 1 details the weight and criteria for each use case complexity value:

Complexity	Description	Weight
Simple	#steps <= 3	5
Average	4 <= #steps <= 7	10
High	8 <= #steps <= 13	15
Extra High	#steps >= 14	20

Figure 1: Use Case Complexity Weights

It is important to note that prior to the assigning of complexity values to use cases, it was necessary to normalize the data sets to ensure that, as much as possible, the steps in the use case descriptions were of the same relative orders of magnitude. This type of normalization can be accomplished through interviews with the developers of the use case or through review of lower-level use cases if they are available.

Actor complexity is determined by examining how each actor is expected to interface with the software. Interface through a well-defined API will generally be simpler than an interface that involves reading and parsing streams of data. Certainly any interface that needs to accomplish intelligent communication with humans presents the most challenges. Figure 2 details the weight and criteria for each actor complexity value:

Complexity	**Description**	**Weight**
Simple	External system interfaces through an API	1
Average	External system interfaced through a text based protocol	2
Complex	End user interfacing through Graphical User Interface (GUI)	3

Figure 2: Actor Complexity Weights

At this point in the research we had function point counts for each component and use case and actor complexity values for each of the use cases in the components. The next step required determining how to sensibly combine this data into a single metric, Use Case Conversion Points that we could then correlate to the function point data available. The following three sets of relationships were experimented with:

Use Case Points with Actors
UCP1 = Weighted Use Case + Sum(Weighted Actors)
For each use case the weighted use case value was added to the sum of the weighted actor values

Use Case Points with no Actors
UCP2 = Weighted Use Case
For each use case the value of the weighted use case is assigned

Use Case Points with Simple Actor
UCP3 = Weighted Use Case + Average(Weighted Actors)
For each use case compute the average of the weighted actors and add this to the weighted use case value

The use case points for each component were summed and compared to both the function point count and unadjusted function point count for that component. Unadjusted function point counts do not include any influence from the general system characteristics (characteristics not related to inputs, outputs, inquiries, or logical files). This analysis was repeated for each component using each of the three relationships above. Of all the cases tested, the use case points with actors (UCP1 above) showed the best and most consistent correlation with Unadjusted Function Points. Figure 3 shows a sample of the analysis for a set of components that were of a consistent application type.

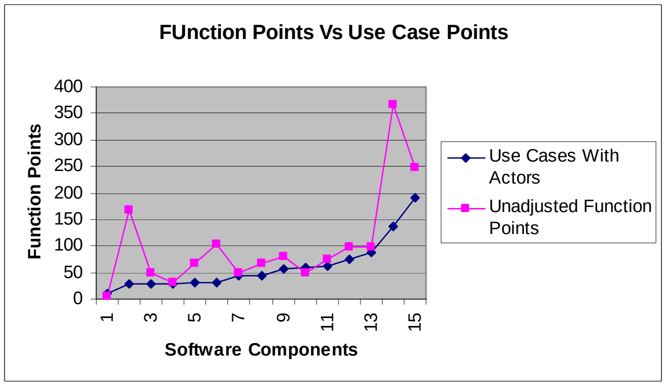
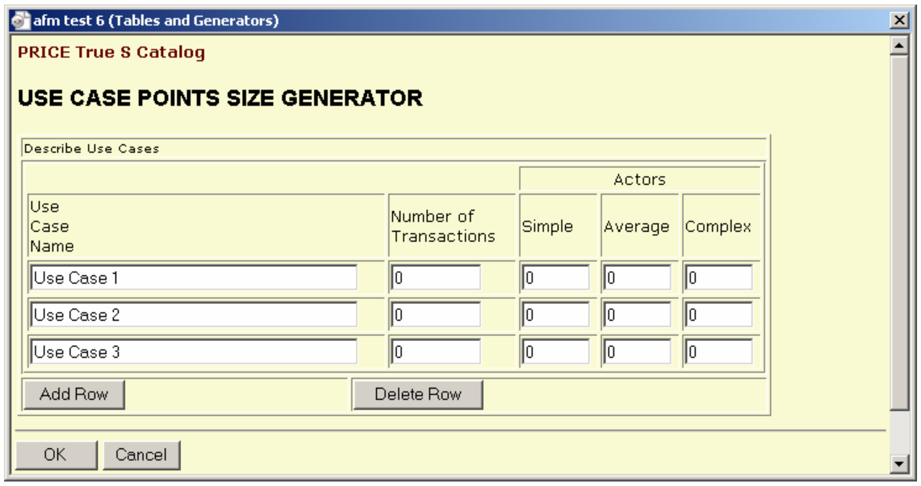


Figure 3: UFP vs. UCP1 for Fed Civilian Apps

From this research a relationship was developed that predicts an unadjusted function point count from use case and actor complexity. Comparisons between predicted and actual unadjusted function point counts across the entire study give an r-squared value of .62. While this is not a great result it shows promise. The analysis results improve dramatically when the few data points that were obvious outliers were removed. This indicates a future direction for this research to be an analysis of these outliers to identify what factors make them outliers and what information might be available in the early stages that would identify these factors.

Figure 4 shows what questions an estimator would have to answer to utilize this relationship in the True S model.


afm test 6 (Tables and Generators)
PRICE True S Catalog
USE CASE POINTS SIZE GENERATOR
Describe Use Cases
Add Row
Delete Row
OK
Cancel

Use Case Name	Number of Transactions	Actors: Simple	Actors: Average	Actors: Complex
Use Case 1	0	0	0	0
Use Case 2	0	0	0	0
Use Case 3	0	0	0	0

Figure 4: Interface for Use Case Conversion Points

Limitations and Caveats

While this research has promise, there certainly are limitations to the extent to which the results should be used. First of all, the data set studied is limited in both number of data points and range of application types. While in theory the type of software being developed is something that is outside the confines of what is traditionally considered software size, and is certainly something that can be accounted for by other cost drivers in an estimating model, the reassurance that this is in fact the case needs to come from a thorough investigation of software projects of many different application types and domains. Additional data points are required to deliver this level of confidence to the Use Case Conversion Point metric.

An area that is not covered explicitly by this research but that is a reality to estimators is the fact that it is not always correct to assume that there is a one to one correspondence between use cases and function points. The functionality covered by one function point count may be utilized in the delivery of functionality required for several use cases. It is important for estimators to look for instances of such an overlap and adjust appropriately by adjusting complexities for use cases and actors to eliminate the affects of these overlaps.

Lack of standardization of use cases can make the process of estimating from use cases problematic. Because some use cases are written to a far greater level of detail than others, the number of steps in the use case can differ substantially among use cases intended to describe the same behavior. It is important to keep use case steps to a consistent level of detail when using a measure that is based on use case description. This may require reengineering of lower level use cases to achieve a consistent level of detail. Having organizational standards in place for the development of use cases simplifies and greatly improves the success of a methodology that estimates from use case artifacts.

It is important that consumers of this research understand what is does and does not deliver. Any estimate performed early in a project lifecycle using top-level requirements information is not going to deliver the same kind of fidelity as an estimate conducted once the project team has made decisions about the architecture and other implementation details. Regardless of the methodology used, estimators and project planners should understand that these estimates are intended to establish order of magnitude analysis and should be used for trade-off and bid/no-bid type decisions. Estimates that require more fidelity should incorporate additional project information based on a thorough understanding of the project. This being said, it is not unrealistic to project situations where an organization with excellent data collection and use case standardization policies could calibrate such a methodology to deliver quality estimates with a high degree of fidelity based on use case artifacts.

Finally, there are limitations imposed by the fact that this research focused on predicting function points from use case artifacts rather than predicting effort. Clearly it would have been better to conduct the research with effort data and other cost driver information. The transitional step of predicting function points from use case information adds an additional level of abstraction to an already abstract process. As is often the case, the research was constrained by the available data.

Conclusions

Use cases have proven to be an effective language for describing software requirements in a way that facilitates communication between the developers of software systems and the consumers of these systems. They present an ideal forum for the kind of back and forth dialog that should take place when requirements are being elicited and understood. Use cases can be available very early in the lifecycle of a software project and thus offer promise in addressing the needs of project managers who require estimates early on before significant architectural and implementation decisions have been made.

This research has demonstrated an approach for using use case artifacts, available with the development of high-level use case descriptions. An algorithm has been developed that predicts a function point count from an assessment of use case and actor complexity. To date this approach has shown promise when applied to a limited set of data points. Since its implementation in True S, the approach has been used on additional data sets, outside the domain of the original data sets, with favorable results. Additional work is required to develop confidence in this approach. Research is underway to identify factors driving outliers in the original data sets and incorporating these factors into the methodology. Work is also being done to add a calibration factor to the methodology to aid organizations in the tailoring process adapting this methodology to use cases written to varying levels of detail. Finally, as additional data is made available, the algorithms will be refined to reflect additional research.

While the results of this research indicate that estimates from use case artifacts is possible, it is important to reiterate the premise that such estimates should be used within the context of the level of detail from which they are derived. An estimate performed with only use case information and no knowledge of what the software architecture will be, what programming language(s) will be employed, what the skill of the development team will be, and what technologies will be employed will not deliver the same level of fidelity as an estimate performed later on in the project when those decisions are made. It is imperative that estimators use this estimate as such, interpreting the results within the context of the risks associated with estimates from early in the lifecycle

References

[1] Cockburn, A., "Use Cases, Ten Years Later", STQE Magazine, Mar/April 2002.
[2] Fowler M., *UML Distilled*, Addison Wesley, Reading Massachusetts, 1997.
[3] Globaltester, "Use Case Points", available at www.globaltester.com/sp7/usecasepoint.html
[4] Banerjee,G., "Use Case Points – An Estimation Approach ", available at http://www.info-knowledge.nl/docs/Proj%20Man/UseCasePoints.pdf
[5] Gottesdiener, E., "Use Cases: Best Practices", Rational Software white paper, June 2003
[6] Smith,J., "The Estimation of Effort Based on Use Cases", Rational Software white paper, available at http://www.uml.org.cn/RequirementProject/pdf/finalTP171.pdf

About the Author

Ms. Minkiewicz leads the Cost Research Department as Chief Scientist at PRICE Systems. In this role, she is responsible for the research and analysis necessary to keep the suite of PRICE Estimating products responsive to current cost trends. She works with industry leaders to collect and maintain cost research data and offers analyses of this data to the cost estimating community through the PRICE products.

Arlene's most recent accomplishments include the development of a catalog of cost estimating relationships for hardware and systems projects that will be delivered to the cost estimating community as part of the TruePlanning suite.

Arlene frequently publishes articles on estimation and measurement in publications such as Software Development Magazine and Crosstalk. She speaks frequently on these topics at conference such as STC, ISPA, SCEA, IEEE Aerospace Conference, SEPG, and many others. Her 'The Real Costs of COTS-Based Software Systems' paper was recognized in 2004 by ISPA and SCEA as Best Paper in the Software Track. Her paper "A Case Study and Assessment of a COTS Upgrade for a Satellite Ground System", co-authored with Marilee Wheaton of the Aerospace Corporation, received Best Paper in Software Track in 2006 by SCEA and her paper "The Evolution of Hardware Estimating" received Best Paper in the Hardware and EVM Track at the 2007 ISPA/SCEA joint conference.

Arlene can be contacted at arlene.minkiewicz@pricesystems.com

More information on software cost estimating and the TruePlanning Suite can be found at:
www.pricesystems.com

PRICE Systems World Headquarters
17000 Commerce Parkway, Mt. Laurel, NJ 08054
voice 1.856.608.7200 / fax 1.856.608.7247 / www.pricesystems.com
Please contact us at www.pricesystems.com/contact/contact.asp for more information.

PRICE Systems - Washington, D.C.
1700 N. Moore Street, Suite 1100, Arlington, VA 22209
voice: 1.703.740.0087 / fax 1.703.740.0088

PRICE Systems International
Meridian Office Park, Osborn Way, Hook
Hampshire RG27 9HY England
voice 44.1256.760012 / fax 44.1256.762122