ESTIMATION USING USE CASE POINTS

Mel Damodaran
Computer Science Program, University of Houston-Victoria
Victoria, Texas 77901, USA
damodaranm@vic.uh.edu

and

Aqua Netta E. Washington
3131 Hayes Rd. #1001
Houston, TX 77082
AWashi4119@aol.com

Abstract

Use case models are increasingly being used to capture and describe the functional requirements of a software system. There are different approaches and methods to successfully estimate effort using use cases. A few researchers have tested the use case points method and analyzed their findings. The results, though not conclusive, indicate that the use case points method has potential to be a reliable source of estimation, much like the function point method, and it can have a strong impact on estimating the size of software development projects, especially when it is used along with expert estimates. Also, since use case modeling is increasingly being utilized as the method of choice to describe the software and system requirements and as a basis of design, development, testing, deployment, configuration management and maintenance, it makes sense to have an estimation method that makes use of them. This paper looks at the potential of successful application of the use case points method for estimating the size and effort of software development projects, including the major limitations and offers some possible remedies.

Keywords: Use case points, software development effort estimation, software project estimation.

1. INTRODUCTION

To capture the functional requirements of a software project, use case models are often employed. Use case modeling is a technique that has been widely used throughout the industry to describe and capture the functional requirements of a software system. Use case points is a new method for estimating software development. Since use cases and scenarios are developed as a normal part of requirements gathering and analysis, and since they capture an accurate representation of the users' requirements, it makes sense to base the more difficult task of estimation of size and resources on them, as opposed to any other technique such as function points, lines of code etc. Another advantage of the use case based estimation is that use cases are maintained with two-way traceable capability using modern requirements management tools. Two-way traceability may be maintained between use cases and many other software engineering artifacts including design, code, test documents, configuration management, architecture and deployment models. Use cases are at the heart of the "4+1" model of the Unified Process. Use cases are usage-based and user centered, rather than system- or design-oriented; it describes the "what-"s rather than the "how-"s, hence they are more robust and less changeable than function points or lines of code. Use cases are artifacts, units that make sense to the user in his/her domain.

Bente Anda (Anda 2001) compared the use case points method with expert estimates made by experienced software developers. The use case points method gave an estimate that was close to the actual estimates produced by experienced software developers. The estimation method gave a satisfactory level of the magnitude of relative error and the mean magnitudes of relative error were within the estimated constraints. The

results of this study indicate that the use case points method can be successfully used to estimate the development of software effort.

Gustav Karner of Objectory AB (later acquired by Rational Software) developed the use case points method of software estimation (Karner 1993). Its influences came from the classic function point method. Arnold and Pedross (Arnold and Pedross 1998) have also studied and reported their experiences using the use case points method. Their particular method was similar, but not identical to the one inspired by Karner. The results that they found solidifies their initial hypothesis that it is a reliable source for estimating software development effort; on the other hand, they did notice that the analyzed use case models differed in the aspect of details and they concluded that the measured size might have differed accordingly. They also found that because use cases do not have a standard format it is difficult to measure the software size based solely on them.

There are also some alternative methods for estimation based on use cases. One approach is to use the use case model as a means for counting function points, which can then be used to obtain an estimate of effort. As yet another alternative, the use case model can also be a means for estimating the number of lines of code and this value can then be used to estimate effort (Anda 2001). These two methods are attempts to make use of the industry's extensive experience with estimation using function points and lines of code.

Industry use of Use Case Points method of estimation is very rare. One noteworthy example is the use of this method by Rakesh Agarwal et al in Infosys, Bangalore India for an Internet project they completed for a client. (Agarwal 2001). The authors report that they used the Use Case Points method "as a means of arriving at a ballpark estimate of the effort involved in developing the system." One possible reason why there has not been more widespread use of this method in industry is that this method has not been incorporated in the popular project estimation tools. Another reason may be the relative newness of use case points, even use cases as a standard method of describing requirements; without good historical productivity figures, it is not possible to use them to estimate effort and cost.

One would assume that based on the characteristics of the use case model that it would be possible to form estimates of size and effort. Since the use case model is a widely used method to capture the functional requirements, one would think that there should be a use case based equivalent of the function point method. However, there are several difficulties that tend to prevent a use case based equivalent (Smith, 1999):

- There are many variations of use case specification styles and formalities which make it very difficult to define metrics.

- Use cases should represent an external actor's view of a system, and so a use case for a 500,000 SLOC system is at a quite different level to a use case written for a 5,000 SLOC subsystem
- Use cases can differ in complexity, both explicitly as written, and implicitly in the required realization;

- A use case should describe behavior from the actor's point of view, but this can be quite complex, especially if the system has states, as most do. So to describe this behavior may require a model of the system that can lead to too many levels of functional decomposition and detail, in an attempt to capture the essence of behavior.

The question remains, should we avoid use cases for estimation and rely instead on the analysis and design realizations that emerge? This presents another problem in that it delays the ability to make estimates and will not be satisfactory for a project manager who has chosen this particular technology to estimate his project. Early estimates will need to be employed; therefore other methods would have to be used. It is more efficient for the project manager to be able to obtain estimates early for planning purposes, and then improve them iteration by iteration, as oppose to delaying the estimation process and proceeding in an unplanned fashion (Smith 1999.)

There is a large amount of published work on describing and formalizing use cases, but there is very little on obtaining estimation metrics from use cases. (Graham 1995) proposes the idea of using 'task scripts' as a way to overcome the problems with use cases such as their varying lengths and complexity. Graham's 'atomic task script' is the basis for collection of a 'task point' metric. The problem with this task script is that it is very low-level, not further decomposable, ideally a single sentence. John Smith (Smith, 1999) tends to think that the root tasks look very similar to low-level use cases and the atomic task scripts like steps in use cases. Therefore, the problem of "level" still remains.

In the rest of this paper, we first look at Karner's method of use case points. Then we compare it with the function point method and analyze its implications as an estimation method. We discuss its potential as an estimation method, including some suggestions for its wider acceptability.

2. KARNER'S METHOD

We will now briefly explain the steps in the use case points method as used by Karner (Karner 93.) First, categorize the actors in the use case model as simple, average or complex and calculate the total *unadjusted actor weight (UAW)* by counting the number of actors in each category, multiplying each total by its specified weighting factor, and then adding the points. Next, categorize the use cases as simple, average or complex, depending on the number of transactions, including the transactions in alternative flows. Then the *unadjusted use case weights (UUCW)* are calculated by counting the number of use cases in each category, multiplying each category of use case with its weight and adding the products. The *UAW* is added to the *UUCW* to get the *unadjusted use case points (UUPC).*

Next, the use case points are adjusted based on the values assigned to a number of technical factors and environmental factors. Each factor is assigned a value between 0 and 5 depending on its assumed influence on the overall project. This step will produce 3 different formulas:

The Technical Complexity Factor:

$$TCF = 0.6 + (.01 * Tfactor)$$

The Environmental Factor:

$$EF = 1.4 + (-0.03 * Efactor)$$

Adjusted use case points:

$$UCP = UUCP * TCF * EF$$

Finally, the UCP is multiplied by a historically collected figure representing productivity, such as a factor of 20-staff hrs per use case point, to arrive at a project estimate. The result is an estimate of the total number of person hours required to complete the project.

3. IMPLICATIONS OF THE USE CASE POINTS METHOD OF ESTIMATION

Although the use case points method was influenced by the function point method, they differ in several ways. First, the function point method doesn't require a standardized notation. Conversely, use case points are based on the use case model, which allows easier development of estimation tools that automatically count use case points. Secondly, there are international standards counting function points. Use case points, on the other hand, is still not a standard or standardized practice. Therefore, there may be discrepancies from organization to organization on how use cases are developed and how use case points are calculated.

Use case models themselves have a direct impact on the estimation process. For example, the number of actors affects the estimates by combining actors with similar descriptions into one actor, the superactor. This increases the precision of the estimate and hence counting the actors only once. Another problem is whether included and extended use cases should be counted when estimating software projects. In some studies these particular types of use cases were omitted and in others they decided to count the included and extended use cases. It is clear that more research should be done on this particular topic because the results are too inconclusive. Another factor that affects estimation is the level of detail that is involved in the use case descriptions. The number of transactions within that use case measures the size of each use case. It is very difficult to determine the appropriate level of detail in each transaction when constructing a use case. There are no previous standards by which to determine the correct level of detail for all projects. The level of detail has been determined to affect the number of transactions, which consequently has a direct impact on the estimate obtained by the use case points method.

On the other hand, it does not seem very difficult for an organization to develop its own standards for granularity of analysis use cases and classify use cases as simple, medium or complex. Each such use case thus becomes a unit of functionality that must be incorporated into the software, and in totality can give an indication of the scope of the project, much like function points determining the scope of a project. In both cases, non-functional requirements and other constraints must be included as adjustment factors to arrive at the final estimate of the size of the project. If the organization has standardization in its use of use cases across projects and over time, it is possible to have a good database of historical measurements from which reliable cost and resource estimates may be determined.

4. CONCLUSION

In conclusion, use case points method of effort estimation is a very valuable addition to the tools available for the project manager. The method can be very reliable or just as reliable as other effort estimation tools such as COCOMO, function point and lines of code. All of the estimation methods are susceptible to error, and require accurate historical figures for productivity in order to be useful within the context of the organization. Use case points method is especially valuable in those system development projects where use cases are produced anyway. It is comparable to the function point method that has become quite respected throughout the industry. It provides estimates that are sometimes better than what experts can provide to the industry. Expert estimates should not be excluded from the process of estimation; rather it should be used in conjunction with use case estimates to ensure an accurate estimate. Lastly, with standardization and kind of national and international efforts that have helped the function point method become widely accepted, this

method also has the potential to become a mature and widely accepted estimation tool.

References

Agarwal, Rakesh, Santanu Banerjee and Bhaskar Gosh, 2001, "Estimating Internet Based Projects: A Case Study." Quality Week 2001, Paper 6W2

Anda, Bente, D. Dreiem, Dag Sjøberg and Magne Jørgensen, 2001, "Estimating Software Development Effort Based on Use Cases - Experiences from Industry", In M. Gogolla, C. Kobryn (Eds.): UML 2001 - The Unified Modeling Language.Modeling Languages, Concepts, and Tools, 4th International Conference, Toronto, Canada, October 1-5, 2001, LNCS 2185 Springer

Arnold, P. and Pedross, P. 1998," Software Size Measurement and Productivity Rating in a Large-Scale Software Development Department." Forging New Links. IEEE Comput.Soc, Los Alamitos, CA, USA, pp.490-493.

Graham, Ian, 1995, Migrating to Object Technology. Addison-Wesley, 1995.

Karner, G, 1993, "Metrics for Objectory". Diploma thesis, University of Linköping, Sweden. No. LiTH-IDA-Ex-9344:21. December 1993.

Smith, John, 1999, "The Estimation of Effort Based on Use Cases. Rational Software.Cupertino, CA.TP-171. October,