Software Size Measurement and Productivity Rating in a Large-Scale Software Development Department

Martin Arnold
Dep. of Software Engineering
EDS/FIDES Informatik
PO Box, CH-8036 Zurich
Switzerland
+41 12 98 69 56
martin.arnold@fides.ch

Peter Pedross
Credit Suisse
Paradeplatz 8
PO Box 100, CH-8070 Zurich
Switzerland
+41 13 34 72 05
peter.pedross@credit-suisse.ch

ABSTRACT
Some current object-oriented analysis methods provide use cases, scenarios or similar concepts to describe functional requirements for software systems. We introduced the use case point method to measure the size of large-scale software systems based on such requirements specifications. With the measured size and the measured effort the real productivity can be calculated in terms of delivered functionality.

In this status report we summarize the experiences made with size metrics and productivity rates at a major Swiss banking institute. We analyzed the quality of requirements documents and the measured use case points in order to test and calibrate the use case point method. Experiences are based on empirical data of a productivity benchmark of 23 measured projects (quantitative analysis), 64 evaluated questionnaires of project members and 11 post benchmark interviews held with selected project managers (qualitative analysis).

Keywords
measuring object-oriented systems, size metrics, requirements specifications, use case, scenario, software development process measurement

1 INTRODUCTION

At the end of 1993 a major Swiss Banking Institute (we call it SBI in this paper) introduced the object-oriented methods for the bank wide software development. For the object-oriented analysis the SBI adapted and extended the use cases of the OOSE-method [5], the CRC-method [12] and the scenarios of the OOAD-method [2] to their individual needs. To measure the productivity of software development organizations, the size of delivered software systems has to be known. Requirements descriptions with use cases, scenarios and object models build an appropriate basis for measuring the size of delivered functionality. The size is determined by the evaluation of scenarios and business objects and the adjustment to the technical complexity. The productivity is computed by dividing the size by the effort. The productivity metric is an important management rate at the SBI because it will be used to

- compare performance between companies (benchmarking),
- compare performance with other development departments,
- compare efficiency of different development technologies and
- do strategic planning based on ratio analysis (outsourcing, innovation, reengineering, continuous improvement of selected development activities, ...).

Use cases and scenarios are the basic descriptions for release planning and software size measurement. Such specifications fit well into the iterative and incremental software process model of OOAD [2], which has been set as a standard for object-oriented software development in SBI projects.

0-8186-8368-6/98 $10.00 © 1998 IEEE

2 COMPARISON WITH OTHER APPROACHES

Several object-oriented methods provide use cases and scenarios. We analyzed the methods and language concepts of OOSE [5], OMT [8/9], OOAD [2], UML [11], OBA [7], ROOM [10] and Syntropy [3]. However, there are as many syntactic and semantic definitions as methods but none of the existing methods provide techniques for size measurement based on use cases or scenarios.

The use case point method developed for the SBI is inspired by the approach presented by Karner in 1993 [6]. Our measurement method is also based on use cases. Therefore we selected the same name as Karner did, but in fact we designed a completely new concept. We introduced the first version in August, 1996 as the standard software size metric at the SBI.

The idea of use case points is inspired by the function point method [1]. It starts with the measurement of the system functionality specified with use cases and scenarios. Technical factors which are significant for the required system functionality of the software product are integrated into the measurement. The resulting use case points can be compared directly with the function points. We used the function point method to calibrate the use case point method. The method is constructed so that the measured use case points range from about 90% to 110% of the comparable function points. This accuracy is sufficient for our purpose because the function evaluation of the function point method can have an inaccuracy of up to 30%, when the size estimation is done by different project leaders [14].

3 FINDINGS, CAUSES OF SUCCESS AND FAILURE

We present some empirical findings and point out success and failure factors discovered at the SBI. Quantitative analysis is based on 23 measured projects in order to test the use case point method. The projects can be characterized as follows:

- All projects have been development projects for banking applications (credit risk management, electronic banking contract management, account management in retail banking, mortgage application, etc.).
- All software systems are interactive applications.
- The size of the application ranges between 200 and 4400 use case points.
- The development duration ranges from 1 to 5 years.

Qualitative analysis is based on 64 evaluated questionnaires of project members and 11 post benchmark interviews held with selected project managers [13].

3.1 Experiences with Software Size Measurement Findings

We found that requirements specification with use cases and scenarios can be used to measure the size of a software system. The application of the use case point method has been well accepted by project leaders. The size measurement of a software system is normally done in one to two hours. The spreadsheet based tool for the use case point method is well accepted by all project leaders. This tool has also been used for documentation and graphical presentation of size and productivity in project state reports. According to our experience with the use case point method, it seems to be an objective procedure and a reliable indicator for the size of delivered functionality.

Success and Failure Factors

Business objects were evaluated without difficulties because modeling of business objects is well understood and well accepted by the developers. The computation of the technical complexity factor seems to be straightforward. The eight weighted factors describe the technical complexity sufficient for our purposes.

The evaluation of scenarios turned out to be a critical step. Important failure factors were

- use cases are not always actualized (often written once and never revised),
- projects differ in the completeness of their use case descriptions and
- scenario modeling is not well understood.

3.2 Quality of Requirements Documents Findings

The analyzed requirements documents show that the use case method is now widely used throughout all development departments. We compared the analyzed requirements specifications with some important quality characteristics stated in the 'IEEE Recommended Practice for Software Requirements Specifications' [4]. We found that the understandability for most documents is satisfactory. Unfortunately, we found that for most documents the unambiguity, correctness and consistency is unsatisfactory and therefore the completeness could not be checked.

Another problem found concerns the degree of detail of requirements modeled: the analyzed documents differ very much in the degree of detail. The measured size can differ according to this degree.

Although the quality of requirements specifications was partially unsatisfactory, the software size measures of the analyzed projects have been sufficient. Nevertheless, requirements specifications have to be improved in order to

be a reliable source for future software size measurement.

Success and Failure Factors

The successful introduction of the use case method and its acceptance can be explained by the simplicity of the method and an 'easy to use' notation. That is why the understandability could be rated as satisfactory.

To explain the weaknesses in important quality characteristics, we found problems in the language/notation, method/process, tool support and integration of the resulting models. We point out some of the most important problems:

- It is difficult to seriously document big sized projects with the concepts given by the use case method due to the lack of defined abstraction mechanisms.
- Despite the high acceptance in practice, the syntax and semantics of the graphical notation to describe the use case model and the scenarios are incomplete, not precise and partially inconsistent.
- The use of free textual descriptions (for example, the descriptions of pre- and postconditions of a use case) makes it sometimes difficult to get unambiguous specifications which are comparable and reproducible.
- Due to the diversity of notations and the fragmentation of requirements information into several models there exist some integration problems. It is difficult, to detect inconsistencies and ambiguities in specifications.
- The process to elicit requirements, describe and review the different models is mainly based on some heuristics and weak rules. Most of the existing object-oriented analysis methods give little help to handle use case- and scenario-modeling appropriately.
- Some project leaders do not accept the importance of requirements specifications.
- Heuristics or rules describing the problem of the 'right degree of detail' are still missing.

4 LESSONS LEARNED, POTENTIAL IMPROVEMENTS AND THESES

4.1 Software Size Measurement

Lessons learned

- The most important lesson we learned is that requirements specification with use cases and scenarios can be used to measure the size of a software system. In our opinion, it is worth doing so. We are able to compare different software systems (and implementations) and we get a higher attention to the requirements specifications by the project leaders („.... it's difficult to reconstruct undocumented requirements...", [13]).
- Business objects are easy to evaluate, because the classification is very simple and the notation of business objects is formal enough.
- We also learned that the use case point method can be well supported by a simple low cost tool. Our home grown use case point tool is accepted and appreciated by the project leaders as it is convenient for their needs.
- Free textual use case descriptions are by no means sufficient to measure the software size.
- Six of the interviewed project leaders are suspicious of size measurement.
- Automatic size measurement is not possible at the moment due to the lack of formalization and supporting tools.

Improvements

We have to make sure that project members understand the benefit of use cases and formal scenario descriptions. Therefore a special use case modeling training and coaching for all project members has to be provided. We will continue to compare the productivity rate in order to make it a part of the daily business of a project leader. The publication of the analyzed project data in the monthly IT journal at SBI should be established to get more transparency within the IT development department.

4.2 Documentation of Requirements

Lessons learned

- It is difficult to adapt existing methods and notations to company-specific needs due to an immature underlying theoretical basis.
- The use of the use case method is accepted, but the language concepts for documentation are not well understood. As a consequence, it would be important to define the language concepts more precisely in advance; the language is the main building material of software documents.
- Management must clearly commit to the standards and enforce their application. Project members must realize that the documentation of requirements is not an optional issue and that it is worth to work seriously on it.

Developers should more frequently work together with their customers because this would lead to qualitatively better requirements documents; completeness should be a first class quality characteristic to users and developers because this is an absolute prerequisite for exact and reproducible size measures.

Improvements

Improvements have to be done in the problem areas mentioned above. First of all, the language concepts for modeling have to be improved. A real abstraction mechanism, more formality and fewer model types are the primary objectives of this task. The process has to be successively made more precise according to future experience. Additionally, organizational improvements have to be done as well. Among others, requirements models have to be updated regularly, the documentation and process standards have to be used in all SBI software projects.

4.3 Theses

The findings and interpretations of several real size projects presented in previous sections have lead us to the following theses.

Systematic measurement of software size and computation of productivity rates based on requirements specifications with use cases

1. increases the quality of software requirements specifications in the long term (principle of positive feedback),
2. creates competition within big software development departments,
3. enables management to objectively compare different software development technologies,
4. increases the discipline of the software developers and
5. increases the global competitiveness (time, cost, quality) of the organization.

We are convinced of the long-term profit of investing in good requirements documents and in size measurement based on such documents.

5 FUTURE WORK

We work on the improvement of the scenario modeling-language and -method. The developed concepts will be permanently validated in practice.

In order to get a broadly usable metric which overcomes the weaknesses stated in this paper, some more quantitative analysis has to be done. As experience with projects grows, we will be able to refine and improve the use case point method. It is an explicitly stated objective to apply the use case point method company-wide in order to continuously improve the overall development process of IT systems at SBI.

REFERENCES

1. Albrecht, A. J. (1979): Measuring Application Development Productivity. Proceedings of the IBM SHARE/GUIDE Applications Development Symposium, Monterey, CA; October 1979. (pp. 83-92)
2. Booch, Grady (1994): Object-Oriented Analysis and Design with Applications. The Benjamin/Cummings Publishing Company, Inc.; 2nd Edition 1994.
3. Cook, S., Daniels, J. (1994): Designing Object Systems: Object-Oriented Modelling with Syntropy. Prentice-Hall, Englewood Cliffs, NJ; 1994.
4. IEEE Recommended Practice for Software Requirements Specifications. IEEE Standard Number 830-1993, IEEE Standards Board; 1993/1994.
5. Jacobson, I., Christerson, M., Jonsson, P., Övergaard, G. (1994): Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, Menlo Park, CA; 4th Edition 1994.
6. Karner, G. (1993): Metrics for Objectory. Diploma thesis at the university of Linköping, Sweden, No. LiTH-IDA-Ex-9344; 21. December 1993.
7. Rubin, K. S., Goldberg, A. (1992): Object Behavior Analysis. Communications of the ACM, Vol 35, No. 9; Sept. 1992. (pp. 48-62)
8. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. (1991): Object-Oriented Modelling and Design. Prentice-Hall, Englewood Cliffs, NJ; 1991.
9. Rumbaugh, J. (1994): Getting started - Using use cases to capture requirements. Journal of Object-Oriented Programming (JOOP), Vol. 7, No. 5; September 1994. (pp. 8-23)
10. Selic, B., Gullekson, G., Ward, P. T. (1994): Real-Time Object-Oriented Modelling. Wiley & Sons; 1994.
11. Booch, G., Jacobson, I., Rumbaugh, J. (1997): The Unified Modeling Language for Object-Oriented Development, Documentation Set Version 1.0, Rational Software Coproration; January 1997.
12. Wirfs-Brock, R., Wilkerson, B., Wiener, L. (1993): Designing Object-Oriented Software. Prentice Hall International, Carl Hanser; 1993.
13. Hunger, Barbara (1998): Zusammenhang zwischen Produktivität und arbeitspsychologischen Variablen in Software-Entwicklungsteams. Diploma thesis at the university of Zurich, Switzerland; submitted for publication at the 'Abteilung für psychologische Methodenlehre'.
14. Baisch, E., Ebert, Ch. (1994): Produktivitätsbewertung im Software-Entwicklungsprozess. In Dumke, R., Zuse, H. (eds.), Theorie und Praxis der Softwaremessung, Deutscher Universitätsverlag, Wiesbaden; 1994. (pp. 1-19)