Assessment of Software System Evolvability

Bente Anda
Simula Research Laboratory and University of Oslo
P.O. Box 134, NO-1325 Lysaker
Norway
+47 67828306
bentea@simula.no

ABSTRACT
The evolvability, the ease of further development, of a software systems is difficult to assess, but may have large economic consequences. Many studies have investigated the relations between particular software metrics and effort on evolving individual classes, but little attention has been given to methods for assessing and measuring evolvability of complete software systems. This paper discusses such methods, and motivates that they should use a combination of structural code measures and expert assessments. This is exemplified in a case study assessing the evolvability of four functionally equivalent systems. The paper also gives with directions for future work on evolvability assessments.

1. INTRODUCTION
Evolvability is an important quality attribute of a software system as it indicates its future potential, including costs of ownership. The evolvability of a system may be particularly important for a software client in the process of acquiring the system, but a software client usually has few means of assessing evolvability. Such assessments are difficult as what constitutes an evolvable software system is not well established. To the author's knowledge there are very few studies on methods for assessing the evolvability of complete software systems, most research in the area of software evolution has focused on how to improve software evolvability [15]

Our interest in assessing the evolvability of complete software systems arose as a consequence of a project conducted by the Software Engineering Department at Simula Research Laboratory where four functionally equivalent systems were developed by four different software development companies. This presented us with the challenge of assessing the evolvability of the four systems as part of the process of deciding which system we would use and possibly evolve in the future.

There are many factors affecting evolvability, for example the technology used, the customer and development organizations and their relationship, as well as the software code. Methods for assessing evolvability must therefore be adapted to specific contexts [13]. In this paper the focus is on evolvable software code.

IWPSE'07, September 3-4, 2007, Dubrovnik, Croatia Copyright 2007
ACM ISBN 978-1-59593-722-3/07/09...$5.00

Assessing the evolvability of a software system is equivalent to making a prediction about future effort on evolving it based on information about the existing system. Studies in the field of software estimation have shown that a combination of expert assessments and formal methods usually provides the best results [11]. Consequently, the four case systems were assessed using structural code measures and expert assessments and these assessments were used to rank the systems according to assumed evolvability. The details of the assessment are described in [1]. The two assessment strategies resulted in different rankings of the systems according to overall evolvability, and they revealed a gap between the aspects of evolvability that may be identified based on structural code measures and those that require expert opinion. The results therefore support the claim that evolvability assessments should combine the use of structural assessments and expert opinion. The assessments also showed that there are many challenges with assessing evolvability of complete software systems

The remainder of this paper is organized as follows. Section 2 describes methods for assessing evolvability. Section 3 presents the evolvability assessments of the four case system. Section 4 concludes and suggests directions for future work.

2. EVOLVABILITY
Evolvability of software can only be indirectly measured, and such indirect measurement is only meaningful if an empirical connection between the directly and the indirectly measured characteristics is established. In order to establish meaningful evolvability measures for particular project contexts there should be a common understanding of the relations "equal evolvability as" and "better evolvability than". However, as long as our understanding of software evolvability is at an intuitive level and not explicitly formulated, we will not be able to establish a complete empirical connection between individual characteristics of the code and evolvability [12]. Therefore, when assessing evolvability, we typically have to choose between a vague definition of evolvability, and use for example expert assessments, and well-defined measures, which may not correspond to our intuitive understanding of evolvability, and use for example structural code measures.

The ISO/IEC 9126 standard measures maintainability, which for these purposes can be considered an equivalent concept to evolvability, by dividing it into a set of measures for effort needed to analyze, change and re-test the system [10]. The measures have

not been thoroughly empirically validated. We did not use this standard in our assessment because the measures require knowledge of the developers who are involved in evolving the systems, and of the changes that will be made to the system. Hence, the standard can not be used in an assessment only based on the software code.

In the related field of software usability, a method for measuring quality has been proposed and used that combines quantitative measures and expert opinion [18]. The result is one usability score, and the method makes it possible to measure and compare the usability of several systems. Methods combining quantitative and qualitative measures may, however, be problematic from a measurement perspective.

2.1 Structural Measures

A large amount of empirical research has been conducted in the area of measurement of structural properties of software, but this research has mostly focused on relations between such measures and evolvability of individual classes or clusters of classes. Briand and Wuest provide an overview of empirical work on structural measures and conclude that measures of size, coupling and cohesion of classes are generally correlated to their evolvability [4]. The set of structural measures by Chidamber & Kemerer, denoted CK-metrics, is probably the most used structural measures and have been empirically validated to be related to evolvability of individual classes [6,7]. One of the measures included in the CK-metrics, Depth of inheritance, has been investigated with respect to evolvability of complete systems [17]. The results of that study showed that the measure itself did not directly affect evolvability. The effects of specialisation classes has also been investigated in [5] where it was found that much use of specialisation classes may be negative [5]. The MOOD set of metrics is another set of metrics intended, but not validated, to provide an overall assessment of a software system [9].

2.2 Expert Assessments

The most commonly used strategies in practice for assessing evolvability are guided and unguided expert assessments [16]. One example of a guided strategy is The Air Force operation Test and Evaluation Center (AFOTEC) pamphlet which provides a very comprehensive set of instructions for evaluating software maintainability [2]. The instructions are, however, not specifically adapted to object-oriented software. Another guided strategy is to search the code for so called code smells. According to Fowler and Beck a defined set of code smells can indicate bad maintainability and a need for refactoring [8]. These code smells have not been validated regarding their impact on maintainability, and in an experiment experts judged the code differently with respect to presence of code smells [16].

A challenge with expert assessments is their reliability which varies in different studies; Schneiderman found little agreement in expert evaluation of code quality where experts had not developed the code [19], but Shepperd, found high reliability within development teams [20]. Another main challenge with relying on expert assessments is that such assessments are dependent on having people who are qualified to do such assessments and who are representative of, or understands the qualifications of, those who will perform maintenance on the products. Experts may also be biased in their opinions, for example by considering designs that they are not familiar with as problematic

There are few studies on the correspondence between measurement-based assessments and expert assessments or on how to combine measurement strategies. Mayrand and Coallier describe an approach for software product assessment used as part of an acquisition [14] which combines structural measures with expert assessments.

Results from the field of software estimation show that expert assessments mostly outperform the estimation methods based on attributes of the system to be developed. The reasons for this are suggested to be that the important variables influencing development effort are not well established, and that only limited empirical data is available to calibrate and validate the methods [11]. The situation is believed to be similar in the area of software evolution.

3. EVOLVABILITY ASSESSMENT

The state-of-the-art of evolvability of complete software systems led us to assess the four case systems, hereafter called A, B, C and D, using a set of structural code measures that had been validated to correlate with evolvability, and expert assessments. The assessments are described in more detail in [1].

3.1 Assessment using Structural Measures

Table 1 shows values for LOC (Lines of code) and NOC (Number of classes) for the four systems, as well as mean values and standard deviation for the metrics WMC (Weighted methods per class), OMMIC (Call to methods in unrelated class), and TCC (Tight class cohesion). Depth of inheritance is not included in as inheritance was relatively little used in the systems; C did not use inheritance, while the other systems mostly used only one level of inheritance The format in the table is mean value/std.

Table 1. Summary statistics for all systems

	A	B	C	D
LOC	7937	14549	7208	8293
NOC	63	162	24	96
WMC	6.9/11.2	7.8/10.3	11.4/12.5	4.9/4.5
OMMIC	7.7/15.8	5.3/11.8	8.6/25	4.7/14.1
TCC	0.26/0.37	0.17/0.31	0.20/0.23	0.11/10.22

The code of the four systems was measured using the complete set of CK-metrics as well as additional metrics. The measurement procedure and all the resulting values are described in [3].

The values in Table 1 show that the systems differ a lot with respect to size, number of classes and how functionality is distributed over the classes, as can be seen from the large differences in the values for WMC. Furthermore, the values for OMMIC and TCC show that the developers of the systems have chosen to implement different trade-offs with respect to focus on good coupling values or good cohesion values. There are very few empirical studies on the effects of these trade-offs in object-oriented design, and therefore it is difficult to combine the values of Table 1 into one overall measure of evolvability for each

system classes. If we assume that the system with the best class-level measures is also the most evolvable we get the following ranking:

- D is assessed as easiest to evolve due to low values and small standard deviation for size of classes and coupling. However, the system is the second largest, and has a relatively high number of classes as well as a low cohesion value.
- A and B are assessed as approximately equally evolvable. The WMC measure does not separate the systems, as A has slightly smaller classes than B, but also has a larger standard deviation indicating a less even design. B has better coupling values, while A has classes with higher cohesion. B may be easier to evolve than A due to lower coupling values, but on the other hand system B is a much larger system.
- C is assessed as the most difficult to evolve due to high values and large standard deviations for size of classes and coupling of classes, something that indicates large and complex classes and an uneven design. The cohesion value is high, but this is probably much due to the size of the classes. The fact that it is the smallest system may, however, mean that it is easier to understand than the other systems.

3.2 Assessments by Experts

The expert assessments were conducted individually by two very experienced Java consultants. They assessed the code from the perspective of experienced java-programmers, who are not in detail familiar with the system. Due to few previous studies on experts assessment of Java software and few empirical studies on the effects of object-oriented design principles on evolvability, it was decided to let the experts chose their own evaluation criteria based on their experience. Although the two experts did not communicate in any way, their criteria and conclusions were very similar. Due to the simplicity of the four systems, the experts were also asked to attempt to see the consequences of design decisions in a larger and more long term perspective.

The characteristics of the code that they considered important with respect to evolvability, and the assessment of the four systems according to these characteristics are shown in Table 2. For each of the characteristics the experts commented on whether it was handled satisfactorily in the system. In Table 2 a satisfactory solution is indicated by 1 and an unsatisfactory solution by 0.

Table 2 show that the experts assessed the evolvability of both A and D to be good. A was assessed as better than D and is consequently ranked in first place. The experts commented, however, that the developers of D had implemented a larger system much because they presumably had started the project with high ambitions for making a system that would be easy to extend and evolve. The ambitions had not been fulfilled, resulting in a low score on some of the characteristics, but D is still likely to be more evolvable than A if the system is to undergo vary large changes.

Table 2 also showed that the experts assessed the evolvability of both B and C as low and much worse than that of systems A and D. B was assessed as better than C and is consequently ranked in third place. Again the experts commented that evolvability is likely to depend on the types of changes that will be required to the system.

Table 2. Experts' characteristics and assessments

	A	B	C	D
Choice of classes	1	0	0	1
Design adapted to system	1	0	0	1
Three-layer architecture	1	0	0	1
Good use of components	1	0.5	0	1
Encapsulation	1	0	0	1
Inheritance	1	0.5	0	1
Good use of class libraries	1	0	1	0
Simplicity	1	0	0	0
Naming	1	0	0	0
Comments	1	1	0	1
Appropriate technical platform	1	0	0	1

The design of system B was too complex and comprehensive for this system, and it may have been more appropriate on a larger product. The developers of C had not emphasized a good design, but since the system is small, it may be easy to perform small change tasks on the system, but larger extensions are not realistic.

Most of the characteristics considered important for the evolvability of the systems by the experts can not easily be measured automatically, especially since many of them include trade-offs that must be made depending on, in particular, the complexity of the system and the competencies of the developers.

4. CONCLUSION AND FUTURE WORK

The ideal method for assessing the evolvability of a software system would provide as result a score of evolvability on at least an ordinal scale ranging for example from very low evolvability to very high evolvability. The ultimate goal, although probably not a very realistic one, is a method which also gives an indication of the economic consequences of the evolvability score.

This paper has given an overview of existing methods for evolvability assessments and has presented an empirical study on the assessment of four functionally equivalent systems. The results support claims that evolvability assessments should combine structural measures of the code with expert opinion, but also that many questions regarding how to assess the evolvability of software systems remain to be investigated. In particular, more empirical studies are needed to investigate

- How different characteristics of Java code impact the effort on evolving the system, including to what extent relations between structural code measures and evolvability for individual classes scale to the system level.
- Which characteristics of software systems are considered important for evolvability by experienced software developers, including to what extent there is agreement among experienced developers on the characteristics and on how they should be assessed for different systems.
- Which qualifications are required to conduct expert assessments

- How can structural measures and expert assessments best be combined.

The evolvability of a software system depends on much more than the software code. Consequently, a method for assessing evolvability will only be applicable to a specific type of systems. Hence, the results of the empirical studies suggested above must all be described with details about for what type of system in terms of programming language and development tools used, organizational context and developer qualifications etc. they are valid. The results of the study described in this paper are expected to be applicable to relatively Java systems and experienced Java developers who are not familiar with the code.

5. ACKNOWLEDGMENTS

I thank Hans Christian Benestad and Erik Arisholm for assessing the code using structural code measures. Per Einar Arnstad and Sindre Mehus are acknowledged for their expert assessments. Dag Sjøberg is acknowledged for obtaining funding for, and for organizing, the project in which the four software systems were developed.

6. REFERENCES

[1] Anda, B. Assessing Software System Maintainability using Structural Measures and Expert Assesments. Accepted for publication at the 23rd International Conference on Software Maintenance, 2007.

[2] AFOTEC Software maintainability evaluation guide. Department of the Air Force, HQ Air Force Operational Test and Evaluation Center, 1996.

[3] Benestad, H.C., Anda, B. and Arisholm, E. Assessing Software Product Maintainability Based on Class-Level Structural Measures. In Proceedings of the 7th International Conference on Product-focused Software Process Improvement (PROFES), edited by Jürgen Münch. Springer-Verlag, pp. 94-111, 2006.

[4] Briand, L. and Wuest, J. Empirical Studies of Quality Models in Object-Oriented Systems, *Advances in Computers*, Vol. 56, pp. 97-166, 2002.

[5] Briand et al. An Experimental Comparison of the Maintainability of Object-Oriented and Structured Design Documents. *Empirical Software Engineering*, 2(3):291-312, 1997.

[6] Chidamber, S.R. and Kemerer, C.F. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, 20(6): 476-493, 1994.

[7] Darcy, D. and Kemerer, C.F. OO Metrics in Practice. *IEEE Software*, 22(6): 17-19, 2005.

[8] Fowler, M. and Beck, K. Bad smells in code. In: *Refactoring: Improving the design of existing code*, 1st ed., Addison-Wesley, Boston, pp.75-88, 2000.

[9] Harrison, R., Counsell, S.J., and Nithi, R.V. An Evaluation of the MOOD Set of Object-Oriented Software Metrics. *IEEE Transactions on Software Engineering*, 24(6): 491-496, 1998.

[10] ISO/IEC. ISO/IEC 9126 Software engineering – Product quality, 2001.

[11] Jørgensen, M. Estimation of Software Development Work Effort: Evidence on Expert Judgement and Formal Models. Accepted for publication in *the International Journal of Forecasting*, 2007.

[12] Jørgensen, M. Software quality measurement. *Advances in Engineering Software*, 30(12):907-912, 1999.

[13] Kitchenham, B.A. et al. Towards an Ontology of Software Maintenance. *Journal of Software Maintenance: Research and Practice*. Vol. 11, pp. 365-389, 1999.

[14] Mayrand, J. and Coallier, F. System Acquisition Based on Software Product Assessment. In Proceedings of the 18th International Conference on Software Engineering (ICSE'96), pp.210-219, 1996.

[15] Mens, T., Wermelinger, M., Ducasse, S., Demeyer, S., Hirschfeld, R. and Jazayeru, M. Challenges in Software Evolution. In Proceedings of the Eighth International Workshop on Principles of Software Evolution (IWPSE'05), pp. 13-22, 2005.

[16] Mäntylä, M.V. and Lassenius, C. Subjective evaluation of software evolvability using code smells: An empirical study. *Empirical Software Engineering*, 11(3):395-431, 2006.

[17] Prechelt, L., Unger, B., Philippsen, M. and Tichy, W. A controlled experiment on inheritance depth as a cost factor for software maintenance. *The Journal of Systems and Software*, Vol. 65, pp. 115-126, 2003.

[18] Sauro, J. and Kindlund, E. A method to standardize usability metrics into a single score. In Proceedings of the SIGCHI conference on Human factors in computing systems. ACM Press, pp. 401-409, 2005.

[19] Schneiderman, B. *Software psychology: human factors in computer and information systems*. Winthrop, Cambridge, Massachusetts, 1980.

[20] Shepperd, M.J. System architecture metrics for controlling software maintainability. In Proceedings of the IEE Colloqium on Software Metrics, April 1-3, 1990.