Measurements of Software Maintainability

Rikard Land
Mälardalen University
Department of Computer Engineering
Box 883
SE-721 23 Västerås, Sweden
+46 (0)21 10 70 35
rikard.land@mdh.se
http://www.idt.mdh.se/~rld

ABSTRACT

In this position paper, we describe the research we have just initiated. We will investigate how the "maintainability" of a piece of software changes as time passes and it is being maintained by performing measurements on industrial systems. We present the notion of "maintainability", our hypotheses, and our approach.

Keywords

Software metrics, Halstead measure, maintainability, modifiability, software architecture, software deterioration.

1. INTRODUCTION

Many resources are spent on software maintenance. Thus, producing software that is easy to maintain may potentially save large costs. The problem of maintaining software is widely acknowledged in industry, and much has been written on how maintainability can be facilitated by e.g. tools and processes (see e.g. the IEEE International Conference on Software Maintenance, ICSM). However, you cannot control what you cannot measure, and there is yet no universal measure of maintainability. Some proposals have indeed been presented, but the very idea of measuring maintainability has inherent problems (these issues are discussed in section 2).

We can in research and practice discern two ways of discussing the term maintainability[1]: either it is used very informally, or it is considered possible to derive a measure directly from source code. To be fair, those adopting the second view admit that any formula describing maintainability as a function of e.g. "relative number of commented lines of source code" is of limited use, and those having the first view have a feeling that maintainability has something to do with program size and complexity.

[1] Although "maintainability" and "modifiability" are similar but by some not considered equivalent terms, will use the term maintainability exclusively throughout the paper.

We have identified two areas where there is little research done. First, it is well known that software systems deteriorate as time passes and changes are made to it. We intend to describe this in terms of how *maintainability changes as a system is being maintained* (see section 3.1), rather than verifying a measure using expert judgment as is usually done. Second, we only know of measurements on code level, and will thus perform *measurements on the architectural level* and compare measurements made on both levels (see section 3.2).

We are therefore about to start performing measures on the history of a number of industrial systems to see how maintainability has changed as changes are implemented. We hope to be able to identify "bad" and "good" types of changes, and learn from that how a system should be maintained.

2. WHAT IS MAINTAINABILITY?

Maintainability has previously been described mainly in two ways, either informally or as a function of directly measurable attributes.

2.1 Informal Descriptions

There are many text descriptions available, which are in essence very similar. We quote the IEEE Standard Glossary of Software Engineering Terminology:

> **maintainability.** […] The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment. [15]

There are other examples of such descriptions [3,4,36]. Such descriptions capture what is intuitively meant with maintainability, but have some problems. They do not in any way guide in how to estimate or measure maintainability. Another problem is that if we follow this approach and try to measure maintainability as "effort", we should bear in mind that the common unit for effort, "man-month", is in itself very dubious [8].

We can also note that Pfleeger describes maintainability as "the *probability* that […] a maintenance activity can be

carried out within a stated time interval […] [it] ranges from 0 to 1" ([26], italics added).

2.2 Maintainability Measures

Despite the subjectivity of any attempt to measure maintainability, great effort has been put into constructing formulas for describing maintainability. Following the opinion that maintainability "is the set of attributes that bear on the effort needed to make specified modifications" [16], we describe maintainability according to this approach as a function of directly measurable attributes A_1 through A_n, that is:

$$M = f(A_1, A_2, \ldots, A_n) \quad \textbf{(1)}$$

On an informal level, this approach is quite appealing – it is intuitive that a maintainable system must be e.g. consistent and simple. However, there may be great difficulties in measuring those attributes and weighting them against each other and combine them in a function *f*. Any such attempt is therefore bound to a quite limited context – a particular programming language, organization, type of system, type of project; the skill and knowledge of the people involved must also be considered then drawing conclusions. Many researchers have tried to quantify maintainability in different types of measures [1,2,10,23,24,36], of which the most noticeable probably is the Maintainability Index, MI [24,32]. The Halstead source code measures proposed in the seventies [13,31] have been used for describing maintainability [31,32] (see section 3.4).

Typically, maintainability measures are validated using expert judgments about the *state* of different systems and modules [10,36], while we rather investigate how the measures *change* as the software is maintained. However, the value of performing such an evaluation before and after a change is implemented has already been discussed [2,10,27] (although the objective of such studies has been to verify cost predictions); we adopt this approach but pursue it even longer by investigating a complete evolution history (very long at least) of industrial systems, measuring such functions after each change.

2.3 Maintainability and Software Architecture

Our research interests includes software architecture [3,6,14,33] and component-based systems [35], in connection with "change". Within the software architecture literature, the terms "maintainability" and "modifiability" are often used informally as a desired feature [3,6,14,33] – indeed, it is one of the very goals with software architecture to make a system understandable, and thus maintainable, by providing abstractions on an appropriate level.

One important goal for research is to make it possible to accurately estimate maintenance costs; such estimations should be done early in the development, i.e. during the architectural design. Such prediction models are often based both on the argument that maintainability must be discussed in the context of particular changes – it might be easy to perform one particular change, while another is virtually impossible. The use of scenarios to evaluate maintainability has therefore been discussed, particularly on the architectural level [3-7,9,17-20]. SAAM [3,19] and ATAM [20] are general scenario-based evaluation techniques with which any quality attributes can be estimated on the architectural level; these have been reported useful in practice [3,18,21]. Bengtsson has suggested one cost estimation model where the type of change (new components, modified components, or new "plug-ins") of each change scenario is taken into account to calculate the estimated change effort [4].

3. OUR RESEARCH

We do not propose a new formula for maintainability, but will rather measure attributes A_i that have been suggested in literature to affect maintainability. We are not interested in comparing systems (i.e. answering whether system A with $M = 82$ is more maintainable than system B with $M = 81$) but rather discuss around how the maintainability of a single system change, thus connecting the notion of maintainability to the recognized problem of software aging and deterioration [25] (see section 3.1). The idea of comparing a measure before and after a change is made has been discussed [2,10]; however, we have not seen a study like the one we are describing in this paper, investigating changes over a long sequence of changes.

We will perform measurements on the architectural level as well (this is described in section 3.2).

3.1 Software Deterioration

It has been noticed that software deteriorates as it ages and is being maintained [3,6,25,33,34]; using the vocabulary of Fred Brooks, a system's *conceptual integrity* [8] degrades as changes are made to it. Our hypothesis is that this is discernible when measuring maintainability of a system during a long sequence of changes. If it is true that the system decays continuously, we should get a graph similar to the one in Figure 1.

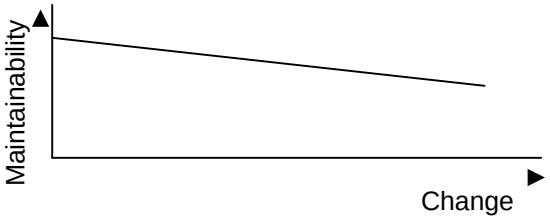

Figure 1. The system deteriorates (measured in maintainability) as changes are implemented[2].

[2] The "change" dimension could be thought of as "discrete time", with which we mean that a number of changes are implemented sequentially to a system.

We will perform measurements on a long sequence of versions of the same system, and investigate if it is possible to discern any trends. However, we do not expect to find a graph that is so easy to interpret as Figure 1; rather we expect to find that the maintainability sometimes increase and sometimes decrease. We hope to be able to correlate increases and decreases in maintainability measures to the descriptions of the changes made (as described by change requests). For example, the logical change "system restructuring" should cause the maintainability measure to increase, as Figure 2 describes.

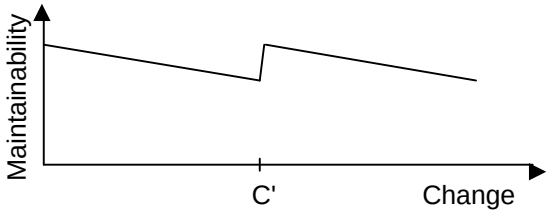

Figure 2. Change C' (a restructuring of the system) causes maintainability to (temporarily) increase.

Since there is no universally accepted definition of how to measure maintainability, we will measure a number of different measures that has been proposed as affecting maintainability. After we have collected data throughout a long history of at least two systems, we will focus on several related questions:

- Can the changes in maintainability be correlated to descriptions of the logical changes done? Are there any specific types of changes that cause the measure to increase or decrease significantly? Are there differences between "maintenance" changes and "modifications"?
- Do different measures display the same trend for the same change? It is quite possible that we cannot find any always-valid correlation – but is it possible to discern any types of changes that make all (or most) measures to show the same trend? For example, do "fault corrections" make two measures decrease or increase simultaneously? When do different measures show the opposite trends?

3.2 Architecture-Level Maintainability Measures

In the field of Component-Based software engineering, a future is pictured where more and more software is built from *components*, meaning binary executables (EXEs, DLLs, etc.) possibly developed out-of-house [35] (this trend is already discernible – we daresay that the absolute majority of systems developed today use existing components such as operating systems, databases, and graphical packages). Therefore, a big question is how to make component-based systems maintainable, and as part of that we must be prepared to measure and estimate maintainability on the architectural level when source code is no longer available. We will thus compare architectural and code-level measures to see if there is any correlation between these.

3.3 Which Systems?

The properties we wish the systems under investigation to have are listed below.

- The system should be much used and much maintained so that there is a long sequence of changes, which means much data.
- To make it possible to investigate the changes made to a system, it should have been developed using some sort of revision control system, from which any earlier version of the system can be retrieved. At least, it should be possible to retrieve all released versions, but this does not allow us to investigate individual changes.
- There should also be a way of tracking how physical changes correspond to logical changes; i.e. which lines in which files the correction of a certain bug affected.

However, a mature use of a revision control system and change requests implies that the developing organization is relatively mature; therefore, we can only expect our results to apply to other development projects with similar maturity.

3.4 Measures

This section lists the candidate measures we will use, and measures we have decided not to use. We have, for convenience, focused on attributes directly measurable from program code (we have therefore neither included measures including subjective ranking techniques, nor measures that includes documentation [1])

There is an abundance of proposed measures of program complexity and maintainability. We have not yet analyzed fully which measures will make sense, but have in literature identified the following candidate measures on "lexical level": Lines Of Code (LOC)[3], number of commented lines, Halstead Length, Halstead Volume, Halstead Effort, purity ratio, number of executable semicolons (the same as executable statements [22]), average variable span, average number of statements between two successive references to the same variable, number of blank lines, number of tokens, number of lines of data declarations, control structure nesting level, average number of commented lines per module, average number of LOC per module [13,22,36]; cyclomatic complexity [29]; readability of source code

[3] There are a number of ways to count LOC – with or without comments, blank lines, compiler directives etc. [11,28]; however, since we are mainly interested in the changes, we can expect the choice of definition of LOC to be of minor importance.

(defined as the ratio between lines of code and number of commented lines) [1]; number of knots [22]. We will also investigate the maintainability measures taxonomy by Oman et al where 92 measures are listed and classified [24].

In addition, we have some ideas of the measurements we will perform on the architectural level, but have yet to perform a thorough literature search. This could include counting dependencies between components; “fan-in” and “fan-out” [12]; number of calls in, number of calls out [22]; and other measures.

There are other “complexity measures”, which we will not use: neither the Function Point measure of software complexity [11,30], the Object Point measure included in the COCOMO 2 method [11], nor DeMarco’s specification weight metrics (“bang metrics”) [11], are directly measurable from source code. Each of these requires a manual moment since not all parameters are measurable from source code. One example it the rating of items as “simple”, “average”, or “complex”. It is of course highly impractical to include manual work to evaluate a large number of subsequent versions, and there is a high risk of mistakes in counting or unfairness in rating. Fenton and Pfleeger list 11 limitations with the Function Point measure [11] (although all of these need not be disadvantages in our case). Also, these measures were rather designed for cost estimations (before source code is available) than of performing measurements.

4. SUMMARY

We will, through measurements, investigate the maintainability of at least two industrial systems. Although there are proposals on how to measure “maintainability” on a given piece of software, we are mainly interested how such measures have changed over time as the software is being maintained. We hope to be able to discern patterns in why some changes make the maintainability decrease and others make it increase. We will also compare measurements on the lexical level and on the architectural level.

We will use this description in our future work on software architecture and components, and hope to be able to describe how to design a system to make it maintainable.

REFERENCES

[1] Aggarwal K. K., Singh Y., and Chhabra J. K., "An Integrated Measure of Software Maintainability", In *Proceedings of Annual Reliability and Maintainability Symposium*, IEEE, 2002.

[2] Ash D., Alderete J., Yao L., Oman P. W., and Lowther B., "Using software maintainability models to track code health", In *Proceedings of International Conference on Software Maintenance*, IEEE, 1994.

[3] Bass L., Clements P., and Kazman R., *Software Architecture in Practice*, Addison-Wesley, 1998.

[4] Bengtsson P., "Architecture-Level Modifiability Analysis", Ph.D. Thesis, Blekinge Institute of Technology, Sweden, 2002

[5] Bengtsson P. and Bosch J., "Architecture Level Prediction of Software Maintenance", In *Proceedings of 3rd European Conference on Software Maintenance and Reengineering (CSMR'99)*, IEEE Computer Society, 1999.

[6] Bosch J., *Design & Use of Software Architectures*, Addison-Wesley, 2000.

[7] Bosch, J. and Bengtsson, P., An Experiment on Creating Scenario Profiles for Software Change, report ISSN 1103-1581, Department of Software Engineering and Computer Science, University of Karlskrona/Ronneby, 1999.

[8] Brooks F. P., *The Mythical Man-Month - Essays On Software Engineering, 20th Anniversary Edition*, Addison-Wesley Longman, 1995.

[9] Clements P., Kazman R., and Klein M., *Evaluating Software Architectures: Methods and Case Studies*, Addison Wesley, 2000.

[10] Coleman D., Ash D., Lowther B., and Oman P., Using Metrics to Evaluate Software System Maintainability, *IEEE Computer*, volume 27, issue 8, 1994.

[11] Fenton N. E. and Pfleeger S. L., *Software Metrics - A Rigorous & Practical Approach*, PWS Publishing Company, 1997.

[12] Grady R.B., Successfully Applying Software Metrics, *IEEE Computer*, volume 27, issue 9, 1994.

[13] Halstead M. H., *Elements of Software Science, Operating, and Programming Systems Series Volume 7*, Elsevier, 1977.

[14] Hofmeister C., Nord R., and Soni D., *Applied Software Architecture*, Addison-Wesley, 2000.

[15] IEEE, IEEE Standard Glossary of Software Engineering Terminology, report IEEE Std 610.12-1990, IEEE, 1990.

[16] ISO/IEC, Information technology - Software product quality - Part 1: Quality model, report ISO/IEC FDIS 9126-1:2000 (E), ISO, 2000.

[17] Kazman R., Abowd G., Bass L., and Clements P., Scenario-Based Analysis of Software Architecture, *IEEE Software*, volume 13, issue 6, 1996.

[18] Kazman R., Barbacci M., Klein M., and Carriere J., "Experience with Performing Architecture Tradeoff Analysis Method", In *Proceedings of The International Conference on Software Engineering, New York*, 1999.

[19] Kazman R., Bass L., Abowd G., and Webb M., "SAAM: A Method for Analyzing the Properties of Software Architectures", In *Proceedings of The 16th International Conference on Software Engineering*, 1994.

[20] Kazman R., Klein M., Barbacci M., Longstaff T., Lipson H., and Carriere J., "The Architecture Tradeoff Analysis Method", In *Proceedings of The Fourth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS), (Monterey, CA)*, 1998.

[21] Land R., "Improving Quality Attributes of a Complex System Through Architectural Analysis - A Case Study", In *Proceedings of 9th IEEE Conference on Engineering of Computer-Based Systems*, IEEE, 2002.

[22] Lanning D.L. and Khoshgoftaar T. M., Modeling the Relationship Between Source Code Complexity and Maintenance Difficulty, *IEEE Computer*, volume 27, issue 9, 1994.

[23] Oman P. and Hagemeister J., "Metrics for Assessing a Software System's Maintainability", In *Proceedings of Conference on Software Maintenance*, IEEE, 1992.

[24] Oman, P., Hagemeister, J., and Ash, D., A Definition and Taxonomy for Software Maintainability, report SETL Report 91-08-TR, University of Idaho, 1991.

[25] Parnas D. L., "Software Aging", In *Proceedings of The 16th International Conference on Software Engineering*, IEEE Press, 1994.

[26] Pfleeger S. L., *Software Engineering, Theory and Practice*, Prentice-Hall, Inc., 1998.

[27] Ramil J. F. and Lehman M. M., "Metrics of Software Evolution as Effort Predictors - A Case Study", In *Proceedings of International Conference on Software Maintenance*, IEEE, 2000.

[28] Rozum, J. A. and Florac, W. A., A DoD Software Measurement Pilot: Applying the SEI Core Measures, report CMU/SEI-94-TR-016, Software Engineering Institute, 1995.

[29] SEI Software Technology Review, *Cyclomatic Complexity*, URL: http://www.sei.cmu.edu/, 2002

[30] SEI Software Technology Review, *Function Point Analysis*, URL: http://www.sei.cmu.edu/, 2002

[31] SEI Software Technology Review, *Halstead Complexity Measures*, URL: http://www.sei.cmu.edu/, 2002

[32] SEI Software Technology Review, *Maintainability Index Technique for Measuring Program Maintainability*, URL: http://www.sei.cmu.edu/, 2002

[33] Shaw M. and Garlan D., *Software Architecture: Perspectives on an Emerging Discipline*, Prentice-Hall, 1996.

[34] Sommerville I., *Software Engineering*, Addison-Wesley, 2001.

[35] Szyperski C., *Component Software - Beyond Object-Oriented Programming*, Addison-Wesley, 1998.

[36] Zhuo F., Lowther B., Oman P., and Hagemeister J., "Constructing and testing software maintainability assessment models", In *Proceedings of First International Software Metrics Symposium*, IEEE, 1993.