Whitepaper Draft
Evaluating Maintainability in Software Architecture

Gautam Kumar
Instructor: Prof. Mark Kochanski

February 7, 2016

Contents

1 Software Maintenance and Maintainability

Software maintenance as defined by the International Organization for Standardization is the process of modifying a software product after delivery to correct faults or to improve any quality attributes [1]

Maintainability is a quality attribute that describes the ease of introducing modifications into the software systems. Maintainability has four sub-characteristics namely analysability, changeability, stability and testability[1].

1.1 Types of Maintenance

ISO defines four types of software maintenance, Corrective, Preventive, Adaptive and Perfective.

Corrective maintenance refers to modifications done fix actual errors in the product while preventive maintenance is performed to introduce changes that can potentially prevent problems during the normal operation of the software product.

Adaptive and perfective maintenance are modifications introduced to enhance the core functionality of the software by improving one or more of its quality attributes.

2 Quantifying Maintainability

Understanding maintainability as a quality attribute is useful in learning the benefits of its presence in a product but to derive value from the process of evaluating software maintainability we require a method of measuring maintainability as a metric.

2.1 UML based metrics

One way to measure maintainability is to analyse the class diagram. This is especially useful in projects which use an object oriented programming language and have an existing UML diagram describing the software structure.

Genero et al [3] [2] describe an elegant way of predicting the maintainability of software early in the design phase by measuring the size and structural complexity of the software project.

UML based metrics cannot independently describe the maintainability of a system because they suffer from the same basic flaw that UML adoption faces. The fact that UML isn't as widely adopted in the software industry is the primary limiting factor. Though UML

is extremely well defined and documented its complexity and the learning curve needed to effectively adopt UML in an organisation setting serves as a serious impediment to its adoption which in-turn reduces the value of using an UML based metric in measuring the maintainability of a software architecture.

2.2 Coupling as a metric

Coupling is the degree of interdependence between software modules [1]. Low coupling is often associated with high readability and maintainability because data and control flow often tends to be well structured in systems with low coupling.

Lindvall et al define[6] metrics to evaluate the maintainability of a software architecture based on the degree of coupling between modules in the software system. The metrics defined by the authors are

1. CBM coupling-between-modules
2. CBMC coupling-between-module-classes
3. CIM coupling inside a module

"CBM" is defined as the number of non-directional, distinct, inter-module references. Similarly "CBMC" is the number of non-directional, distinct, inter-module, class-to-class references for a module. Figure 1 exhibits an example of how "CBM" can calculated from an architecture diagram.

The goal when constructing a software architecture is to reduce Inter module dependencies (CBM and CBMC) at the expense of increasing intra module dependencies. Conversely when comparing architectures the one with lower CBM and CBMC scores are better.

2.3 Related quality attributes

Maintainability has a positive relationship with availability, flexibility, reliability, testability and a negative relationship with efficiency [4]. This means that an increase in flexibility, for example, could potentially improve the maintainability of the software system. An increase in efficiency could potentially hamper maintainability.

Understanding these trade off is important because indirect measurements as a factor of one of the aforementioned quality attributes can play a critical role when direct measurements of maintainability aren't available.

3 Other techniques for evaluating Maintainability

- Maintaining Maintainability [8]
  In this paper the authors attempt to quantify the effects of people, organisation and

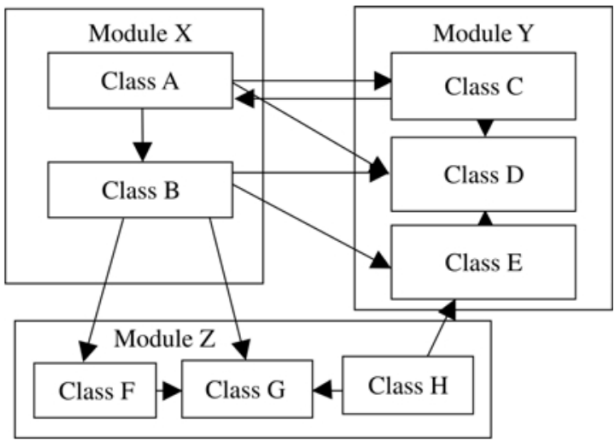

Figure 1: An example of architectural design. Arrows indicate coupling between the classes. Source: [6]

the maintenance process as a metric to evaluate how these external factors affect maintainability in a software system.

- Software Architecture Evaluation Methods for Performance, Maintainability, Testability, and Portability [7]
  This paper is a meta analysis of various methods of software architecture evaluation for to quantify quality attributes such as Performance, Testability, Maintainability and portability.

- SAAM [5]

4 Conclusion

Quality attributes such as performance and maintainability are usually not clearly defined, if at all, in the requirements specification because stakeholders do not generally understand their necessity or value. It is our job as software architects to evaluate and include maintainability as part of the architecture because maintainability offers one of the highest value in the longer run of any project. In this white paper we've highlighted some techniques of evaluating the maintainability of a software architecture before implementation and thus benchmarking the expectations for the production system.

Biblography

[1] , I. International Standard - ISO/IEC 14764 IEEE Std 14764-2006 Software Engineering #2013; Software Life Cycle Processes #2013; Maintenance. *ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998)* (2006), 0_1–46.

[2] Genero, M., Manso, E., Visaggio, A., Canfora, G., and Piattini, M. Building measure-based prediction models for UML class diagram maintainability. *Empir Software Eng 12*, 5 (Mar. 2007), 517–549.

[3] Genero, M., Piattini, M., Manso, E., and Cantone, G. Building UML class diagram maintainability prediction models based on early metrics. In *Software Metrics Symposium, 2003. Proceedings. Ninth International* (Sept. 2003), pp. 263–275.

[4] Karl, W. Software requirements. *Edition-2, Microsoft* (2003).

[5] Kazman, R., Bass, L., Webb, M., and Abowd, G. SAAM: A Method for Analyzing the Properties of Software Architectures. In *Proceedings of the 16th International Conference on Software Engineering* (Los Alamitos, CA, USA, 1994), ICSE '94, IEEE Computer Society Press, pp. 81–90.

[6] Lindvall, M., Tvedt, R. T., and Costa, P. An Empirically-Based Process for Software Architecture Evaluation. *Empirical Software Engineering 8*, 1 (Mar. 2003), 83–108.

[7] Mattsson, M., Grahn, H. a., and M\a artensson, F. Software architecture evaluation methods for performance, maintainability, testability, and portability. In *Second International Conference on the Quality of Software Architectures* (2006), Citeseer.

[8] Ramage, M., and Bennett, K. Maintaining maintainability. In *, International Conference on Software Maintenance, 1998. Proceedings* (Nov. 1998), pp. 275–281.