Bachelor's Programme in Science and Technology

Software metrics in long-term projects

Utilization and deployment of software metrics in long-lasting software projects

---

Santeri Suitiala

BACHELOR'S THESIS

Aalto University
BACHELOR'S THESIS 2018

Software metrics in long-life projects

Utilization and deployment of software metrics in long-lasting software project

Santeri Suitiala

Thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Technology.
Otaniemi, 8 Oct 2018

Supervisor (ABB):	Jorma Keronen
Supervisor (Aalto):	Marko Hinkkanen
Advisor:	Markus Turunen

Aalto University
School of Electrical Engineering
Bachelor's Programme in Science and Technology

Aalto University

Abstract

Aalto University, P.O. Box 11000, FI-00076 Aalto www.aalto.fi

Author	Santeri Suitiala
Title	Software metrics in long-term projects
School	School of Electrical Engineering
Degree programme	Bachelor's Programme in Science and Technology
Major	Electronics and Electrical Engineering — **Code** ????
Supervisor (ABB)	Jorma Keronen
Supervisor (Aalto)	Marko Hinkkanen
Advisor	Markus Turunen
Level	Bachelor's thesis — **Date** 8 Oct 2018 — **Pages** 7 — **Language** English

Abstract

Lorem ipsum.

Keywords software metrics, software quality

urn https://aaltodoc.aalto.fi

Tiivistelmä

Lorem ipsum.

Avainsanat ohjelmistometriikat, ohjelmiston laatu

urn https://aaltodoc.aalto.fi

Contents

1. Introduction

“The notion of ‘software engineering’ was first proposed in 1968” (Sommerville, 2011). Ever since the profession started, software projects have become bigger and more complex. This is because the hardware has been growing even faster and so software engineers have been having a hard time keeping the increasing phase up (Brooks & Kugler, 1987). The increasing code complexity raises new problems with understanding existing code and increases probability of software flaws. Software complexity can be later decreased by refactoring code. Refactoring can mean e.g. removing duplicate code by abstracting, renaming and commenting the code to make it more readable.

Software metric is a quantitative value calculated from a piece of a code or even a whole software project. Software metrics are used to track software quality to determine whether the software has improved or not. Knowing the change of the software quality over time makes it possible to plan and allocate resources to fix software with poor quality.

ABB Drives manufactures industrial drives which are controlled by software. Industrial drives are made to last for a long period of time and the same goes for the software inside the machine. Also new bugs are found and customers demand new features which makes the software evolve rapidly over time. The same effect that Brooks discovered over 30 years ago is something that still exists in companies: software size and complexity increases over time. Without refactoring, the software may become somewhat useless and understandable for the developers. If too much time is spent refactoring and improving the code, no new feature gets developed and customers are left unsatisfied. In a big company, it is difficult to perceive the whole picture and so the happy medium of internal improving and development can be hard to find.

First the reader will get familiarized with the underlying theory of software metrics. Furthermore a part of the task is also to determine the core metrics by getting familiar with the most commonly used metrics and make conclusions. Later thesis tries to solve a small part of this problem and guide the development teams to the right direction by introducing a systematic software analysis. Analysis calculates different core software metrics and ideally represents a long-term graph to determine whether the quality of the software is increasing or decreasing and how rapid is the change.

The ultimate goal of this thesis is to help the company's software development teams to decide when to refactor code and improve tools and dependencies and how much time should be used for it.

2. Theory behind software metrics

Software metric is a quantitative value calculated from a piece of a code or even a whole software project. Software metrics are used to track software quality from many different point of views. The magnitude of the metric does not matter. We are more interested about the change of the metric over a defined amount of time. One of the main reasons to implement software measurements for projects is to replace reviews with metrics to define software quality (Sommerville, 2011). Tracking the quality of software makes it easier and more efficient to see whether software has improved or not. Knowing the direction of the software quality makes it possible to plan and give resources to fix software with poor quality. A simple example of a software metric is source lines of code (SLOC) which simply tells the number of lines of code the whole program has altogether (Nguyen, Deeds-Rubin, Tan, & Boehm, 2007).

2.1 Classification of software metrics

Software metric can be any metric that determines software quality in any way. There are defined sets of metrics e.g Halstead metrics (Al Qutaish & Abran, 2005) but it is also possible to adjust a existing metric for specified software or implement a completely new metrics.

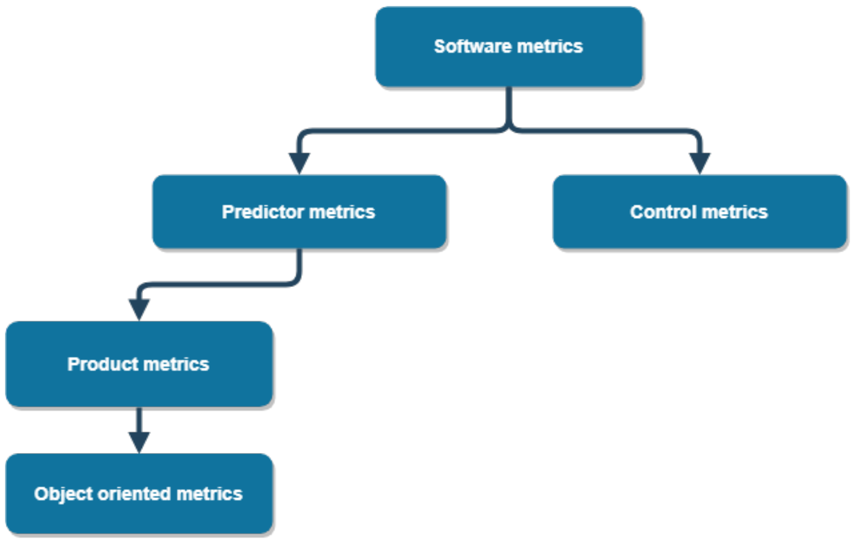

Figure 1: Classification of software metrics

2.2 Halstead complexity measures

Lorem ipsum.

2.3 Object-oriented metrics

Lorem ipsum.

3. Determining useful metrics

Determining the right software is not always a simple task. Measurements can be ambiguous due to the fact that the measurement outputs are just numbers. These number's real correlation to the software quality may be easy to misinterpret.

3.1 Source code examples

To really test if the metrics are correlating software quality we need to first determine low and high quality source code examples. These examples should be good or bad quality code according to the whole development teams opinion. After we have determined what good and bad quality concretely means we can start to figure out which metrics correlate to the quality.

3.2 Trial and error approach

4. Implementing metrics for a development team

After we are sure that the software metrics really are useful and capable of indicating software quality it is time to make use of the metrics. In this chapter I will implement a proof of concept and possibly introduce it for the software development team's use.

4.1 Software metrics in agile development process

Lorem ipsum.

4.2 Essential tools for implementation

For a concrete implementation, some software tools are needed. The preference is that metrics are easily visible and understandable for each developer. Metrics are not meant to be a stressing factor which might happen if the metrics are forced to be seen on a day-to-day basis. Metrics should rather be an auxiliary tool to help developers perceive a bigger picture regarding source code. No comparison was done while choosing the tools and the main idea is to use tools that are easily available or already in use by the company.

4.2.1 CppDepend

CppDepend (https://www.cppdepend.com/) is a commercial tool specifically made to analyze C/C++ code. It provides many of the common software metrics and has a good looking user interface. There are many other promising tools but investigating available options is out of scope on this thesis. Using this tool gives a good indication if software metrics are useful for the software development team or not. If the tool later turns out to be exchangeable with a more usable one, it can easily be replaced.

4.2.2 Jenkins

Jenkins (https://jenkins.io/)

4.2.3 Build server

Lorem ipsum.

4.3 System description

Lorem ipsum.

4.4 Conclusions

Lorem ipsum.

.

.

.

.

.

These citations are temporary for bibtex to show unused citations in the references section.

(Tikka, 2014)

(Coleman, Ash, Lowther, & Oman, 1994)

(Viljanen et al., 2015)

(Zhuo, Lowther, Oman, & Hagemeister, 1993)

(Al Qutaish & Abran, 2005)

(Jamali, 2006)

(Sarker, 2005)

Bibliography

Al Qutaish, R. E., & Abran, A. (2005). An analysis of the design and definitions of halstead's metrics. In *15th int. workshop on software measurement (iwsm'2005). shaker-verlag.*

Brooks, F., & Kugler, H. (1987). *No silver bullet*. University of North Carolina at Chapel Hill.

Coleman, D., Ash, D., Lowther, B., & Oman, P. (1994). Using metrics to evaluate software system maintainability. *Computer*, *27*(8), 44–49.

Jamali, S. M. (2006). Object oriented metrics.

Nguyen, V., Deeds-Rubin, S., Tan, T., & Boehm, B. (2007). A sloc counting standard. In *Cocomo ii forum* (Vol. 2007).

Sarker, M. (2005). An overview of object oriented design metrics. *From Department of Computer Science, Umeå University, Sweden*.

Sommerville, I. (2011). *Software engineering*. Boston : Pearson.

Tikka, A. (2014). *Menetelmiä ohjelmakoodin automaattiseen arviointiin.* Retrieved from `http://urn.fi/URN:NBN:fi:aalto-201501071041`

Viljanen, J., et al. (2015). Measuring software maintainability.

Zhuo, F., Lowther, B., Oman, P., & Hagemeister, J. (1993). Constructing and testing software maintainability assessment models. In *Software metrics symposium, 1993. proceedings., first international.*