Software metrics in long-term projects

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

Santeri Suitiala

Software metrics in long-life projects

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

Santeri Suitiala

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** 6   **Language** English

Abstract

Lorem ipsum.

Keywords software metrics, software quality

urn https://aaltodoc.aalto.fi

Tiivistelmä

Lorem ipsum.

Avainsanat oppimisympäristö, pedagoginen käytettävyys, virheluokittelu, automaattinen tarkastaminen, matematiikan opetus, Stack

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 abstraction or renaming and commenting 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 direction of the software quality makes it possible to plan and give resources to fix software with poor quality.

ABB Drives manufactures industrial drives which are controlled by software. 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. Also if we spend too much time refactoring and improving the code, no new features gets developed and customers are left unsatisfied. In a big company it is hard 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 soft-

ware 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 simple tells how many lines of code 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 but it is also possible to adjust a existing metric for specified software or implement a completely new metrics.
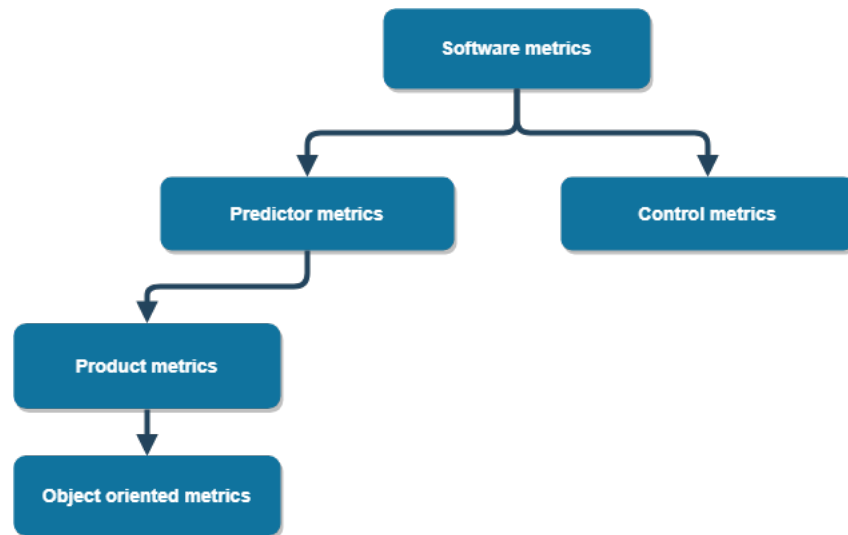
Figure 1: Classification of software metrics

2.2 Halstead complexity measures

Lorem ipsum.

2.3 Object-oriented metrics

Lorem ipsum.

3. Determing 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.

4. Implementing metrics for a development team

Lorem ipsum.

These citations are 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.*