Bachelor's Programme in Science and Technology

Software metrics in long-term projects

Deployments and utilization of software metrics in long-lasting software projects

---

Santeri Suitiala

BACHELOR'S THESIS

Aalto University
BACHELOR'S THESIS 2018

Towards proper errors!

The errors and their interpretation in automatically assessed exercises in mathematics

Hannu Tiitu

Thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Technology.
Otaniemi, 27 Nov 2017

Supervisor:	professor Lauri Malmi
Advisor:	docent Jarmo Malinen

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

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** IL3011

Supervisor Marko Hinkkanen

Advisor Jorma Keronen

Level Bachelor’s thesis **Date** 27 Nov 2017 **Pages** 70 **Language** English

Abstract

Lorem ipsum.

Keywords learning environment, pedagogical usability, error classification, automated assessment, teaching mathematics, STACK

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 code or even a whole software project. 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. Tracking the quality of software makes it easier to plan and budget well beforehand which makes the software more profitable.

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 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 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

Lorem ipsum.

Bibliography

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

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