Ruprecht-Karls-Universität Heidelberg
Institut für Informatik
Lehrstuhl für Parallele und Verteilte Systeme

Masterarbeit

Design and Implementation of a tool for automatic,non-trivial code guideline checking

Name:	Enrico Kaack
Matrikelnummer:	3534472
Betreuer:	Name des Betreuers
Datum der Abgabe:	dd.mm.yyyy

Ich versichere, dass ich diese Master-Arbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Heidelberg, dd.mm.yyyy

Zusammenfassung

Abstract auf Deutsch

Abstract

This is the abstract.

Contents

List of Figures

List of Tables

1 Introduction

This chapter contains an overview of the topic as well as the goals and contributions of your work. Description of the domain Description of the problem Summary of the approach Outline of own contributions and results

1.1 Motivation

1.2 Goals

1.3 Structure

Here you describe the structure of the thesis. For example:

In Kapitel 2 werden grundlegende Methoden für diese Arbeit vorgestellt.

2 Background and Related Work

General description of the relevant methods/techniques

What has been done so far to address the problem (closer related work)

Possible weaknesses of the existing approaches

2.1 Code Quality

Code Quality describes the quality of source code with regard to understandability and readability. Developers can understand well-written code easily. High-quality code has an impact on onboarding new developers, writing new code and maintaining the existing code.

The onboarding of new developers is an investment of time and money in the developer. The faster a new developer can understand an existing code base, the faster the developer can start writing productive code and providing value.

Maintaining existing code and adding features is part of most software today (TODO source). Agile development is a methodology used in software development that reflects this requirement. Source Code is improved and changed with runnable software versions at the end of each iteration. From a business standpoint, the always-changing code is modeled by subscription-based contracts that include new features and bugfixes. The easiness to change source code is business-critical, and a high-quality code can affect this requirement in the following kinds [1]:

1. Well-written code makes it easy to determine the location and the way source code has to be changed.

2. A developer can implement changes more efficient in good code.

3. Easy to understand code can prevent unexpected side-effects and bugs when applying a change.

4. Changes can be validated easier.

The International Organization for Standardization provides the standard ISO/IEC 25000:2014 for "Systems and software Quality Requirements and Evaluation (SQuaRE)"[3].

Code Quality is measured by the following code characteristics:

1. Reliability
2. Performance efficiency
3. Security
4. Maintainability

Besides the mentioned maintainability characteristics, Code Quality also depends on reliability (like multi-threading and resource allocation handling), performance efficiency for efficient code execution, and security (like vulnerabilities to frequent attacks like SQL-injection).

2.2 Clean Code

Clean Code is a concept for high-quality code, coined by the book Clean Code by Robert C. Martin [2]. The root cause for unclean code is chaotic code. Developers produce chaotic code in a conflict between deadline pressure based on the visible output (the functionality of the software) and extra effort to make code more intuitive. The latter is not directly visible as productive output, although an accumulation of chaotic code reduces the productivity over time [2]. A bigger legacy system with chaotic code will slow down later modifications or additions of code. By following the Clean Code guidelines and best-practices, this productivity loss can be minimized.

The Clean Code techniques focus mainly on maintainability by provididng intuitive code. This has a positive effect on the security and reliability aspect as well, since developers can find edge cases in non-logical behaviour more easily in intuitive code. Some of the following Clean Code principles may decrease the performance efficiency, but in many software projects, developer performance is a more valuable ressource than actual runtime performance (TODO source).

2.3 Quantitative Metrics for Code Quality

??Teaching clean code, the paper from one german university

Tools review for all tools that check different stuff

3 Approach

Approach(es) without the quantitative results (or only selected results to motivate the choices/the problems)

What has possibly failed (short)

What has worked and why

What could be possible method extensions/improvements

4 Quantitative Evaluation

Research questions Description of the evaluation environment/setup Results/answers per research question Optional comparison to prior work Discussion and analysis of strengths/problems Note: see

5 Conclusion

An overall short summary of results What was successful, what not (and hypotheses why) Hints for further future work/extension

Bibliography

[1] Robert Baggen, José Pedro Correia, Katrin Schill, and Joost Visser. Standardized code quality benchmarking for improving software maintainability. 20(2):287–307. Publisher: Springer.

[2] Robert C Martin. *Clean code: a handbook of agile software craftsmanship*. Pearson Education.

[3] ISO Central Secretary. Systems and software engineering — systems and software quality requirements and evaluation (SQuaRE) — guide to SQuaRE.