Constructing a Test Code Quality Model and Assessing its Correlation to Issue Handling

Software Improvement Group

Dimitrios Athanasiou
dmitri.athanasiou@gmail.com
Supervisors: Ariadi Nugroho, Joost Visser and Andy Zaidman

TU Delft

1. Introduction

Software testing consumes 30 to 50% of a project's effort [1]. Its benefits include early defect detection, defect cause localization and removal of the fear to apply changes in the code. Thus, maintaining high quality test code is essential.

2. Aim

This study defines a test code quality model and attempts to provide validation of such a model as an indicator of issue handling performance.

- RQ1: How can we evaluate test code quality?
- RQ2: What is the relation between test code quality and issue handling performance?

3. Measuring Test Code Quality

Following the GQM approach, to answer RQ1 we need to answer the following questions:

- How extensively is the system tested?

The higher the test coverage, the higher the confidence that defects are detected throughout the entire system.

- How effectively is the system tested?

Exercising the production code is essential but not enough. Assuring the ability to detect and trace the cause of a defect increase effectiveness.

- How maintainable is the test code?

It is necessary to write highly maintainable test code in order to avoid making the adaptation of the tests to the changes of the code a heavy burden.

4. Test Code Quality Model

	Coverage	Assertion-McCabe Ratio	Assertion Density	Directness	Maintainability Model
Completeness	X	X			
Effectiveness			X	X	
Maintainability					X

	Duplication	Unit size	Unit complexity	Module Dependency
Analyzability		X	X	
Changeability	X		X	X
Stability	X			X

4. Test Code Quality Model (cont.)

- Coverage

Static estimation of test coverage [2].

- Assertion-McCabe Ratio

$$\frac{\#assertions}{McCabe\ of\ production\ code}$$

- Assertion Density

$$\frac{\#assertions}{KLOC\ of\ test\ code}$$

- Directness

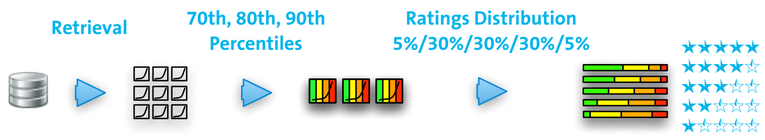
Static estimation of code that is directly called from the tests.

- Maintainability Model

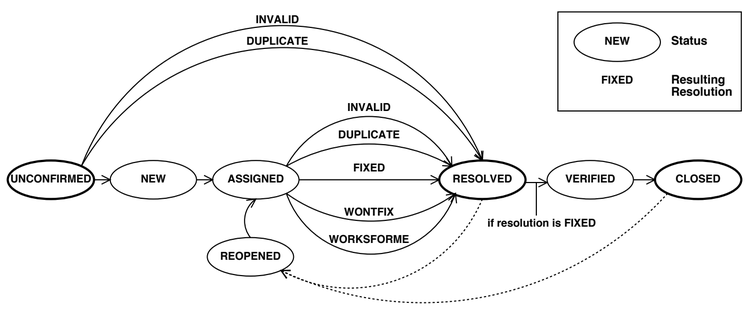
SIG's maintainability model modified and calibrated on test code.

- Calibration

The model was calibrated on 86 Java Open Source and Industry Systems as in SIG's maintainability model [3].



5. Issue Tracking Performance



Issue Report Life-cycle (adapted from [4])

- Defect Resolution Time

The sum of the intervals during which the issue was open until the last time it was marked as "resolved" or "closed".

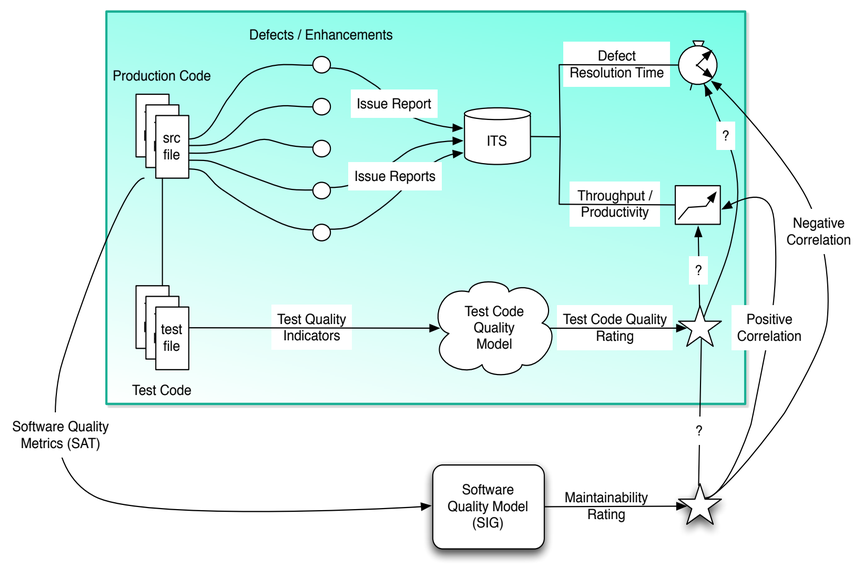
- Throughput

$$\frac{\#resolved\ issues\ per\ month}{KLOC}$$

- Productivity

$$\frac{\#resolved\ issues\ per\ month}{\#developers}$$

6. Design of Experiment



The colored frame focuses on the objective of this study while what is out of the frame is related work performed by Luijten et al. [5] and Bijlsma [6] that forms the basis for this study.

7. Data & Methods

Data

- 18 Open Source Java Systems
- 75 Snapshots
- Snapshots of the same systems were at least 1 year apart and had at least 30% code churn

Methods

- Spearman correlation analysis
- Confidence level at 99% (α = 0.01)
- Bonferroni correction for 4 tests (α = 0.0025)

8. Results

Correlations with Test Code Quality Model Rating

	p-value	cor	N
Defect Resolution Time Rating	0.3256	0.0534	63
Throughput	0.0001	0.5049	54
Productivity	0.0000	0.5133	54
Production Code Maintainability	0.0001	0.4217	75

9. Discussion & Conclusions

- A Test Code Quality Model has been developed, based on direct, static measurements of the source code.
- No significant correlation was found between Test Code Quality and Defect Resolution Time. An explanation would be that test code detects defects during development and therefore most of the detected defects are never reported in an ITS.
- Strong, significant correlation was found between Test Code Quality and throughput and productivity, demonstrating that in systems with higher test code quality issues are being resolved faster at the team and developer level.
- Strong, significant correlation was found between Test Code Quality and Production Code Maintainability, showing that either good code leads to better tests, test-driven development (TDD) leads to better code or just reflects the skill level of the development team.

10. Threats to Validity

- Quality of ITS Data: Poor ITS usage leads to noisy data.
- Confounding Factors: Although we controlled for production code maintainability via multiple regression analysis, other factors such as the popularity of the system exist.
- Generalization beyond Java
- Generalization beyond Open Source Systems

11. Acknowledgments

The author would like to thank colleagues Miguel A. Ferreira and José Pedro Correia for their valuable ideas and insights they have provided throughout the project.

[1] Michael Ellims, James Bridges, and Darrel C. Ince. The economics of unit testing. Empirical Softw. Engg., 11:5–31, March 2006.
[2] Tiago L. Alves and Joost Visser. Static estimation of test coverage. In Proceedings of the 2009 Ninth IEEE International Working Conference on Source Code Analy- sis and Manipulation, SCAM '09, pages 55–64, Washington, DC, USA, 2009. IEEE Computer Society.
[3] Ilja Heitlager, Tobias Kuipers, and Joost Visser. A practical model for measuring maintainability. In Proceedings of the 6th International Conference on Quality of Information and Communications Technology, pages 30–39, Washington, DC, USA, 2007. IEEE Computer Society.
[4] Andreas Zeller. Why Programs Fail: A Guide to Systematic Debugging. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.
[5] Bart Luijten, Joost Visser, and Andy Zaidman. Assessment of issue handling ef- ficiency. In Jim Whitehead and Thomas Zimmermann, editors, Proceedings of the 7th Working Conference on Mining Software Repositories (MSR 2010), pages 94–97. IEEE Computer Society, 2010.
[6] Dennis Bijlsma. Indicators of Issue Handling Efficiency and their Relation to Software Maintainability. MSc thesis, University of Amsterdam, Amsterdam, The Netherlands, 2010.