Is Code Quality Related to Test Coverage?

Jorge Arturo Wong-Mozqueda, Robert Haines and Caroline Jay
School of Computer Science
The University of Manchester
Manchester, United Kingdom
Email: caroline.jay@manchester.ac.uk

***Abstract*—A good test suite is vital for minimising errors, and ensuring that software is easy to maintain. Another factor viewed as being important for the success and longevity of software is code quality. We report on work examining whether there is a correlation between code quality and test coverage, using seven different metrics: lines of code, McCabe's cyclomatic complexity, number of local methods, depth of inheritance tree, coupling between objects, improvement of lack of cohesion in methods and lack of documentation.**

An analysis of three large, open source Java projects showed that all of the response variables had a modest but significant relationship with line coverage, and a stronger relationship with branch coverage: as coverage rose, so did software quality. We propose that writing tests may help people to adopt a 'software quality' mindset, by encouraging them to think about how code will be used as it is written. Testing may improve software sustainability not only by helping to ensure code does not regress, but also by supporting developers in adopting good software engineering practices.

I. Introduction

Test coverage is a metric that tells us how much of a codebase is covered by unit tests. We know that testing is important for checking that the software meets requirements, and for ensuring maintainability, but is the relationship between testing and software quality more complex than that? The aim of the reported study is to discover if code quality is related to the proportion of code that is covered by tests. We hope that by understanding the relationship between test coverage and wider software quality measures, we will gain additional information that can be used for improving software engineering practices, ultimately promoting the development of more sustainable software.

II. Methodology

A. Metric Selection

A set of seven well known metrics were selected from the suite described in [2], formally described in the ISO/IEC 9126 standard [5]. The metrics in this suite are divided into the following categories: size, interface complexity, structural complexity, inheritance, coupling, cohesion and documentation. Metrics that were hypothesized to be related to test coverage were chosen from across these categories.

Lines of Code (LOC) is used as an indication of class size, where a higher value means longer and potentially more complex code. A lower value is deemed desirable as the code is likely to be easier to comprehend and analyse. Shorter classes often follow good design practices, like the Single Responsibility Principle [3].

Number of local Methods (NOM), an indicator of interface complexity, measures the number of methods locally declared in a class. As the interface grows, the class usually becomes more complex, and consequently more difficult to test.

McCabe's Cyclomatic Complexity (CC), is a frequently used measure of structural complexity. It calculates the complexity of a software entity through the number of paths that could be taken within it. As the number of paths increases, the control flow usually becomes more complex and therefore more difficult to test.

Depth of Inheritance Tree (DIT) was selected because inheritance is a basic yet powerful concept of object oriented languages. DIT calculates the complexity of a software entity based on the distance between a node and its root down the inheritance tree [2]. As the code goes down the inheritance tree, the control flow becomes more complex, and consequently more difficult to test.

Coupling Between Objects (CBO) calculates the complexity of a class through its dependencies: a class is considered well designed when it is loosely coupled. As the number of dependencies increases within a class, it increases the complexity and decreases the maintainability of the code, making it more difficult to test.

Improvement of Lack of Cohesion in Methods (ILCOM) provides a measure of class cohesion. ILCOM calculates the number of connected components in a class. High cohesion is a desirable characteristic within a class in object oriented languages, and this metric is a possible indicator of how well a class was designed. It is usually harder to test classes that do not have cohesion between their components.

Finally, *Lack of Documentation* (LOD) was chosen as an interesting metric that considers comments in the code, with at least one comment per method and one per class as a minimum target. Comments often make the purpose of methods and classes clearer, increasing maintainability and facilitating the reuse of the code. Comments in Java code can also be used to automatically build API documentation for a project, so one might expect well maintained code to include at least one comment per method and per class for this purpose.

B. Selection of Projects

Projects were selected from the set of Java projects hosted at GitHub with the highest number of forks [1]. Projects were not included if they: did not run in the standard Java Runtime

[1]Project code and data is available from https://github.com/arturowmex/metrics_merger.

Environment; were deprecated or no longer maintained, or; did not build and run without intervention upon cloning from GitHub. Results are presented here for the top three projects meeting these criteria: Netty, Spring-boot and Alibaba-dubbo. The number of sub-projects and classes for each project, along with the number of contributors and the project creation date, are shown in Table I.

Project	Sub-projects	Classes	Contributors	Creation Date
Netty	19	983	165	2010-11-09
Spring-boot	8	838	180	2012-10-19
Alibaba-dubbo	24	723	10	2012-06-19

TABLE I. ANALYSED PROJECTS

C. Statistical Analysis

Cobertura was used to determine values for the line coverage, branch coverage and cyclomatic complexity of each class. The remaining metrics were gathered using the VizzMaintenance Eclipse plugin [1]. The results of a Pearson correlation analysis can be seen in Table II, which reports R and P values for line coverage (LC) and branch coverage (BC). A star (*) indicates that $P < 0.05$.

III. DISCUSSION

The results demonstrate a relationship between test coverage and code quality, with the quality metrics correlating negatively with test coverage, where a lower value indicates less complex and more maintainable code. Whilst the correlations are modest, the low P values indicate they are unlikely to be due to chance. Branch coverage also appears to correlate more consistently and more strongly with the quality metrics than line coverage. Whilst line coverage tells us whether a statement, such as a conditional, has been tested, branch coverage checks whether every case has been tested, and highlights where they have not. It is better at identifying weak points in the test suite, and is therefore a better indicator of robustly-tested code than line coverage.

Netty	R **(LC)**	P **(LC)**	R **(BC)**	P **(BC)**
LOC	-0.095868*	0.002622	-0.221626*	2.10E-12
NOM	-0.077280*	0.015371	-0.170306*	7.78E-08
CC	-0.030399	0.341030	-0.250631*	1.52E-15
DIT	-0.037586	0.239053	-0.073661*	0.020905
ILCOM	-0.145504*	4.64E-06	-0.388049*	1.12E-36
CBO	-0.074325*	0.019775	-0.240969*	1.88E-14
LOD	-0.232389*	1.61E-13	-0.285402*	7.02E-20
Spring-Boot	R **(LC)**	P **(LC)**	R **(BC)**	P **(BC)**
LOC	0.022020*	5.24E-01	-0.138774*	5.56E-05
NOM	0.002319	0.946549	-0.140619*	4.41E-05
CC	-0.054081	0.117727	-0.328804*	1.41E-22
DIT	-0.137022*	6.91E-05	-0.023515	0.496623
ILCOM	-0.022677	0.512088	-0.110849*	0.001308
CBO	-0.019527	0.572423	-0.036011	0.297754
LOD	-0.143052*	3.23E-05	-0.227835*	2.50E-11
Alibaba-dubbo	R **(LC)**	P **(LC)**	R **(BC)**	P **(BC)**
LOC	-0.043691	0.240661	-0.223800*	1.17E-09
NOM	-0.024219	0.515559	-0.168959*	4.92E-06
CC	-0.100579*	0.006797	-0.358705*	2.24E-23
DIT	-0.245059*	2.39E-11	-0.138653*	0.000184
ILCOM	-0.098263*	0.008193	-0.197943*	8.03E-08
CBO	-0.136590*	0.000229	-0.187666*	3.72E-07
LOD	-0.239555*	6.78E-11	-0.265597*	3.87E-13
All Projects	R **(LC)**	P **(LC)**	R **(BC)**	P **(BC)**
LOC	-0.074303*	0.000176	-0.208734*	1.93E-26
NOM	-0.071241*	0.000323	-0.173464*	1.23E-18
CC	-0.147314*	8.18E-14	-0.340957*	2.81E-70
DIT	-0.163916*	8.80E-17	-0.124169*	3.30E-10
ILCOM	-0.079201*	6.36E-05	-0.220375*	2.36E-29
CBO	-0.059043*	0.002890	-0.169025*	9.24E-18
LOD	-0.220845*	1.78E-29	-0.268367*	3.23E-43

TABLE II. CORRELATION RESULTS

The fact that quality is more strongly correlated with branch coverage than line coverage is noteworthy, as it indicates that it is not merely the presence of tests that appears to improve code, but the presence of 'good' tests. There are two possible explanations for this relationship. One is that a developer who writes high quality code is likely to be the type of developer who writes tests. Another is that the act of writing tests itself causes code to be structured in a particular way, and that this practice results in less complex, and therefore higher quality, software. These explanations are not mutually exclusive, and it is possible that both are in play. The effect of writing tests on software architecture is a strongly-debated topic [6]; here we appear to have some evidence that producing a robust test suite has a positive effect on code structure.

An interesting correlation is with LOD. Comments in the code have no direct effect on the tests as comments are ignored when the code is run. Nevertheless, the results across the three projects show that the classes and methods that are documented have a higher percentage of code covered by tests. The P values of LOD all present a value below 0.05, and in the case of line coverage LOD is the metric with the highest value of R. This suggests that when people write a test, they also write a comment – or vice versa.

Inevitably, this analysis produces more questions than it answers. It appears that Spring-Boot has fewer quality metrics showing significant correlations with either test coverage metric than the other projects, and we need to examine this outcome in more depth to understand why it might be. To confirm the results, it will be important to include more projects in the analysis, and also to understand the results in the context of other factors known to affect quality, such as team size [4]. The current work has a focus on Java. It will be interesting to explore the relationship in other languages.

Nevertheless, the consistent correlation between software quality and test coverage indicates that the value of writing tests may go beyond simply checking correctness and preventing regression, by encouraging developers to produce higher quality, more sustainable code.

REFERENCES

[1] Barkmann, H., Lincke, R., & Löwe, W. (2009, May). *Quantitative evaluation of software quality metrics in open-source projects.* In Advanced Information Networking and Applications Workshops, 2009. WAINA'09. International Conference on (pp. 1067-1072). IEEE.

[2] Lincke, R., & Löwe, W. (2005). *Compendium of software quality standards and metrics.* http://www.arisa.se/compendium/

[3] Martin, Robert C. (2009). *Clean code: a handbook of agile software craftsmanship.* Pearson Education, ISBN: 9780132350884

[4] Bernstein, M. (2014). *Does Team Size Impact Code Quality?* http://blog.codeclimate.com/blog/2014/05/21/does-team-size-impact-code-quality/. [Accessed 10 Aug. 2015].

[5] ISO (2000). *ISO/IEC FIDS 9126-1 Information technology Software product quality. First Edition.* [online] http://www.cse.unsw.edu.au/~cs3710/PMmaterials/Resources/9126-1%20Standard.pdf. [Accessed 10 Aug. 2015].

[6] Holub, A. (2014). *Test-Driven Design* http://www.drdobbs.com/architecture-and-design/test-driven-design/240168102. [Accessed 16 Aug. 2015].