Mythical Unit Test Coverage

Article *in* IEEE Software · May 2018
DOI: 10.1109/MS.2017.3281318

4 authors:

Vard Antinyan
Volvo Car Group
14 PUBLICATIONS **27** CITATIONS
SEE PROFILE

Jesper Derehag
Ericsson
7 PUBLICATIONS **8** CITATIONS
SEE PROFILE

Anna Sandberg
University of Gothenburg
40 PUBLICATIONS **284** CITATIONS
SEE PROFILE

Miroslaw Staron
University of Gothenburg
137 PUBLICATIONS **996** CITATIONS
SEE PROFILE

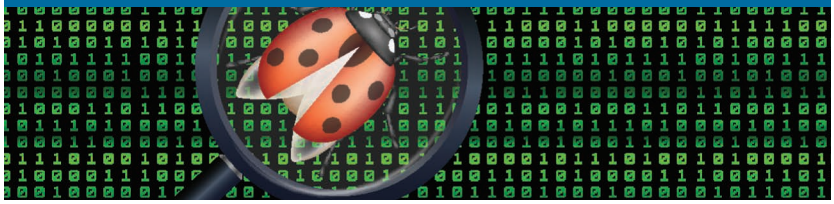
Mythical Unit Test Coverage

Vard Antinyan, University of Gothenburg

Jesper Derehag and Anna Sandberg, Ericsson

Miroslaw Staron, University of Gothenburg

// Ericsson evaluated the adequacy of its unit-test-coverage criterion. The evaluation revealed that high unit-test coverage didn't seem to provide any tangible help in producing defect-free software. //



IT IS A continuous struggle to understand how much a product should be tested before its delivery to the market. Ericsson, as a global software development company, decided to evaluate the adequacy of the unit-test-coverage criterion that it had employed for years as a guide for sufficiency of testing. Naturally, one can think that if increasing coverage decreases the number of defects significantly, then coverage can be considered a criterion for test sufficiency. To test this hypothesis in practice, we investigated the relationship of unit-test-coverage measures and post-unit-test defects in a large commercial product of Ericsson. The results indicate that high coverage did not seem to be any tangible help in producing defect-free software.

The Use of Coverage Measures

Testing is the process of executing a program with the intent of finding errors.[1] Sufficient testing has a decisive role in product delivery. However, as practice has shown, it is rather a complex task to understand whether a product is sufficiently tested or not. There are several unit-test-coverage measures that are used to quantify the sufficiency of testing. Three popular measures are *statement coverage*, *decision coverage*, and *function coverage*:

- Statement coverage is the percentage of statements in a file that have been exercised during a test run.
- Decision coverage is the percentage of decision blocks in a file that have been exercised during a test run.
- Function coverage is the percentage of all functions in a file that have been exercised during a test run.

Simply increasing coverage takes effort from software developers, but the problem is that we do not know whether this effort is justified, because we do not know how much an increase in coverage can decrease the number of defects. There are reports that have indicated that, in theory, simply satisfying coverage criteria can miss important code execution possibilities and leave undetected defects.[2,3] Other reports have proposed tactics, such as assertions and causal analysis, that can improve the defect-finding capabilities of tests independently of coverage.[4,5] In practice, however, the direct effect of test coverage on defect-proneness is still unknown.

Existing Studies

First, we conducted a literature survey to find out what research was available on the subject. The survey gave us 29 articles that most likely were related to our study. After a close examination, we found that only eight of them had a direct relation to the subject of the coverage–defect relationship. These papers' findings are presented in Table 1.

It seems that seven of the eight papers supported the statement that the

0740-7459/18/$33.00 © 2018 IEEE

Table 1. Findings about coverage and test sufficiency.

Reference no.	Context	Summary of findings
6	The paper covered two large industrial products and examined actual defects. It measured block coverage and arch coverage. Cyclomatic complexity and code changes were measured and controlled for.	The correlation between coverage and defects was none or very weak. Moreover, the effort required to increase the coverage from a certain level to 100% increased exponentially.
7	The paper covered 12 small programs. It measured actual defects, block coverage, and decision coverage.	A qualitative analysis found no association between the defects and coverage.
8	Interviews were conducted with 605 practitioners to understand whether coverage measures were used as test sufficiency criteria.	Mixed responses were obtained. Some practitioners used coverage as sufficiency criteria; others stopped testing when they felt the most complex part of the code was tested.
9	The paper covered 12 small programs. It examined artificial defects. It used Monte Carlo simulation to find out the relationship between defects and coverage. Block coverage and defect coverage were measured.	The results did not support the hypothesis of a causal dependency between test coverage and the number of defects when testing intensity was controlled for.
10	The paper covered two large open source products. It examined actual defects. Test suite size, statement coverage, and decision coverage were measured. Defectiveness was measured as a binary variable. Code coverage was not collected as is, but was manually generated and manipulated.	A moderate to strong correlation was found between coverage and defects.
11	The paper covered five large open source products. It examined artificial defects. Statement coverage and (modified) decision coverage were measured. Code size was measured and controlled for.	A weak to moderate correlation was found between coverage and defects. The type of coverage did not have an impact on the results.
12	This paper covered an experiment on a large software product. It examined artificial defects. Block coverage and decision coverage were measured. The correlation of coverage and defects was assessed under different testing profiles.	A moderate correlation was found between coverage and defects. The correlation was different for different testing profiles.
13	This paper covered an experiment on 14 industrial products. It examined both artificial and actual defects. Tests were generated during the experiment. Decision and condition coverage were used. The size of test suites was controlled for.	Coverage measures were weak indicators for test suite adequacy. High coverage did not necessarily mean effective testing.

coverage measures are weakly correlated with the number of defects. Only one paper presented a moderate to strong correlation.[10] Most important, in seven of the eight papers, one or several of the following issues were present:

- artificial defects (mutants);
- uncontrolled confounding factors such as size, change rate, or complexity;
- artificial tests and coverage control; and
- small products and a small number of defects.

These factors reduced the likelihood that the obtained results effectively represented the reality. Essentially, only one paper presented data that was sufficiently close to a practical case.[6] Curiously enough, the authors of this paper argued that there was a correlation between defects and coverage but did not emphasize the fact that the statistical effect size was very small, which is essential in understanding the adequacy of the coverage criterion. Having such inconclusive results, we decided to conduct a case study that avoided artificial conditions in the investigated product and controlled as many confounding variables as possible in order to obtain more conclusive results for practitioners.

The Investigated Product

We studied a large telecom product developed by Ericsson. The product size was about two million LOC.

The organization consisted of about 150 engineers who delivered several major releases of the product each year. The organization used mixed agile and lean development methodologies, relying on incremental code deliveries by semi-independent development teams. As a frontline development organization, it was always eager to identify impediments in its development chain. Thus, it was natural to question the adequacy of test coverage measures that were used as recommendations for test sufficiency.

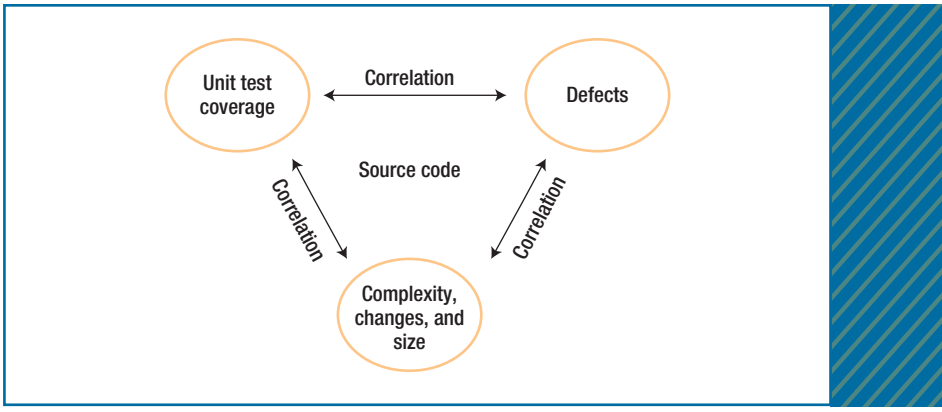


FIGURE 1. The focus of the study.

The Method of Investigation

We collected all defects per file for a year. Generally, if a file was changed due to a defect correction, that change was tagged as "bug fix" in the corresponding development branch. Therefore, it was possible to count how many files had been defective and how many defects had been fixed in a file. The defects we measured were usually found during integration and system tests or were reported by customers. The defects that were found during unit testing were not reported and measured. In contrast, we measured the three coverage measures per file for unit tests, for the same year the defects were measured. Because the coverage data was stable over the given year, we took only a snapshot measurement for the given year.

Considering that unit tests were done earlier than integration and system tests, we could expect that having high coverage during unit testing would reduce the chance of defects emerging during integration and system testing. We could also expect the opposite: less coverage in unit testing would lead to more defects in integration and system testing. This means that if there was a tangible negative correlation (<-0.4) between a unit test coverage measure and the defects per file found later, then the coverage measure could be further analyzed to understand its adequacy of use. If no tangible correlation was found, however, the coverage measure could be regarded as an inadequate indicator of test sufficiency.

We also measured the size, complexity, and changes of files to understand how they affected the test coverage, number of defects, and coverage–defect relationship. Figure 1 depicts an overview of our analysis.

Originally, our analysis focused just on scrutinizing the relationship of test coverage and defects. However, since both coverage and defects are affected by code properties—complexity, changes, and size—we investigated their influence as well. Probably there are other variables that influence the coverage–defect relationship. These could be the developers' experience in coding and testing, the programming language in which the product was developed, and the integrated development environment that offers testing tools. But we assumed that their influence was randomly distributed over the source files, so our results wouldn't suffer noticeably.

Results

The Pearson and Spearman correlation coefficients between the number of defects and other measures and between coverage and other measures are in Table 2. Important values are in bold.

The first important thing is that the correlation coefficients between statement, decision, and function coverage and the number of defects were weak (see rows 2, 3, and 4 under the column "correlation with defects"). Interestingly, the correlation between coverage measures was very strong, indicating that they were very similar to each other (see rows 3 and 4 under the column "correlation with statement coverage"). This is why the correlation coefficients between the coverage measures and the number of defects were nearly equal.

Since the decision, statement, and function coverage were strongly correlated, only one of them was used in the further correlation analysis with the other variables (the last column of the table). The fact that the correlation between the coverage and the number of defects was weak can be regarded as the first indication

Table 2. Pearson and Spearman correlation coefficients between various defect, coverage, change, size, and complexity measures.*

Row	Property	Measure	Correlation with defects	Correlation with statement coverage
1	Defect	No. of defects	1	–0.19/–0.13
2	Coverage	**Statement coverage**	**–0.19/–0.13**	1
3	Coverage	**Decision coverage**	**–0.19/–0.13**	**0.91/0.87**
4	Coverage	**Function coverage**	**–0.18/–0.14**	**0.87/0.86**
5	Change	No. of versions	**0.79/0.62**	–0.14/–0.07
6	Change	No. of developers	0.76/0.63	–0.14/–0.06
7	Change	Amount of changed code	0.61/0.55	–0.11/–0.08
8	Change	Amount of added code	0.58/0.53	–0.11/0
9	Change	Amount of deleted code	0.51/0.55	–0.10/–0.08
10	Change	Age	0.31/0.27	–0.27/–0.21
11	Size	No. of statements	0.62/0.49	–0.17/–0.11
12	Size	LOC	**0.67/0.53**	–0.18/–0.12
13	Complexity	Cyclomatic complexity	0.64/0.48	–0.18/–0.12
14	Complexity	Maximum block depth	**0.42/0.42**	**–0.40/–0.33**
15	Complexity	Parameter count	**0.52/0.45**	0/0
16	Complexity	Percentage of comments	**0/0**	0/0
17	Coverage density	Statement coverage/LOC	**–0.06/–0.25**	–
18	Coverage density	Statement coverage/no. of versions	**–0.18/–0.30**	–

** Important values are in bold.*

that the coverage measures are inappropriate as test sufficiency criteria. There could be a problem with this kind of conclusion, however, because the size of files was not controlled for.

Our analysis assumed that files with equal coverage should have an equal number of defects. But this assumption is not true because a file with 1,000 LOC and 50% coverage has 500 LOC untested, while another file with only 100 LOC but 50% coverage has only 50 LOC untested. Noticeably, the files have equal coverage, but there is a much larger likelihood of finding defects in 500 lines of untested code than in 50 lines of untested code.

As a matter of fact, the strong correlation between the LOC and the number of defects (0.67/0.53) indicated that a larger size was more prone to defects, which is quite intuitive. The same problem emerged for files with different change rates. Files that have more changes (versions) have been under more intensive development and therefore are more prone to have defects.

These problems of size and change create validity threats for our initial results. Therefore, in order to neutralize the effects of size and change on coverage–defect analysis, we created two additional measures—average coverage per LOC (statement coverage / LOC) and average coverage per version (statement coverage /

no. of versions). Lines 17 and 18 in Table 2 show that the Spearman correlation coefficients between the new measures and the number of defects were a little improved (–0.25 and –0.30); however, they still did not gain any tangible value.

At this point, we reached an important conclusion for this study. Increasing unit test coverage did not necessarily decrease the number of defects. Increasing the coverage only created a slight tendency toward decreased defects. This fact is illustrated in Figure 2. Even though the figure shows a downward association between the number of defects and coverage for some files (the red dots), for most of the files, which are closer to the origin of the coordinate system, this association does not exist.

We also found that apart from the maximum block depth, the rest of the complexity and change measures presented in Table 2 were strongly correlated with either the LOC or the number of versions. So, they did not have any additional impact on the coverage–defect relationship, when the LOC and the number of versions were already controlled for.

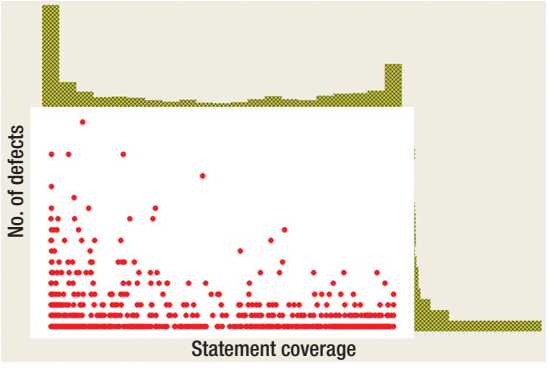


FIGURE 2. A marginal plot for statement coverage and defects.

The Effect of Complexity on the Results

As we mentioned earlier, there was only one measure that was not strongly correlated with the size and change measures: the maximum block depth. This measure shows the maximum level of nesting in a file. Table 2 shows that maximum block depth was the only measure that had a tangible negative correlation with coverage (–0.40/–0.33), suggesting that nesting noticeably hampered increasing test coverage. At the same time, the maximum block depth had a tangible correlation with the number of defects (0.42), indicating that nested code had noticeably more defects than simpler code.

Since the maximum block depth was correlated with both the number of defects and coverage, we found it interesting to understand the coverage–defect relationship in the context of the maximum block depth. Figure 3 shows two contour plots. Figure 3a shows the relationship between the statement coverage, maximum block depth, and number of defects. The dark-green areas indicate no defects, whereas the yellow, orange, and red areas indicate an increasing number of defects.

What is very interesting about Figure 3a is that all of the areas with a large number of defects are on the right, clearly indicating that defects emerged only in places where the nesting got deeper. However, we cannot draw the same conclusion for the coverage–defect relationship, because the defects are along the line-of-coverage axis. It is true that there is more red at the bottom of the plot, but the upper part still contains red and yellow areas as well. It is important to recall that coverage was negatively correlated with the maximum block depth, so some of the files in the bottom-right red area had low coverage because of a high level of nesting. This essentially means that nesting had a double effect on code: it increased the defect-proneness of code and hampered writing unit tests.

It is important to note that the maximum block depth is not a solidly defined complexity measure for source files as entities. It indicates the nesting level only for a block, not for a whole file. Yet it showed a tangible effect on the files. We assume that a more adequate complexity measure based on nesting will indicate that nesting has a much greater effect on coverage and the number of defects.

Figure 3b shows a similar presentation of the relationship between the statement coverage, number of defects, and number of versions. This plot also clearly shows that the files that had the fewest versions (fewest changes) did not have defects, no matter whether they were tested or not. When we substituted the number of versions with the LOC, we got a very similar picture. Thus, the areas of code that were small or had not changed were defect-free. This is quite natural according to the principles of theory of probability, because

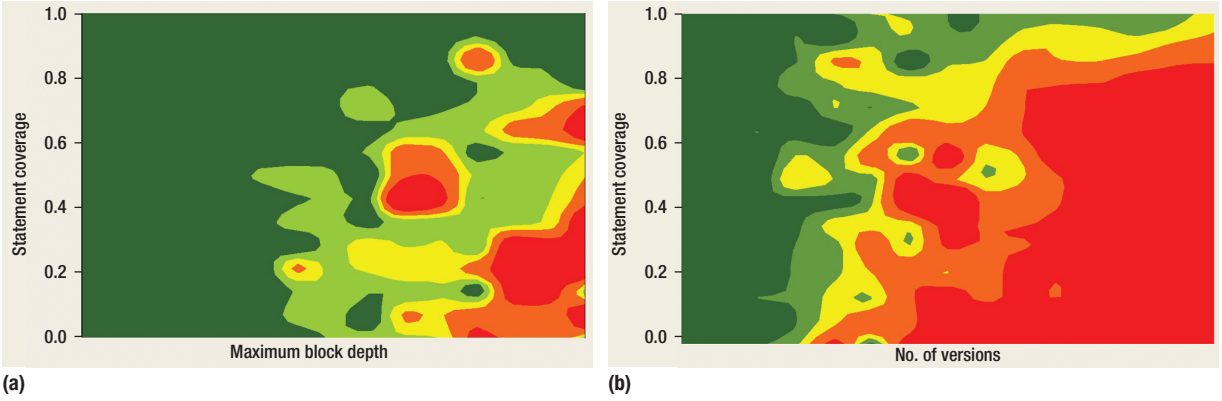


FIGURE 3. Contour plots of the number of defects related to (a) statement coverage and (b) the number of versions. Dark green areas have no defects; the red areas have the most defects.

it is more likely to find defects in larger files or in files that are under intensive development.

Usually, complexity is more manageable than size or change. The latter two properties are not always possible to reduce since they are the core constituents of product development and delivered functionality: no value can be delivered without developing a new piece of code or modifying an existing one. Meanwhile, a certain amount of complexity can be reduced by contrivance and smart coding techniques. Therefore, it is possible to partially control the effect of complexity on the number of defects and the coverage.

Statement coverage, decision coverage, and function coverage are popular measures that purport to indicate the sufficiency of unit testing. One hundred percent coverage, nonetheless, does not entail 100% tested code. Therefore, it is important to understand how adequate the current coverage measures are as criteria for the sufficiency of unit testing. The results of this case study suggest that

- the adequacy of unit test coverage criteria at Ericsson was a myth—developers need more sophisticated criteria for sufficiency of testing;
- it would be well worth it to conduct a few more similar case studies in other domains in order to understand how general this problem is in practice;
- international standards such as ISO 26262, IEC 61508, ANSI/IEEE 1008-1987, and DO 178B need to reconsider their recommendations of unit test coverage measures as criteria for test sufficiency; and
- developers should realize that managing complexity both decreases error-proneness and facilitates testing of code.

References

1. G.J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*, John Wiley & Sons, 2011.
2. A. Glover, "In Pursuit of Code Quality: Don't Be Fooled by the Coverage Report," blog, IBM developerWorks, 31 Jan. 2006; www.ibm.com/developerworks/library/j-cq01316/index.html.
3. B. Marick, "How to Misuse Code Coverage," *Proc. 16th Int'l Conf. Testing Computer Software*, 1999, pp. 16–18.
4. Y. Chernak, "Validating and Improving Test-Case Effectiveness," *IEEE Software*, vol. 18, no. 1, 2001, pp. 81–86.
5. J. Voas, "How Assertions Can Increase Test Effectiveness," *IEEE Software*, vol. 14, no. 2, 1997, pp. 118–119.
6. A. Mockus, N. Nagappan, and T.T. Dinh-Trong, "Test Coverage and Post-verification Defects: A Multiple Case Study," *Proc. 3rd Int'l Symp. Empirical Software Eng. and Measurement* (ESEM 09), 2009, pp. 291–301.
7. M.R. Lyu, J. Horgan, and S. London, "A Coverage Analysis Tool for the Effectiveness of Software Testing," *IEEE Trans. Reliability*, vol. 43, no. 4, 1994, pp. 527–535.
8. B. Smith and L.A. Williams, *A Survey on Code Coverage as a Stopping Criterion for Unit Testing*, tech. report TR-2008-22, Dept. of Computer Science, North Carolina State Univ., 2008, pp. 1–6.
9. L. Briand and D. Pfahl, "Using Simulation for Assessing the Real Impact

ABOUT THE AUTHORS

VARD ANTINYAN is a software quality assurance engineer at Volvo Cars. His research interests include software quality, risk management, software measurement, and complexity. Antinyan received a PhD in software engineering from the University of Gothenburg. Contact him at vard.antinyan@volvocars.com.

JESPER DEREHAG is a software designer at Ericsson. His research interests include software size estimation, quality, and software measurement. Derehag received a BSc in electrical engineering from the Chalmers University of Technology. Contact him at jesper.derehag@ericsson.com.

ANNA SANDBERG is an associate professor in the University of Gothenburg's Department of Applied IT. Her main focus is to drive software change and improve productivity in the businesses she works in. Sandberg received a PhD in software process improvement from the University of Gothenburg. She's a member of IEEE. Contact her at anna.sandberg@volvocars.com.

MIROSLAW STARON is a professor of software engineering at the University of Gothenburg. His research interests include software modeling, design, metrics, and decision support. Staron received a PhD in software engineering from the Blekinge Institute of Technology. Contact him at miroslaw.staron@gu.se.

of Test Coverage on Defect Coverage," *Proc. 10th Int'l Symp. Software Reliability Eng.*, 1999, pp. 148–157.

10. P.S. Kochhar, F. Thung, and D. Lo, "Code Coverage and Test Suite Effectiveness: Empirical Study with Real Bugs in Large Systems," *Proc. IEEE 22nd Int'l Conf. Software Analysis, Evolution, and Reengineering* (SANER 15), 2015, pp. 560–564.
11. L. Inozemtseva and R. Holmes, "Coverage Is Not Strongly Correlated with Test Suite Effectiveness," *Proc. 36th Int'l Conf. Software Eng.* (ICSE 14), 2014, pp. 435–445.
12. X. Cai and M.R. Lyu, "The Effect of Code Coverage on Fault Detection under Different Testing Profiles," *ACM SIGSOFT Software Eng. Notes*, vol. 30, no. 4, 2005, pp. 1–7.
13. G. Gay et al., "The Risks of Coverage-Directed Test Case Generation," *IEEE Trans. Software Eng.*, vol. 41, no. 8, 2015, pp. 803–819.