A Survey on Code Coverage as a Stopping Criterion for Unit Testing

Ben Smith and Laurie Williams
North Carolina State University
[bhsmith3, lawilli3]@ncsu.edu

Abstract

The evidence regarding code coverage as a predictor of software quality is conflicting and inconclusive. However, an estimate of the software testing practices of the majority of professionals can help researchers know how code coverage is being used—or whether it is being used at all. The purpose of this report is to present the results of an online survey we conducted to estimate the percentage of software developers who use code coverage as a stopping criterion for unit testing. We find that a majority of participants 1) perform automated unit testing; and 2) use code coverage, though not always as a stopping criterion. Those people who do not use code coverage, do not find it useful or provide some other reason. Finally, in place of code coverage, we find that most participants stop testing when they have "tested the most important parts of the code".

1. Introduction

Code coverage, defined in 1963 by Miller and Maloney [5] is loosely defined as the percentage of some code structure or artifact which is executed or covered by at least one test [6]. Some researchers have since found that using code coverage to control testing can be an effective way to remove faults [4]; however, others explain that code coverage may only be associated with the number of tested faults because both measures are related to the number of test cases written (test intensity) [1]. Overall, the evidence surrounding the use of code coverage as a software reliability predictor is somewhat inconclusive.

The purpose of this paper is to present the results of an online survey we conducted to estimate the percentage of software developers who use code coverage as a stopping criteria for unit testing. An estimate of the software testing practices of the majority of developers can help researchers know how code coverage is being used—or whether it is being used at all. We also find relevant the rationale behind using or not using test coverage and what alternate forms of stopping criteria are used in its place. In this report, we present the results (in the appendix) of an online survey we conducted which had 605 participants who answered questions about code coverage and their software testing practices. We also briefly survey the related literature to code coverage to explain the relevance of these results and our rationale for conducting the survey.

The rest of this report is organized as follows. Section 2 provides a brief review of the history of code coverage and presents some results in favor and against using code coverage as a stopping criterion. Then, Section 3 presents our method for conducting the survey. Finally, Section 4 presents the results of the survey. The results of the survey can be found in the appendix.

2. Background

Code coverage dates back to 1963, when Miller and Maloney first explained that if a section of a program is not executed by at least one test, the development team has no way of knowing whether that section of code executes correctly or not [5]. Miller and Maloney describe code coverage indirectly by indicating that "there should be no possibility that an unusual combination of input data or conditions may bring to light an unexpected mistake in the program" [5]. Since its inception, many researchers have performed studies to determine whether code coverage (an internal metric) can be used as an indicator of software reliability as measured by the number of failures over execution time (an external metric, or software quality factor). For example, Lyu, et al. performed an empirical evaluation of the effectiveness of code coverage testing as a fault detection technique [4]. Lyu, et al. asked 34 software teams to independently develop

their own versions of an industry-scale critical flight application, and collected the faults the teams introduced for further observation. Lyu, et al. found that a good test case should not be characterized only by its ability to detect more faults, but also by its ability to detect *unique* faults—or those faults which are not detected by any other test case [4]. In addition, Lyu et al. find that code coverage is a good indicator of test case variety, however code coverage is not directly related to fault coverage [4].

Briand and Pfahl indicate that across many earlier studies, code coverage has been significantly correlated with defect coverage (the number of defects uncovered by the test set divided by the total number of defects in the system) [1]. However, Briand and Pfahl explain that this relationship does not indicate that there is a causal relationship between high test coverage and better software reliability [1]. Briand and Pfahl conducted an empirical analysis in which Briand and Pfahl collected 12 program versions which were independently developed from the same specification and tracked the code coverage and number of faults found in each. Then, Briand and Pfahl conducted statistical tests to see whether four common types of code coverage were significantly predictive of the system's defect coverage [1]. Briand and Pfahl conclude by contending that the relationship between defect coverage and code coverage could be due to an external influence: test intensity as measured by the number of test cases [1].

Regardless of the mixed results in its history, code coverage has been incorporated into reliability estimation models [2], and used to prioritize certain parts of a system for testing [3]. Chen, et al. explain that models which estimate the reliability of software systems tend to overestimate because they do not contain a complete or thorough operational profile and assume that each test case improves the reliability of the system [2]. To help make reliability estimates more accurate, Chen et al. incorporate code coverage as a predictor of software quality [2].

3. Survey Method

The initial pool of potential survey participants came from the second authors email address book. She invited all those in her address book she felt were involved in software development to take a short survey and to forward her email to their colleagues. Many invitation recipients indicated they did, indeed, forward her email to their colleagues and posted it on message boards.

Participants were emailed a link to a survey we created on SurveyMonkey.com[1] which contained the following questions:

Q1. Do you (personally) perform automated unit testing?
Q2. Do you (personally) use code coverage as a stopping criteria for automated unit testing?
Q3. When do you (personally) stop writing automated unit tests*?
Q4. Why do you use code coverage*?
Q5. Why don't you use code coverage*?
Q6. What is your role at your organization*?
Q7. What programming languages do you use*?

Each question had to be answered once displayed to the respondent; however, not all questions were asked of each respondent (more details below). The questions with an asterisk (*) allowed the respondent to check more than one of the answers provided in the appendix. The logical layout of the survey is presented in Figure 1.

We only wanted to ask people who use unit testing at all (Q1) whether they use code coverage as a stopping criteria (Q2), therefore if a respondent answered that he or she did not perform unit testing, we asked his or her role and ended the survey. Similarly, for those people who do conduct automated unit testing, we want to know *why* they used code coverage or refused to use code coverage. As a result, when a respondent answered "Yes" to Q2, we transitioned to Q4 and then directly to Q6. However, if a respondent answered that they did not use code coverage as a stopping criteria (Q2) we wanted to know why not (Q5) as well as what they used instead (Q3).

SurveyMonkey.com allows for the adaptive logic presented above and the survey was designed to keep the number of questions asked of each respondent to a minimum. We also asked (Q6) what each respondent's self-reported role is at their organization and rejected any response from a respondent who did not answer any of the roles Tester, Test Lead, Developer, Developer Lead, or Architect.

[1] http://www.surveymonkey.com

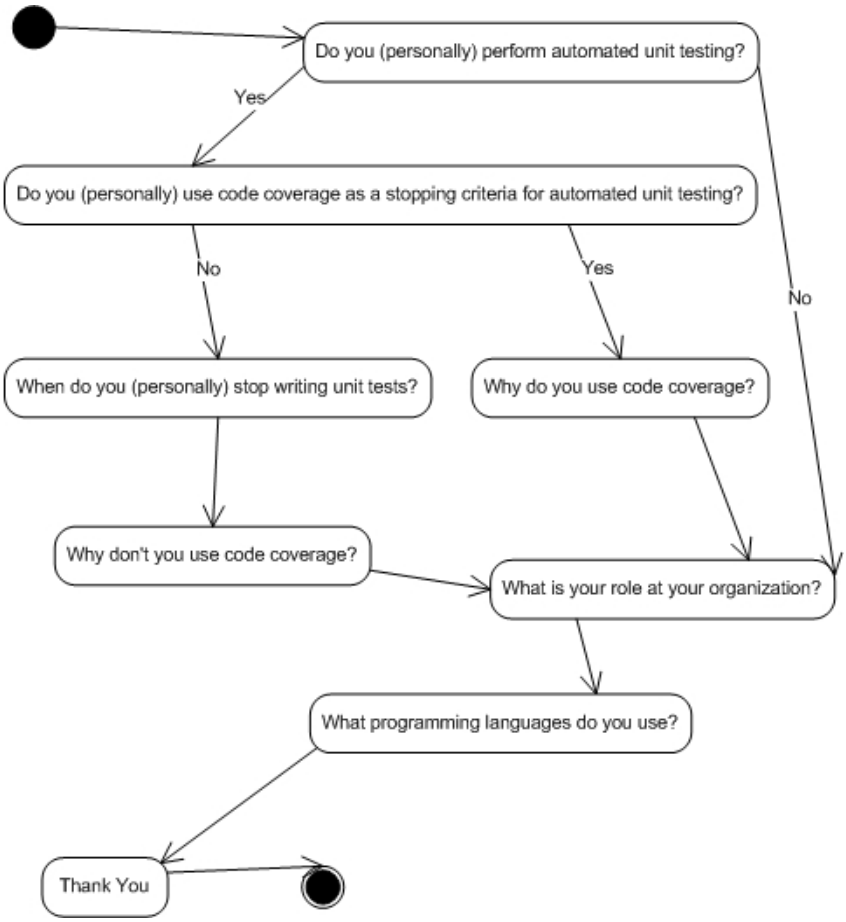

Figure 1: Survey Logic

4. Results

Tables 1-7 present the responses we received from our 605 participants. We divided the responses into two groups: one group who *only* answered that they were either an academic or a researcher; and one group contained participants who identified themselves as having roles traditionally associated with industry. We grouped our responses by isolating anyone who in Q6 marked any of the following: management, test lead, developer lead, developer, architect or marked "other" and typed something similar to "tester" or "quality assurance." For example, if a participant marked "Academic" but also marked "Developer", this response was included in the "Testers, Developers & Management" category, but not the "Academics & Researchers" side.

In Q2 and Q5, many responses indicated that the participant used code coverage, but did not use code coverage as a stopping criterion for writing tests. We isolated these participants in both Q2 and in Q5, with cross referencing. If a participant marked other in Q5, and provided a response similar to "I do, but not as a stopping criterion", this response was marked in the "I do, but not as a stopping criterion" in Q5 and was also removed from the "No" category in Q2 (participants could not reach Q5 without first marking "No" in Q2). We also moved several participants from the "Other" category in Q2 to the "I do, but not as a stopping criterion" in Q2, because these participants' answers for Q2 were something resembling "I do, but not as a stopping criterion."

In Q6, a significant number (37) of participants selected the "Other" option and indicated something similar to either "tester" or "quality assurance." We isolated these responses into a group called "Tester." In Q7, many participants (104) indicated that they programmed in C#, and so we isolated these responses into a new C# category.

Table 1. (Q1) Do you (personally) perform automated unit testing?				
	Academics & Researchers		*Developers, Testers & Management*	
Answer	*Percent*	*Responses*	*Percent*	*Responses*
Yes	44.4%	82	59.2%	248
No	55.9%	104	40.8%	171
Answered Question		186		419
Skipped Question		0		0

Table 2. (Q2) Do you (personally) use code coverage as a stopping criteria for automated unit testing?				
	Academics & Researchers		*Developers, Testers & Management*	
Answer	*Percent*	*Responses*	*Percent*	*Responses*
Yes	35.2%	25	29.5%	66
No	52.1%	37	37.6%	84
I use it, but not as a stopping criteria	11.2%	8	25.5%	57
Other	1.4%	1	7.1%	16
Answered Question		71		223
Skipped Question		115		110

Other answers: sometimes, depends; it is only part of the pictures; use it to identify missing tests; I try to

Table 3. (Q3) When do you (personally) stop writing automated unit tests? (Check all that apply)				
	Academics & Researchers		*Developers, Testers & Management*	
Answer	*Percent*	*Responses*	*Percent*	*Responses*
When I have tested the most important parts of the code	53.8%	21	48.1%	90
I code while I test, so I stop when I am done coding	43.6%	17	48.1%	90
When my coworker or management has approved my test set	12.8%	5	10.2%	19
When I have tested each method once	7.7%	3	6.4%	12
When I have tested the complex parts of the code	33.3%	13	36.4%	68
When I or my team runs out of time	20.5%	8	23.5%	44
Other	20.5%	8	18.7%	35
Answered Question		39		187
Skipped Question		147		232

Other responses: confidence in the tests; test-driven development; other structure: boundaries, paths, use cases

Table 4. (Q4) Why do you use code coverage? (Check all that apply)				
	Academics & Researchers		*Developers, Testers & Management*	
Answer	*Percent*	*Responses*	*Percent*	*Responses*
Lets me know which lines/paths I have involved in a test case	61.9%	13	77.3%	51
Required by management or my organization	19.0%	4	25.8%	17
A respected publication tells me it was important	0.0%	0	1.5%	1
It gives me confidence in the quality of my tests	61.9%	13	69.7%	46
Other (see below)	19.0%	4	15.2%	10
Answered Question		21		66
Skipped Question		165		353

Other responses: find dead, missing or untested code; natural outcome of TDD; tells me what I can refactor; repeatability of sound engineering practice

Table 5. (Q5) Why don't you use code coverage? (Check all that apply)				
	Academics & Researchers		*Developers, Testers & Management*	
Answer	*Percent*	*Responses*	*Percent*	*Responses*
I can't find a good tool for my language	5.9%	2	10.4%	19
Coverage tools are too expensive	11.8%	4	3.3%	6
I don't want to	17.6%	6	6.0%	11
I don't find coverage information useful	20.6%	7	25.1%	46
I do, just not as a stopping criteria	23.5%	8	26.2%	48
Other (see below)	29.4%	10	39.3%	72
Answered Question		34		183
Skipped Question		152		236

Other responses: not a part of the process; tools are poor quality; limited resources; it is ineffective; I work in performance testing; plan to use it; wasn't aware of it

Table 6. (Q6) What is your role at your organization?				
	Academics & Researchers		*Developers, Testers & Management*	
Answer	*Percent*	*Responses*	*Percent*	*Responses*
Academic	57.2%	83	6.4%	27
Developer	0.0%	0	42.5%	178
Management	0.0%	0	20.8%	87
Researcher	38.6%	56	9.8%	41
Architect	0.0%	0	17.7%	74
Test Lead	0.0%	0	12.6%	53
Developer Lead	0.0%	0	26.5%	111
Tester	0.0%	0	8.8%	37
Other	20.0%	29	4.2%	18
Answered Question		145		419
Skipped Question		41		0

Other responses: coach; consultant; manager; founder; analyst

Table 7. (Q7) What programming language(s) do you use? (Check all that apply)				
	Academics & Researchers		*Developers, Testers & Management*	
Answer	*Percent*	*Responses*	*Percent*	*Responses*
Java	66.9%	97	57.7%	236
PHP	9.0%	13	6.9%	35
Perl	9.7%	14	11.7%	55
ActionScript(Flash)	1.4%	2	1.2%	7
C	28.3%	41	16.2%	74
C++	33.8%	49	18.3%	79
C#	6.2%	9	22.6%	95
VisualBasic	2.1%	3	3.6%	19
Python	9.0%	13	9.6%	336
Other	25.5%	37	24.5%	103
Answered Question		145		419
Skipped Question		41		0

Other responses: Ruby; bash; ColdFusion; Smalltalk; Groovy; Assembler; Matlab; JavaScript; JSP; HTML; CSS; SQL; ASP.NET; VB.NET

10. References

[1] L. Briand and D. Pfahl, "Using simulation for assessing the real impact of test coverage on defect coverage," *Software Reliability Engineering, 1999. Proceedings. 10th International Symposium on*, pp. 148-157, 1999.

[2] M. H. Chen, M. R. Lyu, and W. E. Wong, "An empirical study of the correlation between code coverage and reliability estimation," *Proceedings of the 3rd International Symposium on Software Metrics: From Measurement to Empirical Results*, 1996.

[3] M. Gittens, K. Romanufa, D. Godwin, and J. Racicot, "All code coverage is not created equal: a case study in prioritized code coverage," Conference of the Center for Advanced Studies on Collaborative research, pp. 1-15, Toronto, Ontario, Canada, 2006.

[4] M. R. Lyu, Z. Huang, S. K. S. Sze, and X. Cai, "An empirical study on testing and fault tolerance for software reliability engineering," *Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on*, pp. 119-130, 2003.

[5] J. C. Miller and C. J. Maloney, "Systematic mistake analysis of digital computer programs," *Communications of the ACM*, vol. 6, pp. 58-63, 1963.

[6] H. Zhu, P. A. V. Hall, and J. H. R. May, "Software Unit Test Coverage and Adequacy," *ACM Computing Surveys*, vol. 29, 1997.