Becoming Test Certified

Denny Anderson
University of Virginia
dra2zp@virginia.edu

ABSTRACT

The purpose of this paper is to provide a brief overview of how Google tests software, which is based on the book *How Google Tests Software* by James Whittaker, Jason Arbon, and Jeff Carollo [2]. Then, the levels of test certifiability are explained as well as how the Testing Grouplet began awarding certifications to companies. This is based on the article "Test Certified," written by Mike Bland [1]. Finally, the paper concludes by giving a personal account of my current testing practices as well as how my coding practices and view of testing have changed as a result.

1 INTRODUCTION

Early on, testing was something that was rarely done in software development. Only until users would find a bug in their software would they attempt to patch the software and publish a new update, replacing the broken code. However, this is a very expensive way to go about developing software. Thus, the Testing Grouplet made a promising start by making testing more respectable in software companies, giving certifications for various levels of test certifiability.

In addition, the way in which Google tests their software much describes the software itself: state-of-the-art. Software testing at Google is much different than almost every other company out there, and their success can be largely attributed to where they put testing in their design structure.

2 HOW GOOGLE TESTS SOFTWARE

The way in which Google tests software is far different from just about any other software company. Google works hard to ensure that testing is not a "dogmatic, process-heavy, labor-intensive, and time-consuming" process. "Techniques that require too much upfront planning or continuous maintenance simply won't work." Google aims to make testing as easy as possible in order to make the lives of their developers and testers much easier. Google prides itself in being able to produce new, reliable features very often. In order to do this, they must have high test coverage or identify a diverse array of tests in order to find as many bugs and potential places of error as possible. Thus, testing must be quick and easy to do. "Testing must not create friction that slows down innovation and development."

Another aspect that sets Google apart from other companies is that their "testing is interwoven with development to the point that the two practices are indistinguishable from each other." Since developers are heavily involved in the testing process, Google has very few dedicated testers. "The burden of quality is on the shoulders of those writing the code. Quality is never 'some tester's' problem. Everyone who writes code at Google is a tester." This is a very good mindset to have because if developers don't take pride in their code and don't care about testing their code to arrive at a relatively bug-free program, they will likely have many faults, errors, and failures in their code. This will cause testers to become overwhelmed with work if every developer has a bug in her/his code, and they may not find all the errors because it may be too much to test if they have to fix everyone's code. The way developers are involved in the testing process is that they code incrementally. That is, they code a little and then test what they just built. This results in much more manageable code to repair when the developer runs into an issue. The error must be with the small piece of code that was most recently written. This will allow the developer to see their fault much more easily in order to fix that portion of code or alternatively, fix a different piece of code that they wrote previously that is causing their newly written code to fail. "The number of actual dedicated testers at Google is so disproportionately low." As a result, "quality is more of an act of prevention than it is detection."

Although Google does have some programmers who call themselves testers, "their actual mission is one of productivity." Their role in software development, besides writing tests, is to make developers more productive in order to avoid sloppy code. This makes everyone's job easier and more productive.

Furthermore, Google's testers are horizontally structured in the software development cycle. Since there are so few testers, they are responsible for testing software across all of the company's various projects. Making testers horizontal across all product teams helps to ensure that the code that is being developed works across various fields. It can also help testers get a view of what's going on in each part of the company, which may help identify code that interferes with code coming from another product team. Some errors come from one set of developers having their expectations while another set of developers have differing expectations, and this causes code to break somewhere. Having a group of testers foresee and overlook all these product areas helps to spot these errors, making them much more avoidable.

Finally, one way that Google makes their testing process quick and easy is that they attempt to automate every step of it and only write manual tests when absolutely necessary. "Recording technology converts manual tests to automated tests, with point-and-click validation of content and positioning, to be re-executed build after build to ensure minimal regressions, and to keep manual testers always focusing on new issues." In addition, they also automate bug report submissions and the routing of manual testing tasks.

3 TEST CERTIFIED LEVELS

Test certifiability is divided into five levels. Level one includes setting up small, medium, and large tests and identifying nondeterministic tests – tests that produce a different output given the same input due to some level of randomness in the code. This level takes anywhere from one day to about a week to achieve for a given program. Level two prohibits anyone from submitting untested code. This level of certification is achievable without coming anywhere close to the appropriate coverage levels or test size ratio. It takes anywhere from a couple weeks to a couple months to achieve for a given program. Level three is the model for a smooth-running testing process. It consists of high coverage for every developer, coverage goals for each group of tests of a certain size, and low tolerance for broken or nondeterministic tests. Later, levels four and five were added, but the author argues that these aren't quite necessary, as they only incorporate other tools like static analysis to test code.

The certification process involved a member of the team working closely with developers and testers to ensure that the requirements were being satisfied. Then, there would be some peer review, and the team would be awarded Level X Certification. The purpose of this was to move testing to a more respectable role and to encourage companies to test their programs as much as possible before releasing it to users. This is important because errors found earlier during the testing phase in the software development cycle are much easier and inexpensive to fix. Once code is published for real users, there is a high cost associated with fixing broken or buggy code since developers must quickly identify and repair the portion of code, release the new code in possibly a new software update, and users could become frustrated if this occurs frequently, leading to decreased use in the software overall.

4 BECOMING TEST CERTIFIED

I will begin by saying that nearly all the programs I have written have been in class at college. That is to say that I've never written more than several hundred lines of code (except for once, which is discussed later in this section), and they did not really require more than a few number of test cases. I have never submitted code that hasn't been tested before. I'm not saying that I've never submitted buggy code, but I have at least known about the bugs before submitting it (usually due to some sort of deadline). Thus, I would estimate that historically, my level of testing has been around level two. Personally, I would be unable to imagine a scenario in which I wouldn't test code prior to submitting. It just seems like a terribly inefficient idea, and in a company, I'd be afraid of getting fired for poor coding practices.

For a web programming class, I developed a web application (and further improved it and added additional functionality after the class), and the final code ended up being more than 5,000 lines of code. Much like the way Google developers test their software, I would add a single portion of code at a time, and then try it out to ensure that it works. If it worked, I could move on. Otherwise, I would keep working on the buggy code until it was fixed. This is an especially good idea when writing in languages like C or C++ since a program can result in a segmentation fault, which is incredibly difficult to debug. However, if the code is written incrementally, it makes debugging segmentation faults much easier. All the testing that I have done has been informal. I find that writing formal test cases like JUnit can be complicated and overly time-consuming. I simply run the code with inputs and determine if the code runs as expected. While I tested every feature of my web app, I did not write any formal tests or any automated tests. There may certainly be bugs in the program, but at the very least, I've tested every feature of it several times and ensured that it works as expected given proper inputs.

My goal looking forward is to increase my level of testing to around level four. When I begin working at a company, or even if I plan on publishing my web app, the need to write formal tests is of utmost importance. All of my tests in the past have been manual and informal. The need to create formalized, automated tests is fundamental to creating code that has a much higher confidence of being "bug-free." Although bug-free code is certainly the goal, there is no way to be absolutely, 100% sure that the written code is truly free from all errors and will function as expected in all cases. However, by creating more automated tests and increasing test coverage, the likelihood that the code is buggy can be greatly reduced.

CONCLUSION

We have seen the importance that Google places in both its developers and testers. The reason they are able to have such a low number of testers is because all their developers are responsible for performing testing on their own piece of code before they submit it. This gives testers a chance to find real mistakes rather than dumb mistakes that the developer was simply too lazy to fix. We have also seen the role that the Testing Grouplet has had in changing the role of software testing by strongly encouraging companies to perform at least basic software testing and awarding companies that do. This book and article have certainly changed my own view of software testing, and I will continue to test code as it is being developed rather than waiting until it becomes an enormous and arduous task to complete after the entire program has been written.

REFERENCES

[1] Bland, Mike. “Test Certified.” Mike Bland. October 18, 2011. Accessed November 28, 2017. https://mike-bland.com/2011/10/18/test-certified.html.

[2] Whittaker, James, Jason Arbon, and Jeff Carollo. *How Google Tests Software*. Addison-Wesley, 2012. March 2012. Accessed November 28, 2017. http://ptgmedia.pearsoncmg.com/images/9780321803023/samplepages/0321803027.pdf.