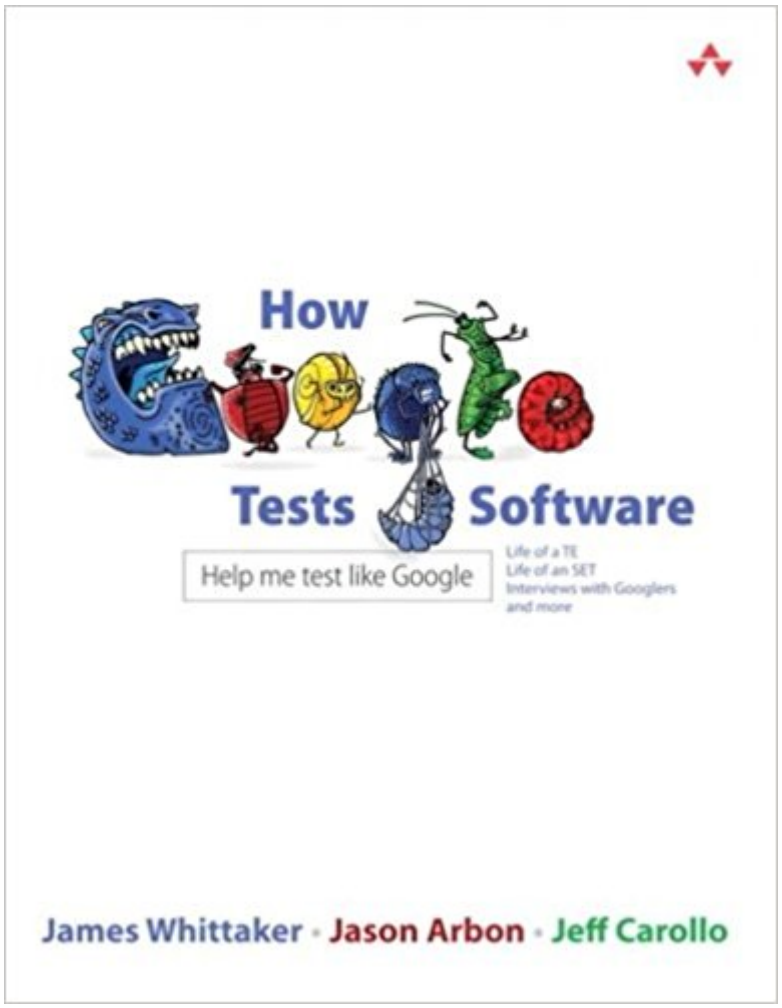
How Google Tests Software



Synopsis

2012 Jolt Award finalist! Â Pioneering the Future of Software Test Â Do you need to get it right, too? Then, learn from Google. LegendaryÂ testing expertÂ James Whittaker, until recently a Google testing leader,Â and two top Google experts reveal exactly how Google tests software, offering brand-new best practices you can use even if youâ ™re not quite Googleâ ™s sizeâ |yet! Â Breakthrough Techniques You Can Actually Use Â Discover 100% practical, amazingly scalable techniques for analyzing risk and planning testsâ |thinking like real usersâ |implementing exploratory, black box, white box, and acceptance testingâ |getting usable feedbackâ |tracking issuesâ |choosing and creating toolsâ |testing â œDocs & Mocks,â • interfaces, classes, modules, libraries, binaries, services, and infrastructureâ |reviewing code and refactoringâ |using test hooks, presubmit scripts, queues, continuous builds, and more. With these techniques, you can transform testing from a bottleneck into an acceleratorâ “and make your whole organization more productive! Â

Book Information

Paperback: 320 pages

Publisher: Addison-Wesley Professional; 1 edition (April 2, 2012)

Language: English

ISBN-10: 0321803027

ISBN-13: 978-0321803023

Product Dimensions: 7 x 0.8 x 9.1 inches

Shipping Weight: 1 pounds (View shipping rates and policies)

Average Customer Review: 4.1 out of 5 stars 54 customer reviews

Best Sellers Rank: #85,597 in Books (See Top 100 in Books) #21 inÂ Books > Computers & Technology > Programming > Software Design, Testing & Engineering > Testing #27 inÂ Books > Computers & Technology > Networking & Cloud Computing > Cloud Computing #91 inÂ Books > Textbooks > Computer Science > Software Design & Engineering

Customer Reviews

James Whittaker is an engineering director at Google and has been responsible for testing Chrome, maps, and Google web apps. He used to work for Microsoft and was a professor before that. James is one of the best-known names in testing the world over. Jason Arbon is a test engineer at Google and has been responsible for testing Google Desktop, Chrome, and Chrome OS. He also served as development lead for an array of open-source test tools and personalization experiments. He

worked at Microsoft prior to joining Google. Jeff Carollo is a software engineer in test at Google and has been responsible for testing Google Voice, Toolbar, Chrome, and Chrome OS. He has consulted with dozens of internal Google development teams helping them improve initial code quality. He converted to a software engineer in 2010 and leads development of Google+ APIs. He also worked at Microsoft prior to joining Google."

I saw James Whittaker speak at STAR West in 2011, and he gave a keynote titled "Test Is Dead". His talk was essentially a teaser for How Google Tests Software that he co-wrote with Jason Arbon and Jeff Carollo. The premise of the book is that testers need to have engineering skills (sometimes to an equal extent as software engineers) in order for the testing discipline to reach first class citizenship on equal footing with development.The argument is aligns well with the movement toward agile software development methods. The book goes on to breakdown testing responsibilities for software engineers (SWEs), software engineers in a test role (SETs), and Test Engineers (TEs). Almost half of the book deals with the roles and responsibilities of the TE, and in the Google model, they do have a higher-level role in testing. In essence, it breaks down like this:* SWEs write unit tests for the software they write* SETs write tools to enable testing without external dependencies and write automated functional tests* TEs coordinate the overall testing activities for a product and focus on the user by doing exploratory testingIn addition, the book also outlines a number of tools (many of which have been open-sourced) that Google uses for testing in the context of these roles. The majority of the content focuses on web applications (it’s Google after all), and some of the ideas won’t apply if the majority of your development is for internal customers to your company - since you probably have user training and rules about frequency of release. However, I would say that you could apply 80% of the ideas in any context and probably adapt at least 10% (if not more) of the others to your situation.Also, there is also a chapter on test managers and directors that has interviews with a number of prominent Googlers. Then, the book ends with a discussion on the future of the SET and TE roles at Google along with some of the errors Google made.Google embarked on the transformation in 2007, and my company is currently trying to do something similar. I hope to be able to leverage these ideas in the months ahead. I recommend it to anyone who is or expects to be involved in such a change. I would also recommend it to any tester in an agile development shop. You may not agree with everything in the book, but tells of the future (if not the present) for much of the software testing industry.

When I found out about the book "How Google Tests Software", it didn’t take long until I had

ordered a copy. I find it quite fascinating to read about how Google does things, whether it is about their development process, their infrastructure, their hiring process, or, in this case, how they test their software. I am a developer at heart, but I have worked for a few years as a tester, so testing is also dear to me.It’s quite an interesting book, and it makes some great points about the future of testing. However, despite the phrase "Help me test like Google" on the cover, it is not as useful as I had hoped when it comes to improving your own testing.The book starts off by describing the key roles at Google: SWE (Software Engineer), SET (Software Engineer in Test) and TE (Test Engineer). Briefly, the SWE builds features for Google’s products, the SET develops testing infrastructure and larger-scale automatic tests, and the TE tests the products from a user’s perspective. After the introductory chapter, there is a chapter each on the SET and TE roles, and there is also a chapter on the TEM (Test Engineer Manager) role. The final chapter is about the future of testing at Google (and in general).Software Engineer in Test (SET)As the different roles are explained in the respective chapters, there is also quite a bit of detail on how the testing is done at Google. The most interesting part in the chapter on the SET role is the part about the infrastructure. There is (of course) extensive support for running tests automatically. There is common infrastructure for compilation, execution, analysis, storage and results reporting of tests. Tests are categorized as small, medium, large or enormous. Small tests are basically unit tests where everything external is mocked out, and they are expected to execute in less than 100 ms.Medium tests involve external subsystems, and can use database access, but generally run on one machine (use no network services), and are expected to run in under a second. Large and enormous tests run a complete application, including all external systems needed. They can be nondeterministic because of the complexity, and they are expected to complete in 15 minutes and 1 hour respectively. A good way to summarize them is that small tests lead to code quality, and medium, large and enormous tests lead to product quality. The common test execution environment for running the tests has been developed over time, and has several nice features. It will automatically kill tests that take too long to run (thus the time limits mentioned above).It has several features to facilitate running many different test concurrently on a machine - it’s possible to request an unused port to bind to (instead of a hardcoded port number that could clash with another test), writing to the file system can be done to a temporary location unique to the current test, and private database instance can be created and used to avoid cross talk from other tests. Further, their continuous integration system uses dependency analysis to run only tests affected by a certain change, thus being able to pinpoint exactly which change broke a certain test. This system has been developed by Google for many years, and has become quite capable and tailored to their way

of working.Test Engineer (TE)The most interesting part in the TE chapter is the description of the process used for developing the test plan for a product. The test plan's purpose is to map out what needs to be tested for the product, and when it is done it should be clear what test cases are needed. It can be a challenge to find the right level of detail for a test plan, but it seems like they have found a good balance at Google.The Google process for coming up with the test plan is called ACC, which stands for Attribute, Component and Capability. Attributes are the qualities of the product, the key selling points that will get the people to use the product. The examples given for Chrome include fast, secure and stable. There won't typically be that many attributes.Next, the Components are the major subsystems of the product, around 10 seems to be a reasonable number to include. Finally there are the Capabilities, which are the actions the system can perform for the user. Whereas there are relatively few attributes and components, there can be quite a number of capabilities. The capabilities lie at the intersection of attributes and components. It is natural to create a matrix with attributes along one axis, and components along the other axis. Then each capability will fit in at the given coordinates. A key property for a capability is that it is testable, and each capability will lead to one or more test cases to verify its functionality. Thus the matrix is an aid in enumerating all the test cases that are needed.The matrix allows you to look at what capabilities affect a certain module. If you look along the other dimension, you will see all capabilities supporting a certain attribute. The matrix is also useful in risk analysis, and when tracking testing progress.In the same chapter, there is also a good story about a 10-minute test plan. James Whittaker did an experiment where he forced people to come up with a test plan for a product in 10 minutes. The idea was to boil it down to the absolute essentials, without any fluff, but still being useful. Because of the time constraint, most people just made lists or grids - no paragraphs of text. In his opinion (and I agree), this is the most useful level - it is quick to come up with and doesn't need a lot of busy-work filling out sections in a document template, and still it's a useful basis for coming up with test cases. The common theme in all cases was that people based the plan on capabilities that needed testing.ToolsThere are other interesting testing tools described in the book too. One such tool developed at Google is BITE - Browser Integrated Test Environment. When testing a browser-based app, like Google Maps, and something went wrong, there was a lot of information to extract and put into the bug report. For example, what actions lead up to the bug, what version of the software was running, how the bug manifest itself etc. The BITE browser extension keeps track of all the actions the tester made in the application, and supports filing a bug report by automatically including all the relevant information. It also has support for easily marking in a screen shot where the bug appeared.Another interesting tool is Bots. It involves automatic tests

where many different versions of Chrome fetch the top 500 webpages on the web. The resulting HTML is compared and detailed "diff"-reports are produced.TipsThere was also a sprinkling of interesting ideas (that can definitely be of use in any test organization) throughout the book. Here are the ones that stuck in my head: When asking people to estimate a value for something (for example the frequency of a certain failure scenario), use an even number of values (e.g. rarely, seldom, occasionally, often). That way you can't just pick the middle value - you're forced to think about it more carefully.Another example in the same area. If you want people's opinion of how likely a certain failure scenario is, you could just ask them about it. But another technique is to assign a value yourself, and then ask what they think. Then you have given them something to argue against. Often, people have an easier time to say what something isn't, then what it is.There is also a quote from Larry Page that is referred to several times in the book (for example regarding the relatively few testers at Google) "Scarcity brings clarity", and (later on), Scarcity is the precursor of optimization. Worth thinking about.As well as describing how the testing is done, and which tools are used, there are also a number of interviews with various people in the test organization. The chapter on TEM (Test Engineer Manager) in particular consists almost entirely of interviews, 8 in total. Most interviews in the book were interesting to read, but many of them weren't that useful in terms of tips or ideas to use in your own testing.The Future of TestingFor me, the best chapter in the book was chapter 5, "Improving How Google Tests Software". It is the last and shortest chapter, only 7 pages. In it, James Whittaker shares some profound insights about testing at Google, and testing in general. One of the flaws he sees with testing is that testers are... testers. They are not part of the product development team. Instead, they exist in their own organization, and this separation of testers and developers gives the impression that testing is not part of the product; it's somebody else's responsibility. Further, the focus of testing is often the testing activities and artifacts (the test cases, the bug reports etc.), not the product being tested. But customers don't care about testing per se, they care about products.Finally, a lot of the testing mindset we have today developed in a different era. When you released a product, that was it. There was no easy way to upgrade it, and users had to live with whatever bugs slipped through. However, these days so much of the software can be fixed and upgraded without a lot of fuss. In this environment, it makes less sense to have testers act as users and try to discover what bugs they might run into. Instead, you can release the software, and see what bugs the actual users encounter. Then you make sure these bugs are fixed and that the new release is pushed out quickly.So his opinion is that testing should be the responsibility of all the developers working on the product. It should be their responsibility to test the product and to develop the appropriate tools (with some exceptions, for

instance security testing). Whether you agree or disagree with this, it is definitely food for thought!ConclusionInitially, when I had just finished reading the book, I felt a little disappointed. It was interesting to read, but there didn't seem to be that much to take away from it and apply to your own testing. Pretty much all of the techniques and tools are tailored for Google and their needs, which is just as it should be. But that means that they may not be applicable to your own situation.However, as I am going through it again while writing this review, I realize that there are quite a few good ideas in it - they just have to be adapted to your specific situation. So while not directly applicable, the ideas in the book serve as inspirations for how testing can be organized and executed.

Handbook of Laboratory & Diagnostic Tests W/ Nursing Implications) Widmann's Clinical Interpretation of Laboratory Tests (CLINICAL INTERPRETATION OF LAB TESTS (WIDMANN'S)) Scholastic Success With Reading Tests, Grade 3 (Scholastic Success with Workbooks: Tests Reading)