Forwards from..."How Google Tests Software" 2012

Buy the book by James Whittaker on Amazon
link to this on Google Docs

Note by (author) James Whittaker: As a scribe and journalist in this process, I owe most of the material to the organization that Patrick has created. And I am not just saying this because he gave me permission to write this book. As my boss, he made me write this book!

Patrick Copeland is the senior director of Engineering Productivity and the top of the testing food chain at Google. All testers in the company report up through Patrick (whose skip-level manager, incidentally, is Larry Page, Google's Co-founder and CEO). Patrick's career at Google was preceded by nearly a decade at Microsoft as a director of Test. He's a frequent public speaker and known around Google as the architect of Google's technology for rapid development, testing, and deployment of software.

My adventure with Google started in March of 2005. If you read Alberto's foreword, you know a bit about the conditions of Google around that time. It was small but beginning to show signs of serious growing pains. It was also a time of rapid technological change as the web world was welcoming dynamic content and the cloud was becoming a viable alternative to the then dominant world of client-server.

That first week, I sat with the rest of the Nooglers topped with a tri-colored propeller hat listening to the founders discussing corporate strategy at a weekly company meeting called TGIF. I knew little about what I had gotten myself into. I was naïve enough to be excited and aware enough to be more than a little intimated. The speed and scale of Google made my previous decade of 5-year ship cycles seem like poor preparation. Worse still, I think I was the only tester wearing one of those Noogler hats. Surely there were more of us somewhere!

I joined Google when engineering was just shy of 1,000 people. The testing team had about 50 full timers and some number of temporary workers I never could get my head around. The team was called "Testing Services" and focused the majority of its energy on UI validation and jumping into projects on an as-needed basis. As you might imagine, it wasn't exactly the most glorified team at Google.

But at that time it was enough. Google's primary businesses were Search and Ads. The Google world was much smaller than it is today, and a thorough job of exploratory testing was enough to catch most quality concerns. But the world was slowly changing. Users were hitting the web in unprecedented numbers and the document-based web was giving way to the app-based web. You could feel the inevitability of growth and change where the ability to scale and get

to market quickly would be the difference between relevance and...a place you did not want to be.

Inside Google, the scale and complexity issues were buckling Testing Services. What worked well with small homogenous projects was now burning out good testers as they leapt from one project that was on fire to the next. And topping all that off was Google's insistence to release quicker. Something needed to give and I had the choice between scaling this manually intensive process or changing the game completely. Testing Services needed a radical transformation to match the radical change happening to the industry and the rest of Google.

I would very much like to say that I drew upon my vast wealth of experience and conjured the perfect test organization, but the truth is that my experience had taught me little more than the *wrong way to do things*. Every test organization I ever worked as part of or led was dysfunctional in one way or the other. Stuff was always broken. The code was broken, the tests were broken, and the team was broken! I knew what it meant to be buried under quality and technical debt where every innovative idea was squashed lest it risk breaking the fragile product it depended on. If my experience taught me anything, it was how *not* to do testing.

In all my interactions up to this point, one thing about Google was clear. It respected computer science and coding skill. Ultimately, if testers were to join this club, they would have to have good computer science fundamentals and some coding prowess. First-class citizenship demanded it.

If I was going to change testing at Google, I needed to change what it meant to be a tester. I used to try to imagine the perfect team and how such a team would shoulder the quality debt and I kept coming back to the same premise: The only way a team can write quality software is when the entire team is responsible for quality. That meant product managers, developers, testers...everyone. From my perspective, the best way to do this was to have testers capable of making *testing* an actual feature of the code base. The testing feature should be equal to any feature an actual customer might see. The skill set I needed to build features was that of a developer.

Hiring testers who can code features is difficult; finding feature developers who can test is even more difficult. But the status quo was worse than either so I forged ahead. I wanted testers who could do more for their products and at the same time, I wanted to evolve the nature and ownership of the testing work, which meant asking for far larger investment from the development teams. This is the one organizational structure I had yet to see implemented in all my time in the industry and I was convinced it was right for Google, and I thought that as a company, we were ready for it.

Unfortunately, few others in the company shared my passion for such profound and fundamental change. As I began the process of socializing my equal-but-different vision for the software testing role, I eventually found it difficult to find a lunch partner! Engineers seemed threatened by the very notion that they would have to play a bigger role in testing, pointing out “that’s what test is for.” Among testers, the attitude was equally unsavory as many had become comfortable in their roles and the status quo had such momentum that change was becoming a very hard problem.

I kept pressing the matter mostly out of fear that Google’s engineering processes would become so bogged down in technical and quality debt that I’d be back to the same five-year ship cycles I had so happily left behind in the old client-server world. Google is a company of geniuses focused on innovation and that entrepreneurial makeup is simply incompatible with long product ship cycles. This was a battle worth fighting and I convinced myself that once these geniuses understood the idea of development and testing practices for a streamlined and repeatable “technology factory,” they would come around. They would see that we were not a startup anymore and with our rapidly growing user base and increasing technical debt of bugs and poorly structured code would mean the end of their coder’s playground.

I toured the product teams making my case and trying to find the sweet spot for my argument. To developers, I painted a picture of continuous builds, rapid deployment, and a development process that moved quickly, leaving more time for innovation. To testers, I appealed to their desire to be full engineering partners of equal skill, equal contribution, and equal compensation.

Developers had the attitude that if we were going to hire people skilled enough to do feature development, then we should have them do feature development. Some of them were so against the idea that they filled my manager’s inbox with candid advice on how to deal with my madness. Fortunately, my manager ignored those recommendations.

Testers, to my surprise, reacted similarly. They were vested in the way things were and quick to bemoan their status, but slow to do anything about it.

My manager’s reaction to the complaints was telling: “This is Google, if you want something done, then do it.”

And so that’s what I did. I assembled a large enough cadre of like-minded folks to form interview loops and we began interviewing candidates. It was tough going. We were looking for developer skills and a tester mindset. We wanted people who could code and wanted to apply that skill to the development of tools, infrastructure, and test automation. We had to rethink recruiting and interviewing and then explain that process to the hiring committees who were entrenched and comfortable with the way things were.

The first few quarters were rough. Good candidates were often torpedoed in the vetting process. Perhaps they were too slow to solve some arcane coding problem or didn't fare well in something that someone thought was important but that had nothing to do with testing skill. I knew hiring was going to be difficult and made time each week to write hiring justification after hiring justification. These went to none other than Google co-founder Larry Page who was (and still is) the last approval in the hiring process. He approved enough of them that my team began to grow. I often wonder if every time Larry hears my name he still thinks, "Hiring testers!"

Of course, by this time, I had made enough noise trying to get buy-in that we had no choice but to perform. The entire company was watching and failure would have been disastrous. It was a lot to expect from a small test team supported by an ever-changing cast of vendors and temps. But even as we struggled to hire and I dialed back the number of temps we used, I noticed change was taking hold. The more scarce testing resources became, the more test work was left for developers to do. Many of the teams rose to the challenge. I think if technology had stayed the same as it was, this alone would have taken us nearly where we needed to be.

But technology wasn't standing still and the rules of development and testing were changing rapidly. The days of static web content were gone. Browsers were still trying to keep up. Automation around the browser was a year behind the already tardy browsers. Making testing a development problem at the same time those same developers were facing such a huge technology shift seemed a fool's errand. We lacked the ability to properly test these applications manually, much less with automation.

The pressure on the development teams was just as bad. Google began buying companies with rich and dynamic web applications. YouTube, Google Docs, and so on stretched our internal infrastructure. The problems I was facing in testing were no more daunting than the problems the development teams were facing in writing code for me to test! I was trying to solve a testing problem that simply couldn't be solved in isolation. Testing and development, when seen as separate disciplines or even as distinct problems, was wrong-headed and continuing down such a path meant we would solve neither. Fixing the test team would only get us an incremental step forward.

Progress was happening. It's a funny thing about hiring smart people: They tend to make progress! By 2007, the test discipline was better positioned. We were managing the endgame of the release cycle well. Development teams knew they could count on us as partners to production. But our existence as a late-in-the-cycle support team was confining us to the traditional QA model. Despite our ability to execute well, we were still not where I wanted us to be. I had a handle on the hiring problem and testing was moving in the right direction, but we

were engaged too late in the process.

We had been making progress with a concept we called “Test Certified” (which the authors explain in some detail later in this book) where we consulted with dev teams and helped them get better code hygiene and unit testing practices established early. We built tools and coached teams on continuous integration so that products were always in a state that made them testable. There were countless small improvements and tweaks, many of them detailed in this book, that erased much of the earlier skepticism. Still, there was a lack of identity to the whole thing. Dev was still dev; test was still test. Many of the ingredients for culture change were present, but we needed a catalyst for getting them to stick.

As I looked around the organization that had grown from my idea to hire developers in a testing role, I realized that testing was only part of what we did. We had tool teams building everything from source repositories to building infrastructure to bug databases. We were test engineers, release engineers, tool developers, and consultants. What struck me was just how much the nontesting aspect of our work was impacting productivity. Our name may have been Testing Services, but our charter was so much more.
So I decided to make it official and I changed the name of the team to Engineering Productivity. With the name change also came a cultural adjustment. People began talking about productivity instead of testing and quality. Productivity is our job; testing and quality are the job of everyone involved in development. This means that developers own testing and developers own quality. The productivity team is responsible for enabling development to nail those two things.

In the beginning, the idea was mostly aspirational and our motto of “accelerating Google” may have rung hollow at first, but over time and through our actions, we delivered on these promises. Our tools enabled developers to go faster and we went from bottleneck to bottleneck clearing the clogs and solving the problems developers faced. Our tools also enabled developers to write tests and then see the result of those tests on build after build. Test cases were no longer executed in isolation on some tester’s machine. Their results were posted on dashboards and accumulated over successive versions so they became part of the public record of the application’s fitness for release. We didn’t just demand developers get involved; we made it easy for them to do so. The difference between productivity and testing had finally become real: Google could innovate with less friction and without the accumulation of technical debt.

And the results? Well, I won’t spoil the rest of the book because that’s why it was written. The authors took great pains to scour their own experiences and those of other Googlers to boil our secret sauce down to a core set of practices. But we were successful in many ways—from orders of magnitude, decreases in build times, to run-it-and-forget-it test automation, to open sourcing some truly innovative testing

tools. As of the writing of this preface, the Productivity Team is now about 1,200 engineers or a bit larger than all of Google Engineering in 2005 when I joined. The productivity brand is strong and our charter to accelerate Google is an accepted part of the engineering culture. The team has travelled light years from where we were on my first day sitting confused and uncertain at TGIF. The only thing that hasn't changed since that day is my tri-colored propeller hat, which sits on my desk serving as a reminder of how far we've come.

Foreword by Alberto Savoia

Alberto Savoia is an engineering director and innovation agitator at Google. He first joined Google in 2001 when, among other things, he managed the launch of Google AdWords and played a key role in kick-starting a developer / unit testing culture in the company. He is also the author of *The Way of Testivus* and of "Beautiful Tests" in O'Reilly's *Beautiful Code*.

Writing a foreword for a book you wish you had written yourself is a dubious honor; it's a bit like serving as best man for a friend who is about to spend the rest of his life with the girl *you* wanted to marry. But James Whittaker is a cunning guy. Before asking me if I'd be willing to write this preface, he exploited my weakness for Mexican food by treating me to a very nice dinner and plying me with more than a couple Dos Equis before he "popped the question." By the time this happened, I was as malleable and agreeable as the bowl of guacamole I had just finished. "*Si senor*," was pretty much all I could say. His ploy worked and here he stands with his book as his bride and I get to make the wedding speech.

As I said, he's one cunning guy. So here we go...a preface to the book I wish I had written myself. Cue the mushy wedding music. Does the world really need yet another software testing book, especially yet another software testing book from the prolific James Whittaker, whom I've publicly called "the Octomom of test book publishing" on more than one occasion? Aren't there enough books out there describing the same old tired testing methodologies and dishing out dubious and dated advice? Well, there are enough of those books, but this book I am afraid is not one of them. That's why I wish I had written it myself. The world actually needs this particular testing book.

The Internet has dramatically changed the way most software is designed, developed, and distributed. Many of the testing best practices, embodied in any number of once popular testing books of yesteryear, are at best inefficient, possibly ineffective, and in some

cases, downright counterproductive in today's environment. Things have been moving so fast in our industry that many of the software testing books written as recently as a few years ago are the equivalent of surgery books containing advice about leeches and skull drilling to rid the body of evil spirits; it would be best to recycle them into adult diapers to make sure they don't fall into the hands of the gullible.

Given the speed at which things are evolving in the software industry, I would not be too surprised if ten years from now this book will also be obsolete. But until the paradigm shifts again, *How Google Tests Software* gives you a very timely and applicable insider's view into how one of the world's most successful and fastest growing Internet companies deals with the unique challenges of software testing in the twenty-first century. James Whittaker and his coauthors have captured the very essence of how Google is successful at testing some of the most complicated and popular software of our times. I know this is the case because I've been there through the transition.

I first joined Google as engineering director in 2001. At the time, we had about two hundred developers and...a whopping *three* testers! My developers were already taking responsibility for testing their own code, but test-driven development and test automation tools such as JUnit were just entering the scene, so our testing was mostly *ad-hoc* and dependent on the diligence of the individual writing the code. But that was okay; we were a startup and we had to move fast and take risks or we couldn't compete with the big established players.

However, as the company grew in size and our products became more mission-critical for our users and customers (such as AdWords, one of the products I was responsible for, was quickly becoming a major source of monetizing websites), it became clear that we had to increase our focus and investment in testing. With only three testers, we had no choice but to get developers more involved in testing. Along with a few other Googlers, I introduced, taught, and promoted unit testing. We encouraged developers to make testing a priority and use tools such as JUnit to automate them. But adoption was slow and not everybody was sold on the idea of developers testing their own code. To keep the momentum going, every week at the company's Friday afternoon beer bash (appropriately named TGIF), I gave out testing awards to the developers who wrote tests. It felt a lot like an animal trainer giving treats to doggies for performing a trick, but at least it drew attention to testing. Could I be so lucky that getting developers to test would be this simple?

Unfortunately, the treats didn't work. Developers realized that in order to have adequate tests, they had to write two or three lines of unit test code for every line of code under test and that those tests required at least as much maintenance as the functional code itself and had just as much chance of being buggy. It also became clear to no one's surprise that developer-unit testing was not sufficient. We still needed integration tests, system tests, UI tests, and so on. When it came to

testing, we had a lot of growing up (and substantial learning) to do, and we had to do it fast. Very fast!

Why the urgency? Well, I don't believe that any amount of testing can turn a bad idea or an ill-advised product into a success. I do believe that the wrong approach to testing can kill the chances of a good product or company or, at the very least, slow down its growth and open the door for the competition. Google was at that point. Testing had become one of the biggest barriers to continued success and coming up with the right testing strategy to keep up with our ultra-rapid growth in users, products, and employees without slowing the company down involved a lot of innovative approaches, unorthodox solutions, and unique tools. Not everything worked, of course, but in the process, we learned valuable lessons and practices that are applicable to any company that wants to grow or move at the speed of Google. We learned how to have attention to quality without derailing development or the speed at which we got things done. The resulting process, with some insights into the thinking behind them and what makes them work, is what this book is about. If you want to understand how Google met the challenges of testing in the twenty-first century on modern Internet, mobile, and client applications, then you have come to the right place. I may wish it was me who was telling the rest of the story, but James and his coauthors beat me to it and they have nailed the essence of what testing is like here at Google.

One final note on the book: James Whittaker is the guy who made this book happen. He came to Google, dug in to the culture, took on big and important projects, and shipped products such as Chrome, Chrome OS, and dozens of smaller ones. Somewhere in that time, he became the public face of Google testing. But, unlike some of his other books, much of this material is not his. He is as much a reporter on the evolution of how Google tests software as he is a contributor to it. Keep that in mind as you read it because James will probably try to take all the credit for himself!

As Google grew from 200 to over 20,000 employees, there were many people who played important roles in developing and putting into action our testing strategy. James credits many of them and they have contributed directly by writing sidebars and giving interviews that are published in this book. However, no one, not me, James, or anyone else mentioned in this book, has had as much influence as Patrick Copeland, the architect of our current organizational structure and leader of Google's Engineering Productivity team. Every tester in the company reports up through Patrick and he is the executive whose vision created what James has documented and contributed to here. If anyone can take credit for how Google tests software today, it's Patrick. I am not just saying this because he's my boss; I am saying it because he's my boss *and* he told me to say it!