Evolution of the Tester

Ben Williams

desertblade@gmail.com

Abstract

The Tester. Who are they, where did they come from, and where are they going?

Testers face different challenges than they did 10 years ago. The business and software landscape has changed dramatically. The modern-day tester needs to evolve or face extinction. Gone are the days of detailed requirements before starting development and of running regression tests after implementation. Often a new feature must be deployed as rapidly as possible, sometimes even the same day it was conceived.

The development world is quickly moving away from the waterfall model, replacing it with agile methodologies. Quality Assurance (QA) engineers find themselves trapped between the trusted but time-consuming ways of ensuring quality and the challenge of providing coverage in the fast-moving agile world. While most software developers have found their groove working in agile, quality engineers have struggled and find themselves lagging behind.

Automation has taken us a long way, but there are still shortcomings. Maintainability, reliability, and re-usability are issues faced in all automation suites. In many agile shops, there remains a high dependence on manual testing and drawn-out regressions, causing the all too common QA 'bottleneck.' The stereotype that QA slows down the process has created a tendency to limit the use or effectiveness of quality engineers or forgo testing completely; trading unnecessary quality risks for speed.

The pace of business has changed. In order to survive, QA must change with it. Reducing the need for manual regressions by expanding test automation, improving coverage reporting, simplifying manual regressions and providing continuous testing are solid first steps. Testers will also need to evolve their responsibilities to include understanding the product and end-goal; making sure that what they are working on is right for the end user and the company.

Biography

Ben Williams has over 10 years of experience working in QA. He started as a manual tester, spent time as a release/build manager and developer, and lead quality management teams for start-ups and Fortune 1000 companies. He is currently consulting as a Senior QA Project Lead for a Fortune 200 company, working in a cross-functional agile environment that relies on continuous integration and automation to provide faster quality turnaround.

Prior to his career in software development, Ben served in the US Army as a Combat Engineer. Along with demolition and bomb detonation he also learned the fundamentals of leadership, the importance of planning, and the ability to adapt to any given situation.

Ben has studied a number of disciplines including computer science, political science, psychology and law. He will graduate from George Fox University in December with a Bachelor of Science in Technology Management.

1. Introduction

A couple of years ago at Google's Test Automation Conference, in the opening keynote, “Test is Dead” Dr. James Whittaker said “If you could take a software tester from 1970 and move them forward in time to, say, 2010, they wouldn't have to learn anything new to test. That's a problem. That's a lack of innovation.” Dr. Whittaker’s statement bothered me. How could QA have changed so little over the past 40 years? His statement was a generalization, but he had a point.

Working in software quality for over a decade, I have experienced a large waterfall environment, a small agile start-up, and an agile development environment that still expected waterfall style processes for QA. While many software shops have embraced agile methodologies and continue to move forward, QA has had a difficult time adopting new methods. Today, the necessity and methods of testing are in question, resulting in our users and time lines paying the price. Testing is still just as important today as it was in 1970. Now is the time to reflect, learn from our past, and look for opportunities to change the future of quality through efficiency and innovation.

2. The Beginning

Software engineering first appeared in the late 1950s. Early on, applications were dependent upon the hardware they were created for, and were more designed than programed. In the late 1960s, hardware costs had fallen, but the cost of software development began to rise. Unfortunately many software projects did not scale, were over budget, late, or just plain failed. The “software crisis” arose out of poor performing applications. The idea of ensuring quality was born.

2.1 Software Development Life Cycle

The modern age of software testing began in the early 1970s. Dr. Winston Royce wrote, "Managing the Development of Large Software Systems", a paper highlighting his views on managing software development projects. The foundation of the waterfall model was based on this paper.

At first, almost all software projects used the waterfall model. Each phase of the model was a separate activity, requiring different teams to work on it. The phases had independent estimates and could not start until the application passed the previous phase.

Royce’s original waterfall phases included:

1. Requirements specification
2. Design
3. Construction
4. Integration
5. Testing and Debugging
6. Installation
7. Maintenance

There could be different phases, depending on the project. Typically testing did not start until the project reached the testing phase. At that time all the features of the application were fully specified, designed and programed. An official release candidate build was delivered to QA and testing began. The application would stay in this phase until testing was complete.

2.2 Testing Models

In their paper, “The Growth of Software Testing”, David Gelpering and Bill Hetzel classified software testing into different time periods. Each period had different goals and methods to define software quality.

Modern testing was built on a combination of these periods. We are now entering a new, not yet defined period.

2.2.1 Pre 1956 – Debugging oriented

The first software was dependent on the underlying hardware. The terms debugging and testing were interchangeable. Put simply, applications were written and then checked to ensure everything worked.

2.2.2 1957-1978 – Demonstration oriented

1957 was the first year that testing was distinguished from debugging by Charles Baker. Testing during this period was focused on problem-solving and demonstrating that the application behaved as expected.

2.2.3 1979-1982 – Destruction oriented

In his 1979 book, *The Art of Software Testing*, Glenford Myers defined testing as "...the process of executing an application with the intent in finding errors." In this period, the primary intent of testing was to cause fault in the application.

2.2.4 1983-1987 – Evaluation oriented

In 1983, the National Bureau of Standards published guidelines for federal information processing systems. The process integrated analysis, review and testing to evaluate the application.

2.2.5 1988-2001 – Prevention oriented

In 1985 a new methodology was written by the Institute of Electrical and Electronics Engineers (IEEE) unit, "Systematic Test and Evaluation Process" or STEP. It included test activities such as test planning, analysis, test design, and test plans with the goal of preventing defects.

2.3 Determining Quality

Manual test cases were—and are commonly still—the primary method in determining quality. Once product requirements were finalized and development began, QA started preparing test cases for the release. A handful of test case writers would crawl through all the product specifications. They would then create new tests that would validate new features and update any of the regression tests.

These tests were rolled into a test run that was used to validate the application. During the test phase, all the new feature tests cases, as well as the regression tests were executed. When all of the tests passed, the product was considered to have 'good quality'.

Sometimes requirements shifted during development for various reasons. This caused tests cases to become inaccurate. When executing the cases many testers would fail the case and file a defect. This created extra work in verifying the requirements and updating the test case. Another issue with executing tests was the expectation of following the steps verbatim. This limited the testers and resulted in missed bugs that were not on the specified testing path.

2.4 Waterfall Does Work

The waterfall model has been proven to work. It was the primary development method for over 30 years and many companies still depend on it.

If you work on projects with limited configurations, that are difficult to update (remember floppy disks?), or your industry requires checkpoints before moving into the next phase, then using a phased model like waterfall is the right approach. Some industries and companies will continue to use the waterfall model well into the future. Many software development projects today, however, do not fall into any of these scenarios. Waterfall is not always the right model.

3. The Evolution

We have entered into a new period in testing that is still being defined. Since 2001 the industry has begun shifting to new methods for development and different tools have appeared to help with testing.

3.1 Introducing Agile

The idea of incremental software releases can be traced back to the beginning of software development. Starting in the 1990s there were a few iterative methodologies proposed, such as eXtreme Programming (XP) and Scrum. In February 2001, 17 software developers met to discuss software development methods. They came up with the Agile Manifesto, which provided a method for a lightweight software development cycle to deliver software quicker than waterfall. Since then, multiple methodologies are now grouped under the agile method, including Scrum, Kanban, XP, and Lean.

These agile methods allow shorter delivery times and the ability to adjust to changing business demands. Developers and product managers have eagerly adopted this new approach to develop software. Instead of three, six, or 12-month turnaround times, agile cycles are typically two to four weeks. This shift to releasing more frequently means QA does not have the luxury of an independent test phase.

3.2 User Stories

Requirements no longer live in large unwieldy documents that are read once and forgotten. They are now captured in user stories, which are brief descriptions of a feature used to inform development. The story size is estimated to help the business understand the cost of the feature. Once the cost is estimated, the business can weigh the feature story against other stories to determine if it delivers appropriate return on investment.

The story should have acceptance criteria. It is usually captured in the story template, for example, "As a *role,* I want *goal* so that *benefit*" or as a “Done When.” User stories can be business-facing tests that address business requirements. Ideally each user story becomes an automated acceptance test, feeding into an automated regression suite that can run on a regular basis.

3.3 Exploding Configurations

Initially, most software testing was targeted to a specific hardware platform, for example, a mainframe computer. This limited the configuration matrix to a single axis. A tester only needed to test the application for the hardware platform it was designed to run on.

Then, in the 1990s, the home PC market exploded, with Microsoft-based operating systems leading the way. By the year 2000 there were five Windows lines; Windows 95, Windows 98, Windows 2000, Windows ME, and Windows NT. At that time Microsoft targeted their operating systems to the intended user; business or personal. Businesses mostly worked on Windows NT, with some migrating to Windows 2000. For personal PCs, Windows 95/98, both MSDOS based, were popular and Windows ME had a sliver of market share.

In 2001, Microsoft released Windows XP, with two flavors, Home and Pro. These operating systems were not MSDOS based and had slightly different targets. Pro was primarily focused for business use. Home

was clearly targeted for personal use. Pro did include advanced networking, which attracted some personal users for in-home networks.

The Windows Vista release in 2007 was the first mainstream release that included both 32-bit and 64-bit versions. In 2009, with the release of Windows 7, 64-bit installs became more commonplace. The last release, Windows 8 in 2012, included the revamped interface called Metro and changes to the operating system that impacted the behavior of many existing applications.

Along with operating systems, Internet Browsers were also evolving. In the early 2000s, Internet Explorer accounted for 85% of all browsers, with Netscape Classic rounding out the last 15% (and declining.) At that time, testing a web-based product meant there were only one or two configurations to consider. Most sites where optimized for the more popular browser, Internet Explorer 6. Internet Explorer 6 was so popular that it took over 10 years for its usage to significantly decline.

Around 2005, alternatives to Internet Explorer started to gain popularity. By the end of that year, Firefox and Safari had gained noticeable share, at 23.6% and 1.56% respectively. Google Chrome, released late in 2008, rapidly became a popular browser. That was about the time Microsoft was releasing newer versions of Internet Explorers including 7, 8, 9 and 10. While Firefox, Safari and Chrome would auto-update, Internet Explorer required a manual update, causing many users to stay on the browser that came pre-installed on their PC.

In 2007 the modern era of mobile platforms was introduced with the iPhone, followed by Android, Windows Mobile, and Blackberry. All the devices supported native applications and used mobile-based web browsers.

All of the change occurring in a span of less than 10 years has created complexity for testing. Whether you are testing a native application or web-based application, there are now multiple configurations to consider. Today, we have six desktop browsers with over 4% market share, multiple mobile browsers, and many shops still support a few of the outdated browsers (looking at you IE6 and IE7).

Web-based testing can easily require over 10 different browser combinations that need to be tested with each release. Application testing can have six or more combinations. The idea of doing a full regression across all of the supported configurations can significantly impact time and resources.

And we haven't even covered the explosion of locales and languages. Today we work in a global economy meaning many applications support multiple regions. Each supported region adds another configuration that requires validation.

3.4 Manual and Automated Test Cases

In the last several years, we have seen an increase in automation test frameworks. Not all testers know or want to learn programming so many of the frameworks offer record and playback functionality to create automated test scripts. Unfortunately record and playback automation is far from what the vendor promises. While the tests are easy to create and will probably work, they are not typically reusable and if anything changes, they will have to be re-recorded. Record and playback does offer some advantages particularly for quick short-term use testing and as a stepping stone for detailed scripting.

Not all testing is suitable for testing through automation. Some testing, such as exploratory and user acceptance, are better suited for manual testing. Other testing, including component testing, API testing or performance testing, are easier with automation or specialized tools. Knowing what is suitable for automation and what is better to test manually will help testers choose the right approach and resources to test efficiently and sufficiently. (Crispin and Gregory 2009)

Running a full manual regression test on a release gives the team confidence that everything is working as expected and any changes introduced did not adversely impact existing behavior. Full manual regressions, however, hamper the team's ability to release quickly or make changes during the testing

phase. When testing new functionality, those test cases may be added to the regression suite and over time the tests become unmanageable, timely to update and difficult to trace.

3.5 Quality Efficiency

Testing alone is not sufficient to ensure a quality product. Most individual testing methods, which include manual testing, unit testing and regression testing, have only a 35% effective rate at catching defects. If testing is the sole preventative measure for a large scale project, it will seldom top 80% of defect removal. A combination of quality methods, including pre-test methods such as static analysis for defect detection, can bring defect removal rates up to 95%. This is the difference between good quality and mediocre quality. The later you catch a bug in the process, the more costly it is to fix. It is expensive in both people hours and project time. (Jones, Subramanyam, and Bonsignour 2011)

With so much focus on testing, and the cost of catching a bug late in the process, we are inadvertently creating the dreaded QA bottleneck that slows the momentum of agile development. Even working in an agile environment with daily builds, manual regressions begin just before releasing to a production environment. If a defect is raised at that point, it may be in code written early on, be more complicated to fix, and have a negative impact on the release date and ultimately on the end user.

3.6 Typical Mistakes

3.6.1 Agile is not mini waterfall

Attempting to condense a waterfall workflow into a two-week sprint is not agile; it is a shorter waterfall. An example of this is having requirements ready before the sprint, developing the stories for the majority of the sprint, and delivering a build to QA for testing at the end of sprint. This leaves little time to meet the sprint commitment. With so much quality work packed into so little time, quality and efficiency suffer and necessary testing slips to the next sprint. This can lead to a backlog of features and defects. As a result, the final release is large and unmanageable, ultimately impacting the team's ability to release on time. (Crispin and Gregory 2009)

3.6.2 Lack of (any) documentation

The Agile Manifesto states: “Working software over comprehensive documentation.” Unfortunately this is usually interpreted, “Working software and no documentation.” A lack of documentation makes it difficult to verify functionality and reference changes. On small projects, personal interactions may be enough, but on larger agile projects the tickets are an integral part of interaction. Informative tickets will reduce confusion and time spent filling in missing details.

3.6.3 Quality ownership

Who is responsible for quality? Everybody.

I have worked on numerous projects at a number of companies where members of the team did not know how to use the software they were developing. Everyone should test. Testing provides insight into how the customers and end users see and interact with the finished product.

3.6.4 Loss of traceability or coverage

In an agile environment, comprehensive or exhaustive documentation should not be necessary and often does not exist. But, many organizations still rely on comprehensive and exhaustive test plans in a test case management tool, like Quality Center, to provide regression coverage. Manual testing is already taking too long; maintaining test plans only increases project cost.

Taking a paragraph-length user story and turning it into multiple, large test cases defeat the goal of working in agile to begin with. That time could be spent in other, more valuable areas, like actual product testing and creating test automation. Since automation tests are reusable, focusing on them instead of manual testing will pay for itself in the long run. (Crispin and Gregory 2009)

Since the software world has started the shift to agile, many new tools now exist to help track the new workflows. Unfortunately, to date, there has not been a revolution for creating a traceability matrix or test cases. Nearly all requirement coverage and test case tools are still based on traditional waterfall methods to manage quality.

4. The Approaching Stage

Quality has come a long way over the last 10 years. Shorter development cycles are proven to return higher quality in comparison to longer-running waterfall projects. The previous transitional period, however, has left resource, process, and documentation gaps that prevent teams from delivering high-quality projects within an agile timeline.

4.1 Determine Coverage

In an agile environment, software change occurs on a consistent basis. Compounded with a reduced tester-to-developer ratio, there are more people introducing change and less people validating it. In this scenario, it is virtually impossible for teams to successfully execute all of the changes introduced into an application.

4.1.1 Traceability matrix

To move forward, we need a more robust traceability matrix to determine coverage. It needs to be lightweight, integrated into ticketing systems, and create and track automated and manual test runs. The matrix could align to user stories, incorporating newly defined capabilities and the existing capabilities of the application.

Google practices the ACC model for rapid test planning and requirements tracking.

Attributes: What types of behavior the application has. Examples are Fast or Secure.

Component: The different components of the systems.

Capability: Intersect of attributes and components, and lists the expected functionality in that intersection. With each capability there is an assigned risk factor that will determine the priority of testing.

With the list of capabilities combining attributes and components, we can tie automated and manual testing to cover each individual capability. We can identify the high-risk, high-value testing areas to prioritize testing and automation efforts. The capability can be validated through different unit testing, automated testing, or through a scenario test case. (Whittaker 2012)

ACC is only one example of a process for identifying capabilities of applications. The ability to tie tests back to an individual capability allows the tester to calculate test coverage. By tracking the risk, testing high risk areas first, we can prioritize and reduce risk in each release. Parsing the application into consumable test focus areas (the capabilities) means we can create tests that validate each capability.

4.1.2 Manual test cases

In a perfect world, all regression tests would be automated. We don't live in a perfect world. That means keeping manual tests cases light reduces the need for constant updating and duplicate testing. If the tests are too detailed, testers will spend more time writing and maintaining the test cases than actually testing the application. Step by step test cases also typically keep testers on a specific path, limiting exploratory testing.

There are tests scenarios in which automation is not ideal, for example, usability testing. Part of usability testing is looking at the page across supported browsers. Having a lightweight test case will result in quicker execution and allow the tester more time for exploratory testing. With the multiple configurations that need to be tested, creating an execution matrix will give traceability and insight to what was tested.

4.1.3 Cross-functional teams

When building a cross-functional team you look for specialized skills. There may be a product owner, a UX designer, front-end developers and back-end developers. Test engineers bring valuable skills and a quality mindset to the table. They will watch builds, run automation, provide test environments, and work closely with each member of the team; always in the mindset of making the best product possible.

Requirements can introduce 20% of all the software defects. (Jones, Subramanyam, and Bonsignour 2011) Making assumptions about how the software works, or not having fully considered all of the possible scenarios, will create unwanted design defects that are usually discovered during testing. Test engineers tend to be more familiar with the product, or type of product, than anyone else on the team. Having them present during design and planning will help expose design defects before they are implemented. This will help the team avoid unnecessary rework efforts.

4.1.4 Quality at all times

Working in agile, continuous integration can mean the difference between having a great product or having no product. A continuous integration server needs to build every time there is code committed into the repository. With each commit of the code it needs to be scanned for language guidelines (linted) and have static analysis performed to detect potential coding issues. Faster running tests, like unit tests, should be executed with every build. Slower tests, like automated regressions, may run once a day during off hours, to be reviewed first thing the next morning.

Since every code change is built and deployed on a regular cadence, test engineers will monitor the builds and automation, providing immediate feedback to developers. Many regressions defects are found less than 24 hours after the offending code is introduced. When a defect is discovered, the team should create a test to verify it and prevent it from happening again. Ideally this will be as close to the actual cause of the defect as possible, such as a unit test of a component or function. (Jones, Subramanyam, and Bonsignour 2011)

Reducing the feedback loop will allow issues to be fixed faster. The status of the build and testing should be made public, and a passing build needs to be the top priority of the team. Having the application in an always-releasable state allows you to release anytime, without waiting for regressions to run, and gives you confidence that the application will work as expected.

4.1.5 Maintainable/reusable automation

Having a robust and easily-maintained system is vital to working in an agile environment. Automation should always pass, unless there is a defect. Time spent digging through logs or manually checking failed tests negates the value of automation. There should be logic in place to address instability that is common with certain types of testing. That might include adding hooks on the web page to signify loading, or running a two-out-of-three pass/fail test.

Far too much cross-browser testing is done manually. Tools like webdriver can run across all browsers, but they only test functionality. This means the web page might render incorrectly and automation will not discover any issues. A page that does not render correctly is a serious issue. The ability to test for this through automation would allow greater coverage in less time. Comparing the layout of browsers can be integrated into the automation with image-compare programs like Sikuli and ImageMagick. There are issues with this approach, including maintaining the browser-rendered images, but it can be useful to validate page layouts quickly.

4.2 Skills in Demand

4.2.1 Open source

If you ever work in a start-up or with cutting edge technology, you will most likely be working in Open Source. The hottest automation technologies (Cucumber, Watir, Selenium, Robot Framework, Sikuli) are all Open Source projects. They may take a little longer to incorporate, but their capabilities usually surpass that of expensive alternatives.

4.2.2 SQL

Companies today are investing in data mining, data gathering, and other metrics to inform and make key business decisions. The need to validate raw data is increasing. For the next generation of QA, the ability to read and create SQL queries is vital.

4.2.3 Object-Oriented (OO) languages

Record and playback automation, and basic scripting automation are not sufficient for complicated, modern applications. Full-fledged programing languages can be used to make automation reusable and maintainable, and should be Object Oriented. Common OO programming languages used for automation are Ruby, Java, and Python. Testers should seek out opportunities to learn and use these languages. There are excellent online resources as well as traditional college courses.

4.2.4 Unit test writing

The closer the tests can live to the actual code the better. Unit tests are able to provide coverage, testing a small portion of the code, and should be implemented with every code change. A tester with unit test writing can provide additional coverage while learning more about how the application works.

4.2.5 Web technologies

It is important to have an understanding of all the pieces required by a web application. Knowing the difference between a JavaSript error and a CSS error means faster troubleshooting, well-written tests, and the ability to provide accurate details for defects.

4.2.6 Virtualization

The QA environment used to include physical hardware supporting all of the configurations needed to test. In today's QA, with many configurations required to test, the only sensible solution is virtualization. This cuts hardware costs, enables environments to change quickly and allows multiple environments to run side by side. There are many different virtualization solutions, in varying price ranges, with local or cloud accessibility. For popular platforms, Microsoft offers test virtual machines for its browsers. (http://www.modern.ie)

4.2.7 Mobile

Mobile devices are commonplace today. This year, it is expected that Internet connected mobile devices will surpass the number of connected desktops and laptops. (Standage) From native applications to mobile-specific web pages, mobile is an important segment that cannot be ignored. The tool Appium (http://appium.io), based off of webdriver, recently emerged for automated mobile application testing.

4.2.8 Social media

Everyone is on it and everyone wants a piece of it. People working on a given project should be familiar with applicable social media sites, understand the difference between the sites, and have a basic understanding of how the sites work. Many websites now incorporate social logins and social sharing, both of which require testing.

5. The Next Generation

We have witnessed a significant transformation over the last 10 years and there is much more to come in the next 10. The industry is constantly changing, with new methods and tools. Looking forward, there are a few changes I feel confident we will see.

5.1 The testers

The ability to design and build reusable automation in a continuous integration environment allows companies to deliver software faster. The demand for purely black-box, manual testers will continue to decrease, replaced by a new, hybrid tester who can implement automation, has programming know-how, and is cognizant of the end-user experience. With the typical agile team size of five to eight, the hybrid tester will also need critical thinking and leadership capabilities to advocate for the quality of the application.

5.2 The leadership

As companies move towards cross-functional teams, QA will become more embedded in the development team. As that happens, the need for a dedicated QA team will slowly decrease and eventually cease to exist. This will reduce the need for QA managers, placing the onus on individual QA leads to represent and promote testing as part of the development team. Larger organizations, with QA testers spread over multiple development teams, may implement a QA Center of Excellence (COE) to span all of QA, creating efficiencies, training and best practices. The QA COE will be responsible for the overall QA strategy and automation tool sets.

5.3 The tools

Web automation is becoming robust, with the ability to provide excellent coverage. All major browsers now support webdriver making it easy to write cross-browser automation. Verifying web page layout across browsers will eventually be simplified through unification of the rendering engines, better web standards and better comparison tools. As we hone and innovate in this area, I foresee the next phase of web automation as self-testing web pages. Some of the testing could be built in, like link checking, or if an image appear, which would not require any test setup. More complicated tests may become test code, embedded on the web page or in a separate file, which the browser can execute to test and provide results on the spot. Each code change will be instantly tested with potential issues known immediately. This will allow for rapid defect detection and resolution resulting in faster development.

6. Conclusion

Software development is an ever-changing field. The constant introduction of new technologies is the new norm. In an industry that is constantly evolving, the test engineer must evolve with it or fade away. Many start-ups today forgo hiring testers, relying (hopefully) on other methods to ensure quality. Many of these organizations are beginning to realize testing is a unique and valuable skill set. The biggest challenge testers face today is the expectation of high-quality output in an relatively short period of time. In the agile environment, manual testing alone is not a viable option.

Over the past 40+ years software quality methods have changed. Shorter development cycles, methods on discovering defects, and automation have reshaped how testers work. Unfortunately we are still holding on to some legacy methods that aren't working, or aren't working well. By doing so it holds back innovations for quality. We are entering a new period for software quality. Some of the old ways and tools used in testing no longer apply. I believe it is important for us to understand our roots, learn from past experiences, and find new ways to move forward. Software teams, including test engineers, need to redefine how they determine quality and create new methods to help them achieve it.

7. References

Savoia, Alberto and James Whittaker. "Test is Dead" Recorded October 26 2011. Google Test Automation Conference. Web, http://www.youtube.com/watch?v=X1jWe5rOu3g.

Royce, Winston. "Managing the Development of Large Software Systems" *Technical Papers of Western Electronic Show and Convention* (WesCon) August 25-28, 1970, Los Angeles, USA. http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf (accessed August 3, 2013).

Gelpering, David, and Bill Hetzel. "The Growth of Software Testing." *Communications of the ACM*. no. June (1988): 687-695. http://clearspecs.com/joomla15/downloads/ClearSpecs16V01_GrowthOfSoftwareTest.pdf (accessed August 3, 2013).

Jones, Capers, and Olivier Bonsignour. *The economics of software quality*. Upper Saddle River, NJ: Addison-Wesley, 2011.

Crispin, Lisa, and Janet Gregory. *Agile testing : a practical guide for testers and agile teams*. Upper Saddle River, NJ: Addison-Wesley, 2009.

Whittaker, James A., Jason Arbon, and Jeff Carollo. *How Google tests software*. Upper Saddle River, NJ: Addison-Wesley, 2012.

Standage, Tom. "Business: Live and unplugged | The Economist." *The Economist - World News, Politics, Economics, Business & Finance*. N.p., n.d. Web. 10 Aug. 2013. <http://www.economist.com/news/21566417-2013-internet-will-become-mostly-mobile-medium-who-will-be-winners-and-losers-live-and>.

"Browser Statistics." *W3Schools Online Web Tutorials*. N.p., n.d. Web. 10 Aug. 2013. <http://www.w3schools.com/browsers/browsers_stats.asp>.