Quality Engineering For DevOps Customers

Dwayne Thomas

thomas.dwayne@gmail.com

Abstract

The TV software company (the company) of this experience report creates interactive software that overlay TV program feeds. The company's DevOps professionals mainly perform software deployment activities and IT operations. (The combination of these two activities form the acronym DevOps.) DevOps help to more frequently scale software solutions to the business market. DevOps professionals also perform production testing and scale interactive software experiences to TV viewers. Testers at the company recently started supporting the company DevOps team more actively, since the DevOps team now consumes the most of the products of testing and development activities. The company quality engineering (testing) team is faced with a question: how do testers improve software quality for DevOps customers?

Recently, many learning opportunities for DevOps professionals have arisen. Due to the 24/7 nature of the TV industry, the company DevOps professionals are on call to monitor the software deployments. The question of how to improve software quality engineering for DevOps customers is fertile for exploration because the DevOps role has only formally existed for 5 years (Wikipedia, 2013). As Gene Kim (2014) says, "DevOps is more like a philosophical movement, and not yet a precise collection of practices, descriptive or prescriptive." Because it has existed informally for 10 years other experienced professionals in the software world assert that there are rich practices to be mined to support the DevOps role (Kowolowski, May 7 2014).

This paper provides relevant test processes and tools for quality engineering for DevOps customers. This paper analyzes author's own most fruitful quality engineering strategies and those of his colleagues. The company pivoted from marketing software solutions offering the software tools to DevOps personnel. Case studies of testing practices of a few celebrated software development companies serve as reference for this paper. Oregon's vibrant community meetups provide supplementary information. This paper is framed to help agile software testers to support software testing in a DevOps focused software development environments.

Biography

Dwayne was a Senior Quality Engineer for Cloud Services and IP devices of Ensequence, a TV engagement solution company(2013-2014). Dwayne whet his appetite to write about quality engineering in his UO Applied Information Management capstone paper titled "The Role of Testers in an Agile Software Development Life Cycle within B2B Companies." Dwayne previously taught middle school math for several years and was a tax/engineering analyst. Dwayne enjoys combining software disciplines to solve software quality challenges. Currently, Dwayne is quality engineer for CrowdCompass by Cvent.

1. Introduction

The TV software company (the company) of this report offers a solutions-based approach to television program providers. The company DevOps professionals maintain the software and hardware infrastructure for the enhanced TV viewer experience. The company has a deep understanding of software development lifecycles. The approaches offered in this paper offer the most benefits when combined with transparent continuous improvement practices. Organization-wide, the company developed production releases which ensure the DevOps resources are protected and improved. The company leaders supported many of these practices that complemented existing approaches. This paper's sections include redefining testing environment; expanding testing roles; software tools selection processes; lessons learned; industry trends.

Testers observed the DevOps workflow early and often at the company. DevOps routed their operation issues through the testing team as well, which focused on documenting the issues, tracking them, and testing the development team fixes. Most of the tools natural to testers described in this paper are used in conjunction with our established browser-based functional testing tools, such as Google's Chrome development tools. The development team helped execute many of the initiatives mentioned in this paper. Developers and testers were keen to document their procedures in order to maximize the efficiency of new processes. Developers provided testers secure shell (SSH) permissions for servers and git code repositories. Gitlab ™ enabled testers to contribute to development discussions on software code discussions.

Testers started testing each development effort in order to validate it as shown in the diagram below, rather than individual pieces of software produced. The company more broadly defined the Microsoft (2014) identified development effort to include working software, application program interfaces, hardware, and database configurations, see Figure 1. The company also defined stakeholders to include internal and external customers.

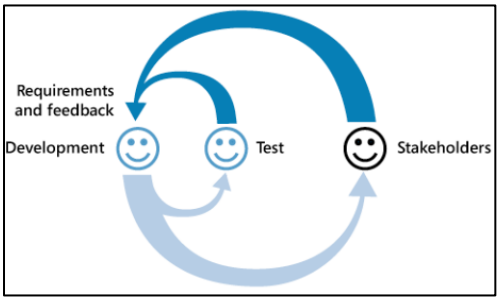

Figure 1. Quality verification of development effort

2. Redefining Testing Environment

Because scaling the company software environment configurations was often rocky, testers corralled the testing environment as an important factor for the development process.

2.1. Deployment And Monitoring the Environment

Testers and development agreed on a testing infrastructure similar to the company production (DevOps controlled) environment. Testers used Rackspace™ (the cloud server provider) tools to deploy new testing servers, databases, and load balancers. Modeling production servers required understanding which services and databases should be paired. Testers used the command-line bash programming language and Unix servers to deploy new branches of software code into servers, mimicking the DevOps professionals setup. Testers support DevOps professionals monitoring tools for detecting and fixing hardware issues. Solarwinds™ complements our internal monitoring tools and is a model for our own internal monitoring software. The company investigated how to test the integrated system of software

devices. Testers mocked more of the infrastructure in many-to-one system tests. After testing cycles the software code and environment variables were both migrated to the company a staging environment. Testers and development provide more robust documentation and intuitive software for the DevOps team but still are ready with answers to DevOps questions.

2.2. Testing The Larger Environment

Deploying the larger testing environment meant longer test cycles in the short run, as more components were vetted in more combinations. A software ecosystem is made up of software solutions that facilitate the activities in associated social or business (Bosch, 2009). Testers prepared for longer test cycles by carving out software project time for integrating all the products and communicated this request to organization stakeholders. But experiencing the integration cycle still unsettled the testing team as a whole. Some testers shone with their debugging and application programming interface (API) skills. Fiddler an API desktop trafficking tool helped to understand the data pipelines of the software stack. Still, the new infrastructure debugging brought the Postman browser tool to the forefront as a more user-friendly interface. Testers of various experience levels with API testing helped isolate development issues. The API testing also helped DevOps workarounds before the graphical user interface (GUI) software was completely stable.

3. Expanding Testing Roles

Testers sometimes performed a small but significant series of responsibilities that are typically reserved for technology specialists in other software development life cycles: primarily in technical writing review, peer code review, database administration, and software configuration.

3.1. Development Documentation Verification

Testers reviewed existing development documentation and used it to complete deployments. API documentation helped testers identify development issues more specifically and quickly. By a series of comparisons, testers identified when API procedures and documentation were not consistent. For example, when developers inadvertently changed Javascript Object Notation (JSON) objects structures, testers suggested restoring the expected state of the objects. Even in cases where development documentation was robust, testing discoveries helped prompt development to deliver more mature graphic user interface tools.

3.2. Code Level Understanding

Testing the company technology infrastructure required in depth knowledge of it and—likely—its alternative systems. The development team encouraged testers to understand and to peer review unit tests. Unit tests are building blocks for assuring pieces of business knowledge were always available. Unit tests also delivered snapshots of the pieces of software function to testers, helping to form mental models of the technology infrastructure. Quality engineers used their access to Git (code version management software) to understand the specific nature of every code commit. It also helped suggest when modules might be especially susceptible to bugs. Tracking down buggy areas of the software was occasionally as simple as finding unreadable javascript code. One javascript comparison of 3 values was more complicated than it needed to be and provided unreliable software. (The comparison should have read like the mathematical arrangement of three variables x<y<z.)

3.3. Database Observations

The testing team used database administration access to experiment in short cycles in development - shorter cycles than permissible in production settings. Therefore it was important for testers to configure the databases. For example, DevOps professionals collected relational table formatted database production metrics on 15-minute intervals using the MySQL Workbench ™ tool. However, testers mainly shortened those intervals to 1 minute in testing using Mongo database tool. Testers also imported

production data into the testing databases to get snapshots of television viewing data. Database observations helped detect whether representational state transfer (REST) methods of data were successful. Mongo™ Nosql (nonrelational document formatted) database tools helped testers get authorization for the user information quickly. In one instance the database should have helped testers detect when a service was out of date. Database observations revealed that a datapoint that should have been reported in the software disappeared due to the wrong time stamp. Understanding differences between relational database unique identifiers (longer integers) and non-relational identifiers again helped speed the fixing of a few development issues.

3.4. Load Testing

The company considered launching its product suite during some of the most celebrated live TV events. Such events might have included the Grammy's. The company executives needed load testing proof that the TV application infrastructure was scalable in order to confirm business contracts with external customers. The load testing initiative helped justify that the company offerings could withstand large increases in the requests of TV viewers. Amazon web services supports the "Bees with machine guns" approach to load testing their own servers. Netflix ™ offers the "Simian Army" suite of tests for maintaining the availability and reliability of their cloud services. Testing also helped establish the financial cost to the company in the case of spikes in the use of the application.

3.5. Verification of Third Party Services

The company interactive experience leverages common third party services such as Facebook ™ and Twitter. As these services created their own ecosystems testers inadvertently grew to understand them better. On occasion when a service did not run for testers, it was due to the social networks changing their authentication protocols. This realization helps DevOps and development anticipate shortcomings in the social services, which should shorten the fix cycle on the production environment. Twilio ™, a telephone messaging, third party service offers extensive documentation of their offerings online. When the company monitoring tools did not report on text failures to landlines, testers used the Twilio documentation as the basis of feature requests to developers. The company projects also harness open-source software packages such Node Package Manager (NPM). Software configuration managers might be the ones to police compatible versions of software, however testers were sometimes on the front lines of compatibility issues. NPM made unexpected production changes to their packages that the company testers helped shield from the company technology architecture.

4. Software Tools

Testers added to their browser-based testing toolboxes throughout this quality engineering effort. The main testing considerations for selecting tools were usability and development support. For example testers preferred Postman to Fiddler, because Postman has a more intuitive user interface.

4.1. Development Support

More often than not, other colleagues at the company were already using the tools to some degree but testers incorporated them into testing workflow. The DevOps team needed Microsoft Windows ™ desktop computers for maintaining some legacy hardware of business customers. The development team needed to work on Mac computers. The computer operating systems determine the testing tools that are supported. Testers at the company use both of these two operating systems. The Gitbash tool helps windows machines SSH into Unix servers. Computer sharing software Real VCN ™ help testers access their Mac computers remotely from their windows ones. Testers might have wanted to use Microsoft Word for all their documentation but the hypertext markup language Markdown was most expedient for both the development and testing teams. Testers were versed in the Python language for their own efforts but read Javascript in order to understand development code. Foundational understandings of relational

(MySQL) and non-relational databases (Mongo™) helped testers reconcile the databases in the testing environment.

4.2. Testing-Specific Coding Tools

Testers already maintained the use of two languages in their own workflow. Python automation scripts help setup the testing environment. Object oriented programming also help make the automation scripts easy to maintain. The Python scripts facilitated data imports into the television software authoring tools. Windows batch scripts facilitated deploying the new software builds and making limited choices about the environment. The scripts also enabled testers with various levels of experience to support the deployment activities. Traditionally batch files take more time to understand because scripts more directly communicate with computers. These tools helped testers made expand the testing toolset.

5. Software Industry Advanced Test Practices

The software industry leaders Linked-in, Facebook, and Google suggest practices to help the company testers to deploy higher quality software even faster.

5.1. Linked-in

Linked-in's test strategies are quite mature and have a diagram, see Figure 2 (Mohapatra, 2014). While the company makes contextual decisions, Linked-in has a strategy diagram that helps determine which products are shipped. The TV software solutions company would benefit from test plans that reliably identify test strategies for release. Further, continuous integration automation tests would likely improve the quality of our software. The company built an automation framework using Selenium and Google Chrome but needs to insist on that framework as part of its integration processes. The tests should be tracked on a dashboard and compared to production issues to make sure that the tests are reliable and robust.

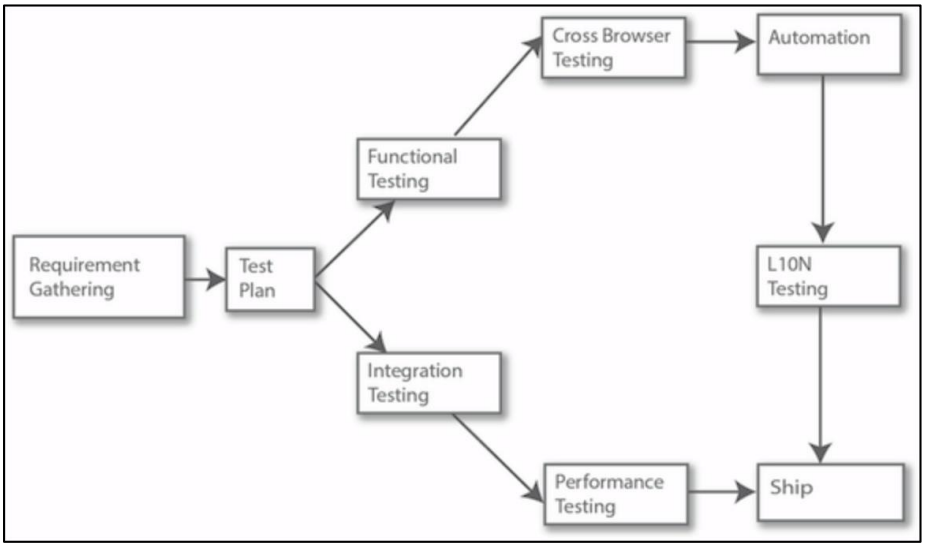

Figure 2: Linked-in test strategy diagram.

5.2. Facebook

The company could apply Facebook's small random sample of testing on its television viewers (Boz, 2014). The TV software solutions company treats its published advertisements as the most effective ones, when the software solutions company could be learning from television viewers to increase the interaction rate of its advertisement assets. External tests can reduce some of the internal debates among the development team and increase the investment return of the products.

5.3. Google

Google has specialized its code-based testing practices among testers into two roles (Whitaker, et al 2012). The first role focuses on programming practices throughout the organization. The second role focuses on testing as a feature, which impacts users and the software product. Additionally, the second role writes production code that focuses on increasing the verification of external inputs. By including quality features in the product itself, the features become more resilient. Test engineers might lead exploratory testing sessions or manage a beta testing effort. They might be focused on programming user scenarios. The TV software solutions company might need to similarly define its testing roles in order to make more scalable software. The company might also need more customized testing tools similar to ones that Google created.

6. Lessons Learned

Regular preparation and verification of all development changes improved the product development flow to one that facilitated the company success and an enhanced television viewer experience. At the time of this paper the company had passed several market trials and had amassed one of the largest footprint of interactive TV advertising in the U.S.

6.1. Bridges To Higher Quality Software

Full stack technology testing decreases risks to internal DevOps and external customers. Many of the consequences of this additional testing were not predicted, but all were beneficial to the company's software quality goals. The products constantly received helpful feedback on its quality from testers. The supporting software tools were readily available to testers for download. Documentation of the software is available online. Software courses by MongoDB and the javascript course on Code School helped fill some information gaps as well. Software quality success shortens deployment cycles and helps increase external demand for the company software solutions. Yet there are also quality of life benefits to quality software for DevOps professionals who experience more predictable deployments and operations. The benefits to testers include increased learning and collaboration (Novak, June 2014).

6.2. Testing New And Legacy Technologies

Testing engineers were not thorough about integrating new and old technology infrastructure. Specifically, database compatibility between Mysql and Nosql integrations were blindspots. Test engineers did not mock the real usage of the software solution. DevOps professionals were wary that test engineers only tested the latest versions of the services. In cases like this one, testers relied on development to more explicitly support the DevOps team.

6.3. Areas For Further Exploration: Security Testing

The Open SSL heartbleed virus hamstrung much of the software community earlier this year (Wikipedia, July 2014). Testing precautions might have helped with this condition. SQL injection attacks are popular among testers. However, XML injection attacks are also feasible and need to be more thoroughly vetted by testers (Owasp.org, July 2014). Increasingly, the company will hold more personal customer information and will need to be sure that passwords or other confidential information are always guarded.

7. Conclusion

The TV software solutions testers found that DevOps-focused product development benefit internal customers and believed that these same focused development practices would also benefit external customers. Prior to important product releases, testers reduced the DevOps questions from a few dozen to several focused readily answered ones. The testing initiative in this paper took place on technology stacks with various histories. Some of the software and hardware tested had existed for months before they were incorporated into the revamped testing process. On the other hand, many of the assets tested were brand new to the company product suite. Implementing targeted testing and verification achieved more software quality goals sooner for development, DevOps teams, and external customers.

References

Bosch, Jan. August 2009. From Software Product Lines to Software Ecosystems. Accepted for SPLC 2009 (13th International Software Product Line Conference)

Boz. August 10, 2014. Building and testing at Facebook. https://www.facebook.com/notes/facebook-engineering/building-and-testing-at-facebook/10151004157328920

Gene Kim. DevOps Cookbook. June 2014. http://www.realgenekim.me/DevOps-cookbook/

Gene Kim. April 21, 2014. DevOps Patterns Distilled - Implementing The Needed Practices In Four Practical Steps. Meetup: FutureTalk with Gene Kim.

Louis Kowolowski. May 7, 2014. "What is DevOps and How do I become one?" Meetup: Hack the People.

Microsoft Development Network. August 10, 2014. Testing in the Software Lifecycle. http://msdn.microsoft.com/en-us/library/jj159342.aspx

Mohapatra Sony. August 10, 2014. Linkedin's Testing Methodology. http://engineering.linkedin.com/testing/quality-control-linkedins-testing-methodology

New Relic. June 2014. The Benefits of DevOps: What's in It for You? http://blog.newrelic.com/2014/06/06/DevOpsbenefits/

Owasp.org. July 29, 2014. Testing for XML Injection (OWASP-DV-008) https://www.owasp.org/index.php/Testing_for_XML_Injection_%28OWASP-DV-008%29

The Top 11 Things You Need To Know About DevOps. Gene Kim. IT Revolution Press. June 2014. http://www.thinkhdi.com/~/media/HDICorp/Files/White-Papers/whtppr-1112-DevOps-kim.pdf

James Whitaker, Jason Arbon, and Jeff Carollo. 2012. How Google Tests Software. http://it-ebooks.info/read/1200/

Wikipedia. July 29, 2014. Heart Bleed. http://en.wikipedia.org/wiki/Heartbleed