White Paper

A Broadcom Company

DevOps and Testers

DevOps is part of an overall approach that organizations use to deliver software frequently and with high quality. The most obvious outcome of successful DevOps implementations is the reduction in the time it takes for software changes to transition from an idea to production.

What Does DevOps Mean for Testers

Background

The DevOps movement (for want of a better label) is progressing rapidly. Like many other movements in the industry, the speed of adoption accelerates faster than the definition of the movement itself. DevOps is still not well defined and the nuances of culture, the emergent capability of new technologies, and range of (mostly successful) case studies means that the issues at hand are still widely debated.[1]

Depending on who you talk to, DevOps can be a solution to a problem or a goal in itself. In some businesses, the goal is going digital, and DevOps is part of the overall approach to delivering frequently and with high quality. But in the marketing of DevOps-related technologies and services, this goal can be obscured. The challenge of making the cultural change (or more concretely, making the behavioral change) required for success is frequently underestimated.

Often, the testers involved in and affected by DevOps are new to the whole idea. This article is an introduction to DevOps for these testers and is a discussion of its impact on test practices.

If you are an experienced DevOps practitioner, we hope that you might still find the article useful. If you are not a tester, we hope that you will at least see the tester's perspective.

What Is DevOps?

Simplistically, DevOps is a label to describe an ecosystem in which development teams and systems operations teams work more closely together. In the so-called delivery pipeline, from committing source code to putting code into production, developers accommodate and automate some of operations activities. Operations has more visibility into, and some influence over, the activities of developers. The motivation for this is primarily to speed up the deployment and implementation of software. Bringing Dev and Ops closer together in an agile team effectively implements agile operations.

The most obvious outcome of successful DevOps implementations is the reduction in the time it takes for software changes to transition from an idea to production. When a developer says a software change is done, the transition to production is performed with the aid of pervasive automation.

Automated tools and processes are used in system configuration, the build process, testing, the deployment to test, staging and production environments, post-deployment monitoring, evaluation, and operations.

Is DevOps Just About Tools?

At one level, the goal of DevOps is to eliminate bottlenecks in the delivery pipeline through automation. But the automation of staged processes still requires governance. Most automated processes are not really autonomous; they cannot complete their tasks without human intervention in maintenance or in handling exceptions. A fully automated DevOps process is meaningless without consideration of the human factor. Although tools do a lot of heavy-lifting, it is the people running the process that make it work or fail.

Is DevOps Just Dev and Ops Working More Closely with the Aid of Tools?

No, DevOps is not that either.

The hand-offs between automated processes often involve other processes, usually testing of one kind or another.

1 "What is DevOps," The Agile Admin, http://theagileadmin.com/what-is-devops/

It might be painful for you, as a tester, to trust developers and automation to do the testing job properly. If it hurts, you must do it more often.

Is DevOps Just About Tools? (cont.)

Automated tests are created by developers and testers. The output of these tests is focused on providing sufficient information to other processes, or just as often to people, to make transitions between stages in the pipeline.

Testers and developers who test provide the assurance that the DevOps process delivers successfully and reliably.

What is DevOps, Really?

The question is posed and discussed at length in an excellent post on the *Agile Admin*, "What is DevOps"[2]. DevOps is an evolving, emergent discipline. So the definition of DevOps is still not settled. Perhaps it never will be.

What does that mean for testers? It means that there is still no *one* true way and that your role in a DevOps regime that is evolving (and every regime is evolving) is not yet fixed. There are two main contributions you can make:

- You need to pay attention to the things that hurt and work to make them less painful.
- You need to identify the opportunities and interventions that will add value to the DevOps process.

2 "What is DevOps," The Agile Admin, http://theagileadmin.com/what-is-devops/

If there is one mantra that best describes the drive towards DevOps it is, "If it hurts, do it more." It might be a bit of a cliché, but use that as the context for implementing and improving DevOps test practices.

If It Hurts, Do It More

The difficulty or pain we experience when doing a particular job influences us adversely. If we do not like to do a task, we tend to put it off. When we finally take the task on, it is more painful. This is true for visiting the dentist, cleaning the garage, integrating software, doing testing, and so on.

Our experience is commonly that the less frequently we perform a task, the more traumatic the task is when we actually do it. Martin Fowler suggests three reasons why the frequent or even continuous execution of certain tasks reduces the pain.[3]

- First, larger, more complex tasks are hard to plan, manage, and control. Breaking up large tasks makes them easier to do, less risky to do and, if something does go wrong, easier to unwind.
- Second, many tasks (and testing is the shining example) provide feedback. Feedback, if received early and often, helps us to rapidly address problems before any further time is wasted.
- Thirdly, if we practice an activity more frequently, we get better at it. We learn how to do it efficiently. We might also see opportunities to automate it in some way.

From the tester's perspective, the mantra, "It it hurts, do it more," forces us to take the notion of automation in the testing process much more seriously.

3 "Frequency Reduces Difficulty," Martin Fowler, http://martinfowler.com/bliki/FrequencyReducesDifficulty.html

If there are manual interventions, typically between the automated stages in the DevOps process, we will see these as the pain points. They are bottlenecks, the causes of delays, and the potentially less reliable and error-prone aspects of the process.

Manual testing is painful. Yes, you might love exploratory testing; and you might fear that only you, as a human, can find those gnarly bugs that automation will never find, that you as the tester are the only person trustworthy enough to prevent disaster happening.

It might be painful for you, as a tester, to trust developers and automation to do the testing job properly. If it hurts, you must do it more often.

Tests, Automation, and Trust

There is much debate around the meaning of checking and testing[4], and the reliance we can place on testers, on checks, and on automation[5, 6].

Can we place all our faith in automated checks?

We certainly need more sophistication than that. But we can, for the purpose of this article, at least separate tests and test execution activity into four components.

- Checks that can be automated by developers as part of their component-level check-in and continuous integration processes.
- Checks that can be automated (typically by system testers) to exercise API-level, link, or end-to-end transactions.

4 "Testing and Checking Refined," James Bach, Michael Bolton, http://www.satisfice.com/blog/archives/856
5 "A New Model for Testing," Paul Gerrard, http://dev.sp.qa/download/newModel
6 "The New Model and Testing v Checking," Paul Gerrard, http://blog.gerrardconsulting.com/?q=node/659

Tests, Automation, and Trust (cont.)

- Tests that can perform compatibility checks to demonstrate compatibility across browsers, operating systems, and platforms.
- Tests that can only be performed by a human.

In this article, we offer only a few suggestions on how to make these distinctions because every environment is different, of course.

The more germane question to ask is, how does the tester let go of late, manual checking?[7] Letting go requires proactive effort and trust.

These areas will be the main focus of your efforts:

- Wherever possible, manual checks that could be performed at a component level should be pushed forward to the developers. As a tester, you might suggest these tests in a pairing or whiteboard session. You might have to write these tests yourself and include them in the continuous integration regime.
- End-to-end or user interface tests might require automation. These tests need to be minimized, as they tend to be slow to run, brittle, and frequently require maintenance. Consider whether they need to be run at every code check-in or if they could be reserved for use on larger, less frequent releases only.
- What manual-only tests could be run on components that are not yet integrated into a release candidate? Can the manual testing be performed in pairing sessions with developers? Are there alternatives to this testing? Could story-boarding or BDD-style prototyping help? Could UI checks be performed on mock-ups or wire-frames?
- Which checks need only be run once, manually, as opposed to checks that need to be retained for regression purposes, and are candidates for automation?
- We mentioned the notion of trust previously. Another way of looking at this is to speculate on how a system could be reliably tested if there were no late manual testing at all.
- Imagine an environment where all of the testing was done by tools. Would your concerns be dominated by the fact that you simply do not trust the developers to do a good job of testing? Moving testing thinking to the left should reduce the doubt.
- If, as a tester, you act more as a pathfinder to identify risks and assess them, to select tests and to ensure that they are incorporated into the development and automation, your concerns could be minimized.

Certainly, you have to stop believing you are the gatekeeper of quality, the last defense, the only person who cares. You have to think more like a visionary, risk-identifier, risk manager, pathfinder, facilitator, and as a coach and mentor.

Practice, Monitoring, and Improvement

With all of our good intentions in reducing or eliminating reliance on late manual checking, bugs will still get through. When software is released into production, problems arise.

One of the key disciplines of DevOps from the operations point of view is a deeper level of monitoring. Monitoring happens at every layer, from components and simple transactions in the applications, through integration and messaging and the infrastructure itself.

Certainly, you have to stop believing you are the gatekeeper of quality, the last defense, the only person who cares. You have to think more like a visionary, risk-identifier, risk manager, pathfinder, facilitator, and as a coach and mentor.

One goal of monitoring is to raise alerts on failure before users experience the impact of them. This is rather ambitious, but this is the ultimate goal.

When problems are encountered in production, the task then is to use the analytics derived from monitoring to not only trace the cause and resolve it, but also to refine the test process, automated or manual, to reduce the likelihood of similar problems in the future. The role of testing and analytics across the entire pipeline process was introduced and discussed here.[8]

We could call the automated tests in the DevOps process monitoring. Coupled with monitoring in production, monitoring throughout the DevOps process an and into production enlarges the scope of testing. DevOps, therefore, does not diminish the role of testers.

7 "How to Eliminate Manual Feature Checking," Paul Gerrard webinar, http://blog.gerrardconsulting.com/?q=node/622

8 "Thinking Big: Introducting Test Analytics," Paul Gerrard, http://blog.gerrardconsulting.com/?q=node/630

Conclusion

You might be concerned about whether DevOps is here to stay and whether testers should take notice? The answer is simple. Yes.

Why wouldn't you want developers and operations people talking to each other? Why wouldn't you want to put more reliable builds and deployments into test and production?

Why wouldn't you want the best of technology to support more accurate, efficient, and informative pipelines?

DevOps is a good thing but not always easy to achieve. Needless to say, it requires cultural change and that isn't always easy. For the tester, DevOps gives us greater influence in the early stages of projects, forces us to think more seriously about automation in testing, information provisioning, and decision-making. Testers need to embrace DevOps because it provides opportunities to be proactive, and to gain more authority and respect in our project teams.

For more information, please visit our website at broadcom.com/continuous-testing.

CS200-181159-0520 May 4, 2020