WHITE PAPER | MARCH 2016

DevOps and Testers

In a series of articles Paul Gerrard, a testing guru and consultant, discusses a range of testing topics. Here he talks about the adoption of DevOps from a tester and testing perspective. DevOps is part of an overall approach that organisations 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.

Paul Gerrard
Gerrard Consulting

Sponsored by
ca technologies

What Does DevOps Mean for Testers?

Background

In this article, I want to discuss the adoption of DevOps from a tester and testing perspective. The DevOps movement (for want of a better label) is progressing rapidly. Like many other moves the industry has made, 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 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. This is the context I'll assume in this paper. But in the marketing of DevOps-related technologies and services, this goal can be obscured. The challenge of the cultural change (or more concretely, behaviour change) required for success is frequently underestimated.

The other assumption I'll make is that the testers involved in and affected by DevOps are new to the whole idea. I'll shape this article as an introduction to DevOps for these testers as well as a discussion of its impact on test practices. If you are an experienced DevOps practitioner, I hope you still find the article useful. If you are not a tester, you will at least see the tester's perspective.

For the Uninitiated: What is DevOps?

Simplistically, DevOps is a label put on the notion of development and systems operations teams working more closely. In the so-called delivery pipeline, from source code commit to operation in production, developers accommodate and automate some of operations activities. Operations have more visibility of and some influence over the activities of developers. The motivation for this is primarily to speed up the deployment and implementation of software. Bringing Ops and Dev closer together—effectively into an agile team—implements what might be called "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 operation. When a developer says a software change is "done," the transition to production usage 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 operation.

So, DevOps is just about tools then?

At one level, the goal of DevOps is to eliminate bottlenecks in the delivery pipeline through automation. But automation of staged processes still require 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.

So DevOps is just Dev and Ops people working more closely with the aid of tools then?

No, it's not that either. The handoffs between automated processes often involve other processes—usually testing of one kind or another. Automated tests need to be created by developers and testers. The output of these tests are focused on providing sufficient information for other processes, or just as often people, to transition between stages in the pipeline. Testers and developers who test provide the assurance that the DevOps process delivers successfully and reliably.

"My head hurts, what is DevOps really?" I have to say, it's an evolving, emergent discipline. The question is posed and discussed at length in an excellent post here.[1] That debate took place just a few weeks before writing this article. So you can see that 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 not "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:

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

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

If it Hurts, Do it More (Often)

The difficulty or pain we experience when doing a particular job influences us adversely. If we don't 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, testing and so on. Our experience is commonly that, the less frequently we perform these tasks, the more traumatic the task is when we actually do it. Martin Fowler suggests three reasons why frequent or even continuous execution of certain tasks reduces the pain.[2]

The first is that larger, more complex tasks are hard to plan, manage and control—breaking up large tasks makes them easier to do, less risky and, if something does go wrong, easier to unwind. The second is that many tasks (and testing is the shining example) provide feedback. That feedback, if received early and often, means that problems can be addressed rapidly and certainly 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 may also see opportunities to automate it in some way.

From the tester's perspective this mantra forces us to take much more seriously the notion of automation in the testing process. If there are manual interventions (typically between automated stages in the DevOps process) these will be seen to be the pain points—the 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; 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 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, for example, checking and testing,[3] and the reliance we can place on testers, on checks and automation.[4, 5]

I am not saying we can 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.

1. Checks that can be automated by developers as part of their component-level check-in and continuous integration processes.
3. Checks that can be automated (typically by system testers) to exercise API-level, link or end-to-end transactions.
4. Tests that can perform compatibility checks to demonstrate compatibility across browsers, operating systems, platforms.
5. Tests that can only be performed by a human.

I can only offer a few suggestions on how to make these distinctions in this article—every environment is different, of course. The more germane question for this article is, "How does the tester 'let go' of late, manual checking?" I have talked about the elimination of late, manual checking before.[6] It requires proactive effort and trust.

These will be the main focus of your efforts:

1. 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 them yourself and include them in the continuous integration regime.
2. End-to-end or user interface tests may require automation. These 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 could be reserved for use on larger, less frequent releases only.
3. 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, BDD-style prototyping help? Could UI checks be performed on mock-ups or wire-frames?
4. 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?

I mentioned the notion of trust above. 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 don't trust the developers to do a good job of testing? Moving testing thinking to the left (as suggested in my previous article) should reduce the doubt. If, as a tester, you act more as a pathfinder to identify risks and assess them, to select tests and 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, a facilitator and a coach/mentor.

Practice, Monitoring and Improvement

With all the good intentions of 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 at every layer, from components and simple transactions in the applications, through integration and messaging and of course the infrastructure itself. 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.[7]

One could call the automated tests in the DevOps process "monitoring." Coupled with monitoring in production, one could say that monitoring throughout the DevOps process and into production enlarges the scope of testing. DevOps, therefore, does not diminish the role of testers.

Conclusion

I was recently asked "When should DevOps not be attempted in an organisation?" It's a good question, but I think what's behind it is concern over whether DevOps is here to stay and whether testers should take notice? My answer is simple.

Why wouldn't you want developers and operations people talking to each other? Why wouldn't you want 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 provision and decision-making. Testers need to embrace DevOps because it provides opportunities to be pro-active, gain more authority and respect in our project teams.

About the Author

Paul Gerrard is a consultant, teacher, author, webmaster, developer, tester, conference speaker, rowing coach and a publisher. He has conducted consulting assignments in all aspects of software testing and quality assurance, specialising in test assurance. He has presented keynote talks and tutorials at testing conferences across Europe, the USA, Australia, South Africa and occasionally won awards for them.

Educated at the universities of Oxford and Imperial College London, in 2010, Paul won the Eurostar European Testing excellence Award and, in 2013, won The European Software Testing Awards (TESTA) Lifetime Achievement Award.

In 2002, Paul wrote "Risk-Based E-Business Testing" with Neil Thompson. Paul wrote "The Tester's Pocketbook" in 2009. Paul co-authored "The Business Story Pocketbook" with Susan Windsor in 2011 and wrote "Lean Python" in 2014.

In 2014, Paul was the Programme Chair for the EuroSTAR Conference in Dublin.

He is Principal of Gerrard Consulting Limited, Director of TestOpera Limited and is the host of the Test Management Forum.

Mail: paul@gerrardconsulting.com
Twitter: @paul_gerrard
Web: gerrardconsulting.com

For more information visit **Develop & Test** with CA Technologies.

Connect with CA Technologies at ca.com

CA Technologies (NASDAQ: CA) creates software that fuels transformation for companies and enables them to seize the opportunities of the application economy. Software is at the heart of every business, in every industry. From planning to development to management and security, CA is working with companies worldwide to change the way we live, transact and communicate – across mobile, private and public cloud, distributed and mainframe environments. Learn more at **ca.com**.

References

1. "What is DevOps," The Agile Admin, http://theagileadmin.com/what-is-devops/
2. "Frequency Reduces Difficulty," Martin Fowler, http://martinfowler.com/bliki/FrequencyReducesDifficulty.html
3. "Testing and Checking Refined," James Bach, Michael Bolton, http://www.satisfice.com/blog/archives/856
4. "A New Model for Testing," Paul Gerrard, http://dev.sp.qa/download/newModel
5. "The New Model and Testing v Checking," Paul Gerrard, http://blog.gerrardconsulting.com/?q=node/659
6. "How to Eliminate Manual Feature Checking," Paul Gerrard webinar, http://blog.gerrardconsulting.com/?q=node/622
7. "Thinking Big: Introducting Test Analytics," Paul Gerrard, http://blog.gerrardconsulting.com/?q=node/630

CS200-181159_0316