Testing in DevOps

Frank Faber

Abstract DevOps can be explained by a set of principles and these principles indicate that Testing in DevOps should be looked at differently. The customer must be at the center of testing and more specialized tests must also be used. To ensure that there is continuous feedback, automation should be used as much as possible. All these tests are the responsibility of one team. To achieve this, a test engineer in DevOps must become a T-shaped engineer.

Keywords Software testing · Software quality · Test automation · Test engineer · DevOps · Agile testing

1 Introduction

In many IT organizations there is a “Wall of Confusion” [1] between development and operations. On the one hand, development wants change, but on the other, operations wants stability. Between them is a wall with a lot of confusion.

There has been a shift to break down this “Wall of Confusion.” More teams are starting to work with a DevOps mindset. Engineers from different organizational silos are working together to create value for the customer. The DevOps mindset places more emphasis on delivering quality in a service, with end-to-end responsibility.

To assure the quality of the service, testing was and will be an essential part of the process. With DevOps, changes are necessary to make sure testing really focuses on the quality in the delivered service.

F. Faber
ALTEN Nederland, Eindhoven, The Netherlands

2 What Is DevOps?

The term DevOps consist of nothing more than the contraction of the words: *Development* and *Operations*. DevOps started in 2009 with the DevOps Days, a conference organized by Patrick Dubois [2]. Although many have tried to come up with a definition for DevOps, there is not one universally accepted definition. The DevOps Agile Skills Association (DASA) has formulated six principles which cover most definitions of DevOps.

1. Customer-Centric Action
2. Create with the End in Mind
3. End-to-End Responsibility
4. Cross-Functional Autonomous Teams
5. Continuous Improvement
6. Automate Everything You Can [3]

The six principles help understand what DevOps really is about.

2.1 *Customer-Centric Action*

DevOps is usually mentioned together with Agile because they share some core ideas. The Agile Manifesto [4] shows a way of working in which a small incremental piece of software adds value for the customer. DevOps builds on the same principles by putting the customer in the center of the action. DevOps teams work directly for the customer and should base all their activities on the mindset that they should add value for the customer.

To be able to accomplish this, it is inevitable to have feedback from the customer. There should be a short feedback loop where the customer can influence the actions of the team. Teams can experiment with new features for services. With a properly implemented feedback loop, customers can give quick feedback on the usage of these features.

2.2 *Create with the End in Mind*

The scope of work within DevOps teams has changed. Before DevOps, the wall of confusion between Development and Operations prevented engineers to work with the scope of the entire Software Development Life Cycle (SDLC). Development created software and ceased after tests were successfully executed. Operations picked the software up for deployment on production with some knowledge on what happened before they got their hands on the software. With DevOps, Development and Operations expertise should be present in a team. A DevOps team should invest in a process where they own the entire SDLC of a functional service.

2.3 End-to-End Responsibility

Responsibility can give teams positive satisfaction in the work they do. It is one of the motivators for work as described by Frederick Herzberg [5]. As part of the two-factor theory, Herzberg distinguishes motivators and hygiene factors for job satisfaction.

Within DevOps, there is an End-to-End responsibility for the service(s) a team delivers. That implies that teams are accountable from “concept to grave” [3]. Teams are responsible for the entire SDLC and they should be aware and act on all changes the service undergoes. Security and performance, for example, could become a higher priority for the team, because they are responsible for maintaining the service. Security, performance, and all other quality attributes are not the responsibility for specialists anymore but should be built in the SDLC process from the start by the entire DevOps teams.

This responsibility can motivate teams and lead to positive satisfaction.

2.4 Cross-Functional Autonomous Teams

In DevOps, teams should be cross-functional and autonomous. All the skills and knowledge to deliver and maintain a service with End-to-End responsibility should be available within the team. This doesn’t mean that engineers should be superhumans and should be able to perform every task available. The skills and knowledge of an engineer can be linked to the T-shaped model [6]. The T-shaped model consists of two bars where the vertical bar stands for the depth of knowledge and skills in a specific area. The horizontal bar shows the knowledge and skills in many areas where the engineer is no expert. This should make it possible for an engineer to pick up simple tasks in their team from outside their expertise. Sharing within the team is important because it will allow engineers to grow in the horizontal bar of the T-shaped model due to the knowledge and skills of their team members. This however is only possible with the vertical bar of the T-shape from other team members. There should be expertise available in depth to be able to share this with other team members.

2.5 Continuous Improvement

New customer demand and changes in the environment are reflected in the work a team needs to do. Teams need to be continuously focused on improvements in their service. This is implemented by the “The Second Way” from The Phoenix Project [7]. “The Second Way” is about implementing a constant flow of feedback. As a team you want to have as much feedback as possible, as soon as possible. Customer collaboration helps with the feedback of the customer. Testing and monitoring is

another way of giving teams feedback on the way their services operate. Teams should be able to react quickly on failed tests or changes in performance as shown in monitoring.

2.6 Automate Everything You Can

Many teams, including DevOps teams experience that they have to perform the same task multiple times in succession. To enable flow in the SDLC, many teams turn to automation. Automation allows them to remove error prone manual steps. When well implemented, it offers a faster process and can free up time for the team to focus on delivering quality in their service.

In the current market there are many companies that use the term DevOps to promote their tools for automation. Although tools can help a team transition to DevOps, DevOps won't start with a tool but with a change in the process.

3 Testing in DevOps

“The process consisting of all lifecycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.” [8]

This definition of software testing is known throughout the world. In DevOps these activities still need to be undertaken to verify quality of a product. Before DevOps, testing tended to be a phase in the SDLC which ended before the product was delivered to Operations. Testing, however, should not be a phase. It should be present from the start and should be a responsibility shared by the entire (delivery) team [9]. Done right, testing helps to “Build quality in” as how W. Edwards Deming described this [10]. The shared responsibility together with embedding it in the entire process should lead to a good quality product.

3.1 Scope

The scope change that comes with DevOps means that teams have an end-to-end responsibility over the quality of their service. Activities that assure the quality of their service are part of this end-to-end responsibility. You could easily argue that all test activities should be done in a DevOps team. The tests a team performs should be in theory end-to-end testcases, because it should cover the entire scope of the teams' service. This can be difficult when a team is responsible for a service that connects with many other services. The definition of end-to-end for a team can differ from the

definition a customer has. Business processes can cover multiple services, and end-to-end testing will test the services of multiple teams. The responsibility of these end-to-end tests can cause a problem because they can be owned by multiple teams.

One way to deal with this problem is to use contract-based testing [11]. DevOps lets you look at software as a service and it is common to create a contract for a service. Contract-based testing is verifying if the service upholds that contract. The tests should imitate a customer (or another service) and make sure there is no contract breach. If a company consists of multiple services and all teams have successfully implemented contract-based testing, you could even argue that there is no need for end-to-end testing anymore. The functionality should be covered in all contract and therefore in the tests. This, however, is only possible when the contract matches with the customer's needs and no mistakes are made in the chain of contracts. There should be something to cover any discrepancies between contracts. End-to-end testing could be used to mitigate the risk in these discrepancies. In a microservices architecture [12] contract-based testing is a more applicable test strategy, because the architecture consists of multiple services. In a monolithic architecture it can be more difficult to apply contract-based testing, because it is more difficult to identify services that can be individually tested.

End-to-end testing that cover multiple teams could be made from small tests connected to each other in a framework until they cover the entire business process. Teams will be responsible for a small part of the tests and should offer help to other teams to make sure the test chain will work. With this approach it is still possible to perform end-to-end testing with DevOps teams. It requires a lot of coordination and a testing framework that supports the input of multiple teams.

In certain situations, it can and will be more fruitful to make one team responsible for end-to-end testing when it covers multiple services. This team can deliver an end-to-end testing service for multiple teams. DevOps teams should provide criteria for the tests for the parts that cover their service and should act on negative results from the tests. The end-to-end testing team should be responsible for making sure the tests will be executed, but the DevOps teams are still responsible for their own service in these tests. Whatever you choose, it is best to ensure that a team is created with the principle of customer-centric action first.

3.2 Customer-Centric Testing

The Agile Manifesto was the start of changing software development. It focusses highly on the process. In his book *Specification by Example*, Gojko Adzic writes: “building the product right and building the right product are two different things” [13]. He makes the distinction between the process and the outcome of the process. While your process can be as you need it to be, there is no guarantee that the outcome of your process – your product/service – has the right quality. To achieve quality, a direct focus on the customer is needed, because they determine what the requirements of your service should be. In this context it makes sense that the first

DASA DevOps principle is about Customer-Centric Action [3]. Testing could help focus on the customer when you apply customer-centric testing.

The first step in customer-centric testing is collecting the requirements, because they will lead to test cases. *Specification by example* [13] is a term which describes a way of dealing with requirements. Another term which shares the same set of principle and ideas is Behavior-Driven Development [13, p. xiii]. Adzic writes that specifying should be done collaboratively to be most effective. In Agile, the term “The Three Amigos” started, where a business analyst, a developer, and a tester worked together on the requirements of a new feature [14]. This process reduced complexity and tried to get the broadest view on requirements with a small group. In DevOps a team draws up requirements in a collaborative manner, whereby all the team’s expertise is considered. It takes it a step further from The Three Amigos when you focus on the expertise the different engineers bring in a team. These kinds of expertise can shape the quality of your service.

Specification by example starts with the customer perspective when writing requirements. One format for doing that is the Given-When-Then format, also known as Gherkin language [15, 16]. This way of writing down requirements in terms of behavior is a way of putting the customer in the center. It’s also a format which is much easier to understand by a customer, either internal or external. Using multiple of these statements will give a set of examples which identifies the functionality a customer wants. It enables discussion between the customer and the DevOps team on a level both can relate to. Examples are a way to illustrate the behavior of a customer and these examples take requirements a step toward a more customer-centric approach.

From the Given-When-Then format to the actual tests is a small step to take with multiple test tools. Tools like Cucumber [17] make it possible for engineers to make test cases with this format. In these tools, Given, When, and Then act as keywords and can be linked to test code which is used to run the tests. Cucumber and other tools keep the requirements visible, so they can be part of ongoing discussions on the requirements and test cases. In test execution reporting, the same format can be used to share the results of tests to stakeholders. Teams must consider the management of the tests and glue code in Cucumber, because it may result in a lot of extra work if it is not manageable.

DevOps does not demand teams to implement *Specification by Example* or use Cucumber as their primary test tooling. Both the method and tool are merely examples of another way of testing where the approach is to look directly at the customer when creating requirements and test cases. It can help create a mindset for Customer-Centric Testing.

3.3 *Specialized Testing*

More specialized tests like security and performance testing should also be done within a team. This can potentially pose a problem, because what do you do when

the knowledge of this kind of testing is not present in the team? There are multiple approaches to face this problem depending on the situation in the team.

As mentioned previously, DevOps doesn't mean engineers should be superhumans. Engineers will probably lack knowledge and skills in certain areas. In general engineers with a testing expertise will know a little of security and performance testing, because they are considered to be specialized fields. Being a T-shaped engineer in DevOps, however, is also about challenging an engineer to expand his knowledge and skills. It can be possible for an engineer to become potent in both security and performance testing. If you also consider that all the knowledge of the service the team provides is within the team, team members should be able to assist the engineers in this. Operations expertise could help understand the performance on server level. Development expertise could help understand framework and methods of security in code. Engineers should not necessarily become experts in security and performance testing, but they should be able to create tests which properly test the requirements set for their service.

Another approach can be to use a testing service for performance and security. End-to-end responsibility does not mean teams have to do everything themselves, but they should keep control. In bigger enterprise organizations it is more common that teams deliver a security or performance testing service. What should be considered though is that working with these testing services can prove to be difficult to fit in the way of working of the teams. There can be different priorities between the teams, which could result in not getting the testing service at the moment you need to. It should also be clear what the expectations are on both sides of the testing service. DevOps teams should deliver a clear request on what they want to have tested and should provide the necessary knowledge. On the other side it should be clear for DevOps teams what is expected from the tests.

Between these two approaches some hybrid approaches are possible. Having an engineer with security/performance testing knowledge in the team for a short period of time or having security/performance testing teams educate the DevOps teams to the required knowledge level are just two examples.

With each approach it is very important for DevOps teams to understand that the scope of their testing has changed. They are responsible and should keep control of all kinds of testing their service needs.

4 Automation

DevOps will not work without automation. Error-prone manual tasks can and should be replaced by automation. DevOps teams require fast feedback and automation is the way to get this to the team. It can speed up the existing processes and make sure the team receives feedback about the process as soon as possible. When automation is working, team members can focus on tasks which do require human intervention.

Automation can play a role in the breakdown of the "Wall of Confusion." It could be possible that Development and Operations used their own set of tools for

deploying and other processes. Within DevOps it is best if teams start using the same tools for the entire SDLC. This can be a way of bringing team members together and make the SDLC clear and coherent. Different skills present in the team can shape the automation to where it fits the needs of the entire team.

4.1 Test Automation

In testing, more and more tests are being automated. Test engineers work more with testing tools and automation supporting their tests. It creates fast feedback loops which drives development design and release [9]. In DevOps you want to automate as much as possible. The testing quadrant diagram, as created by Brian Marick, was adopted by Lisa Crispin and Janet Gregory to create the Agile Testing Quadrants [18]. The quadrants show different kinds of tests where automation can play a role. Technology facing tests supporting the team, like unit tests, are automated. Business facing tests supporting the team, like functional tests, can be automated or done manual. These functional tests in DevOps should be automated, based on the DevOps principle “Automate everything you can” [3]. These functional tests are the tests that should be part of customer-centric testing as mentioned before. Technology facing tests critique to the project, however, require mostly tools and are therefore already automated.

The automation of the first three Agile Testing Quadrants should leave time and space for the last quadrant with business facing tests that critique to the product. These tests should be done manual and cannot be automated. With multiple skillsets in a DevOps team, it would benefit the team to perform these tests with the team during the time they saved with implemented automation.

The test pyramid can help the implementation of test automation. It makes a distinction between tests that can be executed fast on low levels (unit tests) and tests that are slower to execute (UI tests) [19]. The lower-level tests are most suitable for automation, which is why they are usually fast to execute. The test pyramid combines tests originally performed by developers (unit tests) and those performed by test engineers (service, UI tests). This is a testing strategy that will work with DevOps because it is a cross-functional strategy. Engineers in a DevOps team should share their knowledge and expertise to fully implement this test strategy. This strategy also helps teams making testing a shared responsibility within the team.

4.2 Continuous Testing

Automated testing can be part of a deployment pipeline and can be part of Continuous Integration, Delivery, or even Deployment. Deployment pipelines are the “automated implementation of your application’s build, deploy, test and release process” [9]. The deployment pipelines are a way to empower teams to take control

over their deliverables. A pipeline can help a team deploy their service and verify the quality in an automated way. In DevOps teams, it should empower all team members to deploy any version on any environment with the right controls in place. The pipeline can limit the complexity of deploying and testing a service. The knowledge difference between team members can be smaller when every team member could deploy and test a service with a push of a button.

Continuous testing can act as continuous process where every time the pipeline is started tests are being executed. Tests can act as go or no-go points in the pipeline to go to the next step in the process. It also gives the team up-to-date feedback on the quality of their service in different stages of development. Testing can also be used to test the automation. It can help understand if the automation executes the correct steps in a correct way. This includes the automation used for deploying services on different environments. With deployment testing, a team can take control of the infrastructure and check whether it is in the state where it should be. Testing will give teams control on their automation when they are relying much more on it.

4.3 Monitoring

It is increasingly common to arrange monitoring from the start of software development. With an end-to-end responsibility in DevOps teams, monitoring is a way to get feedback from the service on production environments to the teams. Monitoring can vary from technical monitoring on server level: measuring CPU usage, memory, etc., to more functional monitoring: how many users are logged in etc. Functional monitoring gives to a certain degree insight into customer perception of the service. It can allow teams to track customers through their usage of the service.

It could be argued that monitoring can replace parts of testing. If a DevOps team has reached “the third way” [7], they gather feedback continuously and experiment with their service. Monitoring can help the team in the gathering feedback. The third way is when teams are mature enough to deliver new features fast and experiment to see what fits the needs of the customer.

Monitoring should be a more reactive way of getting feedback where testing is a more proactive approach. That is mainly because monitoring is focused on production environments and therefore a later step in the process. Testing can start at an early point in the process and gives teams more options to adapt to the outcome of testing. Monitoring can act as a way of testing when teams are able to adapt quick to the outcome of monitoring. If teams can create new features and deploy them fast, using, for instance, Continuous Deployment, teams can react fast. Monitoring can act as business facing tests critique of the product and would fit in the Agile testing quadrant [18].

5 The Role of a Test Engineer

A test engineer was and is someone who can act as intermediate between business and IT. A test engineer can connect the requirements from the business side to the technical implementation of IT side. This can be an enabler for an ongoing conversation between Business and IT. With DevOps, this conversation must keep going to make sure the customer is in the center of the action. In DevOps it is more likely to have engineers with different skillsets present in a team. The function of a conversation enabler is required. A test engineer can bridge the gap between Dev and Ops, because quality is what connects these two. Test engineers can play a vital role in this conversation. They can facilitate that quality is a shared responsibility in the team. Quality can be the most important topic that brings teammates closer together. Engineers can give their view on quality based on their own expertise. A test engineer can make sure that all these different views are combined in a single test strategy.

Test engineers in DevOps are not the only one responsible for preparing and executing test cases. T-shaped engineers with different kinds of expertise should also be able to prepare and execute test cases. Test engineers can act as coaches to their team members and help them understand how to test. The test expertise from a test engineer should be shared within the team.

5.1 *T-Shaped, Test Shaped*

A T-shaped test engineer should start with making sure his or her own expertise fits in DevOps. The test expertise should contain knowledge and skills to gather the correct requirements for the system under test. From these requirements test cases can be created and executed using test techniques. The test expertise should also contain knowledge on how to choose and use the correct tooling in the process. This expertise differs not from the existing role of test engineer outside DevOps.

With all the automation present in DevOps, a test engineer needs to have technical skills and knowledge to implement testing in Automation. From Test Automation to Continuous Delivery, a test engineer must be able to fit in where the team has implemented automation. This usually means that a test engineer needs to have some basic coding skills and understand the structure of most programming languages. Next to technical skills a test engineer must understand Test Automation and be able to implement it in a fitting test strategy. Following the T-shaped model you could say that programming knowledge is part of the horizontal bar and need not be an in-depth knowledge and skill. Test automation and complementary test techniques should be part of the vertical bar and the Test expertise of an engineer.

Test engineers can act as intermediaries between Business and IT or between different kinds of expertise in a DevOps team. This enables them to gain knowledge from different parties present in the team. They should be able to gain knowledge, and this can help them expand the horizontal bar of their T-shape.

5.2 *Soft Skills*

Although the focus now seems to be on the technical skills that a test engineer needs, it is also important that a test engineer has good soft skills. Soft skills are needed to get the conversation between different parties going and to keep it going. If a test engineer is going to be an intermediary, then people skills must be in order. These skills make it possible to take a leading role.

6 Conclusions

The practice of testing in DevOps starts with the same foundations as testing in a non-DevOps environment would have. The end-to-end responsibility that a team has for a service means that quality requirements for both development and operations must be considered. Quality in the Operations section of the SDLC was usually not included but is now part of the scope of testing. The scope for testing in DevOps is on the entire functional service a team delivers. End-to-end testing therefore can take a different form. More specialized tests, like performance and security tests, will part of the new scope, although on a smaller scale. Due to the specialized nature of these tests, however, it is possible that these tests will be executed as part of a testing service outside the team. DevOps teams should take ownership of these tests and should make sure they are executed. Functional tests in DevOps should focus more on the customer as part of customer-centric testing. This makes sure the quality a customer wants is put central in the work a team performs.

In DevOps you want to automate everything you can. Automation in testing must be used where it can be used. With automation teams can get continuous feedback on their service and the steps they take to add new features to their service. Monitoring can be a valuable addition to testing and help to get quick feedback on the functional and technical levels on the service a team delivers. This feedback can be used to shape the service in a way it fits the customer.

The role of a test engineer changes to a DevOps Engineer with a test expertise. The test expertise as part of the T-shaped model consists of knowledge and skills to implement and teach test strategies in a team. The test expertise should be able to connect Business and IT and different kinds of expertise in a team to get all the requirements for quality in a service. Responsibility for testing must be shared with the entire team. The engineer with test expertise can take a leading role in this and act as a coach.

References

1. Shafer, A.: Agile infrastructure. Speech presented at Velocity Conference (2009)
2. Mezak, S.: The origins of DevOps: What's in a name? <https://devops.com/the-origins-of-devops-whats-in-a-name/> (2018). Accessed 24 January 2018
3. 6 Principles of DevOps. <https://www.devopsagileskills.org/dasa-devops-principles/> (2017)
4. Manifesto for Agile Software Development. <https://agilemanifesto.org/> (2001)
5. Herzberg, F., Mausner, B., Synderman, B.B.: The Motivation to Work. Wiley, New York (1959)
6. Guest, D.: The hunt is on for the renaissance man of computing. *The Independent* (1991, September 17)
7. Kim, G., Behr, K., Spafford, G.: The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win. IT Revolution Press, Portland, OR (2013)
8. International Software Testing Qualifications Board: Certified Tester Foundation Level Syllabus Version 2018 Version. <https://www.istqb.org/downloads/send/51-ctfl2018/208-ctfl-2018-syllabus.html> (2018)
9. Humble, J., Farley, D.: Continuous Delivery. Addison-Wesley, Upper Saddle River, NJ (2011)
10. Deming, W.E.: Out of the Crisis. MIT Press, Cambridge (2000)
11. Aichernig, B.K.: Contract-based testing. In: Formal Methods at the Crossroads. From Panacea to Foundational Support, pp. 34–48. Springer, Berlin, Heidelberg (2003)
12. Richardson, C.: What are microservices? <https://microservices.io/> (2018)
13. Adzic, G.: Specification by Example: How Successful Teams Deliver the Right Software, p. 3. Shelter Island, NY, Manning (2012)
14. Dinwiddie, G.: The Three Amigos: All for One and One for All. Better Software. <https://www.stickyminds.com/sites/default/files/magazine/file/2013/3971888.pdf> (2011). Accessed November/December 2011
15. North, D.: Behavior Modification. Better Software (2006, June 5).
16. Gherkin Reference. <https://cucumber.io/docs/gherkin/reference/> (n.d.)
17. Cucumber. <https://cucumber.io/> (n.d.)
18. Crispin, L., Gregory, J.: Agile Testing. Addison-Wesley, Boston, MA (2008)
19. Fowler, M.: Bliki: TestPyramid. <https://martinfowler.com/bliki/TestPyramid.html> (n.d.)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.