Continuous Testing vs Test Automation

Sara Ersson
Miguel Müller

April 2019

Contents

1 Introduction

Today almost all companies are involved in IT somehow. It does not matter if you are an airline company, a hamburger chain or a drug store, you still need to meet the customers' needs and expectations. This means that as applications are becoming more and more complex, regarding architecture, development and usage, the business impact of the software failures and the need for complex testing are increasing rapidly [1].

What it comes down to is that to be sure that a program works properly it needs to be tested somehow, and writing tests which are good can, like writing programs, be a difficult task. Tests should accomplish many things such as testing what they are intended to test, not including redundancy, checking requirements, being reusable, being independent of other tests, and being clear and easy to understand.[8] However, it is not only what tests are run, but when they are run. Tests can either be run manually or they can be set up to run automatically. The problem is that many testers today feel that there is too much overhead to maintain the automated tests and that it does not work that well these days. The tests just cannot be automated and run the same way anymore. [1] 1 Introduction Today almost all companies are involved in IT somehow. It does not matter if you are an airline company, a hamburger chain or a drug store, you still need to meet the customers' needs and expectations. This means that as applications are becoming more and more complex, regarding architecture, development and usage, the business impact of the software failures and the need for complex testing are increasing rapidly [1]. What it comes down to is that to be sure that a program works properly it needs to be tested somehow, and writing tests which are good can, like writ- ing programs, be a difficult task. Tests should accomplish many things such as testing what they are intended to test, not including redundancy, checking requirements, being reusable, being independent of other tests, and being clear and easy to understand.[8] However, it is not only what tests are run, but when they are run. Tests can either be run manually or they can be set up to run automatically.

2 Test Automation

The process of controlling the execution of the tests is called *test automation*. The key to success is the comparison of the predicted output and the actual output, which reflects the quality of the software. Test automation is especially good for repetitive tasks and those which are difficult to perform manually. [7]

Within test automation there are a couple of different approaches out there practised. The two most used ones are however GUI testing and API testing. The first is the process of testing the graphical user interface, as the name suggests, of the software to ensure that it works and looks the way it is supposed to. The latter, API testing, tests the application programming interfaces of the software. Because APIs do not have a GUI the testing, unlike GUI testing, is performed at the message layer. What is especially examined during API testing is the quality of the functionality, reliability, performance and security of the software package as a whole. [4] This however does not seem to do the trick anymore. Should the software really be tested in the final stage before release? [1]

3 Continuous Testing

The term continuous testing has been increasingly used during the last decade. So what does it bring to the table? The general idea is to bring software tests to each stage of the integration/delivery pipeline in order to obtain feedback as early as possible in the development cycle. This includes repeatedly executing automated tests against both the code base and against all the deployment environments.

The feedback received from continuous testing is supposed to focus on business risks, and whether the software is qualified to be released, or in other words, whether the software has an acceptable level of business risk. Thus, the adaptation of continuous testing is more of a cultural shift that needs to happen in organizations so that they can meet today's process demands, according to Wayne Ariola, Chief Strategy Officer at Parasoft.[2]

The automated continuous testing pipeline should contain many different types of tests, such as unit tests, static code analysis, security code analysis, integration tests, load tests, performance tests, regression tests, etc. Most, if not all, of these types of tests have been extensively used before the introduction of the term continuous testing. But they used to only be written in a traditional bottom-up fashion, meaning that new tests were added mainly when new functionality was introduced. With continuous testing, this line of thinking will not disappear, but rather be complemented by the introduction of top-down testing which first and foremost examines the (expected) user experience. The more user-centric philosophy of continuous testing and the focus on business risk is what differentiates it from traditional end-to-end tests.

3.1 Continuous Testing in DevOps

As many organizations nowadays expect shorter and shorter software delivery cycles, the adaptation of DevOps practices has significantly increased during the last few years. The main goal of DevOps is a faster path to deployment/delivery. Adapting continuous testing in the DevOps lifecycle enables exactly this, through a faster feedback loop for developers.[3]

3.2 Tools

One can divide the DevOps life cycle into eight stages: plan, code, build, test, release, deploy, operate, monitor. Continuous testing tools belong in at least three of these stages: build, test, and release. Naturally, the build automation tool(s) that one is using (Gradle, Maven) need to be able to run all your desired tests in an automated manner. Nevertheless, the need for frameworks that help with test automation will come into play here. Selenium is an example of such a framework, with the purpose of testing web applications without the need to manually open a web browser and manually click buttons to see if the software works as intended. One very important part of the test stage that shouldn't be forgotten is the use of a test coverage tool, to measure how much of the code the test suite(s) cover. The data you get from the test coverage tool should then be used both to optimize coverage based on what's needed for compliance, but also to optimize coverage on what's needed to reduce business risk.[2] One central part of DevOps, and in continuous testing, is the use of a continuous integration server such as Jenkins, which runs tests in all the desired environments to verify the build before preparing it for deployment or delivery.

As long as the mentioned tools are correctly integrated with each other, they should be enough to begin utilizing continuous testing, according to online education platform edureka.[6] Continuous testing does have slightly different definitions depending on which expert you ask. However, there is consensus that all tests has to be a part of the pipeline. The tools mentioned shouldn't be seen as tools that automate some tests, or even most tests. *All* tests should be integrated in the DevOps pipeline. If something in your pipeline relies on manual testing, you're not doing continuous testing properly.

4 Continuous Testing vs Test Automation

People might say that continuous testing and test automation are the same thing, but this is not true at all. Both testing strategies are test automation-based, but there are several ideas which differentiate the two strategies, and they can be divided into three different categories; *risk*, *breadth* and *time*.

4.1 Risk

While test automation is all about testing user stories and functionalities to see if they are correctly implemented and work properly according to the specifications, continuous testing focuses on testing on a higher level. The meaning of the latter is that tests are written mainly with the purpose of testing whether the software is too risky to release or not. [1]

To explain what is meant by *risk* in this case one could use a grocery store example for instance. Before, a grocery store literally offered groceries. It had a location, products, commercials and customers. These days the majority of all grocery chains have their own website and application available for download, where customers can become a member, collect points, have access to recipes, search for products and get discounts, just to mention a few. An application could be bad in many ways. It could have a poor user interface, or not work properly, for instance taking a long time to load pages or not caching information so that the customer has to re-do actions. Having a bad application would be a huge risk to a company. Because of software failures they might lose customers to some other store with better software or better software ideas.

4.2 Breadth

OK so people talk. What differs continuous testing from test automation also is that small errors are taken much more seriously because of the damage it could make to a company's reputation. Small errors could for example be errors affecting the user experience in a bad way, like re-sizing or moving buttons or change the behaviour of them so that the user has to struggle a bit because of the change. It is not necessarily the customer him-/herself which is lost, but many others being lost because of brand damage on social media for instance. [1] People do not just tell their friend or their neighbour if something bad has happened anymore. Today there are endlessly many social media applications where people can say exactly what they want, and people usually do say what they want. That is why the tests need to be broad enough to notice when an application change impacts functionalities which users have come to rely on, without purpose [5].

4.3 Time

In the early beginning of test automation tests were created and ready to be run after the development phase. Now that more and more organizations are adopting Agile and DevOps, the intervals between releases are becoming shorter and shorter. With shorter time between the releases it becomes even more important to obtain almost instant feedback from testing, to know whether your functionalities are implemented the right way or not. Otherwise the user stories might not be ready within the deadlines. And to obtain instant feedback the tests need to be run in parallel with the development. [1]

5 Analysis and Conclusion

Continuous testing and test automation are a little bit like C++ and C. Yes, continuous testing is based on test automation, but there is so much more to it than that. Its primarily goal is assessing business risk coverage rather than just checking if certain functionalities are working. Continuous Testing, if adopted properly, seems to help teams during the development process, protecting the user experience and avoiding poor headlines. [5]

Over all, the earlier you find errors, the more time and money will be saved. Continuing building on something broken could easily escalate exponentially in workload to be done during the aftermath. By always knowing the quality of the software, and that the package as a whole works all together, teams could spend that extra time on other important things like coming up with more new ideas, not working extra hours and stress less.

The feedback received from continuous testing is useful not only for developers, but also for others in the organization. Throughout the software development process, business stakeholders can have instant access to feedback on whether their requirements and expectations are being met, which enables them to make more informed decisions.[2]

What is important to realize is that there are no tools which single-handedly can ”provide” continuous testing on their own. Rather, many software tools already established in the DevOps pipeline combined with robust testing tools and frameworks forms the technical basis for continuous testing. But it is more of a working approach; how processes are structured, how the tools are integrated with each other, how the tests are maintained, what technologies are used, and how the technologies are used.[1] Adapting continuous integration is most probably not something which can happen over the course of a day, or even a week. But it is definitely something to invest time and money on, as it provides instant insight on the state and risks of the software builds.

References

[1] Wayne Ariola. *Continuous Testing vs. Test Automation: 3 Key Differences.* June 2017. URL: https://devops.com/continuous-testing-vs-test-automation/ (visited on 04/26/2019).

[2] Wayne Ariola. "DevOps: Are You Pushing Bugs to Your Clients Faster?" In: *PNSQC 2015 Proceedings.* 2015. URL: http://uploads.pnsqc.org/2015/papers/t-007_Ariola_paper.pdf (visited on 04/28/2019).

[3] Adam Auerbach. *Continuous Testing Is Not Automation.* Nov. 2018. URL: https://www.techwell.com/techwell-insights/2018/11/continuous-testing-not-automation (visited on 04/29/2019).

[4] Joe Colantonio. *Why API Testing is Important (GUI is No Longer King).* Sept. 2014. URL: https://www.joecolantonio.com/the-gui-is-no-longer-king-why-api-testing-is-important/ (visited on 04/30/2019).

[5] *Continuous Testing Definition: 14 Key Points.* URL: https://www.tricentis.com/products/what-is-continuous-testing/ (visited on 04/26/2019).

[6] edureka. *Continuous Testing in DevOps Training.* URL: https://www.edureka.co/continuous-testing-devops-training (visited on 04/30/2019).

[7] Margaret Rouse. *automated testing.* Dec. 2018. URL: https://searchsoftwarequality.techtarget.com/definition/automated-software-testing (visited on 04/30/2019).

[8] Manish Verma. *How to write good Test Cases.* URL: https://www.softwaretestingmentor.com/writing-good-test-cases/ (visited on 04/29/2019).