DZone

THE DZONE GUIDE TO

Automated Testing

Improving Application Speed & Quality

VOLUME I

BROUGHT TO YOU IN PARTNERSHIP WITH

applitools
AI powered Visual Testing & Monitoring

ca technologies

functionize

Panaya

Perfecto

PLUTORA

rainforest

SAUCELABS

Usetrace

DEAR READER,

As software professionals, there are a lot of things that we take for granted. Automated testing is one of those things. There was a time that manual testing dominated any efforts to ensure software quality, usually as a result of excuses like "the tooling doesn't exist," "it will take too long on this platform," and "we don't have time." I'd like to think that the software development industry has matured and we now understand that such excuses don't cut it anymore.

The benefits of having an automated testing strategy in place become clear once you actually move into production. If a customer is having an issue, you can easily check if it was an issue in development. If you don't have a test covering the issue, you can add it to your test suite. And once the fix is applied, your suite will ensure that the issue has been resolved.

It goes beyond just having unit tests that developers have provided. Even though these are a core part of any good quality assurance plan, we all understand that developers are not the best-placed people to test the code they've written. Despite all your best intentions as a programmer, you are likely to avoid one or two areas that are either difficult to test, or likely to have issues. That's where a dedicated QA team comes in, using tooling like Selenium, or any other UI test automation tools appropriate to the platform. Rather than this being an afterthought, it has to be part of the culture of the team.

If you're reading this guide, chances are that you've probably already bought into the power of automated testing. If you're not yet convinced, you will be once you reach the end. This guide provides a lot of what you'll need to know to ensure that testing becomes part of the culture of your organization. It even includes a checklist to do just that. Covering topics from mobile testing in Kotlin, going beyond what Selenium provides for web application testing, and helping guide developers in structuring code for automation, the guide has something for everyone.

Happy reading!

BY JAMES SUGRUE

DZONE ZONE LEADER AND CHIEF ARCHITECT, OVER-C

TABLE OF CONTENTS

PRODUCTION

Chris Smith
DIRECTOR OF PRODUCTION

Andre Powell
SR. PRODUCTION COORDINATOR

G. Ryan Spain
PRODUCTION COORDINATOR

Ashley Slate
DESIGN DIRECTOR

Billy Davis
PRODUCTION ASSISSTANT

MARKETING

Kellet Atkinson
DIRECTOR OF MARKETING

Lauren Curatola
MARKETING SPECIALIST

Kristen Pagàn
MARKETING SPECIALIST

Natalie Iannello
MARKETING SPECIALIST

Miranda Casey
MARKETING SPECIALIST

Julian Morris
MARKETING SPECIALIST

BUSINESS

Rick Ross
CEO

Matt Schmidt
PRESIDENT

Jesse Davis
EVP

Gordon Cervenka
COO

SALES

Matt O'Brian
DIRECTOR OF BUSINESS DEV.

Alex Crafts
DIRECTOR OF MAJOR ACCOUNTS

Jim Howard
SR ACCOUNT EXECUTIVE

Jim Dyer
ACCOUNT EXECUTIVE

Andrew Barker
ACCOUNT EXECUTIVE

Brian Anderson
ACCOUNT EXECUTIVE

Chris Brumfield
SALES MANAGER

Ana Jones
ACCOUNT MANAGER

Tom Martin
ACCOUNT MANAGER

EDITORIAL

Caitlin Candelmo
DIRECTOR OF CONTENT AND COMMUNITY

Matt Werner
PUBLICATIONS COORDINATOR

Michael Tharrington
CONTENT AND COMMUNITY MANAGER

Kara Phelps
CONTENT AND COMMUNITY MANAGER

Mike Gates
SR. CONTENT COORDINATOR

Sarah Davis
CONTENT COORDINATOR

Tom Smith
RESEARCH ANALYST

Jordan Baker
CONTENT COORDINATOR

Anne Marie Glen
CONTENT COORDINATOR

Want your solution to be featured in coming guides?
Please contact research@dzone.com for submission information.

Like to contribute content to coming guides?
Please contact research@dzone.com for consideration.

Interested in becoming a dzone research partner?
Please contact sales@dzone.com for information.

Special thanks to our topic experts, Zone Leaders, trusted DZone Most Valuable Bloggers, and dedicated users for all their help and feedback in making this guide a great success.

Executive Summary

BY MATT WERNER
PUBLICATIONS COORDINATOR, DZONE

In the DevOps world, a concept that follows terms like Continuous Delivery or Integration is often Automation. Since Continuous Delivery itself is a kind of adaptation of manufacturing processes for the world of software development, it makes sense that developers and their organizations would try to automate as much as they can, as manufacturers do. So, the questions that follow are what code can or should be automated, and what are the benefits? To find out what real developers are doing and experiencing in their organizations, DZone surveyed 434 developers and tech professionals to find out.

WHAT'S BEING AUTOMATED?

DATA The most commonly automated tests are integration tests (61%), component (or unit) tests (58%), and performance tests (56%). Manual testing is still popular for user acceptance tests (78%), usability tests (70%), and story-level tests (63%).

IMPLICATIONS While "automate everything" has become a common refrain at DevOps events and blogs, organizations are finding it difficult to automate any tests that require real user input to test functionality and validate user stories. The most popular automated tests, such as unit and integration tests, are those that test a small part of the application and are simple to implement.

RECOMMENDATIONS When adopting automated testing, you should first focus on automating simple, repeatable processes and tests. This is both proven to be a successful strategy in manufacturing processes like software development and can create results very quickly. Trying to automate user acceptance, usability, or story-level tests is far more difficult due to the traditional reliance on a human opinion, so spending time and money on this may not be the best use of either. For a fun illustration that explains each kind of testing and why it's popular to automate or manually test each of them, refer to our infographic, "The Crossroads of Testing" on page 26.

TESTING AND DEVOPS: TOGETHER FOREVER

DATA For organizations with a designated DevOps team (41%), 49% are focused on introducing automation across the SDLC. Those with a dedicated DevOps team were more likely to automate these tests than those that did not: integration (49% vs. 30%), component (51% vs. 29%), and performance (51% vs. 29%).

IMPLICATIONS Unsurprisingly, those who are making efforts to transition to DevOps processes and Continuous Delivery are more likely to automate their testing efforts by a considerable margin. Those who automate their tests are also more likely to perform push-button or automated deployments by similar margins: integration (51% vs. 28%), component (53% vs. 28%), and performance (50% vs. 32%).

RECOMMENDATIONS Adopting test automation seems to go hand-in-hand with other common DevOps processes such as the creation of a DevOps-specific team and performing push-button or automated deployments. Automating tests will be a significant factor in reducing time to market and increasing quality, both of which are major goals of DevOps processes. For a high-level overview on the relationship between DevOps and automation, refer to Soumyajit Basu's article on page 21.

SHIFTING LEFT

DATA Respondents who automated their tests were more likely to try to build performance and quality into an application from the start. Those who automated integration tests were 11% more likely to do so (44% vs. 33%), those who automated components tests were 13% more likely to do so (45% vs. 32%), and those with automated performance tests were 16% more likely to do so (47% vs. 31%).

IMPLICATIONS When tests are automatically performed, it creates more incentives for developers to try and develop applications correctly the first time so that their code will pass the tests. This leads to programming styles like Test-Driven Development (TDD), where code is written in order to pass tests, or maybe pair programming practices, where one developer codes while another developer watches and offers critiques along the way.

RECOMMENDATIONS For those who want to improve their time to market and software quality, implementing automated application testing early on the process will do wonders. This puts responsibility for the quality of code on developers even before it reaches a more traditional QA department. Jim Holmes' article, "How to Write and Structure Code for Automation," is a great beginner's guide to several of these concepts, and can be found on page 18.

Key Research Findings

BY G. RYAN SPAIN
PRODUCTION COORDINATOR, DZONE

DEMOGRAPHICS

434 software professionals completed DZone's 2017 Automated Testing survey. Respondent demographics are as follows:

- 41% of respondents identify as developers or engineers, and 27% identify as developer team leads.
- The average respondent has 14 years of experience as an IT professional. 56% of respondents have 10 years of experience or more; 21% have 20 years or more.
- 40% of respondents work at companies headquartered in Europe; 34% work in companies headquartered in North America.
- 18% of respondents work at organizations with more than 10,000 employees; 20% work at organizations between 1,000 and 10,000 employees; and 28% work at organizations between 100 and 1,000 employees.
- 83% develop web applications or services; 49% develop enterprise business apps; and 36% develop native mobile applications.

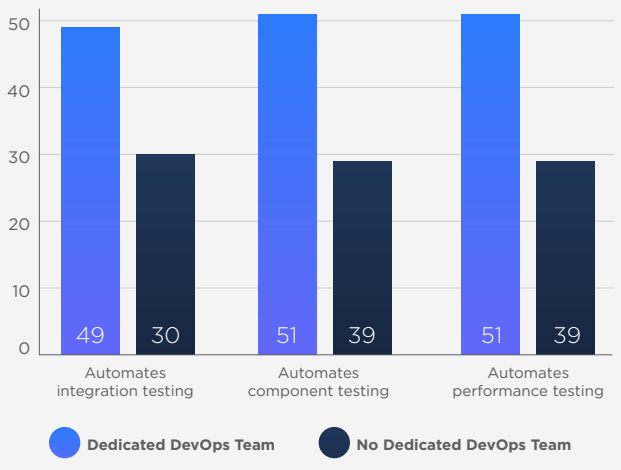
AUTOMATED TESTING

We asked survey respondents which tests in their organization's pipeline(s) are automated and which tests are performed manually. The most popular automated tests were integration (61%), component (58%), and performance (56%). While 22% of respondents automate none of these tests, 17% automate one of the three, 25% automate two, and 36% of respondents automate all three. For manual testing, the most common responses were user acceptance (78%), usability (70%), and story-level tests (63%). Across all manual and automated testing, manual testing had 36% more responses than automated testing. We also asked about a wide array of tools for automated testing. The most popular tools amongst our respondents were JUnit (61%), Selenium (46%), JMeter (45%), SoapUI (29%), and Cucumber (21%). 44% of respondents say their organization's Continuous Integration processes extend into an automated Continuous Delivery pipeline from code check-in to production deployment.

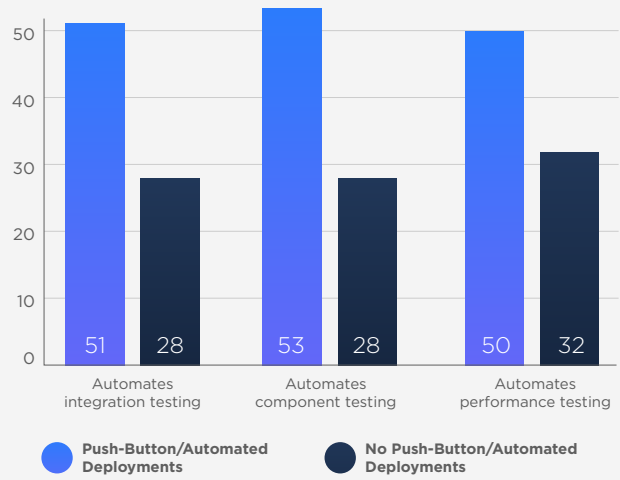
DEVOPS TRENDS

It's no surprise that automated testing and other DevOps practices go hand in hand. 49% of respondents working

Does your organization have a dedicated DevOps team?



Does your organization have either push-button or automated deployments?



at organizations with dedicated DevOps teams said one of that team's goals was introducing automation across the entire SDLC. Looking at the three most popularly automated tests, we found that respondents who said their organization automated these were much more likely to have one of these dedicated DevOps teams. 49% of respondents whose org automates integration tests said they have a DevOps team, as opposed to 30% who said their org does not automate integration tests. For component tests and performance tests, the difference was 51% with dedicated DevOps teams compared to 29% of respondents at orgs not automating these tests. Respondents answering that these tests are automated were also much more likely to say their organization performs push-button or automated deployments; for example, for integration tests, this difference was 51% vs. 28%. These respondents were also significantly more likely to believe their organization has fully achieved Continuous Delivery.

TOOLS AND AUTOMATED TESTING

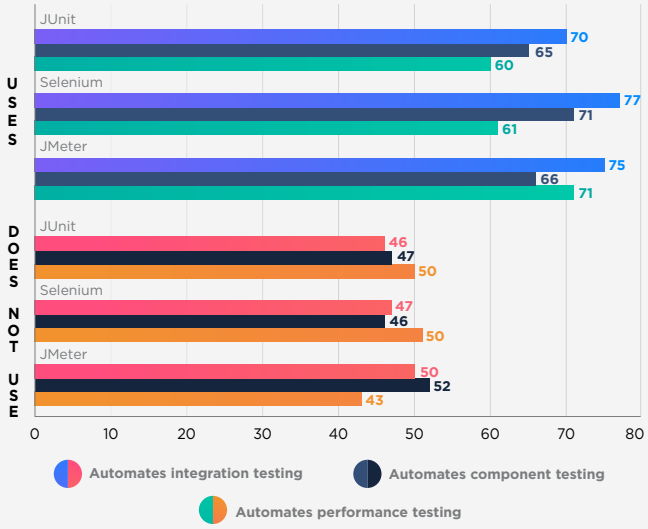
Responses regarding the most popular testing tools were also connected with these commonly automated tests. 70% of respondents whose organization uses JUnit said they automate integration tests, compared to 46% of non-JUnit users. For JMeter this difference was 75% vs. 50%, and for Selenium it was 77% vs. 47%. These trends apply to component and performance testing as well. Performance tests, while not as dramatically different as the others for users of JUnit and Selenium, were automated by 71% of JMeter-users, vs. 43% of non-users. Considering it was more likely for respondents to automate more than one of these popular tests, even starting test automation in one area seems to have an impact on other tests.

TOOLS AND LANGUAGES

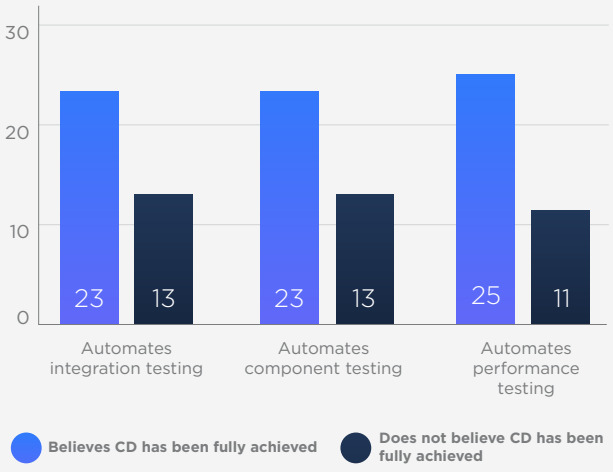
Given how language-specific testing tools are, the popularity of JUnit and JMeter amongst our respondents makes sense. 86% of respondents work at an organization that uses the Java ecosystem, and 62% of respondents work at organizations where Java is the primary programming language. 68% of respondents working at an organization that uses Java at all said their org uses JUnit, and 50% said they use JMeter. Of the respondents who work at primarily Java organizations, 77% said they use JUnit, and 53% said they are using JMeter. So these two open source testing tools are taking hold in the Java world.

**Margin of error calculated with 95% confidence interval*

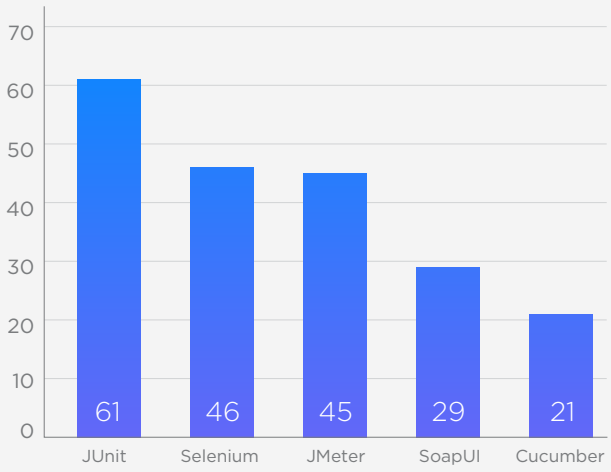
Does your organization automate integration, component, or performance testing?



Do you believe continuous delivery has been achieved in your organization?



Which of the following testing automation tools does your organization use?



A brief history of web and mobile app testing.

Find out how Sauce Labs can accelerate your testing to the speed of awesome.

For a demo, please visit saucelabs.com/demo
Email sales@saucelabs.com or call (855) 677-0011 to learn more.

SAUCELABS
Testing at the speed of awesome.

Automated Testing Is Essential To DevOps And Continuous Delivery

Much has been written and said about DevOps, less so about the role of automated testing in a DevOps environment. In order to reap the full benefits of a healthy DevOps practice, organizations must integrate automated software testing into their Continuous Delivery pipelines. It is the only way to ensure that releases occur at both a high frequency, and with a high level of quality.

In order to integrate continuous testing effectively into a DevOps toolchain, look for the following essential features when evaluating an automated testing platform:

- **Support for a variety of languages, tools, and frameworks**. The programming languages and development tools that your DevOps teams use today are likely to change in the future. Look for a testing solution that can support a broad array of languages, tools, and frameworks.
- **Cloud testing**. On-demand cloud-based testing is the most cost-efficient option because it obviates the need to setup and maintain an on-premises test grid that is underutilized most of the time. It also reduces the resource drain associated with identifying and resolving false positives, or failures due to problems in the test infrastructure.
- **The ability to scale rapidly**. Your testing platform should be able to perform tests as quickly as needed, and be able to do so across all required platforms, browsers, and devices. It should also be highly scalable to support as many parallel tests at one time as you require.
- **Highly automated**. DevOps teams achieve their speed and agility in part by automating as much of the software delivery process as possible. Your testing solution should work seamlessly with other components of your toolchain, most notably your CI and collaboration tools.
- **Security**. In a DevOps environment, all members of the team have an important role to play in keeping applications secure. Testing platforms, therefore, need enterprise-grade security features.

A software testing platform that includes these qualities will empower your organization to derive full value from its migration to a DevOps-based workflow by maximizing the agility, scalability and continuity of your software delivery pipeline.

WRITTEN BY LUBOS PAROBEK
VP OF PRODUCTS, **SAUCE LABS**

PARTNER SPOTLIGHT

Automated Testing Platform

SAUCELABS

Sauce Labs accelerates the software development process by providing the world's largest automated testing cloud for mobile and web applications.

CATEGORY
Automated Testing Infrastructure

NEW RELEASES
Daily

OPEN SOURCE
Yes

CASE STUDY

GoDaddy is a growing web hosting and domain registration company that serves more than 16 million customers across the globe. Based in Scottsdale, Arizona, the firm is the world's largest domain name registrar, with more than 70 million domain names under management. To ensure that websites created by its customers stay up and running around the clock, GoDaddy is committed to delivering the highest-quality software to support its backend systems. GoDaddy sought to increase cross-browser coverage and reduce the time and money it took to run its internal software testing environment. The company met its needs by selecting Sauce Labs, and now utilizes dozens of Sauce Labs virtual machines in parallel to run thousands of tests every day. The tests are automatically run as part of a CI/CD pipeline from GoDaddy's Jenkins CI server. The company now continuously tests all popular browser and OS variations for its products and services, and can also easily test more browser variations as needed, thanks to the on-demand nature of Sauce Labs.

STRENGTHS

- Enterprise-grade cloud-based test infrastructure provides instant access to more than 800+ desktop browser/OS combinations, ~200 mobile emulators and simulators, and 1,000+ real devices
- Highly scalable, on-demand platform reduces testing time from hours to minutes when tests are run in parallel
- Optimized for CI/CD workflows, testing frameworks, tools, and services
- Single platform for all your web and mobile app testing needs

NOTABLE CUSTOMERS

- Salesforce
- VISA
- Starbucks
- GoDaddy
- WalMart Labs
- Slack
- Yahoo!

WEBSITE saucelabs.com | **TWITTER** @saucelabs | **BLOG** saucelabs.com/blog

When Is a Test Case Ready for Test Automation?

BY **MACIEJ GRYKA**
LEAD SCIENTIST, **RAINFOREST QA**

QUICK VIEW

01 Testing and QA are critical parts of a mature software product.

02 Forward-thinking practitioners agree that QA is needed, but there's no agreement on best practices.

03 One of the difficulties is choosing between manual and automated testing, aiming to minimize risk with minimum investment.

04 Automation can work, but the ongoing maintenance is an often-overlooked cost.

05 Knowing common causes of automation breakages can help you make the right decision.

There are so many things you need to think about when working on software projects: what problem you want to solve, if software is actually the best way to do it, what technology to use, how to structure your teams, which development methodology to use, and — finally — what and how often to test?

That last part is often left as an afterthought, but that's not always a bad decision! After all, going from "no solution" to "some solution" is the first step and it does not, strictly speaking, require having good test coverage. However, as your product and your organization mature, testing and Quality Assurance become more and more critical to your success. Learning this lesson took us, as an industry, time and some painful lessons.

While these days most professional software developers agree that QA is important, we are still working out the best way to implement it. What seems clear, however, is that the ideal QA strategy should fit right into a modern continuous integration pipeline, and therefore be fast and reliable. Today, we'll discuss how to identify test cases that are ready for testing automation as part of your QA strategy, and how to gain the most from its benefits while minimizing its downsides.

CREATING A BALANCED TEST AUTOMATION STRATEGY

One of the choices you have to make when thinking about your QA strategy is between automatic and manual testing, and it's almost always a balance; there are few situations really suitable for a 100% approach one way or the other.

First, to get the obvious out of the way: some things, like unit tests, are unambiguous. You should always have some form of automated unit tests! Other things, like penetration testing and usability research, almost always require a human touch. The boundary between these two is where things get interesting.

So what are the trade-offs? Well, I'm glad you asked! That's something I tend to think about a lot, so let me try to give you a breakdown of what we've got.

FUNCTIONAL TESTS: TO AUTOMATE, OR NOT TO AUTOMATE?

Let's focus on functional testing, which in the world of web applications, usually means testing the UI and flow of your app. Once you have a working app with a UX flow you are comfortable with and ship it to production, you'll want it to have some functional test coverage. This coverage is designed to make sure you can safely iterate and improve it over time without breaking the ability of your users to achieve things. Should you test it using an automated or manual approach?

Automated tests are great, because executing them is free and they are usually fast. These are huge bonuses when you are trying to save development time, which is usually one of your most precious resources. Once you have, for example, a working Selenium script, you can run it as often as you want and get the results fairly quickly. The problem is that these tempting savings only come in after the initial (and often significant) investment of — you guessed it — dev time to write such scripts. Worse still, some automated tests tend to break unexpectedly as you change your application, so you need to continually re-invest into keeping them up-to-date.

Manual testing doesn't have this problem. It will generally

work as expected, giving you reliable results without you having to specify every tiny detail. The drawbacks, however, are significant: manual testing generally creates high, persistent costs and long turnaround times.

From this, some clear pictures start to emerge. If you have a fairly stable component that you need to test often, automation is probably the way to go. Every time you execute your test suite, you will rake in a return on your initial investment. Assuming the unexpected breakages don't happen too often, the continuous maintenance will not hurt you either, because you only need to fix your test scripts when things break. On the other hand, if you are testing something that tends to change frequently and especially if you don't test often (which you should be doing, but life is not always simple) you might be better off going through a manual testing phase before every release.

DETERMINING WHEN TO USE TEST AUTOMATION

The next question is, how can we assess when and how often something is going to break? This is a tough question to answer and it heavily depends on your specific situation. Luckily, smart people have done some work for us already! For instance, this paper discusses the most common ways in which automated tests of web applications tend to break (while they mostly talk about record/replay tests, the same principles apply to hand-crafted testing scripts). The authors created suites of automated tests for early versions of five different open source web applications, and then executed these tests for each subsequent version. Every time a test broke, they recorded the breakage reason, fixed the test and continued testing. This way, they created a very unique dataset, which makes it possible to explore the most common failure reasons.

Broadly, they describe five main types of test breakages. Thinking about these potential breakage risks should give you some idea which areas of your applications might be suitable for test automation, and which are better served by manual QA.

1. ELEMENT LOCATORS-BASED BREAKAGE

When your test script validates something, it usually needs to find and inspect a DOM element, whether by its attribute, text, or place in the hierarchy. Each of these ways can break: changing page styles, updating copy, or restructuring your layout can all threaten your ability to find the same thing reliably. This is by far the most common cause of breaking automated tests, and one that is very difficult to mitigate. This is one of the reasons why humans are still much more robust testers than machines!

2. VALUE-BASED BREAKAGE

When testing text input fields, you often want to make sure they accept the correct values and raise appropriate errors when the provided values are invalid. Whenever you change your backend, e.g. the requirements for passwords when testing a password input field, you need to update your test to prevent it from breaking. Similarly, if you add a new, required input field, your old automation script will try to submit a form with missing data unless you update it.

Another common case is asserting that a specific error message appeared and then updating the error message, which will also break the script.

3. PAGE RELOADING

Depending on your backend, going through some user flows might require page reloads. On the other hand, reloading a page at the wrong time can often break the test! Changes to your application can often result in broken reload login in test scripts, because of: 1) reloads being required at different points; 2) reloads no longer happening (while the script is stillexpecting them); 3) reloads potentially requiring different amounts of time then previously (since explicit "sleep" times are often used to allow the page to reload before continuing, changing the logic behind the reload might render the original wait time insufficient).

4. USER SESSION TIMES

Many applications implement logic to log out inactive users after a set amount of time and use test scripts that are intentionally inactive for a while and validate that the expected action (warning and/or logging the user out) happens when expected. Such tests are likely to break whenever the allowed session time changes.

5. POP-UP WINDOWS

Pop-ups, whether windows or alerts, are often used in web applications and need to be tested. It's therefore common for test scripts to assert either the presence or absence of pop-ups, which will break whenever the application is updated to change the relevant behavior. While the paper authors have found this to be one of the least-frequent reasons for broken tests, here at Rainforest we have seen our customers being affected by this particular issue often enough to develop a specific set of rules for testers to follow when testing pop-up behavior.

THE TAKEAWAY: LEVERAGE TEST AUTOMATION STRATEGICALLY

Test automation is great for development — it not only gives you a peace of mind about being able to serve your customers well and without hiccups, but also allows you to move at a much faster pace, because your team spends less time dealing with QA overhead and the consequences of bugs in production. When done well it can truly supercharge your development efforts, but it is not without pitfalls. Thinking carefully about your QA strategy and about which parts of your test suite can be automated will definitely pay off.

Maciej Gryka leads the Science Team at RainforestQA, overlooking the efforts to develop smarter testing algorithms, and manages the testing crowd and business intelligence activities. He holds a PhD in Computer Vision and Machine Learning from University College London and often speaks at international ML conferences about data science and related topics.

functionize

AI-POWERED TESTING FOR DEVELOPERS

Still messing with test scripts? Functionize provides completely automated testing — no coding required.

Functionize's AI-powered testing solution lets developers automatically generate, maintain and execute complex cross-browser test suites. Let our cloud-based machine learning technology simplify your QA effort. Spend time coding your product instead of your tests.

Learn More

demo@functionize.com
Functionize.com

Use AI For Easier Testing, Without Scripts

For years, developers and QEs have been writing test scripts, relying on various testing "automation" CI tools. At first, these tools were better than manual testing for the complex, cross-browser requirements all marketers now have. However, as site complexity and testing demands have grown, this coding approach to QA automation is failing.

The reality is that writing test scripts with Selenium or Robot requires too much time and maintenance. For developers and QA teams, scripting adds a high level of complexity — taking time away from the more important goal of completing products on schedule. Scripting is also cumbersome and slow, making it difficult to keep up with today's agile releases.

So, Functionize asked these questions:

1. Is coding test scripts really automation?
2. Shouldn't valuable developers be coding product, not tests?
3. Couldn't modern AI and cloud technologies improve testing automation?

Once we thought about these questions, the answers were obvious.

We believe developers and QA teams shouldn't have to write test code. Functionize provides a solution to the hassles and slowness of manual test scripting by applying AI-powered QA in the cloud.

Functionize dramatically reduces your testing effort with these key technologies:

1. **AI and Machine learning.** Test creation is now truly automated by letting machines do the work for you.
2. **Smart bot technology.** Let bots execute your tests for complete coverage of even the most complex sites and apps.
3. **Easy maintenance.** Maintaining tests with Selenium is a nightmare. With Functionize, it's a dream.
4. **Instant cross-browser scalability.** Our elastically-scalable SaaS solution lets developers run any number of tests in 20 minutes or less.

To Functionize customers, true automation means complete cross-browser testing, in the cloud, without having to write a a single testing script. Give it a try — we think you'll never go back.

WRITTEN BY TAMAS CSER
FOUNDER & CEO, **FUNCTIONIZE, INC.**

PARTNER SPOTLIGHT

Functionize

Functionize is AI-powered Testing for Developers. Create and execute complex cross-browser and mobile tests, in the cloud, without coding.

CATEGORY
Testing Automation, AI, cross-browser testing

CASE STUDY
A major SaaS CRM provider found coding test scripts to be expensive and inflexible. Developer and QE time was being wasted writing and maintaining brittle, hand-scripted test suites. Agile releases meant test coverage was always incomplete, with many UX details left unexamined. With hundreds of frequently-updated web and mobile pages across dozens of services, they needed a better way.

With Functionize, the company is transforming their QA practice: replacing hard-to-maintain testing code with AI-powered test creation and execution in the cloud. Dev teams now complete broad test coverage in minutes without using coding tools, like Selenium or Robot.

The result: Developer time spent on testing has dropped by 75%.

Test results that used to take days to generate now take 20 minutes.

NEW RELEASES
Continuous, SaaS delivery

OPEN SOURCE
No

STRENGTHS
- Automated QA in the Cloud
- Rapid test creation and execution
- Complete cross-browser testing
- No coding

NOTABLE CUSTOMERS
- Zenefits
- Riffyn
- Diamond Media

EMAIL demo@functionize.com | **TWITTER** @Functionize | **BLOG** functionize.com/blog

Testing Automation Without Writing Test Scripts

QUICK VIEW

01 Selenium solved many low-level problems, but didn't address the core problems we had with test automation.

02 We built a high-level test automation framework that enabled us to move from source code to data-driven testing.

03 Machine learning presents so many new opportunities for testing automation that it is safe to say it is the future of all testing automation.

BY **TAMAS CSER**
FOUNDER AND CEO, **FUNCTIONIZE**

Like many developers charged with creating browser-based testing for our products, we used to start by creating test scripts in a CI tool and writing a bunch of code.

Writing test scripts seemed like a step forward over existing approaches because scripting tools provided good support for the latest browsers. As our projects grew, however, we ran into many challenges, including time-intensive, on-prem set up and maintenance to run test "automation" tools, like Selenium or Robot, properly at scale. All the code we had to write and the amount of time engineers had to spend on testing kept on growing.

So, we decided there was room for improvement, and made some key discoveries about how the testing process could be improved. In this article, we will discuss the opportunities we found to improve browser-based testing, specifically the ability to speed up testing execution with cloud-based providers targeting multiple browsers, and providing better insight into applications under test.

OUR INITIAL CHALLENGE

We had several, highly visible projects with low tolerance for bugs and performance issues. Unfortunately, as is often the case, budget and resources for testing were limited. At the time, there were two options: hire manual testers or try to automate. The manual testing route would be too slow and inaccurate for the number of updates we were pushing, so we decided that we had to figure out a way to automate. We were building JavaScript-heavy applications, and traditional automation tools were both out of date and far too expensive for our needs. Selenium ended up being our choice because of its support for many different browsers, ongoing updates, and cloud-based runtime (as opposed to the outdated Win32 / desktop tools on the market).

We were excited to expand our usage of test scripts and built a framework around PageObjects that enabled us to get a high degree of modularity and extend the tests quickly. However, the more we dug into writing our own testing scripts, the more critical shortcomings we discovered in this approach. Ultimately, we embarked on a journey to take our own testing automation to the next level. It's this experience and our results that I would like to share in this article.

QUESTIONING ASSUMPTIONS

As we got further down the path of script-based testing automation, we realized that we had been making several false assumptions about the whole process.

Specifically, we started to question the following:

1. Was it actually a good thing that we had to spend so much time coding to create and maintain test cases, especially in a rapidly changing application?
2. In today's JavaScript-heavy applications with changing

contexts, multiple div layers, hovers, mouseovers, asynchronous behavior, callbacks, and more, simple selector-based scripting felt very two-dimensional in a three-dimensional world. Was this really the best we could do?

3. Execution was slow. Why was testing automation stuck in a pre-cloud, pre-elastic-scaling mode of operation?
4. Why did we have to write multiple classes in order to target the different browsers we cared about, for the exact same test cases?
5. We were also having to spend a lot of time creating load and performance tests. Given existing frameworks and cloud services, we were creating multiple instances of the same type of test for the same user workflow, simply to test across functional, load, and performance areas. This seemed odd.

Many more questions occurred to us, but these were the big ones that we felt needed to be solved. We looked at every product on the market, every open source project, and we couldn't find anything that was questioning this set of assumptions in the same way that we were.

STARTING DOWN A NEW PATH

We had an idea. We knew now that the coding-centric paradigm of script-based testing automation was simply too slow and primitive to scale to the needs of our dev team going forward.

The next step was to reimagine how testing automation should be done. We made a few observations:

1. You should be able to code but you shouldn't have to code in order to create and maintain test cases.
2. IT is moving to the cloud and this should include testing.
3. The cloud enables browser-driven workflows. Testing should be managed via a browser and it should be visual.
4. The cloud means more than just running on remote servers. It's about elastic scalability, and testing should never be a bottleneck.
5. Machine learning is democratizing, and it's possible for even small teams to take advantage of it in order to build systems that learn.
6. Browser and testing automation should be "write once, run anywhere."

With these observations in mind, we had the beginnings of a vision for how a new testing automation practice would look. Selenium still ends up being a part of the final solution — but our system produces the code, not the developer.

> "We decided that there was room for improvement and made some key discoveries about how the testing process could be improved."

WORKFLOW CAPTURE

Starting with our #1 observation, that you shouldn't have to code, it became clear that this is a revolutionary concept but that it has been tried many times before. One of the core value propositions of legacy Win32 testing automation platforms was that you don't have to code. The downside is that each user needs their own desktop license since this is on-premise software.

On the other hand, you have many projects that aim to create test cases in the browser, visually. They do a decent job for simple tasks, but their limitations become clear quickly when you start dealing with complex applications. We mentioned some of these challenges above in the "Questioning Assumptions" section but, suffice to say, it was impossible to find any visual recorder technology that let us capture sophisticated user workflows and edit those workflows again without having to re-record everything.

There are, it turns out, multiple ways of capturing user workflows in a novel way but we decided to focus on making recording work as a first step. And, as it also turns out, we massively underestimated how challenging this would be. At the time, we thought it was possible to build a recorder in three months that would handle 80-90% of what we needed.

Three years later, we finally had a tool that worked in the way we envisioned. There were far more edge cases to cover than we had originally imagined. Areas requiring extra attention included accurately capturing

mouse position after hover elements were selected, complex navigational menus, and automatically creating modules, similar to PageObjects, during the workflow capture process.

CROSS-BROWSER TARGETING

Another area of difficulty is targeting multiple browsers with a single workflow capture. It is easy to imagine how difficult this is, simply because when you're writing test code, you'll usually do this with entirely separate scripts. Our aim was to abstract this away entirely, providing what we call an Action Log for the entire test case and a browser-specific Action Log for each browser targeted by the test.

An additional hurdle was to enable the ad hoc addition of new browsers to an existing test case, even after the test case had already been created. We were able to extend our Action Log abstraction in this case and enable real-time screenshot and video capture along the way.

RUNTIME EXECUTION

As we mentioned above, the whole point of the cloud (in our opinion) is not simply about running tasks on remote servers but also to spin up quickly to scale elastically to any sized workload and then spin back down as soon as the task is complete. This single innovation has revolutionized computing and it was high time for it to revolutionize testing as well.

We were able to achieve this in our project by building a custom runtime scheduler on top of both Google Cloud and Amazon Web Services. Both cloud providers have their advantages and disadvantages. Our runtime scheduler allows our team to kick off a test suite execution of any size, guaranteeing completion in 20 minutes, max. This completely changed the game for us because we were tired of waiting for hours at a time for our test suites to complete on traditional, instance-based models.

REPORTING AND RESULTS

Another benefit of cloud-based, visual testing orchestration is that you move away from source code and into data. When you stop thinking in terms of Selenium scripts, you start thinking more about the data and insights you can derive from the testing process and how that might improve things overall. Key things that you can start tracking automatically are the number of test cases created by different engineers, how often tests fail (again by engineer), isolating systemic problems faster by categorizing test cases by modules, and more.

INTEGRATIONS

The future of cloud applications in general is integration, and the same is true of testing automation as well. The most important integration to get right is with a good continuous integration (CI) tool, such as Jenkins, CircleCI, or any number of options. One of the things we realized, however, was that there is still some bureaucracy in many IT departments in getting even simple integrations such as CI set up, so we also built and included a mini-CI tool inside our new platform. This allows end users and developers to quickly run an entire suite of tests and we included webhook-based APIs as well in order to provide lightweight programmatic access.

Other important integrations include repositories such as Github, alerting tools such as PagerDuty, and chat tools like Slack.

MACHINE LEARNING OPPORTUNITIES

The final big opportunity we found in our research was, again, tied to this shift away from a source code-centric model to a data-driven approach, which the cloud helps facilitate. There is a ton of data created by the testing automation process and it is a shame that so little of it is properly exploited in the code-driven model.

What we view as the holy grail for testing automation is a platform that can learn on its own from rapidly changing applications and adapt tests to those changes. This is a long road, even given current technology, but this is the big opportunity for the industry.

RESULTS SO FAR AND NEXT STEPS

In our experience with this new approach, we've had a lot of successes as well as many lessons learned. The biggest benefit we've seen so far has been in a dramatic reduction in time spent on creating and maintaining test cases. Developer teams that we've worked with on this thus far love this aspect: creating a new test case goes from a matter of hours or even days to something they can create in minutes. Updating a test cases is something that can done in 5 minutes or less.

Going forward, one of our bigger challenges is in helping quality engineering teams to see the benefits they can derive from a data-driven approach and a move beyond testing scripts as the core technology in the stack of testing automation. Full-stack testing automation is the future as far as we are concerned and our goal is to help as many teams as possible realize this vision.

Tamas Cser is the Founder and CEO of Functionize, the AI-powered testing automation platform for developers. Tamas built Functionize to revolutionize and streamline how websites and web applications are tested because of his frustration with the difficulties and learning curve involved with coding test scripts. His goal is to help developers slash the time and cost associated with QA.

Diving Deeper

INTO AUTOMATED TESTING

TOP #AUTOMATEDTESTING TWITTER ACCOUNTS


@martinfowler

@jezhumble


@angelovstanton


@VijayShinde


@SoftwareYoga


@jamesmarcusbach


@jcolantonio


@julianharty


@bridgetkromhout


@RealGeneKim

AUTOMATED TESTING-RELATED ZONES

DevOps
dzone.com/devops
DevOps is a cultural movement, supported by exciting new tools, that is aimed at encouraging close cooperation within cross-disciplinary teams of developers and IT operations. The DevOps Zone is your hot spot for news and resources about Continuous Delivery, Puppet, Chef, Jenkins, and much more.

Agile
dzone.com/agile
In the software development world, Agile methodology has overthrown older styles of workflow in almost every sector. The Agile Zone is your essential hub for Scrum, XP, Kanban, Lean Startup, and more.

AI
dzone.com/ai
The Artificial Intelligence (AI) Zone features all aspects of AI pertaining to Machine Learning, Natural Language Processing, and Cognitive Computing. The AI Zone goes beyond the buzz and provides practical applications of chatbots, deep learning, knowledge engineering, and neural networks.

TOP AUTOMATED TESTING REFCARDZ

Declarative Pipeline With Jenkins
To fully recognize the benefits of Jenkins Pipeline, this Refcard will help you build a Jenkins Declarative Pipeline and manage it with Blue Ocean. Learn the details of creating a Jenkinsfile, branches and pull requests, pipeline fundamentals, and using the agent directive.

Getting Started With Appium
In this Refcard you will learn everything you need to know about getting started with this open-source tool, from installing the Appium server to running your first tests. Download this Refcard now to see why Appium is "Mobile App Automation Made Awesome."

Deployment Automation Patterns
Covers seven useful deployment automation patterns, antipatterns and code snippets.

TOP AUTOMATED TESTING PODCASTS

Arrested DevOps
arresteddevops.com
A podcast that helps you achieve understanding, develop good practices, and operate your team and organization for maximum DevOps awesomeness.

Test Talks
joecolantonio.com/testtalks
TestTalks covers news found in the testing space, reviews books about automation and speaks with some of the thought leaders in the test automation field.

Testing Podcast
testingpodcast.com
Individual audio podcasts on software testing.

TOP AUTOMATED TESTING BOOKS

The Phoenix Project
by Gene Kim
Learn how to recognize problems that happen in IT organizations; how these problems jeopardize nearly every commitment the business makes in Development, IT Operations, and Information Security, and more.

Continuous Delivery
by David Farley and Jez Humble
Getting software released to users is often a painful, risky, and time-consuming process. This groundbreaking new book sets out the principles and technical practices that enable rapid, incremental delivery of high quality, valuable new functionality to users.

Automated Software Testing
by Elfriede Dustin, Jeff Rashka, and John Paul
This book teaches you the path to understanding how to apply microservices architecture with your own real-world projects.

Continuous Testing

Enable continuous testing across your software delivery lifecycle.

Adopt next-generation testing practices to test early, often, automatically, and continuously.

Only CA offers a continuous testing strategy that's automated and build upon end-to-end integrations and open source. Enable your DevOps and continuous delivery practices today.

Explore ca.com/continuous-testing

ca technologies

Drive Testing at the Speed of Agile

It's hard to believe that after 30 years, 70% of testing is still performed manually. A major bottleneck in the SDLC, legacy testing remains a barrier to speed and quality – unable to keep up with today's agile, continuous testing model.

But as more organizations adopt test-driven, agile development methods, they gravitate towards test automation – enabling test teams to automatically generate reusable test assets like test cases, test data, and test automation scripts right from requirements.

Model-based testing (MBT) helps you avoid costly defects that stem from poor requirements. It enables you to automate testing activities, shortening testing time dramatically. And so, your high-quality apps are delivered faster at lower costs.

The question then becomes, in what ways does MBT drive test automation effectiveness?

Start with modeling requirements as an active flowchart, versus writing them in inefficient text-based methods. With 64% of defects coming from poorly-defined requirements, modeling using a flowchart eliminates unclear requirements – while boosting collaboration and communication.

Next, generate optimized sets of test cases automatically. This means creating test cases, test data, and test scripts automatically, right from the flowchart as user stories are created, and testing the functionality at maximum coverage with the smallest set of tests.

Finally, automate the 'change in requirements process.' This cuts the time wasted on manually finding and fixing tests when requirements change, because as changes occur they automatically initiate impact analyses and create or repair tests to maintain test coverage – while building up a library of reusable test assets that can be run or rerun as test automation artifacts.

CA offers comprehensive solutions that automates the most difficult testing activities – from requirements engineering through test design automation and optimization. These capabilities help you test at the speed of agile, enabling you to build better apps, faster.

WRITTEN BY GEDEON HOMBREBUENO
PRINCIPAL PRODUCT MARKETING MANAGER, **CA TECHNOLOGIES**

PARTNER SPOTLIGHT

CA Agile Requirements Designer

Improve software quality, reduce testing costs, and speed up application delivery.

CATEGORY
Continuous Testing

NEW RELEASES
Continuous

OPEN SOURCE
No

CASE STUDY

Williams is a Fortune 500 energy infrastructure company, providing natural gas processing and transportation. Based in Tulsa, Oklahoma, it employs 5,600 people through operations across the US.

To keep pace with business demands, Williams needed to deliver higher quality applications and updates more quickly and with fewer defects. Manual testing processes were hampering its ability to achieve this.

Williams implemented a suite of CA Technologies solutions to automate and improve software testing. CA Services also provided customization, education, and implementation support to provide end-to-end release management.

Williams has improved the speed and quality of software delivery, which frees up resources for new business projects, and will help the company achieve its business goals of providing the best service for less cost.

See their story here.

STRENGTHS

- **Requirements Engineering**. Map requirements to a visual, active flowchart, and reduce requirement ambiguity by 90 percent and software defects by 56 percent.
- **Test Design Automation**. Automatically generate the smallest number of test cases needed for 100 percent functional coverage and test automation scripts, linked to the right data and expected results.
- **Enable Agile Testing**. Automatically generate test cases, test automation scripts, and test data for all functionalities being delivered in every sprint.
- **Test Case Optimization**. Import test cases, remove any duplicates, and shorten test cycles by 30 percent.
- **Manage changing requirements**. Automatically identify the impact of a change and update test cases in minutes.

NOTABLE CUSTOMERS

- Williams
- GM Financial
- Sprint
- Citigroup
- Level 3 Communications

WEBSITE ca.com/us/products/ca-agile-requirements-designer.html

TWITTER @CAinc

BLOG ca.com/en/blog-highlight

Building Testable Apps

QUICK VIEW

01 Automated testing at the code level isn't something that can be "bolted on" after your code's complete—you need to start with testable code from the beginning.

02 Ensure you have clear acceptance criteria and great communication with your customers.

03 After that you need to focus on clean design, effective dependency management, and good craftsmanship principles like SOLID.

BY **JIM HOLMES**
EXECUTIVE CONSULTANT, **PILLAR TECHNOLOGY**

TAKING YOUR APPLICATIONS' TESTABILITY TO THE NEXT LEVEL

Nearly all successful teams understand at least the fundamentals of testable system code: good automated unit and integration/API tests have finally reached the point of being an accepted—or even required—part of a solid delivery process. What's still missing with many teams is an understanding of how small changes to systems and user interfaces can dramatically improve test automation at the functional level.

Because of its focus on business value, functional system testing is one of the most critical parts of a project's overall quality and value delivery. Wrapping this critical, high-value testing in sensible automation helps project teams ensure they're keeping their overall.

MAKING THE CASE FOR TESTABILITY

Getting testing involved earlier isn't just about reducing the cost of testing later in the project. It's also about making testing easier from the start. Far too many teams miss the fact that making testing easier can be dramatically impacted by making the system itself easier to test. High-performing teams look at testability as one of their fundamental design considerations.

DEVELOPER-LEVEL TESTABILITY

Good software design focuses on simplicity and maintainability. Testability below the UI layer dovetails with good craftsmanship principles and practices like low complexity, dependency injection, and low coupling. Test-driven practices drive out this simplicity by their very nature; however, good design practices keep the system more testable regardless of when test automation is completed.

EASING FUNCTIONAL AUTOMATION WOES

Functional test automation is by its very nature far more complex and brittle than automation at the unit or integration level. Not only are there complexities of logic, dependencies, etc., the very technologies and toolsets for user interfaces drop in additional challenges for automation. These challenges often leave teams with brittle, low-value functional automation tests.

Thankfully, a few simple approaches can greatly improve testability, making it far easier to have high-value, low-maintenance automation suites that check the system's functionality. The approaches listed here start out with simple techniques for interacting with the UI, escalate to simplifying asynchronous situations, and finish off with complex configuration of the system.

STARTING EASY: LOCATORS

Every automated functional test tool, regardless of platform, relies on finding things to interact with on the user interface: buttons, fields, text, controls, etc. The testing tool has to locate those objects to click them, inject text, compare text, and numerous other actions. Exact mechanics for this location process vary greatly across platforms; however, the concepts are the same regardless.

Various properties, attributes, or metadata can be used for

this location process—appropriately, these are referred to as locators.

GOOD IDS

HTML gives one of the simplest mechanics for good locators: the ID attribute. Generally speaking, IDs are fast for tools to locate, they're unique on a page (if the page is valid HTML!), and they're also very easy for developers to customize.

As an example, consider a table used to display contacts from a Customer Relationship Management (CRM) system. Adding an ID to an element on the page is normally a very simple task for a developer working on the page's code. The result may not seem earth-shattering, but it makes all the difference in the world for test automation:

With this simple addition in place, one statement will enable a WebDriver script to quickly locate the table:

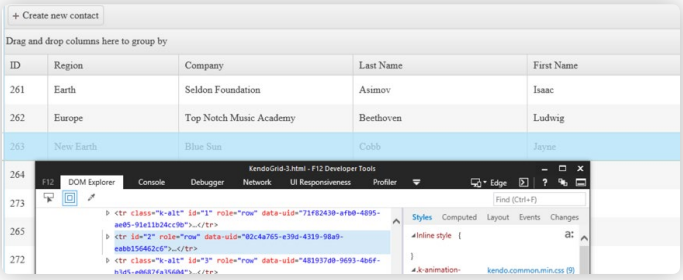
```
WebElement table = browser.FindElement(By.Id("contacts"));
```

Again, the mechanics for implementing the HTML attribute vary by platform. Controls on the page may offer the ability to easily add attributes to their rendered output. Various frameworks offer great control over their output as well.

DEALING WITH DYNAMIC IDS

While HTML IDs are a terrific locator, you can't use them thoughtlessly. Sometimes they're not the best locator, especially if they're dynamically created.

Going back to our example of a contact list, imagine a grid/table control that dynamically creates IDs for each row in the grid.

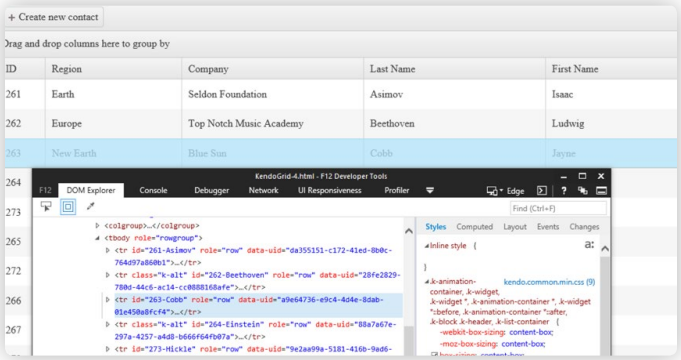


Suppose you're writing a test to confirm data for the contact Jayne Cobb. A WebDriver statement to locate that row, given the above HTML, could look like this:

```
WebElement JayneRow = browser.FindElement(By.Id("2"));
```

That would work well for the first run, but what would happen if the source data changed? The row you're looking for would likely appear elsewhere in the grid—your script would break and your test would fail.

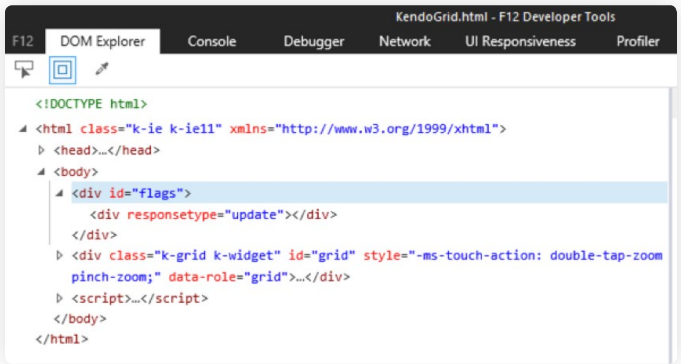
A short conversation between testers and developers can solve this. The underlying system can be altered to render an ID that contains information pertinent and specific to the data you're looking to locate. In the case below, the ID attribute is a composite of the contact's ID and last name.



This dramatically changes the functional test script, enabling us to locate by the desired test row's data:

```
WebElement JayneRow = browser.FindElement(By.
CssSelector("[id$=Cobb"));
var contactSource = new kendo.data.DataSource({
            requestEnd: function (e) {
                var node = document.
getElementById('flags');
                while (node.firstChild) {
                    node.removeChild(node.firstChild);
                }
                var type = e.type;
               $('#flags').append('<div responseType=\'' +
type + '\'/>');
            }
```

This snippet causes an empty DIV element to appear on the page with the specific action (post, update, delete, etc.) as an attribute within the element:



Now it's easy to use your functional testing tool to wait for the actions to complete. Pseudo code for a test using this approach might look like:

```
//Navigate to screen
//Locate and edit an existing contact
WebDriverWait wait.Until(ExpectedConditions.
ElementExists(By.CssSelector("#flags >
div[responseType='update']")));
```

TACKLING THE HARD THINGS: SYSTEM CONFIGURATION AND SERVICES

How do you solve hard problems in testing? Apply the same smart, thoughtful engineering approaches you do for solving hard software problems. Too often teams forget they can make major system changes to ease testing burdens.

Creating software switches to deactivate areas of the system that are hard to test is one of the best ways to improve testability. Creating stub or mock services is another.

Approaches like this take time and hard work, but they make total sense for high-value, high-risk areas of the system.

FEATURE SWITCHES

Experienced test automation folks regularly get asked "How do you write test scripts for CAPTCHA?" A common response is "Don't."

By its very nature, CAPTCHA is meant to prevent automation. This is why it's used for guarding things like user registration or account creation. Trying to write test automation scripts to deal with it is crazy; however, the functional slice of creating a new account absolutely has to be covered. How to deal with this gap?

Easy: Cheat.

Teams can work to create a feature switch within the system itself that can totally disable CAPTCHA and enable the registration/creation process to proceed without having CAPTCHA as part of the flow. As mentioned earlier, this can be hard work, and it brings its own complexity to the situation. However, if the team has decided this particular flow merits the investment, then it's worth it to have a configuration file, API service endpoint, or some other means that turns off CAPTCHA. Obviously similar code will be needed to return the system to its normal state...

This same approach can be used for similar concepts:

- **Third-Party Controls**. Don't write tests for those controls—the vendor should have tested them. If those controls are hard to interact with, then use a feature switch to swap out for an easier route. TinyMCE has long had a history of being hard to automate. Swap it out for a simple text box.
- **Email**. Need to check validity of outgoing mails? Don't ever, EVER use a functional test to log on to Gmail or some other service. Instead, configure an SMTP sink such as NDumbster or similar tool.

STUB OUT ENTIRE SERVICES

What is your functional test flow really dependent on? How many external services do you really need if you're focused on the high-value business related part of your test?

For example, consider a check to ensure your order system enables you to search for a part, add that part to your shopping cart, and proceed to check out. There are a number of ways to consider testing this, but let's take these considerations to work with:

- User must have an account created, and be logged on as that user
- User searches for a specific part known to be in the system
- User adds that resulting part to their shopping cart
- User proceeds to checkout. Part remains in shopping cart.

This test is not about checking out, it's confirming you can search for a part, add it to the cart, and head to checkout. This test is also not about validating searches, it's about adding a result from the search.

There are a number of things we don't need to concern ourselves with as part of this test:

- **Logon**. Tested elsewhere and is not central to the functionality of the search results to cart flow.
- **Part Search**. Again, should be tested elsewhere and isn't central to the flow.

Spending the time to stub or mock out these services would be a perfect use of a team's time. The overall flow is high-value, so it needs wrapping with an automated check. Stubbing the services would let the team write the automated tests to work in any environment the mock could be established in—another great advantage, as external systems often aren't available in lower environments for development and testing.

CLOSING

Functional testing is one of the most critical aspects of software development. It focuses on the end user and business needs. Taking time to make systems more testable at the functional level reaps high rewards for teams disciplined and supported enough to do the work.

Jim Holmes is an Executive Consultant at Pillar Technology, where he works with organizations to improve their software delivery process. He's also the owner/principal of Guidepost Systems, which lets him engage directly with struggling organizations. He has been in various corners of the IT world since joining the US Air Force in 1982. He's spent time in LAN/WAN and server management roles in addition to many years delivering great systems. When not at work you might find Jim in the kitchen with a glass of wine, playing Xbox, hiking with his family, or practicing guitar.

Introduction to App Automation for Better Productivity and Scaling

QUICK VIEW

01 Begin to understand how to efficiently and intelligently analyze code stability in the SDLC.

02 Understand different methods of performing test automation and how to incorporate them into a DevOps process.

03 Learn the limitations of manual regression testing.

BY **SOUMYAJIT BASU**
QA/DEVOPS ENGINEER, **3Y3 DIGITAL LABS**

Test automation can improve the development process of software in several cases. The automation of tests are initially associated with an increased effort, but the related benefits will quickly pay off. Automated tests can run fast and frequently, which is very cost effective for software products with a long shelf life, and, therefore, high maintenance costs. When testing in an agile environment, the ability to quickly react to changing software systems and requirements is necessary. New test cases based on updated features can be added simultaneously to automation in parallel to the development of the software itself. The automation process also leads to a shorter development cycle combined with better quality software, and thus the benefits of automated software testing help quickly outrun the initial costs.

SOFTWARE TESTING LIFE CYCLE (STLC)

The STLC is made of several phases of software testing that are followed in a sequential manner to ensure better software quality and delivery. The phases of a STLC process includes Test Strategy, Test Development, Test Execution, Defect Management, and Delivery. Essentially, "Automation" for the application happens in the Test Strategy and Test Development phase.

In the Test Strategy phase, the technical and the functional aspects have to be chalked out with respect to automation development. In the Test Development phase, the development of the automation framework or application starts with the proper documentation of the functionalities that need to be automated as a guide, along with the technical approach that needs to be taken in order to create the automation code block.

LIMITATIONS TO MANUALLY CHECKING REGRESSIONS FOR AN APPLICATION

In today's era of software development, especially following the agile model, deployment of functionalities to an application has become very regular. So, keeping track of all existing functionalities and the new functionalities coming aboard is very tough, unless we have some application do the job for us to check the regression of the application's deployment over a certain interval of time.

REGRESSION: CONCEPT AND TYPES

Regression is a term that states that the existing modules within the software system should not break when introducing new modules within the application. Regression can mostly be classified into three types:

1. **Functional regression**: Running an entire regression across the application to determine if all the functionalities are working fine.
2. **Unit regression**: Running specific regressions on unit modules to check if each module making up the entire application is working properly.

3. **Visual regression**: Visually checking on the layout of an application on the web or mobile and all the components on the web and mobile.

CI IS IMPORTANT TO THE DEVOPS PROCESS

When it comes to testing in DevOps, Continuous Integration plays an important role in the software release process. The main concern of Continuous Integration is to resolve integration problems. In Extreme Programming, Continuous Integration was intended to be used in combination with automated regression tests written through the practices of automated test-driven development. Initially, this was conceived of as running all regressions and tests in the developer's local environment and verifying they all passed before committing to the mainline. This helps avoid one developer's work-in-progress breaking another developer's work. In the same vein, the practice of Continuous Delivery further extends Continuous Integration by making sure the application checked in on the mainline is always in a state that can be deployed to users and makes the actual deployment process very rapid.

Continuous Integration thus provides an important aspect in today's era of software development, as it helps to recover integration bugs and detect them as early as possible, keeing track of the application's stability over a period of time.

Setting up a Continuous Integration process can be achieved through tools like Jenkins, TravisCI, Atlassian Bamboo, and CircleCI. Jenkins and Travis CI are open source.

Reference: Setting up a CI job using Jenkins

TEST AUTOMATION - A CRUCIAL ASPECT IN THE DEVOPS PROCESS

A matured testing process is a key differentiator of the overall DevOps infrastructure. A test automation platform, whether it be a test data driven framework or a behavioral driven framework, plays a pivotal aspect in the DevOps process, not only to ensure or verify changes in the code structure or that the code gets integrated properly, but also by ensuring that the stability of the software is not hampered. Continuous testing is a connecting link between the development and operational flow. The operational aspects come into the flow once the continuous testing gives a "Go Green". The development phase finishes only when the continuous testing has all its tests passed and the development phase has an assurance that the new code merged has no negative effects on the existing codebase.

AUTOMATION PRACTICES AND TRENDS

These are a few of the development trends being followed today:

- Behavioral Driven Development (BDD)
- Test Driven Development (TDD)
- Visual Regression
- Executing automated tests on the cloud
- Headless automation

> "The development of the automation framework or application starts with the proper documentation of the functionalities that need to be automated as a guide, along with the technical approach that needs to be taken in order to create the automation code block."

BEHAVIOR-DRIVEN DEVELOPMENT

Behavior-Driven Development, or BDD, is a synthesis and refinement of practices from the Test Data-Driven Development (TDD) and Acceptance Test Data-Driven Development (ATDD). It utilizes a format that is known as the Gherkin language, which helps in automating test scenarios in a behavior-specific format where the tests are formatted in simple English based on the behavior of the elements in the application. The expected benefit of using BDD is that it offers more precise guidance on organizing the conversation between the developers and testers and the domain experts. The notations originating in the BDD approach, in particular the given-when-then canvas, are closer to everyday language, which everybody on the team can understand. Besides, the tools and frameworks targeting a BDD approach generally afford the automatic generation of technical and end user documentation from BDD specifications.

TEST DRIVEN DEVELOPMENT

Test-driven development (TDD) is an evolutionary approach to development that combines test-first development where you write a test before you write just enough production code to fulfill that test and refactoring. One goal of TDD is specification and not validation. In other words, it's one way to think through your requirements or design before you write your functional code. Another view is that TDD is a programming technique. As Ron Jeffries likes to say, the goal of TDD is to write clean code that works.

> "Continuous testing is a connecting link between the development and operational flow."

VISUAL REGRESSION

Visual Regression is a technique that is utilized to automate the process of capturing UI/UX-based issues over the long-run for multiple devices. This can be achieved by using different tools and techniques. Visual Regression can take a very important part in the delivery process.

A visual regression testing tool performs front-end or UI regression testing by capturing the screenshots of web pages/UIs and comparing them with the original images (either historical baseline screenshots or reference images from the live website). If the new screenshot deviates from the baseline or reference screenshot, the Visual Regression tool sends notifications about the test failures in an HTML report. In the context of web applications, Visual Regression tools perform visual regression tests mainly on CSS changes. Some of the frameworks and libraries that can be used to carry out Visual Regression are Galen, Wraith, Backstop JS, and PhantomCSS. These are all open source initiatives and can be used directly by installing their respective binaries. There are other licensed tools as well, including the popular Applitool.

Reference: *Visual Regression using Galen*

RUNNING AUTOMATION TESTS ON THE CLOUD

BrowserStack is a cloud-based cross-browser testing tool that enables developers to test their websites across various browsers on different operating systems and mobile devices without requiring users to install virtual machines, devices, or emulators.

Reference: *Setting up your automation on BrowserStack*

There is another well-known cloud platform where you can run your tests, called SauceLabs.

HEADLESS AUTOMATION

Headless refers to the running of automated tests without allowing users to see any direct browser interaction, so most of the functionalities run from the backend. So, basically, a headless browser is a web browser without a graphical user interface. Google started using headless browsing in 2009 in order to index the content of websites using Ajax.

Headless automation can be achieved using PhantomJS, CasperJS, or by using SlimerJS. The concept of headless interaction was brought up by Ariya Hridayat, who designed a webkit that could run like a browser without having a UI. PhantomJS is available with Selenium and can be easily used with Selenium as well.

References:

- *Headless automation using CasperJS*
- *Headless automation using Selenium and PhantomJS driver*

CONCLUSION

All kinds of testing in the STLC can be handled through automation, depending on the project and other aspects such as ETA, sprints, deployments, etc. But the quality control team definitely needs to do an ROI before even starting automation because the client needs to be sure of the fact that the time and effort invested in developing an automation framework will benefit the project life cycle. But with my experience in the IT industry, automation does give an edge to continuous project management and delivery. It is quite scalable because you can use any language to get started with an automation framework because you have APIs relevant for each library and package.

Soumyajit Basu is a QA/DevOps engineer at 3Y3 Digital Labs. He is a quality keeper by profession and has a diverse knowledge on test automation in the domain of Java, JavaScript, and Python. Soumyajit enjoys helping organizations adopt test automation techniques, continuous integration, and promoting and utilizing the cloud for test automation techniques. He also has a passion for writing blogs here and is an article contributor on DZone.

33

IS THE ANSWER.

OR IS IT?

Thirty-three is the number of web platforms required to achieve 80% market coverage.

But if you consider previous, current and beta browsers in a responsive web environment and factor in the number of possible OS versions maybe that's not enough? Read on to find out!

Perfecto

perfectomobile.com/is33theanswer

Why is 33 the Answer?

The strategy pillars for testing responsive websites are clear.

1. Automate (Duh)
2. Use the open source Selenium framework, or one based on it
3. Use traffic analytics to determine appropriate lab size
4. Extend functional test automation to include visual verification across break points
5. Embrace TDD and then progressively test every build, every merge, every night
6. Deliver fast feedback

While much has been written about the need to automate and the de facto automation framework of choice — Selenium — more work is needed to define guidelines on building the right strategy.

Most cite the key advantages of a single code base and consistent user experience across platforms. These advantages also create a multiplier effect on testing:

1. Platform matrix (web and mobile)
2. Visual validation across CSS break points, and
3. Differing web and mobile navigation options

Selenium is well suited to address most of these challenges – high volume parallel execution. Most teams look for extensions to visual validation and a new strategy for enabling efficient analysis of test results able to connect to the defect management tool of choice. The Achilles heel undermining confidence is a reliable lab offering supporting real mobile devices and web browsers matched to user traffic.

So why is 33 the answer? Thirty-three is the number of web platforms required to achieve 80% market coverage. But think about applying a "current, beta, last" approach across five browser platforms forgets about mobile devices. So, full coverage requires adding in thirty-two mobile devices. Turns out the answer isn't 33, it is 55!

WRITTEN BY CARLO CADET
DIRECTOR OF PRODUCT MARKETING, **PERFECTO**

THE CROSSROADS OF TESTING

Automated testing is seen as one of the key components of achieving Continuous Delivery in an organization. Thought leaders often suggest to automate everything. However, as of right now, not everything can, or should, be automated, particularly tests that rely on real-user input. So, developers stand at a crossroads: which tests can be automated now, and why? We asked almost 400 developers about which tests they performed manually and which were performed automatically.

MANUAL TESTING

USER ACCEPTANCE TESTS ensure that a user is satisfied with a product's function. These can be used to test user stories and ensure that they have been implemented correctly. Since this is based on the subjective opinions of end users, this is not conducive to automation.

78% of respondents perform manual user acceptance tests.

USABILITY TESTS These are tests that are performed directly with users to determine how easy it is to use a piece of software. Because results rely on the opinions of real users, this cannot easily be automated.

70% of users perform manual usability tests.

POST-DEPLOYMENT TESTS can vary between smoke checks to ensure the application is running and testing any major bugs that become apparent once the application is live. Because of the troubleshooting and unique circumstances involved, this is not an easy task to automate.

63% of DZone's audience perform manual post-deployment tests.

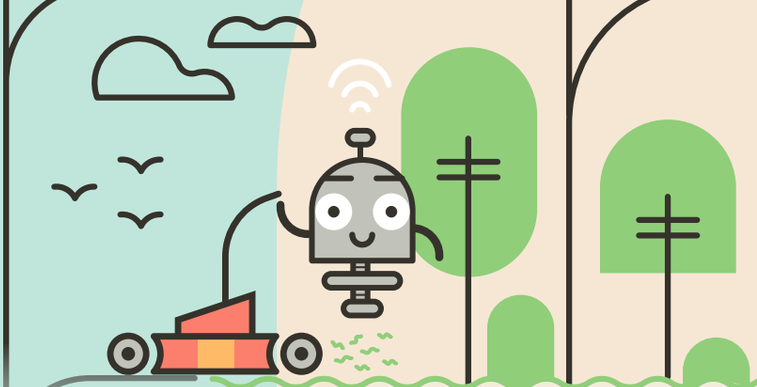
AUTOMATIC TESTING

Also called component tests, **UNIT TESTS** take individual pieces of an application's source code, called units, and ensures they are operating properly. Since these units don't rely on external dependencies, automating them is a relatively straightforward process.

58% of respondents perform automated unit tests.

INTEGRATION TESTS are performed when two pieces of software are combined and tested as a single unit. It should be an easy task to ensure that two pieces of software can communicate between each other, and can be easily automated.

61% of readers perform automated integration tests.



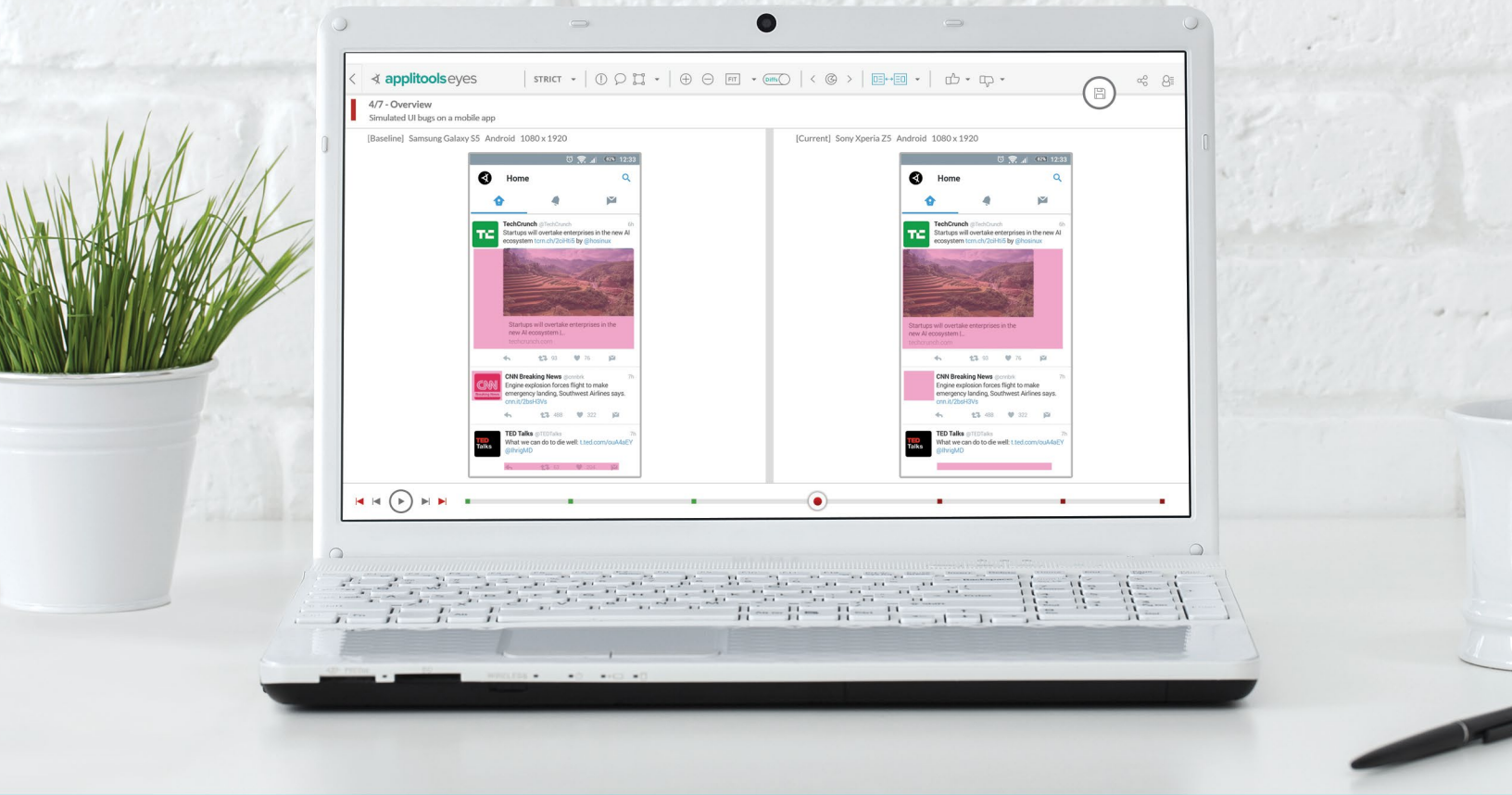
PERFORMANCE TESTS determine the speed or effectiveness of an application or network. Performance tests may be manually performed to determine the source of a performance bottleneck or diagnose an issue, but automated performance testing is useful to get a consistent, up-to-date picture of an application's performance. task to automate.

55% of users automate performance tests, while **45%** perform them manually.

applitools

AI powered Visual Testing & Monitoring

Deliver a flawless user experience across all devices, browsers, and screen sizes with automated visual testing



Start visual testing now - click here to open your free account

Trusted by

AMERICAN EXPRESS · hp · LVMH MOET HENNESSY · LOUIS VUITTON · intuit · CODE · cisco · MasterCard · SONY · salesforce · WELLS FARGO · SAP · WIX

When Responsive Web Design Goes Bad!

3 Most Common RWD Testing Mistakes That Will Ruin User Experience and Harm Your Brand

According to reports from Google and IBM, half of web traffic comes from mobile devices. Combine that with the fact that 4 out of 10 purchases involve multiple devices, and it's obvious why top brands are adopting a responsive web design (RWD) approach.

However, developing responsive apps and sites – and putting them through rigorous testing — is an elaborate process with many potential pitfalls.

Here are the 3 most common mistakes developers and testers should avoid before releasing responsive apps and sites:

(1) RUNNING FUNCTIONAL TESTS – BUT NEGLECTING TO TEST VISUAL ASPECTS

Functional tests confirm that features work according to the code, but they can't validate that the UI is aligned correctly across all browsers and devices, nor can they ensure that text, links, and buttons are perfectly visible.

And while functionality can be tested within a limited set of environments, UI should be tested across all environments, screen sizes, form factors, browsers, and operating systems, validating all possible changes in fonts, colors, text alignment, and position changes from portrait to landscape mode, and anything else that's visible to your customer.

This is where visual testing comes into play in full force — and it's your best line of defense against UI defects, ensuring that all UI elements appear correctly to the end user, across all the devices and layout modes you are testing.

Key Takeaway: Visual Testing is the most efficient method to ensure the UI has not broken across environments; it's best to use automated tools that support CI-CD processes. Try it out here.

(2) FORGETTING THAT NOT ALL MOBILE DEVICES ARE CREATED EQUAL

When doing responsive web design, developers will define different layouts for different types of devices (i.e. mobile, tablet, laptop/desktop), but will sometimes use one layout for all smartphones, even though smartphones vary in size.

This means that using the same RWD layout for iPhone 4 and iPhone 6+ can potentially cause text and images to display incorrectly. It is recommended to always test on the smaller end of the range, simply because images tend to get cut off or misaligned on smaller screens. Smaller devices do tend to be older, but that's no reason to neglect them in your responsive web design process.

Key Takeaway: When designing and testing your responsive website for a smartphone, always consider the smallest screen size in your size range.

(3) NOT TESTING ON REAL, CONNECTED MOBILE DEVICES

As mentioned, mobile devices come with various screen sizes and resolutions and are up against a daily grind of ever-changing user conditions, such as: switching networks, phone call interruptions and heavy user load. These factors will affect how your website appears on the screen.

Different devices have different OEM add-ons, customizations, variations (which affect screen size, breakpoints, pixel ratio, and view-port size), and in addition the physical size of the device affects the overall display size. Emulators cannot account for all these variables, therefore give a very limited account of what your customer will experience when using your app.

Key Takeaway: While emulators are good for basic testing, they cannot mimic true user experience. Test your app on real devices - and on as many of them as possible.

To learn more, watch this webinar on Advanced Test Automation Techniques for Testing Responsive Apps and Sites.

WRITTEN BY MOSHE MILMAN - COO & CO-FOUNDER, **APPLITOOLS**

applitools
AI powered Visual Testing & Monitoring

Selenium Grid Using Docker

QUICK VIEW

01 Traditional Selenium grids made out of individual VMs present maintenance headaches and scalability issues.

02 Running Selenium Grid on individual VMs in the cloud can be costly.

03 Docker gives us easier way to spin up Selenium grid on a single VM.

04 Docker vastly improves the efficiency of the automation and proves to be an ideal environment for distributed testing.

BY **SLAVEN SLUGIC**
LEAD SOFTWARE DEVELOPMENT ENGINEER IN TEST, **SMARTSHEET**

Docker has vastly improved the efficiency of automation and provides an ideal environment for distributed testing. Before Docker containers, spinning up and maintaining selenium grids was time consuming and required constant maintenance as the machines had to be kept up and running at all times. The introduction of Docker containers removed the need for this maintenance. Automation has been able to scale at a much faster rate. Using Docker with EC2 instances allows us to provision the VMs, spin up the grids and tear them down all in a single automation run in less than two minutes.

We began working on UI automation about 3 years ago and one of the first tasks for the automation was to spin up a Selenium grid. Although our Selenium grid consisted of only two virtual machines, it was essential to have it up and running so our tests could run against it. We quickly realized that this would only be a temporary solution and that we would have to figure out a means to distribute our tests to increase the execution speed. As we continued growing our test base, we also increased the amount of virtual machines on the grid. At the beginning of 2017, we had about 40 Amazon EC2 instances running so that we could accommodate fast execution of the 4000 or so UI regression test cases that we have. We managed to get the full regression run to finish in under two hours, but we ran into limitations with this approach, and while it was viable, we had to figure out a better way to tackle this problem.

As the company grew, the demand for full regression runs by QA personnel increased with it, and we began running into scaling issues. Every time a QA Engineer ran a full regression it would utilize all 40 of our virtual machines, which meant that if anyone else wanted to run automation at this point, they would have to wait two hours. Spinning up additional Selenium grids was also not an ideal solution, as it would require quite a lot more resources on top of the headache of maintaining the latest Selenium binaries and drivers on each and every one.

On top of this, running so many EC2 instances at all times would become expensive very quickly. This ultimately lead to the realization that this approach is not the solution, so we began to look for a better one.

Docker is a platform that allows you to run containers within the same virtual machine. You can think of these containers as isolated virtual environments. Without going too much into Docker, after researching it we realized that it was what we needed all along. We wanted to be able to provision a virtual machine and run Selenium Grid inside Docker containers. This will allow us to have only a single virtual machine with the same Selenium Grid that required 40 virtual machines before. Each Selenium node would now be represented by a Docker container. If we needed 20 nodes we would just run 20 Docker containers within seconds. At last we no longer have to worry about scalability of our Selenium grids. Each time a tester wants to run a full regression now, we will provision a new EC2 VM and spin up brand

new Selenium Grids running as Docker containers. Once automation is finished with the full regression run, we will terminate the EC2 instance. This solution solved the maintenance, cost and scalability problems we were running into. Since everything is now done on demand, we no longer need the EC2 instances to be permanently up. If you need to update your Selenium Grid deployment, Docker has a simple definition file called a Dockerfile that contains all the commands the user needs to assemble the image.

The Selenium community has already written everything for us so there is no need to write your own Dockerfile from scratch unless you'd like to personally customize it. This makes it super easy to spin up your own Selenium Grid using Docker within minutes. The following snippets of code are pretty much all you need to do to have your grid up and running, assuming you have already provisioned your EC2 instance and you have Docker installed on it.

THE FOLLOWING COMMAND WILL RUN THE SELENIUM HUB:

```
docker run -d -p 4444:4444 -v /dev/shm:/dev/shm -e
GRID_TIMEOUT=30000 -e GRID_MAX_SESSION=5  --name
selenium-hub selenium/hub:3.4.0-dysprosium
```

THE FOLLOWING COMMAND WILL RUN THE SELENIUM NODE:

```
docker run -d -e NODE_MAX_INSTANCES=5 -e
NODE_MAX_SESSION=5 -v /dev/shm:/dev/shm --link
selenium-hub:hub selenium/node-chrome:3.4.0-
dysprosium
```

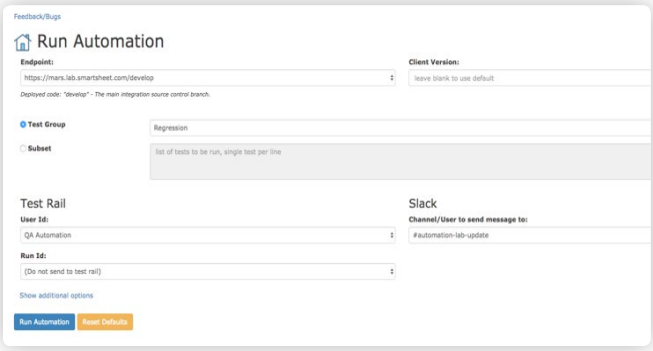
If you want to run 20 nodes then you will have to repeat the command above to connect more nodes to the hub. This can become a bit tedious, so instead you will want to write your own Docker compose file. Once you have a Docker compose file, you will run something like the following command to scale up:

```
docker-compose scale chrome=15 firefox=15
docker-compose.yml
hub:
  image:selenium/hub
  ports:
    -"4444:4444"
firefox:
  image:selenium/node-firefox
  links:
    -hub
chrome:
  image:selenium/node-chrome
  links:
    -hub
```

This command will spin up 15 instances of Chrome and 15 instances of Firefox.

As you can see, within a few commands and a little bit of knowledge of Docker, you can completely transition to Docker and save yourself from the problems we ran across. Now, our testers can run almost a limitless number of regressions and will not have to worry about whether or not our one and only grid is available to use. With Docker in place, we can now begin working on new improvements, such as reducing our full regression time from two hours to possibly 30 minutes or less! With Docker, scaling has become a non-issue, which has opened many other opportunities for improvement that may not have been possible before.

Currently our process is very simple and straightforward. Running automation is just a click of a button and testers don't have to worry about anything but the results at the end.



Our custom automation control board allows our testers to specify the branch, test group and TestRail information. We use TestRail for test case management and we automatically update their test runs when the automation is finished.

When you trigger the run, we will provision a brand new EC2 instance and spin up the Selenium Grid with N number of nodes depending on the size of the test group you are running. As you can see, using Docker along with Selenium gives us full control on how we want to set up our grid at run time.

Slaven Slugic is a lead software development engineer in test working at Smartsheet. He has been in the industry for about 11 years and has worked for companies like Microsoft, Expedia, and Walt Disney. Slaven graduated from Western Washington University in 2006 with a Computer Science Degree.

PLUTORA TEST

Next-Generation Enterprise Test Management

Selenium and JIRA Integration

Integrate with Selenium to pull in your existing library of test automation scripts, and run those scripts on any browser directly in Plutora Test. Set up a two-way sync with JIRA for both requirements and defects so developers can keep using their own tools while testers view progress in Plutora Test.

Reporting and Insights

Get unprecedented insights into software quality and delivery pipelines with comprehensive metrics, reports, and leaderboards. Measure progress with complete visibility across projects and teams, and update stakeholders with crisp-looking yet comprehensive reports.

Requirements Traceability Matrix

Drill down into live data with a single view of requirements, test cases, and defects. No need to export to Excel and wrestle with pivot tables.

Learn more about **Plutora Test** and the **Plutora Platform** on **www.plutora.com**

Start your 30-day free trial now!

go.plutora.com/dzone

Drive Automated Tests from Selenium in a SaaS-based Test Management Tool

Automated testing tools like Selenium dramatically reduce the time required for cross-browser compatibility, improving efficiency of repetitive tests and re-verifying build time tests against a variety of data sets. While this provides a great benefit, integrating test automation into your process isn't without its own challenges like the need for additional coding or manual processes.

To address the challenges that test automation presents, Plutora Test includes integration with test automation tools, such as Selenium, to enable a whole new world of testing possibilities. Test teams can now run both automated and manual tests from within a single tool.

IMMEDIATE BENEFITS FOR TEST AUTOMATION

Test automation can be imported via Bulk Upload, which maps the automation scripts back to test cases and their associated requirements. These scripts are run during test execution, logging results and defects directly back into Plutora Test. Test teams are free to allow automation experts to improve the test process for the entire team – without the need for additional tools. The development team can also move lengthy test automation scripts out of the build process to the test management team, ensuring those tests are run while reducing check-in times. In-tool comment streams are also available to enable contextual collaboration, further improving efficiency.

UNPARALLELED VIEWS AND COLLABORATION

Plutora Test manages the complete software testing process across any development approach. Testing design, execution, defect tracking, and progress reporting are all supported and offer collaboration every step of the way.

Try it now: go.plutora.com/dzone

WRITTEN BY SEAN HAMAWI
CO-CEO, **PLUTORA**

PARTNER SPOTLIGHT

Plutora Test

PLUTORA

Next-Generation Enterprise Test Management

Enhance DevOps Performance • Complete Test Solution in Single Repository • JIRA & Selenium Integration

CATEGORY
Enterprise Test Management

NEW RELEASES
Every 2 weeks

OPEN SOURCE
No

CASE STUDY

Plutora Test is a modern enterprise test management tool that supports the complete software testing process across all types of development methodologies. Teams can manage test design, planning, manual and automated execution, defect tracking, and progress reporting all from a single tool. The consumerized onboarding process is completely revolutionizing how test teams get immediate value. Traditionally, legacy test management solutions require complicated implementation and expensive consultants for configuration. Using Plutora Test, a large multi-national travel provider went from initial purchase to test execution in less than 3 days at scale. The average Plutora Test customer has over 800 test cases for each test plan.

STRENGTHS

- End-to-end – Track all aspects of the test process
- Centralized – A single view of testing metrics across teams regardless of software development methodology
- Mobility – Works on any device including Apple iWatch
- Reporting – Rich, interactive insights not found in any other tool
- Integrations – Including JIRA and Selenium

NOTABLE CUSTOMERS

- Merck
- Box
- Red Ventures

WEBSITE plutora.com | **TWITTER** @plutora | **BLOG** plutora.com/blog

Automated Android Testing With Kotlin

BY **NIKLAS WÜNSCHE**

QUICK VIEW

01 Learn how to set up Kotlin, together with the Espresso and JUnit frameworks, in Android Studio.

02 Learn how to write unit and UI tests in Kotlin.

03 Learn how to use extension functions to simplify your code.

04 Read about other frameworks and programming techniques for developing your app (UIAutomator, functional programming...).

As of Android Studio Version 3.0, Kotlin's officially joining the Android language family. However, we don't want to lose our ability to test our app by using a different language. Luckily, we can use our Java knowledge and many new techniques to test our apps in Kotlin.

After reading this article, you should be able to:

1. Set up Kotlin in Android Studio.
2. Write unit tests in Kotlin.
3. Write UI tests in Kotlin.
4. Use extension functions.

WHY KOTLIN?

Kotlin's simple and elegant. If you have a Java background, you're ready to develop in Kotlin. Some of the best Kotlin features are:

- 100% compatible with Java and all of its libraries.
- Extension functions.
- Awesome frameworks for Android development.

Of course, we will talk about all of these features later on.

WHY SHOULD WE CREATE TESTS?

Testing should be an integral part of every developer's routine. If you release an app, there should be no bugs left. You definitely don't want your users to find bugs, because this will damage your reputation as a developer. This alone should be more than enough motivation for you to start testing, if you haven't done so already.

So let's learn how to set up Kotlin in Android Studio.

SET UP KOTLIN IN ANDROID STUDIO

Since Kotlin is a language developed by JetBrains, the support for Kotlin in Android Studio is superb. First off, you'll have to install the Kotlin Plugin in Android Studio. To do this, just install it in the `File > Settings > Plugins > Install JetBrains plugin...` dialog. Now, we'll need to import or create a Java app.

After you've done this, you have to convert the Java files into Kotlin files. Just open a Java file and press `Ctrl + Alt + Shift + K`. By the way, Java and Kotlin can work side-by-side, so if you don't want to convert all your files, that should work just fine.

Right now, Android Studio will probably tell you that Kotlin isn't configured. To do this, just press `Configure > Android with Gradle > OK` and sync Gradle. Voilà, you are ready to develop in Kotlin.

OUR EXAMPLE APP

Before we can start testing an app, we actually have to create one. For this guide, we will test an app which takes the name of the user and creates a greeting.

The app consists of two input views, one output view, and a button. You can find the finished project on GitHub.

We use one input view for the first name and one for the last name. When you click the button, the message `Hello $firstName $lastName!` will appear in the output view. Underneath the surface, the message will be computed by a function fun `greeting(firstName: String, lastName: String): String`, which we're going to test later on.

In the next section, we are going to create unit tests for this app.

UNIT TESTS

Unit tests check the concrete functions and their output, but don't interact with the UI. The default framework for writing unit tests is Android's JUnit, so we're going to use it too.

Our app has two tests cases which need to be checked. First, we have to test that our `greeting()` function returns the right message if the first and last name parameters aren't empty. Second, we want to check that the same method returns an empty string if the first or last name is empty.

With this in mind, let's look at the default Android Studio unit test class.

OUR DEFAULT TEST CLASS

After we convert the default unit test class into Kotlin, it should look something like this:

```
class ExampleUnitTest {
    @Test
    fun addition_isCorrect() {
        assertEquals(4, (2 + 2).toLong())
    }
}
```

As you can see, unit tests in Kotlin look like the ones in Java, so we can reuse and extend our knowledge. As we've discussed earlier, we have two cases we have to test.

Without further ado, let's get to testing.

OUR UNIT TESTS

CHECK NORMAL BEHAVIOR AND EXTEND JUNIT

```
class ExampleUnitTest {
    @Test
    fun testNormalGreeting() {
        greeting(firstName = "first", lastName = "last")
                equals "Hello first last!"
    }
}

infix fun Any?.equals(o2: Any?) = assertEquals(this, o2)
```

If you've written any unit tests before, this should look pretty familiar to you. We check `greeting()` with a first and last name and test whether or not `greeting()` returns our desired message.

We've also added an extension function. It helps us to make our code more readable by hiding redundant code.

The extension function converts `assertEquals()` into an infix function. You could do the same, for example, for `assertThat()` or `assertArrayEquals()`. Next, let's test `greeting()` with empty parameters.

CHECK EMPTY PARAMETERS

```
class ExampleUnitTest {
    //...
    @Test
    fun testEmptyFirstName() {
        greeting(firstName = "", lastName = "last") equals ""
    }

    @Test
    fun testEmptyLastName() {
        greeting(firstName = "first", lastName = "") equals ""
    }
}
//...
```

When you look at this case, you shouldn't forget that we have two parameters that can be empty. Nevertheless, for both tests the `greeting()` should return an empty message.

And that's it for our unit tests. We created all necessary tests for `greeting()`. Now, we can close our unit test files and step right into UI tests.

UI TESTS

Up until now, we have just created tests for the internal mechanisms inside our app. Unfortunately, if our UI's broken, unit tests won't warn us. They don't rely on the UI, but users do! For this reason, we have to test it.

UI tests interact with the UI, not with the underlying method calls. They check that everything works hand-in-hand when the user's interacting with the app.

We will use the Espresso framework for our UI tests. It's already part of our app and works really well.

HOW TO SET UP ESPRESSO

As you can see, there isn't only a default test class for unit tests, but a class for integration tests too! The latter one can be used for our UI tests. However, we need to prepare the class for Espresso by adding a rule. Otherwise, our tests will fail.

In the end, it should look something like this:

```
@RunWith(AndroidJUnit4::class)
    class MainActivityTest {
        @Rule @JvmField val activity =
ActivityTestRule<MainActivity>(MainActivity::class.java)
        @Test
        //...
    }
```

Next off, we can consider for which cases we have to create UI tests.

OUR UI TESTS

The test cases are still the same as in our unit tests, but this time, we are going to check the UI. UI tests are really easy to read, so even if you haven't seen any UI tests before, you will understand the following example:

CHECK NORMAL BEHAVIOR

```
@RunWith(AndroidJUnit4::class)
    class MainActivityTest {
        @Rule @JvmField val activity =
ActivityTestRule<MainActivity>(MainActivity::class.java)

        @Test
        @Throws(Exception::class)
        fun testNormalGreeting() {
            val firstName = "first"
            val lastName = "lastName"
            val expected = "Hello $firstName $lastName!"
            onView(withId(R.id.firstNameView)).
perform(typeText(firstName))
            onView(withId(R.id.lastNameView)).
perform(typeText(lastName))
            R.id.messageButton.click()
            onView(withId(R.id.messageView)).
check(matches(withText(expected)))
        }
    }
    fun ID.click() = onView(withId(this)).
    perform(ViewActions.click())
```

And here's our very first Espresso test! This test writes the first and last name into the input view, presses the button, and checks that our output view contains the right message.

As you can see, the extension function can help us create UI tests too. In the next example, we will use many more.

CHECK EMPTY INPUT VIEWS

If one of the input views is empty, the output view should remain empty. However, we also want to display a Toast which tells the user what went wrong. Because a Toast lacks an ID, we have to find it by its message. This message is: `You forgot to write your full name!`

```
@RunWith(AndroidJUnit4::class)
class MainActivityTest {
    //...

    @Test
    @Throws(Exception::class)
    fun testEmptyFirstName() {
        R.id.firstNameView.write("")
        R.id.lastNameView.write("last")
        R.id.messageButton.click()
        R.id.messageView.textEquals("")
        activity containsToast "You forgot to write your
full name!"
    }

    @Test
    @Throws(Exception::class)
    fun testEmptyLastName() {
        R.id.firstNameView.write("first")
        R.id.lastNameView.write("")
        R.id.messageButton.click()
        R.id.messageView.textEquals("")
        activity containsToast "You forgot to write your
full name!"
    }
}
```

(You can find all extension functions on GitHub.)

As you can see, most Espresso methods consists of two parts:

1. Find the view you need.
2. Do something with this view.

You have many, many choices for both parts. But all of them work in the same way. You can find more methods on this site.

NEXT STEPS

Now you know the basic concepts on how to test your app in Kotlin. You know how to set up Kotlin in Android Studio, how to write unit tests with JUnit and UI tests with Espresso, and how to simplify them by using extension functions.

But there are many techniques left to learn. You can learn how to use functional programming, UIAutomator, Kotlinx, and KotlinTest in your tests. And always keep in mind, that if your Kotlin code doesn't look good, there's probably a better way.

Niklas Wünsche has been studying computer science at the TU Dresden, Germany for almost two years. In his free time, he loves developing Android apps, mostly in Kotlin. You can find most of his apps in the Google Play Store. He is also the author of Flying Bytes, a blog where he writes guides about Java 8 programming techniques. When he's offline, he can be found playing the guitar.

Nine Critical Considerations for Testing Responsive Websites Using Selenium

BY **CARLO CADET** - DIRECTOR PRODUCT MARKETING, **PERFECTO MOBILE**

Responsive website design is becoming a method of choice for many organizations. Among the primary motivations for embracing responsive design are:

- Consistent user experience across all platforms
- Improve marketing results by being mobile friendly
- Lower maintenance cost

One code base across so many platforms and form factors raises the bar on quality and therefore the testing strategy.

In the below checklist, you can find the most critical testing consideration for a RWD that will ensure good UX, and sufficient test automation coverage. All of the below considerations can be automated using Selenium framework or cloud-based tools.

1. VISUAL VALIDATIONS

Does your website look right across all platforms like desktop browsers, smartphones, tablets, and IoT-supported devices?

Does the site look okay in various orientations, like portrait and landscape, as well as in various languages?

2. ENVIRONMENT CONDITIONS

Validate performance across all expected conditions. Factor in variables such as incoming events, background apps, location services, and changing network conditions (Data, Wi-Fi, Airplane mode, etc.)

3. NAVIGATION

Validate CSS breakpoints across different form factors and orientations. When the site is launched across these displays, the navigation and the content of what is being displayed to the user changes (above the fold and beyond the fold content, hamburger menus, etc.).

4. PLATFORM COVERAGE

Analyze web traffic to determine coverage strategy, identifying the mandatory platforms and OS versions to be tested throughout the SDLC.

5. ACCESSIBILITY COMPLIANCE

Assess compliance with accessibility requirements across the market you serve. Using tools like WAVE that can be integrated into your Selenium scripts or be a stand-alone tool for your test engineers, is a good choice (out of few others) to adopt.

6. PERFORMANCE

Performance optimization is key, especially considering Google's recent prioritization of mobile-friendly sites. Performance testing for source and data loading, caching controls, and functional scenarios are proving to be effective tools for achieving critical performance gains.

7. LOCALIZATION

Validate location services scenarios. Design scenarios for both location specific data and the "traveling user".

8. SECURITY

Personalization strategies are driving an increasing quantity of private user data process by sites. Add data privacy scenarios to test suites. This includes authentication rules and types, cleaning of private data upon session termination.

9. DON'T FORGET QUALITY ANALYSIS & VISIBILITY

Testing a RWD site across multiple platforms, means, dealing with large amount of test data. Having a quality dashboard after each test automation execution that is tag-driven for easy filtering enables data-driven and risk-based decisions.


Like Automation...
But Good.
Machine Efficiency with Human Context
Rainforest QA helps agile and continuous delivery engineering teams move faster with the industry's only AI-powered crowdtesting platform. Our platform leverages 60,000 qualified testers to deliver on-demand, comprehensive and machine learning verified regression test results. Rainforest customers spend less time and money testing so they can ship better applications faster.
Learn more at www.rainforestqa.com
rainforest

The "Total Automation" Myth

"The fetishization of automated tests as the magic bullet that will allow you to deliver software quickly is wrong. The opposite is the case!"
- **Sally Goble**, Head of Quality, **The Guardian**

Testing automation is often held up as the QA endgame. Teams mistakenly believe that if they can just crack the automation code, all of their quality problems will be solved. But even when implemented perfectly, automated tests are brittle, flaky, and require significant upkeep. Automation is a critical QA tool, but it's not a magic bullet. It's simply not flexible enough to keep up with fast-evolving applications.

THE COST OF AUTOMATION

Stable automated tests can be used quickly and repeatedly without additional cost, but the fast pace of development makes stability a moving target. As a result, "cost-effective" automated tests become huge resource drains, eating up developer time and budget.

DYNAMIC QA ECOSYSTEMS: A NEW APPROACH TO AUTOMATION

In order to maximize their impact, automated tests should not be considered the "final" form of the test, but the preferred method for stable tests. Approaching automation as just one QA tool to accelerate testing prevents the creation of suites of rapidly-outdated automated tests. Treat automation as one state within an ecosystem of test execution types.

In order to maximize their impact, automated tests should not be considered the "final" form of tests, but the preferred method for stable tests.

When a test fails, it is rolled back to manual execution until it is stable again. This practice keeps test suites healthier and more reliable overall. Use a rapid-feedback crowdtesting solution to run tests that are not quite automation-ready to keep things running smoothly.

By taking a more dynamic approach to their automation strategy, fast-moving teams will gain more consistent insights into quality without sacrificing speed.

WRITTEN BY RUSSELL SMITH
CTO AND CO-FOUNDER, **RAINFOREST QA**

PARTNER SPOTLIGHT

Rainforest QA

rainforest

Like Automation but Good: Machine Efficiency with Human Context.

CATEGORY
Continuous Delivery and Integration

NEW RELEASES
Continuous

OPEN SOURCE
No

CASE STUDY

As TrendKite, a PR analytics company, scaled their customer base, code base, and team, QA became a major bottleneck to their fast-paced Continuous Delivery cadence.

TrendKite's team has been able to focus their own QA efforts on more complex challenges, while letting Rainforest take care of repetitive regression tests. "With Rainforest, we're able to automate the very routine parts of our smoke tests and allow our own testing to be more specialized. We can spend more time focusing on things like user experience and usability testing, to make sure the change is going to make sense to customers," says David Perdue, VP of Development.

Rainforest alleviates many of the QA growing pains felt by fast-moving companies as they scale. "Rainforest lets us move faster by allowing us to keep more of our budget on feature development instead of feature verification."

STRENGTHS

- Crowdtesting for agile teams. Write test cases and get results on-demand in under 30 minutes.
- Cross-browser coverage on-demand, any day of the year, without worrying about your testing bandwidth.
- Machine learning-verified results and detailed video reports help your team understand and fix issues fast.
- Vetted, experienced testers to dig deep into your web and mobile apps on-demand.
- Exploratory testing to improve quality and fill in the natural gaps in your tests coverage strategy.

NOTABLE CUSTOMERS

- Adobe
- BleacherReport
- TrendKite
- Oracle
- StubHub

WEBSITE rainforestqa.com | **TWITTER** @rainforestqa | **BLOG** rainforestqa.com/blog

Executive Insights on the State of Automated Testing

QUICK VIEW

01 Automate the entire CI/CD/DevOps process to reduce the amount of human interaction.

02 Companies who don't automate are paying a "stupid tax."

03 The future of automated testing is integrating AI/ML into the CI/CD/DevOps process so that testing is completely automated.

BY **TOM SMITH**
RESEARCH ANALYST, **DZONE**

To gather insights on the state of automated testing today, we spoke with 20 executives who are familiar with automated testing. Here's who we talked to:

MURALI PALANISAMY EVP, CHIEF PRODUCT OFFICER, APPVIEWX

YANN GUERNION DIRECTOR OF PRODUCT MARKETING, AUTOMIC

ERIC MONTAGNE TECHNOLOGY PM, BARCLAYCARD

GREG LUCIANO DIRECTOR OF SERVICES, BUILT.IO

AMIT PAL QA MANAGER, BUILT.IO

DONOVAN GREEFF HEAD OF QA, CURRENCYCLOUD

SHAHIN PIROOZ CTO, DATAENDURE

LUKE GORDON SENIOR SOLUTIONS ENGINEER, DIALEXA

DANIEL SLATTON QA MANAGER, DIALEXA

ANDERS WALLGREN CTO, ELECTRIC CLOUD

CHARLES KENDRICK CTO, ISOMORPHIC

BRYAN WALSH PRINCIPAL ENGINEER, NETAPP

DEREK CHOY VP OF ENGINEERING, RAINFOREST QA

SUBU BASKARAN SENIOR PRODUCT MANAGER, SENCHA

RYAN LLOYD VP PRODUCTS, TESTING AND DEVELOPMENT, SMARTBEAR

GREG LORD DIRECTOR OF PRODUCT MARKETING, SMARTBEAR

CHRISTOPHER DEAN CEO, SWRVE

WOLFGANG PLATZ FOUNDER AND CHIEF PRODUCT OFFICER, TRICENTIS

PETE CHESTNA DIRECTOR OF DEVELOPER ENGAGEMENT, VERACODE

HARRY SMITH TECHNOLOGY EVANGELIST, ZERTO

KEY FINDINGS

01 The key to automating testing to improve speed and quality is strategic automation and developer adoption. You must have a platform in place that enables you to use automated test development on a consistent basis and understand how to write automated test cases that automate repeatable core functionality. Automate the entire CI/CD process to reduce the amount of human interaction. This will accelerate the SDLC and lead to more reliable and secure code and applications. Increase speed and quality at every step by learning from unit and regression test results.

Developers need to take accountability for their work by integrating the proper unit tests at the beginning of the SDLC. Do not think of testing as someone else's job. Inspect quality from a functional and non-functional point of view from the beginning.

02 There have been several changes to automated testing recently: **cloud-based testing solutions, the speed with which data is available, and the rise of container services.**

Cloud computing has boosted the performance of automated testing. It makes it much easier to run load tests with many users at one time. Cloud/grid-based Selenium testing is becoming increasingly mature and popular. Services like Xamarin Test Cloud and Testdroid now provide physical device hardware through a web interface in a cloud-based Device-as-a-Service model.

The digital transition, including mobile, is accelerating development testing. This impacts both the frontend and the backend of an organization. Test processes are automated more quickly. This provides more data-driven insights from the tests and improves the stability of the build.

Containers are a big deal because they allow you to push around code in a completely fixed environment. The rise of container

services like Docker have brought great changes to automated testing, reducing many of the environmental risks.

03 Many technical solutions are used for automated testing since there are a lot of different aspects to automated testing. Of the 33 different solutions mentioned, Selenium was mentioned most frequently: Selenium with a Capyvara layer on top making the automation more portable, Selenium WebDriver tools, and NodeJS-based Selenium frameworks.

04 Real-world problems solved by automated testing are speed to market and lower costs. Automation reduces development time and reduces the time it takes to discover and fix problems. Automation enables you to complete QA and production in two days instead of two weeks. New versions are rolled out faster and enable human developers and engineers to focus on the complex tasks of software development, product strategy, test strategy, and on the more difficult areas to test within the product. Running a script automatically can be done much faster than a human can run through code or validate functionality.

All of this saved time results in reduced resource costs, since you can go to product faster with automation, and there are fewer resources used to test at scale. Companies save by hiring fewer manual testing resources. Companies who don't automate are paying a "stupid tax."

05 The most common issues affecting the automation of testing is the **brittleness of the automation process, people, and lack of processes**. If the product is dynamic or Agile it becomes a difficult effort to keep automated tests up to date. The real challenge is setting the test up in a manner that allows you to cope with maintenance challenges that come with automation. Comprehensively solve the issue by taking care of the automation, test data management, and service virtualization to provide a stable infrastructure.

People unwilling to change are the number one roadblock. They don't believe a manual test can be automated when any process can be automated. To promote culture change, get the development and testing teams excited about the potential outcomes and let them see how quickly their code gets to production.

Define KPIs and pass/fail criteria for all performance tests. You cannot become Agile without repeatable processes.

06 **The future of automated testing is integrating artificial intelligence (AI) and machine learning (ML) into the CI/CD/DevOps process so that testing is completely automate**d. Leverage AI/ML to automate automation. Learn from the tests to automate the process. This will result in self-healing code. Orchestrate the test process end-to-end and integrate the DevOps CD release cycle. AI/ML based solutions can identify and proactively check areas known to introduce critical issues.

07 The biggest concerns with automated testing today revolve around the **fragmentation in the marketplace, the lack of an end-to-end solution, and the perspective of the people involved**. The automated test space is very crowded. Several third-party solutions keep popping up and it is very difficult to have the knowledge to put together a "best of breed" solution for your company's needs.

Nothing is managed holistically. There is no connection across the siloes. There's need for greater automation to maintain accuracy and reliability of the process. However, there's still a fear of failure. It's not about the technology. It's about understanding the required comprehensiveness of the approach. "It's not the quality of the daily work; it's the improvement in the quality of the daily work.

08 Skills developers need to ensure their code and apps perform well with automated testing are: **1) the ability to write tests; 2) understanding automation; 3) good coding skills; and, 4) a broader vision that includes CI/CD/DevOps and what the company is trying to accomplish**. Create test code and test cases defining what data to test. Don't write bad tests. Write tests in the smallest, most elegant way possible. Developers writing code are the best people to direct the testing on their code.

Be aware of how quality engineers write automation. As you make code changes, think about how the automation will need to change. Put effort into getting automation right. Cut out repetitive work. It may not seem like much now, but it will add years of productivity to your life. Build trust in automation by running it regularly during implementation.

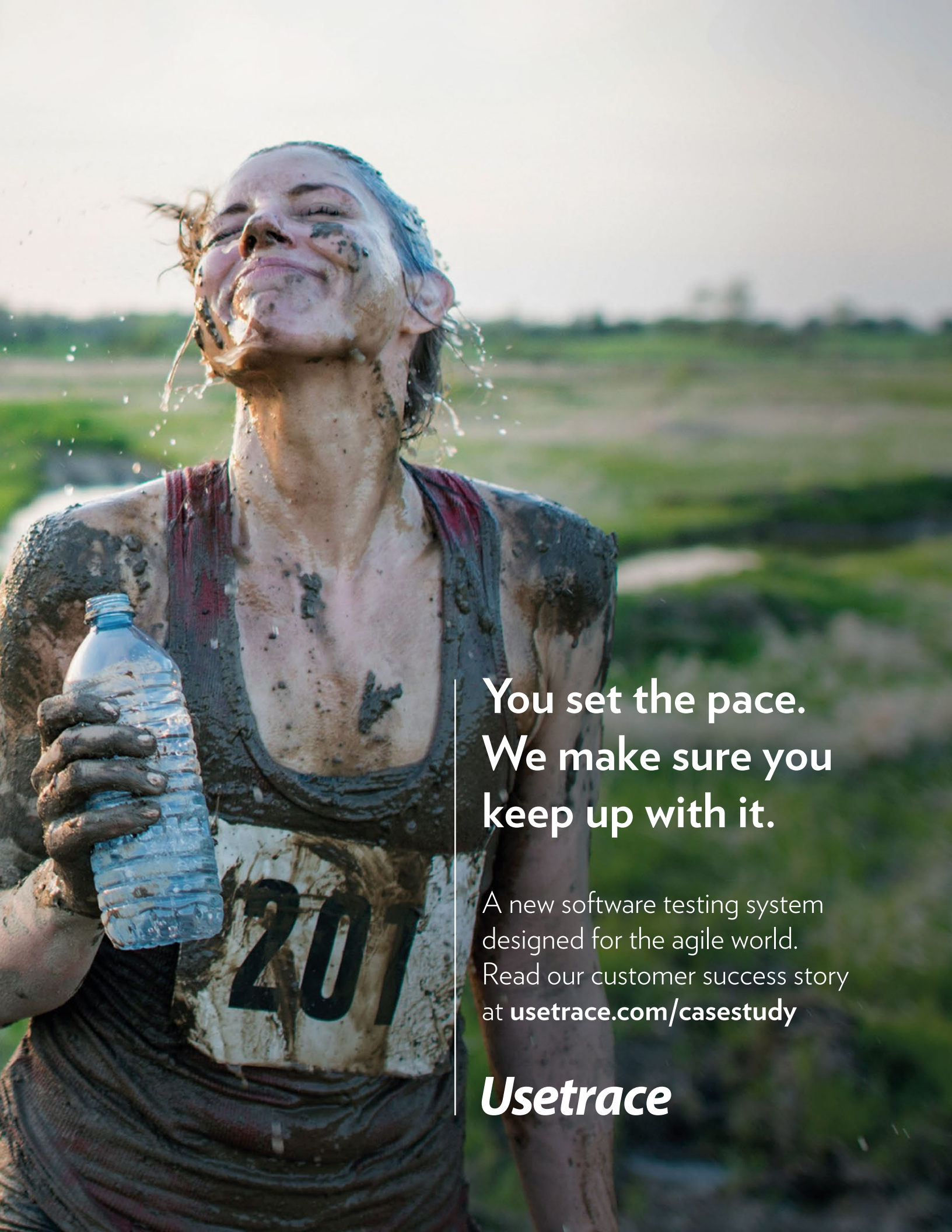
Write code so the next person will be able to see what you've done and make changes as necessary. Code efficiently so you're not wasting someone's phone battery. Follow the coding standards of modular programming, unit testing, exception handling, Selenium, and Appium, and strive for CI/CD. Developers need the "DevOps" attitude – testing is an extension of the development process. Product quality doesn't stop once the coding stops.

09 While there were several different "additional considerations," the **current and future state of automation, as well as the cost and ROI of automation** were mentioned by several respondents.

There's a stigma that automated testing is difficult to do; however, there are plenty of resources available for manual testers to get into automated testing. Everyone from developers, QA, DevOps, product owners, and stakeholders are responsible for automated testing. Everyone should be involved in automation. Do people know how to write automated tests? Testing is a problem with developers, not the testers. Shift left; test early while still in development. How often are software developers involved in the automation strategy for a given project?

Think about the cost of automation. How much do you invest in people, environments, tools, and maintenance? How do you successfully estimate the cost to reduce surprises down the road? We need to be able to quantify the value automated testing provides by optimizing time to market, reducing mean time to recovery, and ensuring applications are secure.

Tom Smith is a Research Analyst at DZone who excels at gathering insights from analytics—both quantitative and qualitative—to drive business results. His passion is sharing information of value to help people succeed. In his spare time, you can find him either eating at Chipotle or working out at the gym.



You set the pace. We make sure you keep up with it.

A new software testing system designed for the agile world. Read our customer success story at **usetrace.com/casestudy**

Usetrace

Cloud-Based Visual Testing Tools Boost Productivity and Improve Reliability

Agile methodology is speeding up the software development process, enabling innovation through shorter release cycles. With continuous integration, it's common to release a new version several times per day. This puts a lot of pressure on QA personnel, just to keep up with the speed of development. Most of their time is spent on trying to keep the test cases current after the application has been changed. A lot of time is wasted in inefficient communication between testers and developers. Cloud-based visual testing tools like Usetrace allow for several key benefits. Developers and testers can collaborate effortlessly, finding and fixing bugs faster. Every test result can be shared and examined right there in the web browser, helping developers reproduce the error in seconds. With no local test infrastructure installed on a specific computer, test resources are automatically shared and usable by everyone. Visual testing tools that minimize coding bring another benefit, as visual test case generation can boost the productivity of less technical testers. Of course, there are always advanced use cases that require help from more technical personnel, but as all resources are shared in the cloud, cooperation between different teams is easier than ever. This combination of cloud-based and visual test case generation will definitely be the next big step in testing. This is only the beginning, as AI will eventually allow test cases to automatically adapt to software and UI changes. It's definitely an exciting time to be in the QA space.

AI will eventually allow test cases to automatically adapt to software and UI changes.

WRITTEN BY ARTO VUORI
CEO, **USETRACE**

PARTNER SPOTLIGHT

Usetrace

Usetrace is a new cloud-based visual testing system, built for agile development.

CATEGORY
SaaS, Cloud-based functional testing tools

NEW RELEASES
Monthly

SUPPORTED PLATFORMS
Web

CASE STUDY

Sanoma Pro, the front-runner in digital education in Finland, moved their platform from yearly to monthly releases. Previously, testing had been done by three to five manual testers, but that was too time-consuming with the new tighter schedule.They evaluated several options to speed up testing before choosing Usetrace. Implementing all test cases in Usetrace took only six weeks compared to several months with the competing solution. Testing all test cases takes only an hour, compared to three to four days of manual testing. According to Sanoma Pro, Usetrace saves them $258,000 (61%) per year compared to full manual testing. Usetrace has also been estimated $218,000 (83%) less expensive than the main keyword-based alternative in the first year alone.

STRENGTHS

- Cloud-based solution built for agile development
- Visual test case editor for non-programmers (extendable with JavaScript)
- Tolerant to UI changes by auto-correcting element locating
- Integrated load testing with realistic end-user modeling

NOTABLE CUSTOMERS

- Sanoma
- LeadDesk
- Clickfunnels

WEBSITE usetrace.com | **TWITTER** @Usetrace | **BLOG** blog.usetrace.com

QUE SERA SERA

(Said no test professional ever)

We know you can't take reckless risks; that you are expected to work swiftly and deliver quality software, while staying safe. That's why Panaya offers solutions to help you introduce change fast and risk-free.

Experience Smart Testing

—

www.panaya.com

Panaya

Augmenting Test Automation for Greater Enterprise Agility

To meet the requirements for digital transformation, organizations need to apply agile test methodologies that deliver quality at speed change, without introducing risk that can create critical application failures. Delivering continual change with quality assurance, requires implementing an end-to-end testing strategy that aims to accelerate business process testing across all test cycles with higher levels of automation coverage.

QA and test strategies must go beyond traditional automated test execution and deliver smart automation throughout the testing process. Automation must be executed holistically from planning, discovery and business knowledge capture to test execution and defect management, while offering collaboration and visibility to all QA and business users involved.

Today's increasingly complex enterprise applications landscapes warrants QA and Testing teams to have functionality that addresses cross functional teams and processes with an automated workflow processes, testing and defect tracking, collaborative communications and intuitive dashboards for both business and IT users. By increasing visibility into the change delivery process and improving collaboration you can significantly speed up test cycle times and guarantee quality.

Automation must be executed holistically from planning, discovery and business knowledge capture to test execution and defect management

Cloud-based solutions like Panaya's end-to-end testing platform inherently deliver these functionalities and extend beyond traditional automation to increase quality assurance, shorten delivery cycles, and position IT as an agile true enabler of digital transformation.

WRITTEN BY RONIT ELIAV
DIRECTOR OF PRODUCT MARKETING, **PANAYA**

PARTNER SPOTLIGHT

Panaya Test Center

Panaya

Empowering Organizations to Continually Deliver Innovation Through Change Across Enterprise Applications

CATEGORY
Test Automation and Management

NEW RELEASES
Monthly

SUPPORTED PLATFORMS
No

CASE STUDY

Panaya Test Center (PTC) is a cloud-based testing platform that improves the speed and quality of application delivery. Built around four principles: automation, collaboration, visibility and compliance, PTC provides an intuitive, easy to use, test automation and management solution to plan, execute and monitor all functional test cycles.

With an innovative and affordable dual stage automation approach based on machine learning and our autonomous testing methodology, PTC, improves adoption of functional test automation beyond the traditional 5-10% while removing the reliance of costly test engineering. The automated process centric and collaborative features address acceleration of manual testing for both IT and business users to ensure the entire end-to-end testing process is faster and safer. Panaya customers realize 30-50% faster test cycles with increased scope.

STRENGTHS

- Zero touch test automation eliminates the need for expensive engineered scripts
- Business process oriented test planning ensures quality release to production
- Cross functional, collaborative test execution strengthens IT and Business convergence
- Risk management through delivery impact and risk mitigation
- Increased visibility into test execution utilizing business-oriented Reports and Analysis

NOTABLE CUSTOMERS

- Shiseido
- Aritzia
- ABB
- FujiFilm
- Eaton

WEBSITE panaya.com | **TWITTER** @panaya | **BLOG** panaya.com/blog

Solutions Directory

This directory of automated testing frameworks, platforms, and services provides comprehensive, factual comparisons of data gathered from third-party sources and the tool creators' organizations. Solutions in the directory are selected based on several impartial criteria, including solution maturity, technical innovativeness, relevance, and data availability.

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
AgileEngine	Screenster	Visual UI testing	Free tier available	screenster.io
Apache Software Foundation	JMeter	Java functional and performance testing	Open source	jmeter.apache.org
Apica	Apica Load Test	Load Testing	30 days	apicasystem.com
Applitools	Applitools	AI-powered automated visual testing	Available by request	applitools.com
Appvance	Appvance UTP	Automated performance and load testing	14 days	appvance.com/appvance-utp
Atlassian	Bamboo	Continuous Integration, Continuous Delivery, automated testing	Available by request	atlassian.com/software/bamboo
BrowserStack	BrowserStack	Automated mobile and web testing	Available by request	browserstack.com
Buildbot	Buildbot	Continuous Integration	Open source	buildbot.net
CA Technologies	BlazeMeter	Performance and load testing for JMeter	Free tier available	blazemeter.com
CasperJS	CasperJS	JavaScript navigation testing	Open source	casperjs.org
CircleCI	CircleCI	Continuous Integration and Continuous Delivery	Free tier available	circleci.com
Cloudbees	Jenkins	Continuous Delivery and Continuous Integration	Open source	jenkins.io
Cucumber	Cucumber	Automated rails testing	Open source	cucumber.io
DevExpress	TestCafe	Functional web testing	30 days	testcafe.devexpress.com

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
eureQa	eureQa	Load and Performance Testing	Available by request	sayeureqa.com
FitNesse	FitNesse	Acceptance Testing Framework	Open source	fitnesse.org
Functionize	Functionize	Automated functional, load, security, and performance testing	Available by request	functionize.com
Gridlastic	Gridlastic	Selenium-as-a-service	Free tier available	gridlastic.com
HipTest	Hiptest	Automated Web Testing	30 days	hiptest.net
HP Enterprise	Loadrunner	Load testing	Available by request	saas.hpe.com/en-us/software/loadrunner
HP Enterprise	HP Unified Functional Testing	Automated functional testing	60 days	saas.hpe.com/en-us/software/uft
HP Enterprise	HP Quality Center	Test planning and management platform	Available by request	saas.hpe.com/en-us/software/quality-center
IBM	IBM Rational Functional Tester	Automated functional and regression testing	Available by request	ibm.com/us-en/marketplace/rational-functional-tester
IBM	IBM Rational Performance Tester	Performance and load testing	30 days	ibm.com/us-en/marketplace/rational-performance-tester
IBM	Urbancode Build	Continuous Integration, build management	Available by request	developer.ibm.com/urbancode/products/urbancode-build
JetBrains	TeamCity	Continuous Integration, Application Release Automation	Free solution	jetbrains.com/teamcity
JS Foundation	Appium	Automated web and mobile testing framework	Open source	appium.io
JUnit	JUnit	Unit testing framework	Open source	junit.org
Loadster	Loadster	Web app load testing	Free tier available	loadsterperformance.com
Micro Focus	SilkTest	Automated mobile and web testing	45 days	microfocus.com/products/silk-portfolio/silk-test
Microsoft	Visual Studio Test Professional	Test case management, exploratory and browser testing	Available by request	visualstudio.com/vs/test-professional
Mobile Labs	Mobile Labs	Automated Mobile and Web Testing	Available by request	mobilelabsinc.com
Neotys	NeoLoad	Automated performance testing	Free tier available	neotys.com/neoload/overview
NUnit	NUnit	.NET unit testing framework	Open source	nunit.org
Panaya	Panaya Test Center	Continuous Testing	Available by request	panaya.com

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
Parasoft	DotTest	Test automation platform	Available by request	parasoft.com/product/dottest
Parasoft	SOAtest	Functional and load testing for distributed apps	Available by request	parasoft.com/product/soatest
Perfecto Mobile	CQ Lab	Automated web and mobile testing	Available by request	perfectomobile.com/solutions/perfecto-test-automation
PHPUnit	PHPUnit	PHP unit testing framework	Open source	phpunit.de
Plutora	Plutora	Release and test management platform	Available by request	plutora.com
Practitest	Practitest	QA and Test Management	14 days	practitest.com
Progress Software	Telerik Test Studio	Automated GUI, performance, load, and API testing	Available by request	telerik.com/teststudio
QASymphony	qTest Platform	Continuous testing	14 days	qasymphony.com/software-testing-tools
Radview	WebLOAD	Load testing	Free tier available	radview.com
Rainforest	Rainforest	Automated Mobile and Web Testing	Available by request	rainforestqa.com
Ranorex	Ranorex	Automated GUI testing	Available by request	ranorex.com
Robotium	Robotium	Android UI testing	Available by request	robotium.com
Robustest	Robustest	Automated Mobile Testing	Available by request	robustest.com
Sahi	Sahi Pro	Automated web testing	30 days	sahipro.com
Sauce Labs	Sauce Labs	Automated web and mobile testing framework	14 days	saucelabs.com
Sealights	Sealights	Continuous Testing	Available by request	sealights.io
Selenium	Selenium	Automated web testing	Open source	seleniumhq.org
Sencha	Sencha Test	Automated Ext JS Testing	30 days	sencha.com/products/test
SmartBear Software	LoadUI	API load testing	Free solution	loadui.org
SmartBear Software	SoapUI	Automated web and API testing	Open source	soapui.org
SmartBear Software	TestComplete	Automated UI testing	Available by request	smartbear.com/product/testcomplete/overview

COMPANY	PRODUCT	PRODUCT TYPE	FREE TRIAL	WEBSITE
SmartBear Software	Cross Browser Testing	Automated Mobile and Web Testing	7 days	crossbrowsertesting.com
Solano Labs	Solano Labs	Continuous Integration and Deployment	14 days	solanolabs.com
Tellurium	Tellurium	Automated Web Testing	Free tier available	te52.com
Test Anywhere	Test Anywhere	Frontend web testing	Available by request	testanywhere.co
Testim	Automate	Automated Regression Testing	14 days	testim.io
TestingBot	TestingBot	Automated Web Testing	100 minutes of testing	testingbot.com
Testlio	Testlio	Automated Mobile and Web Testing	Available by request	testlio.com
TestNG	TestNG	Unit testing framework	Open source	testng.org
TestPlant	eggPlant	Automated functional, performance, network, and integration testing	Available by request	testplant.com/eggplant/testing-tools
Thoughtworks	Go CD	Continuous Delivery/Continuous Integration	Open source	thoughtworks.com/go
Travis CI	Travis CI	Continuous Integration	Free solution	travis-ci.org
Tricentis	Tricentis Tosca	Continuous testing platform	14 days	tricentis.com/software-testing-tools
Turnkey Solutions	cFactory	Automated Mobile and Web Testing	Available by request	turnkeysolutions.com
Ubertesters	Ubertesters	Automated Mobile Testing	Free tier available	ubertesters.com
UseTrace	UseTrace	AI-based automated web testing	Available by request	usetrace.com
VersionOne	VersionOne	Acceptance and Regression Testing	Free tier available	versionone.com
Watir	Watir	Automated web testing	Open source	watir.com
Windmill	Windmill	Automated Web Testing	Open source	getwindmill.com
Worksoft	Worksoft	Automated Mobile and Web Testing	Available by request	worksoft.com
xUnit	xUnit	Unit Testing Framework	Open source	xunit.github.com
Zephyr	Zephyr Enterprise	Real-time test management	30 days	getzephyr.com

GLOSSARY

AMAZON ELASTIC COMPUTE CLOUD (AMAZON EC2) An infrastructure-as-a-service that provides secure, resizable compute capacity to run applications.

ANDROID An open source mobile programming language developed primarily by Google.

CONTAINERS Resource isolation at the OS (rather than machine) level, usually (in UNIX-based systems) in user space. Isolated elements vary by containerization strategy and often include file system, disk quota, CPU and memory, I/O rate, root privileges, and network access. Much lighter-weight than machine-level virtualization and sufficient for many isolation requirement sets.

CONTINUOUS TESTING The process of executing unattended automated tests as part of the software delivery pipeline across all environments to obtain immediate feedback on the quality of a code build.

DEPLOYMENT PIPELINE A deployment pipeline is an automated manifestation of your process for getting software from version control into the hands of your users. (source: informIT.com)

DEVOPS An IT organizational methodology where all teams in the organization, especially development teams and operations teams, collaborate on both development and deployment of software to increase software production agility and achieve business goals.

JAVASCRIPT A programming language used with HTML and CSS for web applications that supports event-driven, functional, and object-oriented programming.

KOTLIN A language that runs on the JVM, developed by JetBrains, provided under the Apache 2.0 License, offering both object-oriented and functional features. It is also an official Android language.

LOAD TESTING The process of measuring a software system's response when handling a specified load.

PAGE OBJECT A feature in Selenium that models parts of a web application's UI as objects in the test code.

MACHINE LEARNING An AI system that is able to learn based on exposure to new data, rather than being specifically programmed.

SCALABILITY The ability of a network, application, or system to process an increasing amount of work or the ability to grow in order to handle that increased work.

SELENIUM An open source framework for automating web applications inside of a web browser for testing.

SELENIUM GRID A feature of Selenium 2.0 that allows developers to run tests on different machines against different browsers in parallel for distributed testing.

SOFTWARE DEVELOPMENT LIFECYCLE (SDLC) The division of software development into distinct phases to improve efficiency and quality.

TEST AUTOMATION The use of special software (separate from the software being tested) to control the execution of tests and the comparison of actual outcomes with predicted outcomes.

UI TEST A test to determine bugs or failures in an application's Graphical User Interface (GUI).

UNIT TESTS A testing method focused on testing individual pieces, or units, of source code to determine whether it is fit for use, rather than the entire application.

VIRTUAL MACHINE (VM) Software that emulates a computer hardware system for the purposes of testing and development for different pieces of hardware.