DZone

THE 2018 DZONE GUIDE TO

# Automated Testing

## YOUR END-TO-END ECOSYSTEM

VOLUME II

BROUGHT TO YOU IN PARTNERSHIP WITH

## Dear Reader,

Welcome to DZone's 2018 Guide to Automated Testing. Automated testing has grown significantly from its early purpose of conducting simple tests ensuring that developer code has not regressed from build to build, to being of product-level importance for engineering software at the quality and speed necessary to succeed in today's software business. Automated testing is what powers today's Continuous Delivery and Deployment pipelines and is a non-negotiable part of a modern software development lifecycle.

In this Guide, we will focus on automation methodologies, automation classifications, how to stratify automation, and the cultural changes necessary to build a successful pipeline and an automation-first engineering team.

In the yesteryears of software, i.e. Waterfall, automation was a task that was usually left for the time-crunched testing phase of the SDLC. If the team was fortunate, they would gather functional tests, primarily focusing on the happy paths, and fold them into the build process. If the team was very fortunate, they might also have a suite of unit, performance, and security tests. Why not do this at the end, right? There is always a support team that will pick up the pieces, right? That world has changed.

In the services world, there is little room for buggy software and there are a lot of options for the customer to choose from. You need to ship high-quality software the first time — not version 3.0. Customers expect software to just work, to be available in the way we expect electricity to flow when a switch is flicked. Customers don't just request quick turnaround on broken issues — they demand it. If your cycle time to get a defect fixed is in the frame of weeks or even months, then you have lost your customer. Cycle times need to be days, hours, or possibly even minutes. Automated testing is the only way to satisfy these needs.

As such, automated tests and automation frameworks are tightly coupled with your automated pipeline. Join us as we look into types of automated tests, types of frameworks, where to integrate these into your pipeline for maximum coverage and throughput, and how to evolve to a hands-off testing world.

Thank you for downloading this Guide. As the software industry continues to push forward at breakneck speeds, check back at DZone to catch up with latest in automated testing trends and technologies. As with any great community, your feedback matters, and we would love to hear it. As much as automation has grown, it is but a piece in the software engineering machine. Be sure to check out more awesome articles at DZone.com.

By Rick Severson
**DIRECTOR OF ENGINEERING AT DZONE**

## Table of Contents



## DZone is...


**BUSINESS & PRODUCT**

MATT TORMOLLEN
CEO

MATT SCHMIDT
PRESIDENT

JESSE DAVIS
EVP, TECHNOLOGY

KELLET ATKINSON
MEDIA PRODUCT MANAGER

**PRODUCTION**

CHRIS SMITH
DIR. OF PRODUCTION

ANDRE POWELL
SR. PRODUCTION COORD.

ASHLEY SLATE
DESIGN DIRECTOR

G. RYAN SPAIN
PRODUCTION COORD.

BILLY DAVIS
PRODUCTION COORD.

NAOMI KROMER
SR. ACCOUNT MANAGER.

JASON BUDDAY
ACCOUNT MANAGER

MICHAELA LICARI
ACCOUNT MANAGER

**EDITORIAL**

CAITLIN CANDELMO
DIR. OF CONTENT & COMMUNITY

MATT WERNER
PUBLICATIONS COORDINATOR

SARAH DAVIS
PUBLICATIONS ASSOCIATE

MICHAEL THARRINGTON
CONTENT & COMMUNITY MANAGER II

KARA PHELPS
CONTENT & COMMUNITY MANAGER

TOM SMITH
RESEARCH ANALYST

MIKE GATES
CONTENT TEAM LEAD

JORDAN BAKER
CONTENT COORDINATOR

ANNE MARIE GLEN
CONTENT COORDINATOR

ANDRE LEE-MOYE
CONTENT COORDINATOR

LAUREN FERRELL
CONTENT COORDINATOR

LINDSAY SMITH
CONTENT COORDINATOR

**SALES**

CHRIS BRUMFIELD
SALES MANAGER

FERAS ABDEL
SALES MANAGER

ALEX CRAFTS
DIRECTOR OF MAJOR ACCOUNTS

JIM HOWARD
SR. ACCOUNT EXECUTIVE

JIM DYER
SR. ACCOUNT EXECUTIVE

ANDREW BARKER
SR. ACCOUNT EXECUTIVE

BRIAN ANDERSON
ACCOUNT EXECUTIVE

SARA CORBIN
ACCOUNT EXECUTIVE

BRETT SAYRE
ACCOUNT EXECUTIVE

**MARKETING**

SUSAN WALL
CMO

AARON TULL
DIRECTOR OF MARKETING

LAUREN CURATOLA
MARKETING SPECIALIST

KRISTEN PAGÀN
MARKETING SPECIALIST

JULIAN MORRIS
MARKETING ASSOCIATE

# Executive Summary

**BY MATT WERNER** - PUBLICATIONS COORDINATOR, DZONE

While "automate everything" has long been a catchphrase of DevOps advocates, it's still not considered the de facto way of doing things, despite the amount of evangelism on the topic and advancements in technology. For our second Guide to Automated Testing, we wanted to find out if attitudes towards automation had changed within our audience. This year, 541 DZone members told us what tests they were automating, as well as other tactics they were using to improve the quality of their software at all stages of development.

## SHIFTING LEFT

**Data**: 54% of respondents are starting to automatically test software in development, compared to 52% in 2017. In addition, 51% are manually testing software in development compared to 48% in 2017. Also, more DZone members are performing both kinds of testing. 11% reported they did not perform automated testing in 2018, compared to 23% in 2017, and 2% reported that they did not perform manual testing in 2018, compared to 23% in 2017.

**Implications**: There is a slow growth in the amount of testing that is happening during development, and a sharp drop in respondents that reported that they did not perform any kind of either automated or manual testing. The sharp drop in those who were not performing any testing could be attributed to an increased knowledge of what kind of testing is happening as well as the utilization of testing tools and best practices.

**Recommendations**: "Shifting left" is a common term for performing more tasks earlier in a product pipeline, specifically development. The earlier testing can happen, the better code generally is, and less time is spent getting it into production.

## TEST-DRIVEN DEVELOPMENT

**Data**: Only 36% of respondents reported that they performed no test-driven development (TDD) at all. Another 36% use TDD in certain teams, and 28% use TDD throughout the whole organization. Only 5% of those who use TDD said they were unsure of the benefits.

**Implications**: 95% of survey respondents who use TDD in some capacity have seen some benefit from adopting the practice. The most significant benefits were improved code quality (76%), less time spent debugging (60%), easily maintainable code (46%), and the ability to use tests as documentation (45%).

**Recommendations**: The amount of people who recognize that TDD has had some kind of benefit for their organization is a strong indicator that the practice is worth looking into for anyone who has yet to adopt it. For more information, check out our fun Test-Driven Development Infographic to learn about some of the basic tenets and why they're important.

## DEVOPS TEAMS AND TESTING

**Data**: Those who have a DevOps team in their organization have reported that they perform more integration tests (57%), component tests (53%), and performance tests (55%) compared to those that do not. Those who do not have designated DevOps teams in their organizations perform these kinds of tests by 14%, 6%, and 10% less, respectively.

**Implications**: While there is some debate on whether dedicated DevOps teams are necessary for adopting DevOps or not, they are clearly encouraging the adoption of both automated and manual testing practices.

**Recommendations**: Even if there is not a designated DevOps team in your organization yet, culture can be influenced from the bottom up. Speak to your immediate managers or upper leadership about adopting DevOps tools and best practices that could help your organization develop and release better software. There are several articles on the culture of DevOps adoption in our DevOps Zone and Research Guide.

# Key Research Findings

**BY JORDAN BAKER** - CONTENT COORDINATOR, DZONE



## DEMOGRAPHICS

For this year's Automated Testing Guide Survey, we received 821 responses, with a 64% (526) completion rate. Here's how the basic demographics from this group look:

- 41% work as a software developer, 19% are developer team leads, and 13% are architects.
- 22% are employed by enterprise-level (1,000-9,999 employees) organizations.
- 65% work on immediate teams of 10 or less people.
- 87% identify as male, and 6% identify as female.
- There were three main geographic areas reported:
  - 18% from the USA.
  - 20% from Europe.
  - 26% from South Central Asia.

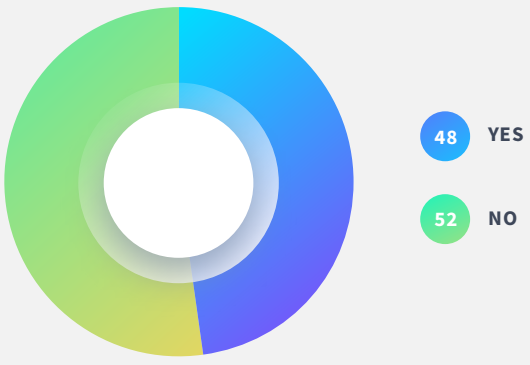
## AUTOMATED VS. MANUAL TESTING

Only 11% of respondents told us that their organization does not practice any kind of automated testing. Out of the 89% who do automate tests somewhere within their organization and SDLC, there were varying opinions as to when it's best to automate tests, and which tests should be automated. When asked at what stage they begin automating their tests, 54% of respondents told us development, 31% in staging/QA/testing, and 4% in deployment.

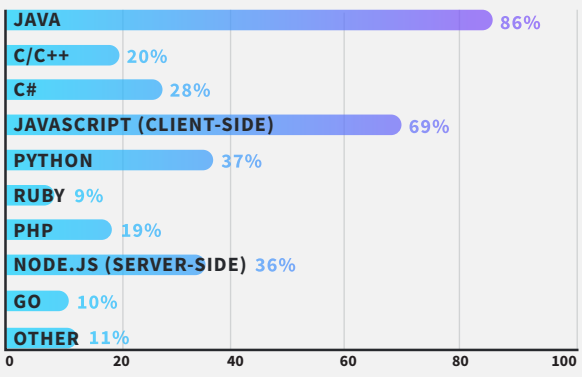
The most popular tests to automate in the development pipeline were integration tests (77%), component tests (52%), and performance tests (51%). Comparing this data to last year's automated testing survey, we see that integration tests are on the rise, while the frequency of component and performance testing is falling. In last year's survey, 61% of respondents automated their integration tests, which, when compared with this year's data, constitutes a 16% increase. Automated performance tests, however, suffered a 5% decrease year-over-year, and automated component tests saw a 6% year-over-over decrease.

Since manual testing still has its place in the development pipeline, and there's a small, but not insignificant, faction of organizations who do not use automated testing, let's quickly look at where and how manual tests are used and how this compares to automated tests. When asked at what stage they begin manual testing, 51% reported development, 40% said staging/QA/testing, and 7% told us deployment. Not too surprisingly, manual tests are used in much the same places in the pipeline as automated tests.

**GRAPH 01.** Do you have an officially designated DevOps team in your organization?




**GRAPH 02.** What programming language ecosystems does your company use?



When it comes the types of tests, however, we see some divergence between manual and automated. When asked, 'Which tests in your organization's pipeline(s) are currently performed manually?', 60% said user acceptance tests, 50% usability tests, 41% story-level tests, and 38% integration tests.

Interestingly, despite the year-over-year decrease in automated component and performance tests noted above, the instances of manual component, performance, and integration tests all decreased. In 2017, 41% of respondents performed manual integration tests, while in 2018 this number fell to 38%; manual performance tests decreased from 45% in 2017 to 34% in 2018; and manual component tests fell from 40% in 2017 to 29% in 2018.

## CONTINUING THE DEVOPS TREND

As automated testing is a component of larger DevOps trends, we asked our community what other DevOps practices they have adopted. Surprisingly, only 48% reported having a designated DevOps team in their organization. Despite this lower-than-expected number, the percentage of DevOps teams who automate their integration tests increased from 49% in our 2017 survey to 57% this year. Additionally, the percentages of organizations with DevOps teams who have automated their component and performance tests rose from 51% in 2017 to 53% for component tests and 55% for performance tests in this year's survey.

Test-driven development and CI/CD were some other popular DevOps methodologies reported in this survey. 64% of respondents work for organizations that have implemented TDD in some capacity, with 36% saying they only use TDD at certain times during the development process. The biggest benefits reported from TDD were improved code quality (77%), less time spent debugging (60%), and easily maintainable code (46%). Additionally, 59% of respondents believe their organization has achieved Continuous Delivery to some degree, and 65% said their organization has achieved Continuous Integration for at least some projects.

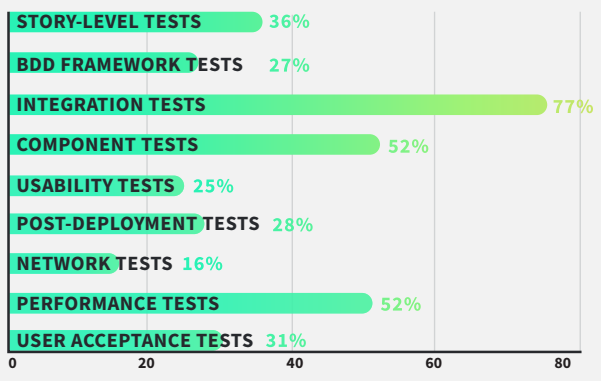
## TOOLS, LANGUAGES, AND AUTOMATED TESTS

Java was far-and-away the most popular language and ecosystem among this year's respondents. 86% of those surveyed reported that their company uses the Java ecosystem, and 65% of respondents actively work in the Java language. Other popular programming language ecosystems were client-side JavaScript (69%), Python (37%), and Node.js (36%). In terms of tools, three clear favorites emerged. The most popular testing tool among our audience is JUnit, with 60% of respondents claiming to use this tool in their testing. Selenium finished second with 52% of respondents using it (up from 46% in 2017), and Apache JMeter ranked third with a 41% adoption rate among our community (down from 45% last year).

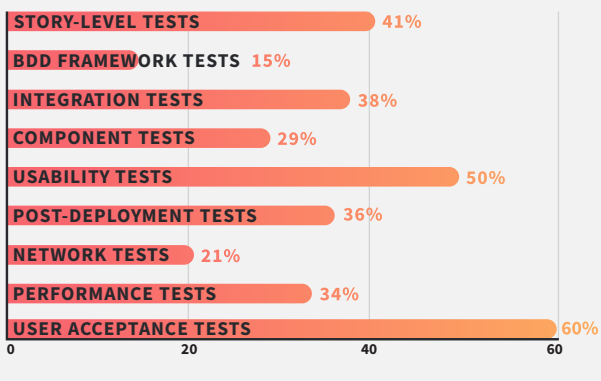
When we compare the number of developers working in the Java ecosystem with these three popular testing tools, we see a similar trend in terms of adoption rates. In 2017, 68% of Java ecosystem developers actively worked with JUnit to automate various tests; in 2018 this number stayed relatively stable at 69%. Selenium, however, saw a decent rise in its adoption, increasing from 48% of Java ecosystem developers in 2017 to 55% this year. This rise in Selenium accompanied (or perhaps caused) an inverse trend in the use of JMeter. In 2017, 50% of Java ecosystem developers reported that they worked with JMeter, but this year this number fell to 45%.

If we compare these three main tools to the three main types of automated test (automation, component, and performance) we see a similar trend as the one we observed earlier in the "Automated vs. Manual Testing" section. Among Selenium users, automated integration tests saw a 6% year-over-year increase (77-83%), whereas automated performance tests declined by 35% (61-58%), and automated component tests witnessed a precipitous decline, falling 20% (71-51%).

**GRAPH 03.** Which tests in your organization's pipeline(s) are currently automated?




**GRAPH 04.** Which tests in your organization's pipeline(s) are currently performed manually?



functionize

A Revolutionary

# Autonomous Testing

Platform in the Cloud Powered By

- Artificial Intelligence / Machine Learning
- Natural Language Processing
- Computer Vision

www.Functionize.com

# Autonomous Testing Is the Catalyst of Continuous Development

The adoption of test automation remains well under 40% in enterprises today. Functionize's autonomous cloud platform is making continuous testing a reality through our AI engine in the following ways:

1. NLP + AI modeling allows testers to submit a user journey in plain English that is automatically converted into a functional test case.
2. Completely autonomous test creation based on live user data.
3. Execution of thousands of test cases in parallel across all browsers.
4. Self-healing test cases via Adaptive Event Analysis™.

The benefits to enterprises that implement Functionize's self-healing testing automation into their DevOps strategies are manifold:

1. **Significant cost reductions**: Labor hours are reduced as scripting and the majority of manual maintenance tasks are depreciated.
2. **Faster time-to-market**: Scale for creation, execution, and maintenance enables product to be deployed on time.
3. **Better UX**: Self-healing allows enterprises to have confidence that major errors won't be introduced, which can irreparably damage brand.
4. **Executive insights**: Actionable insights provided by continuous data feedback allows better resource utilization.

When large-scale projects suffer from massive delays, the C-suite is confronted to reexamine agile deployment processes which are hampered by QA cycles. Functionize enables faster software releases and higher customer satisfaction—while reducing the risks inherent in current QA testing approaches.

**WRITTEN BY TAMAS CSER**
FOUNDER AND CEO - FUNCTIONIZE



**PARTNER SPOTLIGHT**

## Functionize

*Functionize delivers an autonomous testing platform that incorporates AI and machine learning technologies to automate the painstaking software testing process.*

**CATEGORY**
Production-Grade Containers

**RELEASE SCHEDULE**
Continuous, SaaS delivery

**OPEN SOURCE?**
No

**CASE STUDY**

Zenefits provides a human resources (HR) software platform used by more than 10,000 small and midsize businesses throughout the United States. For Zenefits to succeed, certain functions of its HR software platform must work flawlessly. These can't-break functions include things such as employee hiring, onboarding, and termination; benefits enrollment including medical insurance; and payroll including deductions, taxes, raises, and bonuses.

Zenefits was using manual QA testing processes for its HR software platform—and relying on a QA team of three people to do all the testing. "Our manual testing approach was definitely not scalable," said Batraski. Batraski began looking for a QA testing solution that could raise the testing throughput, make testing more accurate, and ease the pressure on the QA team. After evaluating some open-source solutions, Selenium-based testing tools, and all in-one platforms, Batraski chose the Functionize autonomous testing platform. "Functionize has the best all-in-one QA testing product I've seen," said Batraski. "And it's real technology, not vaporware."

**STRENGTHS**

- NLP + AI-test creation
- Autonomous test creation
- Self-healing maintenance
- Elastic scalability across all browsers, including mobile web

**NOTABLE USERS**

- Hyundai
- Zenefits
- Salesforce
- Intuit

**EMAIL** Info@functionize.com **TWITTER** @functionize **BLOG** functionize.com/blog

# Continuous Testing and Progressive Web Apps

**QUICK VIEW**

**01.** It takes automated testing to make continuous delivery possible — and practically speaking, the right test automation at the best possible time.

**02.** Continuous testing is a critical part of gaining efficiencies throughout DevOps.

**03.** Progressive web apps technology has arrived, and when implemented well, proves to streamline SDLC while ensuring great UX.

**BY CARLO CADET**
SENIOR DIRECTOR PRODUCT MARKETING AT **PERFECTO MOBILE**



Continuous testing (CT) is all about getting feedback to developers more quickly. In theory, this should increase their productivity and allow them more time to innovate. Whether you subscribe to an "Agile" philosophy, a DevOps methodology, or the principles of Continuous Delivery, in practice, it takes *automation* to make CT work.

In the current era of digital transformation, organizations are always looking for new ways to offer a better user experience (UX) in their online offerings. Traditional web and native app technologies have steadily given way to responsive web design (RWD); for a few years now, it may have looked as though RWD was the best hope for developing a universal cross-platform experience for end users. Recent developments (especially Apple's decision in March of this year to finally back "service workers" and Google's solution for adding more app-like capabilities to the web experience) mean that developers now have a new technology with which to create a uniform experience across all platforms:  progressive web applications (PWA).

PWAs are a powerful new tool for ensuring a great UX within the confines of mobile web technology; however, they do require a rethink regarding how to implement continuous testing.

Continuous testing begins with the developer, who can run unit tests as part of every local build. Once the code is checked in, integration and other system-level tests are run automatically. If those tests pass, automated end-to-end tests can run in order to ensure that things still work as expected. Other testing — stress tests, performance tests, or other large tests — may also be run before the code is deemed ready for release. After this point, monitoring and alerts are (for some devs, anyway) other types of testing that may happen during production.

Basically, CT is lots of tests, each running at the most appropriate point in the development cycle.

Testers want to create tests that will find issues with the customer experience. They also want to get feedback from tests as quickly as possible; to make this happen, they target their tests to find issues at the earliest possible time. Bugs that can be found by unit tests should be found by unit tests. The same is true for acceptance and integration tests and is especially important with end-to-end tests. While a lot of automation efforts focus on end-to-end tests, they should only exist for bugs that can only be found with end-to-end tests.

Software products with good CT systems enable teams to deliver changes and updates to customers frequently and safely and to use testing at every single integration point to help us determine whether we're heading in the right direction. In the context of CT, one promising aspect of PWAs is that teams can release as frequently as they see fit — directly to the user's device/browser without having to go through the

process of clearing it through the Apple App Store or Google Play Store.

CT is not just about having a lot of automation. It's about having the right automation running at the best possible time to find issues. For some web services, automated tests may be all you need in order to produce a good product. The amount and types of testing you do, your desired shipping frequency, your customers, and various risks all combine to help you make this business decision.

## PROGRESSIVE WEB APPS: NEW TECH, NEW CHALLENGES

Across the board, early reports confirm that progressive web apps improve user experience, grow engagement, and increase conversions. PWAs offer users access to features that, until now, were restricted to native apps: browsing while offline, receiving push notifications, and having access to platform features like cameras, microphone, speakers, data storage, and GPS. These capabilities are propelling PWA adoption and enriching user experiences. Everyone seems to agree: PWA adoption is expected to increase. In a recent survey conducted by Perfecto, 41% of respondents plan to add PWA capabilities to their sites already, followed by a responsive web design over the next 12 months, with an additional 32% actively researching the transition. Gartner has stated that by 2020, 50% of all native apps will transition to PWAs. Teams are seeing the benefits of PWAs and taking notice of their success and efficiency.

Instead of remaining a niche tool for the Android market, with Apple's change of heart, PWAs became a viable way of reimagining everyone's internet presence, on all platforms. As the technology develops, we can expect new levels of creativity and features. For the time being, the data seems to suggest that big companies are using PWAs as a way to start with a fresh/lean codebase while devising ways to cut page load times - and to great effect. Take a quick look at a few examples of companies that are starting to see big payoffs from making the switch to progressive web:

### STARBUCKS

Their PWA achieved results well beyond the main scope of the project. Simply by the nature of the specific requirements of a PWA, the final version of their PWA clocked in at 233KB — 99.84% smaller than their 148MB (that's megabytes) mobile iOS app! Naturally, this enormously improved load times and responsiveness, making the web app a hit with users.

### TINDER

Tinder took three months to build and implement their MVP progressive web app based on React and Redux. The result of their efforts is delivering the core Tinder experience at ten percent of the former data costs for someone in a data-costly or data-scarce market. They managed to shrink their 30MB native app to a 2.8MB progressive web app — which is downright impressive. Additionally, they cut load times from ~12 seconds to ~4.5 seconds.

### PINTEREST

With Pinterest's previous mobile web app, users often had to wait as much as 23 seconds before any UI was usable at all. Pinterest set out (much like the previous examples) to create a fast, light experience that would ultimately lead to better engagement and conversion. The Pinterest team reduced the "time to interactive" by a staggering 75%, from 23s to 5.6s.

## A CONVERGENCE OF EFFICIENCIES: PWAS AND CONTINUOUS TESTING

The teams at Starbucks, Tinder, and Pinterest (and others) have all gotten an early jump on PWAs, and now, with support for iOS (as of 11.3), they are well-positioned to offer all users a delightfully fast and robust experience with their lean apps. Developing a PWA seems to be a challenge that teams are rising to meet — but working PWAs into the continuous testing process is another additive element in the DevOps pipeline. Not only are there slightly different features/behaviors to test but, in addition, there are other native/RWD apps that still need to be supported/tested/developed/monitored. There's no doubt that DevOps teams are subject to ever-increasing levels of intricacy for the entire build-release process. Those teams that pull PWAs into their continuous testing process methodologies are going to be better able to meet user demands for fast, always-available apps that give them the instant gratification they have come to expect.

**CARLO CADET** leads Product Marketing at Perfecto, a continuous testing and monitoring platform for optimizing DevOps pipelines. He is responsible for go-to-market strategy, and manages the Analyst & Public Relations as well as the company's Customer Advisory Board. He speaks at industry events such as StarEast, STP CON, Jenkins User Conferences, as well as hosting webinars. Previously, he held similar responsibilities at RSA Security. During that time he acquired CISA and CISSP certifications. He earned an MBA from the MIT Sloan School of Management and a BSc in Computer Science from Tufts University. He served as a Captain in the United States Air Force.

# Advancing QA
# with machine intelligence.

## easy test automation
Create automated functional
UI tests without scripting.

## auto-healing tests
Tests repair themselves when
changes to your UI occur.

## complete test output
Logs and screenshots provide
fast issue diagnosis & repair.

mabl | Functional UI testing made reliable and easy.
Create a free account at www.mabl.com.

# Modern Development Gives Rise to Intelligent Testing

### THE STATE OF SOFTWARE TESTING IS IN PERIL

The state of software testing has not kept pace with advances in software development over the past 10 years. Faster feature development, shorter release cycles, evolving requirements, and unclear accountability for end-to-end quality all present real challenges for QA. However, a new generation of QA tools are emerging to confront these challenges head on.

### MACHINE INTELLIGENCE IS HERE

Innovations in machine intelligence and test automation have laid the groundwork for next-generation QA tools. These tools will make it easy to create and maintain tests, they will adapt seamlessly to change, run in the cloud, and be fully integrated into the delivery pipeline. They will share test results and insights through Slack, Jira, email, and other common communications channels. They will take feedback and training from the entire team—including product, development, QA, and customer success—and incorporate what they learn into their tests. Most importantly, they will describe tests and results in plain language, and provide useful context to help developers reproduce and fix issues quickly.

### THE ERA OF INTELLIGENT TESTING

Software QA has never been easy, and the quickening pace of software development has made it even more challenging. By incorporating cutting-edge machine intelligence, delivered from the cloud and embedded into the modern development workflow, next-generation QA tools will dramatically improve the effectiveness of QA, ushering in a new era of efficiency and innovation across the software industry.

**WRITTEN BY DAN BELCHER**
CO-FOUNDER - MABL



**PARTNER SPOTLIGHT**

## mabl

*Functional UI testing made reliable and easy*

CATEGORY
Automated Testing

RELEASE SCHEDULE
Continuously

OPEN SOURCE?
No

CASE STUDY

James is the QA and automation engineer for 3 teams at Jobcase. He used to spend more time fixing broken Selenium tests than writing new tests. Because of this, production and many functionalities remained untested, especially dynamic parts of the Jobcase application. James is now running mabl tests on a daily schedule, and on deployment. The data coming in hourly from mabl is useful for monitoring production, and mabl's link crawler proactively uncovers bugs beyond the scope of written test cases; James was able to fix a bug in the apps login flow based on a JavaScript error that mabl found within the first 24 hours of running. James is planning to move away from Selenium completely and do all QA with mabl for its stability and ease of use.

STRENGTHS

- Create automated functional UI tests without scripting
- Automatically repair tests when your UI changes
- Diagnose and repair issues quickly with comprehensive test logs and screenshots
- Automatically compare visual diffs at every step of a test
- Easily Integrate with your existing software delivery pipeline

NOTABLE USERS

- Turo
- Simpology
- JobCase
- PassPortal
- IndoorMedia

**WEBSITE** mabl.com | **TWITTER** @mablhq | **BLOG** mabl.com/blog

# How Your Project Will Take a Leap With Test-Driven Development

## QUICK VIEW

**01.** Examine the fundamentals and rules behind test-driven development (TDD).

**02.** Look at how TDD fits into the Agile, Lean, and DevOps spaces.

**03.** Consider the pros and cons of TDD as well as tips on how to successfully implement the process.

**BY STEFAN THORPE**
CTO AT CAYLENT

## WHAT IS TEST-DRIVEN DEVELOPMENT?

Test-driven development (TDD) refers to an approach in software development geared towards reducing errors and improving flexibility when designing applications. It is a process that encourages a quick, rapid, and fearless testing development style. This approach to programming enables tests to drive the design of the whole system.

The following set of rules (designed originally by Robert C. Martin or "Uncle Bob") defines the best practice guidelines for TDD:

1. You are not allowed to write any production code unless it is to make a failing unit test pass.
2. You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.
3. You are not allowed to write any more production code than is sufficient to pass the one failing unit test.

Abiding by these rules will provide IT teams with a tight feedback loop between production code and test code.

As highlighted earlier, the aim is to first write a unit test before any production code is started. The aim is to develop the system (and high-quality code) gradually through an ongoing sequence of tiny test improvements. By rule two, you obviously can't write much of the unit test itself other than is required for the test code to fail. This approach is generally referred to as red, green, and refactor. Write a small failing test (red), and then help it to pass by writing good production code (green). Then, the final step is to refunction the code (refactor) before it's possible to restart the process. (Callaway and Hunt, 2018.)

## DOES TDD FIT IN THE AGILE, LEAN, AND DEVOPS SPACES?

### WHERE TDD FITS IN THE AGILE AND LEAN SPACE

For projects already affiliated with Agile methodologies of software development, implementing TDD is fairly simple. It is ideally suited for building complex projects with many features to clearly defined requirements — an integral ideal of Agile. TDD will help to improve the efficiency of the development process and encourage teams to produce products with reliable scalability and desired quality. The TDD approach is highly applicable to greenfield projects being developed from scratch.

If the project is the development of a basic app or building on proof-of-concept, TDD is not necessarily justifiable. It can admittedly be time-consuming. On the other hand, it's not realistic to buy a car that hasn't been through any quality control (QC), so there's no real argument for leaving QC out of a software development process, either.

In Lean Software Development, the integral aim is to reduce waste in projects. The application of test-driven development reinforces this core aspect. Furthermore, TDD is useful for building quality into the process by eliminating defects from the very beginning rather than at the end of the system where most QC takes place. Though building quality in is a central Lean principle, it isn't always easy. Implementing TDD will help overcome this hurdle.

### TDD IN THE DEVOPS SPACE

In DevOps, high-performing IT teams have turned to TDD to produce high-quality end products. While the DevOps umbrella can cover many workflows and ideas, here are the ideal practices you can incorporate with TDD involved.

#### EMBRACE AUTOMATION

DevOps processes advocate breaking down heavy complex projects for everyone involved in the project. All stakeholders are continuously involved in the whole software development process and DevOps promotes a flow of work that reduces any negative impact on anyone else down the value stream. In this case, TDD can be implemented by first creating the initial tests and running them manually (as suggested in *The DevOps Handbook*) and once they're written, they can be added to an automation pipeline to save time in the future.

#### BRING END-USER FEEDBACK FRONT AND CENTER

The DevOps methodology entails fostering a culture of collaboration, especially with the ultimate end-users. Involving the end-users early in the process by constantly getting their feedback is crucial to make the final product meet their expectations. Teams can use TDD to engage in initial

tests needed to help design the end-product in accordance with a thorough understanding of the needs of end-users. Having this knowledge will help to develop useful initial tests.

#### ADOPTION OF AN ENTERPRISE TEST MANAGEMENT TOOL (ETMT)

A robust ETMT tool like Jenkins, XL TestView, or Selenium can provide crucial support in handling all the necessary tests in TDD. Such a tool guarantees that everyone receives all the necessary information at all stages of development — a truly DevOps concept. ETMTs help those new to the TDD concept, as well as the entire team, receive all the benefits of a TDD environment. (Kim et al., 2016.)

## PROS AND CONS OF USING TEST-DRIVEN DEVELOPMENT

### ADVANTAGES OF TDD

1. Coding becomes modular when executing small tests at a time. TDD assists in understanding, learning, and internalizing the fundamentals of good modular design.
2. Inherent code structural problems surface much earlier and can be addressed swiftly. Thus, TDD facilitates good architecture.
3. TDD simplifies the maintenance and refactoring of code. It also facilitates clarity during the implementation stage. In addition, it provides a safety cushion during the refactoring step.
4. TDD offers developers greater insight into the perspective of the end-users' POVs.
5. TDD helps reduce the defects rate. The identification of design or requirement issues happens right at the start when it is early enough to be fixed without having any impact down the value stream.
6. Implementing TDD encourages the use of an automated regression test suite. There is no need to spend time testing implementation code after writing unit tests — it's built into the process from the start.

### DISADVANTAGES OF TDD

1. TDD slows development at the beginning, as it necessitates ample time and effort to write unit tests.
2. It does not necessarily guarantee unit test quality. Focus goes on metrics such as code coverage, which does not actually assess quality. (DevOps approaches such as peer reviews and paired programming can mitigate some of this step.)
3. In complicated scenarios, test cases can be tough to calculate and create.
4. If there is a rapid change of design, changing tests is also necessary, which can slow development. These changes can result in a lot of time wasted; e.g. writing tests for features that may get dropped later. (Good product design and planning can address the issue of going off-project.)

## TIPS ON HOW TO IMPLEMENT TEST-DRIVEN DEVELOPMENT

### 1. ENSURE TDD IS THE RIGHT FIT FOR YOUR PROJECT

TDD can provide a much-needed boost of confidence to teams, but it can also be a false source of security. First, ascertain that TDD is the appropriate approach for your project. TDD is ideal for products where you know the exact specifications (or have time to generate these) or where you're trying to stabilize an existing one — in which case, this process can serve as the best route to a quality product.

### 2. DECIDE THE INPUTS AND THE OUTPUTS

Start the process from a high level of determining your eventual product goal. Decide the inputs necessary to achieve that product goal and then outline the necessary outputs/feature steps to reach it. Consider the overall function of the product, as well as the data that the function will require to run and the results that this function will return. This step is an integral parameter to consider before writing any code or test.

### 3. SELECT THE FUNCTION SIGNATURE

Having established what data goes in and the outcome to expect, it's now vital to choose the function signature. These are the parameters that the function takes and the expectations that the function will return a certain result. This step is similar to writing code without TDD. Generally, in code writing, deciding the parameters and return value is a necessary step.

### 4. DECIDE ON ONE SMALL ASPECT OF FUNCTIONALITY

Figure out the most straightforward possible behavior that draws your team closer to the end goal. Using simple behavior can assist to determine the smallest amount of code for each step of your process. This code can bring the function closer to becoming a working end-product.

### 5. IMPLEMENT TEST

It is worth noting that all previous steps were similar to programming without TDD. The significant difference is the focus on implementing the function. This is opposed to how the functions behave concerning certain conditions. Your focus is to test how the functions behave at every step. When you begin testing behavior at every step, the testing gets easier the closer you get to the end-product.

### 6. IMPLEMENT CODE

You should practically write the slightest amount of code to pass each unit test as outlined at the beginning of this article. To advance on the same function, continue with these small steps, repeating from Step 4. TDD gets a lot easier to achieve as your team gets used to the process, and it helps in eliminating tedious and helpless coding after a product launch.

To make your organization successful, adapt to the changing environments and address project improvement through TDD. Spending time on the process early on will boost productivity, output, and market value much later down the line. Test-driven development is the ideal approach to achieving application flexibility, extensibility, and maintainability.

## REFERENCES


- Callaway, J. and Hunt, C. (2018). Practical test-driven development using C# 7. Packt Publishing.
- Kim, G., Humble, J., Debois, P., and Willis, J. (2016). *The DevOps Handbook*. Portland: IT Revolution Press.
- Martin, R. (2005). *The Three Rules Of TDD*. Available here. (Accessed 14 Jun. 2018).



**STEFAN THORPE** is CTO for the DevOps-as-a-Service consultancy management platform Caylent. He is an IT professional and DevOps evangelist with 20+ years of management and hands-on experience providing technical solutions to support strategic business objectives. Stefan leads and fine-tunes development teams to optimize for innovation and growth.


WITH CONTINUOUS TESTING
EVERY STEP MATTERS
Jenkins
XCUITest
Protractor
PWA
Bamboo
slack
appium
Visual Studio
Team Foundation Server
JIRA
Perfecto
perfecto.io/continuous-testing-pwa
DevOps teams are subject to ever-increasing levels of intricacy for the entire build->release process. Teams that pull PWAs into their continuous testing process methodologies are going to meet user demands for fast, always-available apps that deliver the instant gratification they have come to expect. Perfecto has you covered as the first platform to fully support automated continuous testing of PWA

# Leading Retailer Grows with Perfecto to Support Automated Progressive Web App Testing

A leading retailer originally selected Perfecto's Continuous Quality Lab for its native mobile app development. A year later, as the web team embraced responsive design, it saw web traffic from mobile devices climbing. They decided to shift their testing using Protractor from internally managed VMs to Perfecto's cloud-based solution. CI-triggered testing is executed in parallel on twenty-four platforms. A 15-minute smoke test is executed every commit on eight platforms complemented by nightly regression tests executed on all platforms.

The transition to Perfecto's cross-browser testing platform enabled automated testing of desktop and mobile browsers in parallel.

The ability to compare test results side-by-side helped accelerate triaging defects, helping contribute to the team successfully coding, testing and deploying within the same sprint. Integrating Perfecto into itsDevops toolchain, the dev team also laid the foundation for supporting initial experimentation with Progressive Web Apps (PWA), specifically for offering customers offline access.

Early investigation of PWA capabilities suggested that development velocity might slow given the need to revert to manual testing. However, it became clear that Perfecto was able to help them navigate automating testing of PWAs. The team was already familiar with camera-based scenarios, utilizing visual analysis and programmatically degrading network conditions to simulate users transitioning from one to five bars. The team got a head start on tackling automating Progressive Web App testing. Perfecto's recently announced support testing PWA on mobile devices (lauched from the home screen) enabled all scenarios to be successfully automated.

**WRITTEN BY CARLO CADET**
SENIOR DIRECTOR, PRODUCT MARKETING - PERFECTO

**PARTNER SPOTLIGHT**

# Continuous Quality Lab

*Seek Perfection*


**CATEGORY**
Automated Testing Platform

**RELEASE SCHEDULE**
Continuous Deployment

**OPEN SOURCE?**
Yes

**CASE STUDY**

Google is championing Progressive Web Apps (PWA). The good news is that other browser providers are now rallying to support this new(ish) technology.

Early results demonstrate that the combination of adding offline access and native mobile capabilities delivers better experiences and increases conversions.

Perfecto supports key automated PWA test scenarios.

- **Launching PWAs** – Launching PWAs from device home screens creates a new challenge for test automation.
- **Test authoring & execution** – Perfecto provides a single view of all objects in an application under test. As a result, identifying and interacting with both web and native iOS and Android objects in a single flow is fully supported. This overcomes Selenium and Appium limitations.
- **Test audio, camera, and location-based scenarios** – PWAs bring the rich engagement capabilities of mobile devices to websites.
- **User condition test scenarios** – Offline testing is easy with Perfecto's Wind Tunnel™ capability.

**NOTABLE USERS**

- AT&T
- GE
- IBM
- Lloyd's of London
- Toyota

**WEBSITE** perfecto.io **TWITTER** @perfectomobile **BLOG** blog.perfecto.io

# AI in Test Automation

**BY TAMAS CSER**
THE FOUNDER AND CEO AT FUNCTIONIZE

## QUICK VIEW

**01.** If artificial intelligence doesn't excel beyond automating simple, repetitive tasks, then it's not going to meaningfully impact ROI.

**02.** True AI should empower personnel to be more effective, as their focus is shifted away from less repeatable tasks and more toward creativity and innovation.

**03.** With AI, the C-Suite can focus on real-time analytics that impact the bottom line.



Today, almost all IT projects are faced with the challenge of operationalizing and deploying software and services with greater speed and accuracy, creating an unrelenting, high-pressure environment for the project team. Requirements shift daily and there are never enough engineers to make it all happen perfectly. A major part of the burden on project teams is the need for continuous testing.

In this article, I will explore the opportunities I've discovered by applying artificial intelligence (AI) to test automation. AI is meant to make businesses far more capable and efficient. The best companies are using AI to enhance customer and client interactions, not eliminate them. Big data collection and algorithmic advances are vastly extending the scope of testing automation, making it possible for non-technical team members to define and scale tests with levels of capability and sophistication comparable to or even greater than developers'. In short, AI is transforming all facets of test automation by streamlining creation, execution, and maintenance, and providing businesses with actionable insights in real-time that directly affect the bottom line.

## BACKGROUND

More than 12 years ago, I launched a consulting business serving both startup and enterprise clients alike. As I served my clients and looked to streamline the time between committing a change to an application and the change being deployed into live production, I found that the need to ensure quality and reliability demanded greater and greater shares of resources. There is an ever-increasing variety of combinations of innovation, application components, and protocols that interact within a single event or transaction. Over time, I realized that there was a need for something more. Then, in the fall of 2013, I saw how AI could shape the testing landscape, and I wrote the first line of code for Functionize.

## THE ORIGINS AND LIMITATIONS OF TEST AUTOMATION

Test automation is not new. The advent of Selenium in 2004 was a major advancement in empowering developers to take further control of QA. However, the challenges of Selenium and popular record/replay frameworks became readily apparent to developers who sought to use the recorder in complex environments, as the selectors used for identifying elements had to be continuously updated with every code change.

## TEST CREATION GUIDED BY MARKET DEMAND

Automated test creation has often been limited to three methodologies: manual, scripted languages (some with greater degrees of modularity than others), and record/replay tools. Each on their own offers value, but they are all constrained in conspicuous ways:

1. Manual testing is slow and not designed for today's CI/CD pipelines because it can't scale with complex applications.
2. Scripting is laborious, error-prone, and expensive, as engineering resources are usually required.
3. Record/replay tools struggle to capture sophisticated user workflows, and editing those workflows often requires having to re-record everything.

As I began listening to the market and our customers, it became clear that different options for test creation were desirable, but often not presented within the same tool. With Functionize, I sought to offer both conventional and new modes of test creation, all enhanced by AI:

- Simply writing a user journey in plain English or submitting a sequenced set of desired tests to our NLP engine, which uses AI to analyze and model the data.
- Training our AI modeler that learns the application.
- Using our Developer Mode that enables robot-compatible scripting and automatically builds smart Page Object libraries that are modular and portable.

- Fully autonomous test creation that analyzes and generates test cases from live user data.

## HOW AI IMPACTS TEST CREATION

There is a lot of noise in the market around AI in test automation. Below are a handful of examples that serve as a litmus test for judging the degree to which AI is present in test creation.

- Machine vision that automatically locates and identifies hundreds of selectors. This requires a much broader focus and ingest than just HTML and CSS.
- AI and machine learning to continually scan and analyze the DOM and application states for meaningful information, rejecting noise and irrelevancies.
- Page object recognition that happens continually and autonomously, increasing test modularity and scalability.
- Fully autonomous test creation utilizing AI technologies via natural language processing and advanced modeling.

However, even test automation frameworks that go beyond traditional scripting methods and employ an image- or visual-based approach remain constrained. Test creation remains time-consuming, as the tester must manually select and drag the desired element for interaction. There also remains a high degree of selector maintenance due to a pixel/image approach to object recognition. Market leaders are struggling to integrate AI into their automation stack, and the result is confusing jargon that mis-defines AI as *Awesome Integrations*, not *Artificial Intelligence*.

## HOW AI IMPACTS TEST EXECUTION

The dearth of true cloud scale test execution options reveals there is ample room for AI to drive new productivity. On-premise and even cloud technologies like Selenium Grid are still hampered by execution time, based on the number of nodes running, memory, and the number of concurrent tests. The whole purpose of cloud computing is the ability to perform rapid scaling of applications, up and down, dependent on the workload, with information shared across all execution instances.

As testers look for solutions to execute their tests at scale, the bar should be set very high if AI is claimed to be augmenting those processes. We set ourselves the following acceptance criteria:

- Tests should be executable at scale, in the cloud, so they become more efficient and reliable with every subsequent run and release.
- Tests should be executable from anywhere around the globe, from any device, with any bandwidth, and in all types of environments.
- Even the most complex tests should take minutes to execute — not hours, let alone days.

## HOW AI IMPACTS MAINTENANCE

Rapid test creation is only as viable as the resiliency of executed tests. The most efficient way to ensure that test maintenance is not the bottleneck in a deployment pipeline is to identify what is actually happening to the data during test creation. The failure point of test maintenance ultimately resolves to inadequate data modeling during creation. AI can help here:

- **Self-maintenance**: Resultant tests are modeled and thus maintained by the combination of an exhaustive and autonomous set of data points, such as the size of element, location on a page, previously known size and location, visual configuration, XPath, CSS selector, and parent/child elements.
- **Self-healing tests**: Root cause analysis highlights all potential causes for test failure and provides a path for one-click updates.
- **Data modeling:** Selector maintenance should be eliminated by having elements identified by hundreds of data points that are rated and ranked instead of a single selector.
- **Computer vision diagnosis**: AI means visual diagnosis is easy: identifying broken tests should take seconds in a visual environment and shouldn't require digging through scripts.

Applied AI methods such as these ensure that your testing frameworks learn the structure and organization that is unique to your application, minimizing human intervention.

## HOW AI-POWERED TESTING AUTOMATION TRANSFORMS BUSINESSES

Businesses that have a commitment to implementing AI at the enterprise level are already experiencing greater operational efficiency and better product results. Developers are renegotiating their involvement within Agile and DevOps strategies, as smart algorithms are now capable of tackling the most repetitive problems presented in testing automation. Not only is product development significantly streamlined when testing automation changes from the bottleneck to the catalyst within a CI/CD pipeline, but also, executives are provided with business intelligence previously unavailable that directly impacts the bottom line.

Functionize is partnering with Google Cloud to build advanced anomaly detection through canary testing where a small set of users are used for real-world testing of new code. AI is used to compare the experience of these users with those running the existing code. Anomalies can then be identified automatically, and details passed back to the developers.

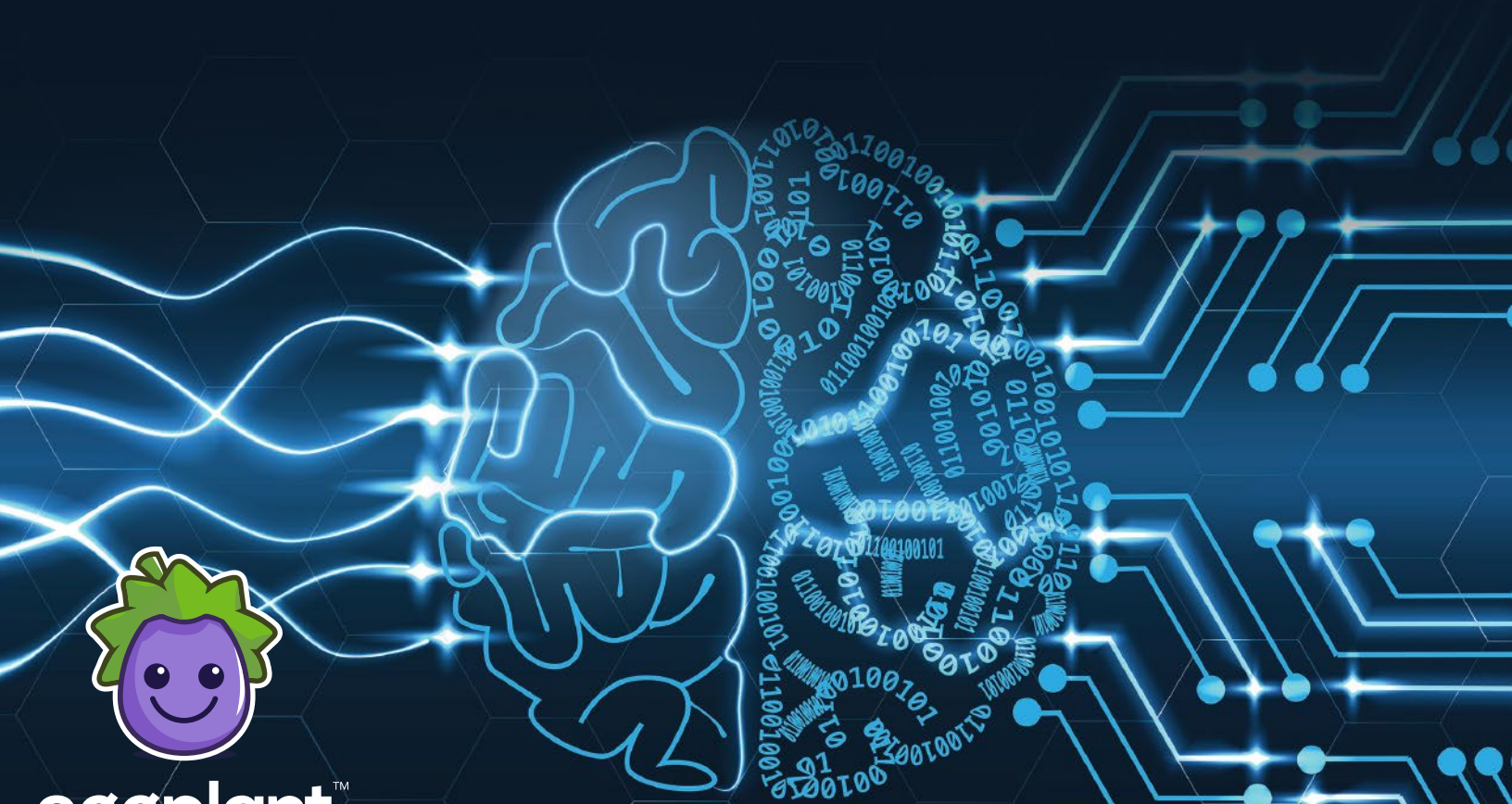
**TAMAS CSER** is the Founder and CEO of Functionize, the AI-powered autonomous testing platform in the cloud. It allows businesses to effectively balance the need to innovate with the need to deliver ever-better user experiences.

# Intelligently Automate Your End-To-End Testing

**Quickly and continuously create amazing digital experiences that drive positive business outcomes. Eggplant Intelligent Automation solutions use AI, machine learning, and predictive analytics to help you:**

- Transform testing from a compliance function to a profit center.
- Shrink time to market by testing new versions, fast.
- Accelerate productivity by scaling test coverage without blowing your budget.
- Keep up with the pace and evolution of DevOps.

## eggplant.io



eggplant™

# True Automated Testing Requires a Modern Approach

Everything about software has changed—how it's architected, developed and produced, what it does, what users want from it. To keep up, organizations are turning to CD and DevOps. Yet product teams still do lots of time-consuming, manual testing.

Achieving true test automation involves modern technology, products, and an approach with five key elements:

- **Test through the eyes of the user.** Engage with devices the way users do, not through the code.
- **Continuously test all aspects of the digital UX.** Test functionality, performance, and usability from a user perspective for a more intuitive way to test.
- **Expand automation beyond test execution using AI, machine learning, and analytics.** Automate more of the testing process to boost team productivity.
- **Report quality in terms of the user with predictive analytics.** Reveal quality specifics in user terms: releasing now will net a 4 out of 5 app rating.
- **Take a coherent approach to monitoring and testing.** Monitor UX in production and emulate real network conditions to get a true, end-to-end view of performance.

**AI and machine learning** build on directed regression tests with automated exploratory testing to find bugs faster and maximize test coverage. **Predictive analytics** combines results from various UX touchpoints to determine if a release is ready for prime time. **Smart image search** automates testing of what users actually see when interacting with an app. Use a hosted or on-premise **device lab** to automate test models simultaneously across platforms, environments and browsers.

All these technologies are key to improving digital experiences and ensuring your applications and workflows hit and exceed your business objectives.

**WRITTEN BY DR. JOHN BATES**
CEO, EGGPLANT



PARTNER SPOTLIGHT

# Eggplant Intelligent Automation

*An intelligent, user-centric, test automation suite that uses AI and machine learning to enhance the quality and performance of the digital experience.*

CATEGORY

Test automation, AI, UX testing, website monitoring, and performance

RELEASE SCHEDULE

Quarterly

OPEN SOURCE?

No

CASE STUDY

NTT DOCOMO is Japan's largest mobile telecommunications company, providing innovative, convenient, and secure mobile services to more than 74 million customers via advanced wireless networks. The end-user quality of internet services is determined by complicated factors including traffic that varies within the network, across a variety of content, with an external cloud environment, and based on the time of day. NTT DOCOMO uses Eggplant to test previously untestable areas, as well as deliver service confirmation and monitoring capabilities. With Eggplant, NTT DOCOMO can now continuously improve the user experience through 24/7 monitoring of live, online applications from the user viewpoint in a stable and proactive manner.

STRENGTHS

- AI, machine learning, predictive analytics
- True UX testing
- Bug-hunting capabilities

NOTABLE USERS

- Boeing
- Cerner
- Lockheed Martin
- P&G
- Walmart

**WEBSITE** eggplant.io **TWITTER** @eggplantio **BLOG** blog.testplant.com

# Embracing the Chaos of Chaos Engineering

**BY CHRIS WARD**
ZONE LEADER AT **DZONE**

## QUICK VIEW

**01.** Modern applications are increasingly growing in complexity.

**02.** A dizzying amount of moving parts, layers of abstraction, reliance on external systems, and distributed systems all result in a stack that few truly fully understand.

**03.** Chaos engineering aims to help you identify the seemingly random errors that plague our complex applications by injecting some if its own randomness.



Modern applications are increasingly growing in complexity. Adding a dizzying amount of moving parts, layers of abstraction, reliance on external systems, and distribution that all result in a stack that few truly fully understand.

Any developer worth hiring now knows the merits of a thorough testing regime, but one of the issues with testing is that you are often testing for predictable outcomes. Despite our "logical systems," show-stopping issues are typically unexpected; situations that no one foresaw.

These unforeseen eventualities are what chaos engineering attempts to account for. It's a reasonably new principle, practiced by Netflix for several years and then formalized in 2015, setting out its principles in a time-honored manifesto.

Naturally, there are critics of the practice, and the comments at the bottom of this TechCrunch article summarize some of them. The typical counter-arguments are that the principle is a band-aid for applications that were poorly planned and architected in the first place, or that it's another buzz-word-laden excuse to invent shiny new tools that no one knew they needed.

Still, it's proponents are a friendly bunch, so in this article, I summarize my findings on the practice and let you decide.

## ORGANIZED CHAOS

In many ways, while the term "chaos" is a good eye-catching phrase, it's misleading, summoning images of burning servers and hapless engineers running around an office screaming. A better term is *experimental engineering*, but I agree that is less likely to get tech blog or conference attention.

The core principles of chaos engineering follow similar lines to those you followed in school or university science classes:

1. Form a hypothesis.
2. Communicate to your team.
3. Run experiments.
4. Analyze the results.
5. Increase the scope.
6. Automate experiments.

Early in the lifetime of chaos engineering at Netflix, most engineers thought chaos engineering was about "breaking things in production," and it is in part. But while breaking things is great fun, it's not a useful activity unless you learn something from it.

These principles encourage you to introduce real-world events and events you expect to be able to handle. I wonder if fully embracing the "chaos" might result in more interesting results, *i.e.*, measuring the worst that could happen. True randomness and extremity could surface even more insightful results and observations.

Let's look at each of these steps in more detail.

### 1. FORM A HYPOTHESIS

To begin, you want to make an educated guess about what will happen in which scenarios. The key word here is "educated;" you need to gather data to support the hypothesis that you'll share with your team.

#### DECIDE ON YOUR STEADY STATE

What is "steady" depends on your application and use case, so decide on a set of metrics that are important to you and what variance in those metrics is acceptable. For example:

- When completing checkout, the majority of customers should have a successful payment processed.
- Users should experience latency below a particular rate.
- A process should complete within a timeframe.

When deciding on these metrics, also consider external factors such as SLAs and KPIs for your team or product(s).

#### INTRODUCE REAL-WORLD EVENTS

The sorts of events to test vary depending on your use case, but common to most applications are:

- Hardware/VM failure
- State inconsistency
- Running out of CPU, memory, or storage space
- Dependency issues
- Race conditions
- Traffic spikes
- Service unavailability

#### RUN IN PRODUCTION

"Testing in production" has long been a tongue-in-cheek reference to an untested code base, but as chaos engineering is likely run in collaboration with a properly pre-tested code base, it takes on a different meaning.

The principles we're working with here encourage you to undertake tests in production, or if you have a genuine reason for not doing so, as close as possible. Chaos engineering principles are designed to identify weakness, so they argue that running in production is fundamentally a good thing.

Some banks are already following these principles, and while engineers behind safety-critical systems should be confident of their setup before embarking on chaos engineering, the principles also recommend you design each experiment to have minimal impact and ensure you can abort at any time.

#### METRICS

While the most tempting hypothesis is "let's see what happens" (much like "let's just break things"), it's not a constructive one. Try to concoct a hypothesis based on your steady state, for example:

- If PayPal is unavailable, successful payments will drop by 20 percent.
- During high traffic, latency will increase by 500ms.
- If an entire AWS region is unavailable, a process will take 1 second longer to complete.

### 2. COMMUNICATE TO YOUR TEAM

As a technical communicator, this is perhaps the most important step to me. If you have a team of engineers running experiments on production systems, then relevant people (if not everyone) deserve to know. It's easy to remember engineers, but don't forget people who deal with the public, too, such as support and community staff who may start receiving questions from customers.

### 3. RUN YOUR EXPERIMENTS

The way you introduce your experiments varies, some from code deployments, others by injecting calls you know will fail, or simple scripts. There are myriad tools available to help simulate these; I've provided links to find them below.

Make sure you have alerting and reporting in place to stop an experiment if needed, but also to analyze results later.

### 4. ANALYZE THE RESULTS

There's no point in running an experiment if you don't take time to reflect on what data you gathered and to learn from it. There are many tools you probably already use to help with this stage, but make sure you involve input from any teams whose services were involved in the experiment.

### 5. INCREASE THE SCOPE

After defining your ideal metrics and the potential effects on them, it's time to start testing your hypothesis. Much like other aspects of modern software development, be sure to iterate these events, changing parameters or the events you test for.

Once you've tried one experiment, learned from it, and potentially fixed issues it identified, then move on to the next one. This may be introducing a new experiment or increasing the metrics of an existing one to find out where a system really starts to break down.

### 6. AUTOMATE THE EXPERIMENTS

The first time(s) you run an experiment, manually is fine — you can monitor the outcome and abort it if necessary. But you should (especially with teams that follow continuous deployment) automate your experiments as quickly as possible. This means that the experiment can run when new factors are introduced into an application, but it also makes it easier to change input parameters for the scope of your experiments.

Again, the resources section below lists places to find tools to help with this.

## QUIETLY CHAOTIC

While engineers and developers are divided on the usefulness of chaos engineering, the most interesting aspects to me are not the technical ones, but rather that it tests and checks ego.

The principles state in many places that if you are truly confident in your application, then you shouldn't fear what it proposes. They force you to put your money where your mouth is and (albeit in a careful and controlled way) prove your application is as confident as you are. I can imagine many insightful debriefing sessions after a chaos engineering experiment.

## TOOLS AND RESOURCES

- The free O'Reilly book on Chaos Engineering
- The comprehensive Chaos Engineering awesome list that features a plethora of useful tools and resources
- The Chaos Community, Mailing List, and Meetups
- Gremlin, whose staff are often behind a lot of the Netflix-independent chaos engineering resources, runs a 'Failure-as-a-service' platform that commoditizes many of the tools and practices featured in this post

**CHRIS WARD** is a technical writer, blogger, and DZone Zone Leader. He is always working on crazy projects and will speak to anyone who listens. His various articles and podcasts can be found at gregariousmammal.com.

## TREASURING

# Delightful Development

In 2018, 56% of DZone survey respondents reported that they were doing either automatic or manual testing during development. One tactic teams can use to increase testing during development is to implement test-driven development, a strategy that ensures that code can pass unit tests. To demonstrate how adopting test-driven development (TDD) can create better code, let's see how a team might reach some buried treasure when they use TDD and when they don't.


**1**

The basic rule of test-driven development is to write code that will pass tests. If the code doesn't pass the test, then it is not built to solve the intended use case or issue. Avoid your software falling into oblivion by making sure your development is test-driven.

**2**

Because TDD focuses so much on passing tests, it can improve the focus of the development team and avoid animosity. If a project has very clear acceptance criteria (i.e. to pass a test), the team is less likely to be pulled in different directions or get distracted with different features. 41% of survey respondents reported that TDD improved developer focus.

**3**

46% of survey respondents mention that using tests as documentation is a key benefit of test-driven development. Reading unit tests is a great way to see how software is intended to be used without having to rely on developers to keep documentation up-to-date for every fix. TDD can help you stay in the loop so that you avoid losing track of your ship.

**4**

79% of survey respondents reported that they saw code quality improve with test-driven development, and 62% said they spent less time debugging as a result. By using test-driven development, teams can accomplish their goals quickly, resulting in faster deploys with fewer rollbacks because the software was built to pass automated unit tests.

"Parasoft SOAtest has allowed us to implement an **automated and repeatable testing process** with the breadth of coverage needed to ensure **every release is stable, meets requirements, and prevents critical errors from reaching production**."

PARASOFT®
Automated Software Testing

# Make Continuous Testing a Reality with a Scalable and Maintainable Testing Strategy

Although it's critical to execute your test automation continuously to deliver software at the speed of agile, the software industry is still struggling to adopt test automation due to the complexity of creating, and time spent maintaining, meaningful test scenarios.

Using test automation tools that easily plug into your environment, you can enable greater levels of test automation and build a scalable and maintainable portfolio of test scenarios by aligning with a software testing pyramid. This starts with a foundation of unit tests, backed up with API/service-level tests and minimal reliance on end-to-end UI testing, which is difficult to maintain and debug.

With Parasoft's test automation tools, you can reduce technical complexity at each layer of the testing pyramid and ensure the tests can be executed continuously for immediate and ongoing feedback

- Parasoft Jtest, reducing the time for developers to create JUnit test cases by up to 50%
- Parasoft SOAtest, leveraging AI to dramatically increase efficiency of API testing and ensure you are testing both "API design" and "API use"
- Parasoft Virtualize, mitigating constraints within the test environment and ensure the test automation can be executed continuously
- Parasoft DTP, aggregating data from up and down the testing pyramid to provide complete visibility and advanced analytics to quickly understand risks coming from changes made to the code

With a focus on technical excellence and ease-of-use, Parasoft helps customers realize the potential of test automation and make Continuous Testing a reality.

**WRITTEN BY MARK LAMBERT**
VP OF PRODUCTS, PARASOFT



**PARTNER SPOTLIGHT**

# Parasoft SOAtest

*Bring artificial intelligence into your functional test automation to save time and money with functional, load, performance, and security testing*

PARASOFT®

**CATEGORY**
API Testing Solution

**RELEASE SCHEDULE**
3x per year

**OPEN SOURCE?**
No

**CASE STUDY**

When the Netherlands' Ministry of Security and Justice determined that quickly delivering reliable and stable applications required building and maintaining automatic checks and verifications for both functional and non-functional requirements, all integrated as part of the continuous integration later in their DTAP environments, they turned to Parasoft.

The team integrated Parasoft SOAtest into their continuous integration pipeline, which provided effective safeguarding of the required provable quality level. With Parasoft's assistance, they now sustain a high productivity level of rapidly delivering new functionality without having to compromise the strong commitment to product quality that is necessary in the judicial information chains.

**STRENGTHS**

- Automate end-to-end test scenarios across multiple interfaces and end-points.
- Create scriptless API tests that can be extended to load/performance and security tests.
- Use artificial intelligence to efficiently configure complex test scenarios.
- Test broadly with support for 120+ protocols and message formats.
- Easily integrate with service virtualization and test data management.

**NOTABLE USERS**

- Alaska Airlines
- AT&T
- CareFirst
- Comcast
- DELL

**WEBSITE** parasoft.com **TWITTER** @parasoft **BLOG** blog.parasoft.com

# Setting Up Automation for Webhooks Testing

### QUICK VIEW

**01.** Webhooks are a new API concept that many companies have begun to adapt

**02.** Write web-service separate from the app so you can easily scale.

**03.** Python is light way and rather easy way to get your web service running.

**04.** Create session keys for each automation run so you can easily retrieve data.

**05.** Consider using third party tools if that is going to save you time.

**BY SLAVEN SLUGIC**
AUTOMATION ARCHITECT/MANAGER AT SMARTSHEET



Webhooks are a new API concept that many companies have begun adapting. Unlike an API, which responds when you make a request, Webhooks are triggered when an event occurs in your application. Automating this process might be a little confusing to those unfamiliar with it, so I will provide some high-level information on how to set up automation for this type of testing.

## HOW IT WORKS

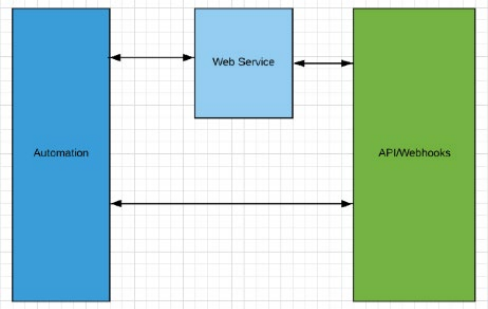
From a high-level perspective, Webhooks simply allow applications to provide real-time information as events occur. Imagine that you have created a spreadsheet and you would like to be notified any time information is added, updated, or deleted. This is where Webhooks come in handy — they send a callback back to your service that notifies you that a change has been made with relevant data pertaining to the change. This allows you to receive real-time updates rather than constantly having to make API calls to check if something has been changed.

## HOW TO AUTOMATE

To automate this process, you will need your framework to actively listen for callbacks from the API service so that you can then verify if the data received is correct. In short, every time you run automation, you need to run a web service along with it in order for Webhooks to work. This approach will work — but scaling issues will become apparent quickly. In order to deal with these issues, you will need to decouple your web service from your automation framework. What this means is that you will need to write a web service that will be running in background independent of your automation. Once it is running, automation will subscribe to the Webhooks API by providing the callback URL of your web service. Once you have the web service in place, your automation is free to make changes in your app through the API or UI and then verify if the information is correct by pulling that data from your web service.

From a high-level perspective, Webhooks simply allow applications to provide real-time information as events occur.

However, if you plan to have multiple automation runs at the same time, then you will need to keep track of that information in your web service. So, in short, you will need to assign a unique identifier to each run. In our automation, we call this the session key; each run needs to create its own session and use that session key to grab the callback data received from the Webhook API server.




## IMPLEMENTATION

Here is an example overview of how this may work:

1. Send a request to your service to create a session. The service will return a session ID.

| Description | Endpoint | Response |
|---|---|---|
| Create new session | /service/sessions/create | { "session_id": "0c6598cce2e24174ccdb9e8fca95e9" } |

2. Call your API to subscribe to Webhooks by providing your call-back URL along with the session key provided earlier. This will be your API endpoint along with the payload, which includes your callback URL. Your callback URL will look something like this: mysrvcurl/service/sessions/listen?session_id={id}.

| | | |
|---|---|---|
| Your service needs to send payload through this call | /service/sessions/listen?session_id={id} | {'msg':'success'} |

3. At this point, you can start testing your application by making changes and verifying that Webhooks are sending you correct information back to the callback URL that you provided. At a very high level, something like this could be done to retrieve the latest callback:

| | | |
|---|---|---|
| Get last service payload | /service/sessions/getlast?session_id={id} | returns last payload sent by your server |

4. Here is a sample of a web service definition:

| Description | Endpoint | Response |
|---|---|---|
| Create new session | /service/sessions/create | { "session_id": "0c6598cce2e24174ccdb9e8fca95e9" } |
| Get All Sessions | /service/sessions | [ { "session_id": "d93ef2f6197e711659fa35acf98bca" }, { "session_id": "1c67ece3a18efcd8f780f8e95e110f" }, { "session_id": "0580e0ceb17beed44a2e4794f7d80e" }, { "session_id": "810c16c97b6feceed4472c7ff28833" }, { "session_id": "0c6598cce2e24174ccdb9e8fca95e9" } ] |
| Your service needs to send payload through this call | /service/sessions/listen?session_id={id} | {'msg':'success'} |
| Get last service payload | /service/sessions/getlast?session_id={id} | returns last payload sent by your server |
| Get all service payload | /service/sessions/getall?session_id={id} | returns all payloads sent by your service server |
| Clear all payloads for the session | /service/sessions/clearall?session_id={id} | {"msg":"cache is cleared"} |
| Configure your response with different status codes | { "setError": { "isOn":true, "code":500 }, "setHeader": true, "setBody":true } | {"msg":"success"} |
| Configure custom headers and data | return specific header or data to be returned to your service | Not yet available |

In short, every time you run automation, you need to run a web service along with it in order for Webhooks to work.

There are various ways that this web service can be implemented, but ours was designed in Python using the webpy module. It is very light-weight and easy to use, and can spin up a web service within minutes.

The following is a sample code detailing how to quickly get your Hello World service running in Python:

```
import web

urls = (
   '/', 'index'    )

class index:
    def GET(self):
        return "Hello World"

if __name__ == "__main__":
    app = web.application(urls,globals())
    app.run()
```

To run this code, run `python filename.py`. This will run on *port 80* by default so you can navigate to localhost:80 to access it.

At this point, we can start defining our operations. The following is an example of all the endpoints you may need to write at minimum:

```
# definition of available endpoints
urls = (
  '/', 'index',
  '/service/sessions','sessions',
  '/service/sessions/create','create',
  '/service/sessions/getlast', 'getcache',
  '/service/sessions/getall', 'getallcache',
  '/service/sessions/clearall','clearcache',
  '/service/sessions/listen','posthook',
  '/service/sessions/configure','confighook',
)
```

Once the endpoints are defined, an implementation for each will need to be written. Here is some sample code of what that implementation may look like:

```
class create:
   def GET(self):
       session_id_list = []
       session_id = binascii.b2a_hex(os.urandom(15))
       response = {'session_id':session_id}
       session_id_list.append(response)

       return session_id

# post method for webhook server callback; We are
taking the header hook challenge value and returning
it back in our response
class posthook:
   def POST(self):
       try:
           session_id = web.input().session_
id.strip()
       except:
           return {'Error':'session_id required'}

       shc = web.ctx.env.get('HTTP_WEB_HOOK_
CHALLENGE')
       challenge_key = shc

       if(configCache and session_id in
configCache):
           flag = configCache.get(session_id)
['setHeader']
           body_flag = configCache.get(session_id)
['setBody']
           if(flag!=False):
               web.header("Web-Hook-Response", shc)

           if(body_flag==False):
               challenge_key=None
       else:
           web.header("Web-Hook-Response", shc)

       payload = web.data()

       return setData(payload,challenge_key,session_id)

// get last payload
def getLastCachedPayload(session_id):
   cLen = len(cache)
   result = cache.get(cLen-1)

   if (result == None):
       return {'info': 'cache is empty. try to post
some payload first'}
   else:

      while cLen > 0:
          cLen = cLen - 1
          result = cache.get(cLen)
```

*code continued on next column*

```
           if result[0]['session_id']==session_id:
              return json.dumps(result[1], indent=4)

# will return all cached data
def getAllCachedData(session_id):
   datalist = []
   for k in cache.keys():
       result = cache.get(k)
       if result[0]['session_id'] == session_id:
           datalist.append(result[1])
```

As you can see by the definition and examples above, Webhooks can be implemented quite quickly. Webhooks are a newer concept, but even so, it is a good idea to see if there already exists something that may fit your needs. Most of what existed at the time when our team began using Webhooks was not open-source, so I decided to write one for us. A UI can be designed for this, as well, allowing your manual testers to have an easy-to-use tool to test Webhooks for you, saving everyone time.

Webhooks are a newer concept, but even so, it is a good idea to see if there already exists something that may fit your needs.

## OTHER CONSIDERATIONS

SSL will be required for this to work and you may not have a certificate right away. One way to get around this to use service like NGROK. The NGROK service allows you to create a secure URL connection to your localhost server and can be used until you get your own certificate.

If you are planning to have a large amount of concurrent automation runs, then you will need to configure your webpy service to handle load more efficiently. There are few ways to do this, such as running it behind NGINX or using Lighthttpd. Lighthttpd is pretty easy to configure and you can find instructions for configuring it here.

**SLAVEN SLUGIC** is an engineer and manager working at Smartsheet. He has been in the industry for about 12 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.

# diving deeper

## INTO **AUTOMATED TESTING**

## twitter

 @martinfowler

 @angelovstanton

 @AutomatedTester

 @TourDeDave

 @jcolantonio

 @jezhumble

 @eviltester

 @Nikolay_A00

 @RealGeneKim

 @wakaleo

## books

**Experiences of Test Automation**
Software test automation has moved beyond a luxury to become a necessity. Check out this book to learn how to achieve optimal automation.

**Selenium Design Patterns and Best Practices**
Learn how to keep up with the changing pace of your web application by creating an agile test suite, improve your programming skills with a step-by-step continuous improvement tutorial, and more.

**The Way of the Web Tester**
Learn how to up your skills in automated testing so that you're freed up to do other, more important things while letting the computer quickly run thousands of repetitive tasks.

## 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.

**Performance** DZONE.COM/PERFORMANCE
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.

## refcardz

**Test Design Automation**
Download this new Refcard to get started with test design automation, explore the many benefits, and find real-world use cases.

**Continuous Testing**
These concepts are certainly connected. Let's push the misconceptions aside and learn exactly what Continuous Testing is, the 10 key elements of Continuous Testing methodology, and the benefits of utilizing this concept.

**Lean Software Development**
This Refcard fills the gap by covering a step-by-step approach to carrying out a lean software development process, including best practices to streamline your technology value stream.

## courses

**Selenium Bootcamp**
Learn how to start off on the right foot with Selenium in this simple, actionable, free one-week Selenium tutorial.

**Automated Testing Courses on Udemy**
There are thousands of courses related to automated testing on Udemy. Peruse them here.

**Automated Testing Engineer Master's Program**
Get on the course to becoming an expert automation test engineer with a focus on software development and quality assurance.

# The **Gartner Magic Quadrant** for **Software Test Automation** is here!

## Download your copy now!

tricentis.com/GartnerMQ


## Software testing reinvented for DevOps

LEXMARK Allianz BOSCH STARBUCKS SIEMENS Deutsche Bank

TRICENTIS

# Test Automation is Essential... But it's not Sufficient

Reinventing testing is an untapped opportunity to accelerate the delivery of innovative software. What's needed to transform testing? Test automation is essential, but it's not sufficient. We need to:

1. Break through the barriers to test automation. The industry average test automation rate is hovering around a dismal 20% (and even lower for large enterprises). This must improve significantly in order to provide the fast feedback required for quality@speed. To get there, we not only need faster and simpler ways to create automated tests; we also need a sustainable way to maintain them—escaping the "maintenance trap" created by flaky, brittle tests.
2. Expand our definition of test automation. Test automation is more than functional testing at the web UI level. Today, a single end-to-end transaction may involve technologies ranging from SAP and other packaged applications, to APIs, to web and mobile UIs. Test automation must embrace all of these technologies—and it must also extend beyond functional testing to include practices such as load and performance testing.
3. Recognize that Continuous Testing is more than test automation. Continuous Testing also involves practices such as aligning testing with your business risk, applying service virtualization and stateful test data management to stabilize test automation for Continuous Integration, and performing exploratory testing to expose "big block" issues early in each iteration. Furthermore, it's not simply a matter of more tools, or different tools. It requires a deeper transformation across people and processes as well as technologies.

**WRITTEN BY WOLFGANG PLATZ**
FOUNDER AND CHIEF STRATEGY OFFICER, TRICENTIS



**PARTNER SPOTLIGHT**

# Tricentis Tosca

*With the industry's No. 1 Continuous Testing platform, Tricentis is recognized for reinventing software testing for DevOps.*

TRICENTIS

**CATEGORY**
Continuous Testing Platform

**RELEASE SCHEDULE**
Quarterly Release

**OPEN SOURCE?**
No

**CASE STUDY**

With a risk-based approach as a key requirement, Dolby Labs selected Tricentis Tosca's end-to-end Continuous Testing platform to reduce test cases while providing end-to-end risk coverage within continuously increasing release cadences. Dolby also needed a test automation solution that supported their large footprint in SAP packaged applications.

Since adopting Tricentis Tosca's innovative, model-based approach to test automation, Dolby quickly developed a standardized approach and a robust maintenance process for all of its testing deliverables. Along with a consistent drop in the number of incidents and support tickets during their initial pilot, the TCoE team at Dolby proved ROI with significant savings in hours per testing cycle.

**STRENGTHS**

- Reduce regression testing from weeks to minutes
- Create dynamic test cases that synchronize with any application under test
- Maximize reuse and maintainability with model-based test automation
- Gain clear insight into the risk of your release candidates

**NOTABLE USERS**

- Allianz
- Experian
- HSBC
- Merck
- Vantiv (now WorldPay)

**WEBSITE** tricentis.com **TWITTER** @tricentis **BLOG** tricentis.com/blog

# End-to-End Tests: The Pinnacle in Test Automation

## QUICK VIEW

**01.** End-to-end business process tests ensure not just that your software works but also that the business works.

**02.** End-to-end tests are not just a series of functional and unit tests strung together.

**03.** Automating end-to-end tests comes with unique challenges that make them especially difficult to maintain and scale.

**BY CHRIS KRAUS**
SR. DIRECTOR PRODUCT MANAGEMENT, WORKSOFT



End-to-end testing can mean different things to different people. Regardless of the definition, end-to-end tests are more complex, harder to automate, and many times carry more risks than functional and unit tests. Functional and unit tests focus on whether or not a specific feature of a single application works. End-to-end tests ensure a request or transaction can flow across an entire system of technologies and applications. Failure of an order-to-cash process or payroll process not only represents a software failure but also a business failure. As companies move to adopt Agile across all of their applications and quickly roll out changes across all of their systems, automating end-to-end tests becomes critical for reducing risks and accelerating change.

### WHAT IS AN END-TO-END TEST?

Depending on who you ask, end-to-end testing can mean different things. In the custom app development space, end-to-end testing is often used to describe an integration test. In this case, a "vertical" test that may start at the presentation/services level is used to test the back-end responses from the System Under Test (SUT). The test runs across the various layers used by the SUT.

Business process testing is used to test enterprise business processes that include multiple steps and, many times, across multiple applications. In this case, the test is usually driven at the UI layer and is used to validate that the end-to-end business process works across systems as a "horizontal" test. Unlike vertical tests that measure performance up and down the layers of the technology stack, horizontal end-to-end tests are used to ensure that mission-critical business processes work. Tests are run at the UI layer and measure whether or not the test proceeds to the next step in the process.

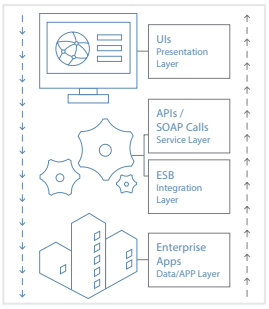



*Figure 1: "Vertical" End-to-End Integration Test*

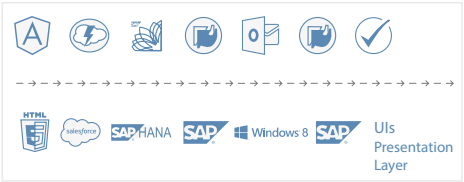



*Figure 2: "Horizontal" End-to-End Business Process Test*

This is a similar debate to test-driven development (TDD) vs. behavior-driven development (BDD). TDD execution is writing a failing test for one very specific bit of functionality, generating a unit or "vertical" test. An end-to-end test or "horizontal" test is similar to BDD when looking at multiple features working together to complete a workflow or epic in Agile terms.

It is also important to note that end-to-end tests are not just several unit and functional tests strung together. The scope of what is being tested and the goal of the tests are considerably different between functional tests and end-to-end tests. Key differences between functional and end-to-end tests include:

- Functional Tests:
  - Testing is limited to a single piece of code or application.
  - Software is tested to ensure it meets acceptance criteria.
  - Test the way a single user engages with the application.
  - Validate the result of each test for inputs and outputs.

- End-to-End Tests:
  - Testing crosses multiple applications and user groups.
  - Testing ensures a process continues to work after a change is made.
  - Test the way multiple users work across applications.
  - Validate that each step in the process is completed.

## WHY ARE END-TO-END TESTS SO HARD TO SCALE?

When we talk about end-to-end business process testing, we're talking about testing just like a user, which means there are going to be hundreds — if not thousands — of tests that have to be managed across multiple applications and UIs. For this reason, just building the automation for an end-to-end test is a challenge. In fact, according to the 2017-18 World Quality Report, only 16 percent of end-to-end business scenarios are executed with test tools.

Included below are some key reasons end-to-end tests are hard to automate, including a couple of examples of customers who have had these challenges and how they have overcome them.

## MAINTENANCE

Most agree that implementing a test automation solution is good for the business. But to be successful with end-to-end test automation, it needs to take less time to create, maintain, and run tests than it does doing things manually. Unfortunately, many organizations that have tried automation solutions for end-to-end testing have had to abandon their efforts because it either takes too long to create the tests or too much time to maintain them.

A recent case study published in SAP Insider regarding Honda R&D Americas is a perfect example of challenges clients have had with automating end-to-end business process tests. They purchased a test automation solution that was script-based and found it took more time to create and maintain the automation than it did to continue to test things manually. They quickly abandoned it, but a move to SAP HANA forced Honda to revisit the idea of test automation. With SAP HANA, they were looking at a three-month patch cycle and there was no way to continue doing things manually. This time, they focused on ease-of-use and ease-of-maintenance in their selection process. With the new solution (Worksoft Certify), they were able to meet their automation needs and scale testing to keep pace with updates.

## ACCESS

Unlike custom app virtual devtest environments that can be spun up in minutes, tests for enterprise applications like SAP need to be run many times in pre-production where the systems are shared and testers can't just spin up a personal instance. Another big challenge is automation that acts like a user needs to act like a real user by logging into the desktop and running applications. This means installing local agents and logging into remote machines to simulate real users. To make things more complicated, you also need to figure out a way to keep the machine awake for the life of the test and prevent things like Windows updates from interrupting tests.

## ORCHESTRATION

You must be able to orchestrate the workflow of an end-to-end test. The tests need to run in the same order as the real system would process the request in production to validate whether or not it will actually run in production. The tests often need to be scheduled in sequence with complex dependencies. It's difficult to manage distributed and diverse testing resources. The ability to just say, "Go run these thousand tests and let me know when they're done," without the right solution will not validate multiple features are working together. The testing window, especially in global situations, is often strictly limited or closely scrutinized.

## SCALE

When rolling out a change to a mission-critical system, businesses don't just want one workflow tested. They want them all tested each time a change is made. As changes happen more frequently, more and more tests have to be documented, created, and run in the same testing window. Automation becomes a critical part of the process. Test automation needs to be generated faster and tests need to be run in parallel across more resources so that they can be completed within a given testing window.

Cardinal Health recently undertook an automation effort in an effort to become more Agile and to better support changes to SAP. They quickly realized that in order to be successful, automaton needed to become everyone's job. They needed to change the way they work and enable business analysts and testers to better collaborate together along with being able to orchestrate and run tests at scale. By implementing an automation solution that enabled them to create and run automation faster, they were able to spend more time planning, detect more defects faster, and better meet the needs of the business.

## CONCLUSION

End-to-end business process tests are complex by nature. They cover a wide variety of systems and applications and can include hundreds to thousands of steps that need to be orchestrated and driven via a UI that requires a real machine. But IT must evolve test automation to include end-to-end testing to meet the needs of their business. Companies must select tools that are easy to use, easy to maintain, and easy to scale in order to ensure their long-term success with test automation.

**CHRIS KRAUS** currently leads the Product Management team at Worksoft and brings with him over 25+ years of software experience. His vision in the testing space has helped transform the way testing tools are architected and developed. As a product manager, he was also an early adopter of Agile, working with multiple agile teams moving from cowboy to Scrum-based sprinting.

# 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

**RELEASE SCHEDULE**
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 USERS**

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

**WEBSITE** ca.com/us/products/ca-agile-requirements-designer.html **TWITTER** @CAinc **BLOG** ca.com/en/blog-highlight

# Executive Insights on Automated Testing

**BY TOM SMITH**
RESEARCH ANALYST AT **DZONE**

## QUICK VIEW

**01.** The keys to automated testing are to save time and improve quality while realizing the goals and objectives of the organization.

**02.** Increased adoption of automated testing is attributed to the maturation of Agile methodologies and adoption of the DevOps culture.

**03.** The ROI of automated testing is three-fold: time saved, fewer defects, and greater customer satisfaction/retention.

- **Gil Sever**, CEO, **Applitools**
- **James Lamberti**, Chief Marketing Officer, **Applitools**
- **Shailesh Rao**, COO, **BrowserStack**
- **Kalpesh Doshi**, Senior Product Manager, **BrowserStack**
- **Aruna Ravichandran**, V.P. DevOps Products & Solutions Marketing, **CA Technologies**
- **Pete Chestna**, Director of Developer Engagement, **CA Veracode**
- **Julian Dunn**, Director of Product Marketing, **Chef**
- **Isa Vilacides**, Quality Engineering Manager, **CloudBees**
- **Antony Edwards**, CTO, **Eggplant**
- **Anders Wallgren**, CTO, **Electric Cloud**
- **Kevin Fealey**, Senior Manager Application Security, **EY Cybersecurity**
- **Hameetha Ahamed**, Quality Assurance Manager, **FileCloud**
- **Amar Kanagaraj**, CMO, **FileCloud**
- **Charles Kendrick**, CTO, **Isomorphic**
- **Adam Zimman**, VP Product, **LaunchDarkly**
- **Jon Dahl**, CEO and Co-founder, **Mux**
- **Matt Ward**, Senior Engineer, **Mux**
- **Tom Joyce**, CEO, **Pensa**
- **Roi Carmel**, Chief Marketing & Corporate Strategy Officer, **Perfecto Mobile**
- **Amit Bareket**, CEO and Co-founder, **Perimeter 81**
- **Jeff Keyes**, Director of Product Marketing, **Plutora**
- **Bob Davis**, Chief Marketing Officer, **Plutora**
- **Christoph Preschern**, Managing Director, **Ranorex**
- **Derek Choy**, CIO, **Rainforest QA**
- **Lubos Parobek**, Vice President of Product, **Sauce Labs**
- **Walter O'Brien**, CEO and Founder, **Scorpion Computer Services**
- **Dr. Scott Clark**, CEO and Co-founder, **SigOpt**
- **Prashant Mohan**, Product Manager, **SmartBear**
- **Sarah Lahav**, CEO, **SysAid Technologies**
- **Wayne Ariola**, CMO, **Tricentis**
- **Eric Sheridan**, Chief Scientist, **WhiteHat Security**
- **Roman Shaposhnik**, Co-founder V.P. Product & Strategy, **Zededa**

**1.** The keys to automated testing are to **save time and improve quality while realizing the goals and objectives of the organization.** Value drivers are quality and velocity, and they need to be prioritized first. 70% of the time, clients are looking for both and it can take multiple meetings to decide what to prioritize. Automation helps customers with a quality assurance process that runs in minutes and hours versus days and weeks, all while removing errors.

The organization must understand its goals. Planning is important. Many organizations are facing the traditional issue of "we're too busy to improve, we don't have time to think about test automation, but we know we are going to fail if we don't have it in place soon." They need to understand the end goal of doing this kind of testing.

There are four key factors of success: 1) level of automation

(must be 90%+ for CI/CD to succeed); 2) maximize coverage of all flows, all datasets, quality of app and UX; 3) creating an efficient feedback loop with data; and 4) adopting an integrated approach and platform for testing across all devices. Organizations need to be able to do end-to-end DevOps testing throughout the SDLC. Testing needs to shift left at conceptualization.

**2.** The most significant change to automated testing in the past year has been **increased adoption**. This is attributed to the maturation of Agile methodologies and adoption of the DevOps culture. Agile-driven teams are no longer separating automation from development. Organizations are realizing as complexity increases, automation is the only way to keep up with all of the changes. Also, the continued evolution of tools and frameworks are easier to work with. Enablement tools integrate automatically, so tests can be built quickly.

**3.** There are **more than a dozen benefits to automated testing** being seen in six key industries. Automated testing is invaluable for: 1) saving time by running tests automatically 24/7; 2) reporting with daily insights and accurate feedback; 3) consistency and accuracy; 4) saving money; 5) reducing resources (i.e. manual testers); and 6) achieving full coverage. Manual testing can achieve 6% code coverage, while well-written automated tests can achieve 100% coverage. Automated testing is helping organizations achieve continuous integration (CI)/ continuous delivery (CD) and is helping legacy enterprises make the digital transformation with microservices.

The industries most frequently mentioned as benefitting from automated testing are: 1) automotive, especially automated cars; 2) healthcare – pharmaceuticals, medical devices, and patient monitoring; 3) telecommunications; 4) financial services – brokerage and algorithmic trading; 5) ecommerce; and 6) the Federal Government.

**4.** The ROI of automated testing is three-fold: **time saved, fewer defects, and greater customer satisfaction and retention.** Organizations can see 20 to 40x cycle time improvements by spreading the work across different machines. Going from weeks to days and days to minutes can yield 20x savings. Conservatively speaking, if you find a defect, it takes 10 times longer to fix in production than earlier in the SDLC. Catching errors earlier and more accurately saves a million dollars per year in developers not having to look for errors. For us, automated testing has had a direct correlation with customer satisfaction. The product is simply running better, and the customers are happier.

**5.** The most common issues affecting the implementation of automated testing is the **corporate culture used to doing manual testing.** Where manual processes have been used, people need to be retrained and turned into programmers — but management doesn't want to ruffle feathers. Companies need to shift left and developers need to learn to write tests. Old style testers are not adapting to, let alone embracing, automated testing and AI. We need one comprehensive, automated, visible delivery process to share the feeback from the different tools.

**6.** The future of automated testing is the use of **AI/ML to improve the process** on several fronts. Testing will be designed for AI/ML to build predictable models and patterns. Testing will be a natural as writing code which will be done by machines. AI/ML will be part of the solution as teams generate more data which will make the models smarter. With AI/ML, testing will be faster, more thorough, and will result in self-healing tests and self-remediating code. You will be able to use AI/ML to run every test imaginable in the least amount of time to ensure your code is always vulnerability and defect-free.

ML will also improve automated security testing, as security teams are able to leverage historical vulnerability data to train ML models to automate the vulnerability verification process, thereby providing developers accurate vulnerability data in near real-time.

**7.** The three primary skills a developer will need to ensure their code and applications perform well with automated testing are: **1) test scripting skills; 2) understanding the use case of the application; and, 3) move left**, beginning testing earlier in the SDLC.

Hone test scripting skills. Write small, simple tests. Recognize the different types of tests you will need to run at different points in the SDLC. Have the ability to write your own unit and regression tests. Know how automated tests are going to be written. Learn to write a model type of code from mentors and previous products.

Think about the use cases first and what the purpose of the code is. Understand users, domains, who they are, and what problem they're trying to address. Have an overarching view of the application, what it's doing, and how that impacts APIs and services.

Get rid of simple, recurring problems by architecting the system to be tested from the beginning. Leverage the testing methodology from the planning phase, and build richer code earlier in the SDLC.

**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.

# 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 |
|---|---|---|---|---|
| **Apache Software Foundation** | JMeter | Java functional & performance testing | Open source | jmeter.apache.org |
| **Apica** | Apica Load Test | Load testing | 30 days | apicasystems.com |
| **Applitools** | Applitools | AI-powered automated visual testing | Available by request | applitools.com |
| **Appvance** | Appvance IQ | AI-driven, unified test automation system | Demo available by request | appvance.com/appvance-iq |
| **Atlassian** | Bamboo | Continuous integration, continuous delivery, & automated testing | 30 days | atlassian.com/software/bamboo |
| **BrowserStack** | BrowserStack | Automated mobile & web testing | Available by request | browserstack.com |
| **Buildbot** | Buildbot | Continuous integration | Open source | buildbot.net |
| **CA Technologies** | BlazeMeter | Performance & load testing for JMeter | Free tier available | blazemeter.com |
| **CasperJS** | CasperJS | JavaScript navigation testing | Open source | casperjs.org |
| **CircleCI** | CircleCI | Continuous integration & continuous delivery | Free tier available | circleci.com |
| **Cloudbees** | Jenkins | Continuous integration & continuous delivery | Open source | jenkins.io |
| **CollabNet** | VersionOne | Acceptance & regression testing | Available by request | collab.net |
| **Cucumber** | Cucumber | Automated rails testing | Open source | cucumber.io |

| COMPANY | PRODUCT | PRODUCT TYPE | FREE TRIAL | WEBSITE |
|---|---|---|---|---|
| **DevExpress** | TestCafe | Functional web testing | 30 days | testcafe.devexpress.com |
| **Eggplant** | Eggplant | Digital performance management & digital automation intelligence | Available by request | eggplant.io |
| **eureQa** | eureQa | Load & performance testing | Demo available by request | sayeureqa.com |
| **FitNesse** | FitNesse | Acceptance testing framework | Open source | fitnesse.org |
| **Functionize** | Functionize | Automated functional, load, security, & performance testing | Demo available by request | functionize.com |
| **Gridlastic** | Gridlastic | Selenium-as-a-Service | Free tier available | gridlastic.com |
| **Hiptest** | Hiptest | Automated web testing | Free tier available | hiptest.net |
| **IBM** | IBM Rational Functional Tester | Automated functional & regression testing | Available by request | ibm.com/us-en/marketplace/rational-functional-tester |
| **IBM** | IBM Rational Performance Tester | Performance & 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 & mobile testing framework | Open source | appium.io |
| **JUnit** | JUnit 5 | Unit testing framework | Open source | junit.org/junit5 |
| **Loadster** | Loadster | Web app load testing | Free tier available | loadsterperformance.com |
| **mabl** | mabl | ML-driven automation testing | Available by request | mabl.com/features |
| **Micro Focus** | LoadRunner | Load testing | Available by request | software.microfocus.com/en-us/products/loadrunner-load-testing/overview |
| **Micro Focus** | Quality Center Enterprise (QC) | Test planning & management platform | Available by request | software.microfocus.com/en-us/products/quality-center-quality-management/overview |

| COMPANY | PRODUCT | PRODUCT TYPE | FREE TRIAL | WEBSITE |
|---|---|---|---|---|
| **Micro Focus** | SilkTest | Automated mobile & web testing | 45 days | microfocus.com/products/silk-portfolio/silk-test |
| **Micro Focus** | Unified Functional Testing (UFT) | Automated functional testing | 60 days | software.microfocus.com/en-us/products/unified-functional-automated-testing/overview |
| **Microsoft** | Visual Studio Test Professional | Test case management, exploratory, & browser testing | N/A | visualstudio.com/vs/test-professional |
| **Mobile Labs** | Mobile Labs | Automated mobile & 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 | 14 days | panaya.com |
| **Parasoft** | Parasoft C/C++ Test | C and C++ development testing | Available By Request | parasoft.com/products/ctest |
| **Parasoft** | Parasoft Continuous Testing Platform | Browser-based continuous testing tool | Available By Request | parasoft.com/products/continuous-testing-platform |
| **Parasoft** | Parasoft dotTEST | Test automation platform | Available by request | parasoft.com/product/dottest |
| **Parasoft** | Parasoft JTest | Java static analysis and testing | Available By Request | parasoft.com/products/jtest |
| **Parasoft** | Parasoft SOAtest | API testing platform | Available By Request | parasoft.com/products/soatest |
| **Parasoft** | Parasoft Virtualize | Service virtualization and virtual test environments | Trial Available By Request | parasoft.com/products/virtualize |
| **Perfecto Mobile** | CQ Lab | Automated web & mobile testing | Available by request | perfectomobile.com/solutions/perfecto-test-automation |
| **PHPUnit** | PHPUnit | PHP unit testing framework | Open source | phpunit.de |
| **Plutora** | Plutora | Release & test management platform | Demo available by request | plutora.com |
| **Practitest** | Practitest | QA & test management | Available by request | practitest.com |

| COMPANY | PRODUCT | PRODUCT TYPE | FREE TRIAL | WEBSITE |
|---|---|---|---|---|
| **Progress Software** | Telerik Test Studio | Automated GUI, performance, load, & 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 QA | Automated mobile & web testing | Demo available by request | rainforestqa.com |
| **Ranorex** | Ranorex | Automated GUI testing | 30 days | ranorex.com |
| **Robotium** | Robotium | Android UI testing | Open source | github.com/RobotiumTech/robotium |
| **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 & mobile testing framework | 14 days | saucelabs.com |
| **Screenster** | Screenster | Visual UI testing | Free tier available | screenster.io |
| **Sealights** | Sealights | Continuous testing | Demo 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 | Available by request | soapui.org/professional/loadui-pro.html |
| **SmartBear Software** | SoapUI | Automated web & API testing | Open-source version available | soapui.org |
| **SmartBear Software** | TestComplete | Automated UI testing | Demo available by request | smartbear.com/product/testcomplete/overview |

| COMPANY | PRODUCT | PRODUCT TYPE | FREE TRIAL | WEBSITE |
|---|---|---|---|---|
| **SmartBear Software** | Cross Browser Testing | Automated mobile & web testing | 7 days | crossbrowsertesting.com |
| **Solano Labs** | Solano Labs | Continuous integration & deployment | 14 days | solanolabs.com |
| **Test Anywhere** | Test Anywhere | Frontend web testing | Free tier available | testanywhere.co |
| **Testcraft** | Testcraft | Regression and continuous testing | Available by request | testcraft.io |
| **Testim** | Automate | Automated regression testing | Available by request | testim.io |
| **TestingBot** | TestingBot | Automated web testing | 14 days | testingbot.com |
| **Testlio** | Testlio | Automated mobile & web testing | Demo available by request | testlio.com |
| **TestNG** | TestNG | Unit testing framework | Open source | testng.org |
| **ThoughtWorks** | GoCD | Continuous integration & continuous delivery | Open-source version available | gocd.org |
| **Travis CI** | Travis CI | Continuous integration | Free solution | travis-ci.org |
| **Tricentis** | Tosca Testsuite | Continuous testing platform | 14 days | tricentis.com/software-testing-tools |
| **Turnkey Solutions** | cFactory | Automated mobile & web testing | Demo available by request | turnkeysolutions.com |
| **Ubertesters** | Ubertesters | Automated mobile testing | 14 days | ubertesters.com |
| **Watir** | Watir | Automated web testing | Open source | watir.com |
| **Windmill** | Windmill Testing Framework | Automated web testing | Open source | github.com/windmill |
| **Worksoft** | Worksoft | Automated mobile & web testing | Demo available by request | worksoft.com |
| **xUnit** | xUnit.net | Unit testing framework | Open source | xunit.github.io |
| **Zephyr** | Zephyr | Real-time test management | 7 days | getzephyr.com |

# GLOSSARY

**AGILE:** A group of software development methods that involve fairly short development cycles with flexible requirements that evolve as the software is built; involves self-organizing, cross-functional teams.

**AMAZON ELASTIC COMPUTE CLOUD (AMAZON EC2):** An Infrastructure-as-a-Service that provides secure, resizable compute capacity to run applications.

**AUTONOMOUS TEST CREATION:** A test creation process that is possible utilizing AI technologies via natural language processing and advanced modeling.

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

**CALLBACK URL:** A URL invoked by the API method that you're calling after you're done.

**CLOUD-SCALE EXECUTION:** A way of executing complex tests in a matter of minutes so that they become more resilient with every run; resource limitations should be abstracted from the client and managed by the cloud provider.

**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 DELIVERY:** A software engineering approach in which continuous integration, automated testing, and automated deployment capabilities allow software to be developed and deployed rapidly, reliably, and repeatedly with minimal human intervention.

**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:** 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.

**FUNCTION SIGNATURE:** The parameters that the end function takes that determine whether it will return an expected outcome.

**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.

**LIGHTHTTPD:** An open-source web server optimized for speed-critical environments while remaining standards-compliant, secure, and flexible.

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

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

**NGROK:** A service that allows you to create a secure URL connection to your localhost server and that can be used until you get your own certificate.

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

**PARAMETERS:** A particular kind of variable used in code to refer to one of the input pieces of data information input.

**PROGRESSIVE WEB APPS:** Web applications that appear to users like native mobile applications or traditional applications that are actually regular web pages.

**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.

**SELF-HEALING MAINTENANCE:** Selector-based maintenance should be eliminated if elements are correctly identified by using hundreds of data points, and then rated and ranked.

**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.

**TEST-DRIVEN DEVELOPMENT (TDD):** A process in software development geared toward reducing errors and improving flexibility when designing applications.

**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.

**WEBHOOKS:** An API concept that delivers data to other applications in real-time.

INTRODUCING THE

# Open Source Zone

**Start Contributing to OSS Communities and Discover Practical Use Cases for Open-Source Software**

Whether you are transitioning from a closed to an open community or building an OSS project from the ground up, this Zone will help you solve real-world problems with open-source software.

Learn how to make your first OSS contribution, discover best practices for maintaining and securing your community, and explore the nitty-gritty licensing and legal aspects of OSS projects.

COMMITTERS & MAINTAINERS

COMMUNITY GUIDELINES

LICENSES & GOVERNANCE

TECHNICAL DEBT

**Visit the Zone**

BROUGHT TO YOU IN PARTNERSHIP WITH Flexera