



**DZone**  
A DEVADA MEDIA PROPERTY

---

**TREND REPORT**

# The Rise of Continuous Testing

Identifying Capabilities and Challenges in the SDLC

BROUGHT TO YOU IN PARTNERSHIP WITH

**WORKSOFT®**

# Table of Contents

**3** Highlights and Introduction  
BY JACOB DOIRON | [Software Engineer at DZone](#)

**4** Key Research Findings  
BY MATT LEGER | [Research Analyst](#)

**11** Leaders in Software Testing  
Alan Page, Renowned Author and Industry Expert, Discusses  
Modern Testing Trends  
BY LINDSAY SMITH | [Publications Content Manager](#)

**14** To Automate or Not to Automate  
The Framework to Continuous Testing Success  
BY ALIREZA CHEGINI | [Site Reliability Engineer](#)

**20** Strategic Continuous Testing  
Integrating Test Automation into the SDLC  
BY PRIYANKAA ARUNACHALAM | [Data Analyst](#)

**24** Functional Testing vs. Robotic Process Automation  
Same Car, Different Paint?  
BY CHRIS KRAUS | [VP Product Management](#)

**27** Diving Deeper Into Continuous Testing

To sponsor a Trend Report:

Call: (919) 678-0300

Email: [sales@devada.com](mailto:sales@devada.com)

# Highlights and Introduction

By **Jacob Doiron**, Software Engineer at DZone

*Continuous Testing*, a term that started to increase in popularity in mid-2019, has made its way into many of today's CI/CD processes used in the SDLC, but what exactly does the phrase mean?

Continuous testing (CT) refers to the idea of automated testing of software as it passes through various stages in the software delivery pipeline. It is a technique employed to provide insight into risks associated with new and upcoming software release candidates before they are released to production. This ideology stands in stark contrast to the old thought process of performing most of the application testing after deployment.

CT often involves many forms of testing different layers in the application — everything from unit, integration, API, performance, systems, security, and functional (UI/UX) testing. Integrating automated testing into the lifecycle of software development helps assess risk to the business, delivers feedback used in every stage of the delivery pipeline, and reduces the length of the feedback loop between development and deployment of software.

A shorter time identifying risky and detrimental defects in software helps ensure a quality product and contributes to higher client satisfaction with the deliverable.

Continuous testing goes hand in hand with CI/CD processes to improve the time between product enhancements and feature updates. CT exposes development defects much sooner in the SDLC, which results in a faster time to patch issues before they make their way into production environments. This efficiency factor leads to more frequent and more manageable deployments, with companies releasing new build deployments on a much shorter cadence — anywhere from once every two weeks to as many as multiple times per day.

Other benefits stemming from continuous testing include assessing the end-user experience, reducing the number of false positives, improved code quality, and emphasizing mitigations to business risk. Continuous testing is a relatively new concept and one that is largely a cultural shift for all parties involved in the development process. CT can be difficult to implement correctly without the right tools, knowledge, and sufficient resources. Nevertheless, continuous testing is an important paradigm shift and will continue to prove its worth as it evolves through AI and machine learning to further improve testing processes and feedback handoff at each stage in the delivery pipeline.

To understand more about these shifts, DZone conducted original research and a survey among members of the DZone Community about their insights into and experience with continuous testing. The following conclusions from our research explore CT's impact on culture, the benefits and challenges of adoption, and how AI and machine learning are predicted to transform CT capabilities over the next six to 12 months. You will also find new articles from contributors, whom shared their experiential knowledge about continuous testing frameworks and test automation throughout the SDLC.

Read on to learn more! 

# Key Research Findings

By **Matthew Leger**, Research Analyst

Businesses that can innovate rapidly, fail quickly, and improve constantly in digital business operations are primed for success in the 21st century economy. With seemingly every business transforming into a “tech company”, organizations can no longer pend months and millions of dollars making large batch updates to their digital products and solutions. Only a few years ago, this method of making large-scale changes to software often resulted in massive amounts of unplanned work, and even more wasted time and resources.

In recent years, we have seen a shift to making software updates in smaller batches more quickly and efficiently. The idea is to minimize disruptions to the business and customer experience by making small changes in a continuous fashion. Beyond rapidly innovating, CI/CD emerged to reduce the risk of imposing massive disruptions to business operations, which often occurred with large batch updates due to the inability of code to integrate with existing software.

However, this “move fast, fail quick, adapt rapidly” approach is not just about deploying code into production faster. If this code goes unchecked — or untested — prior to deployment, we risk breaking apps faster, causing more problems. That is why “Continuous Testing” has emerged as a leading tactic in this modern, fast-moving software development environment. [According to IBM](#), Continuous Testing (CT) is defined as the “the process of incorporating automated feedback at different stages of the software development lifecycle (SDLC) in support of better speed and efficiency when managing deployments.”

To better understand the current state of CT, we surveyed 340 developers on DZone about their experience with this methodology. We wanted to better understand how teams are going about building CT pipelines, as well as reported benefits and challenges. Lastly, we will close by providing a glimpse into what the future holds for CT.

## KEY TAKEAWAYS

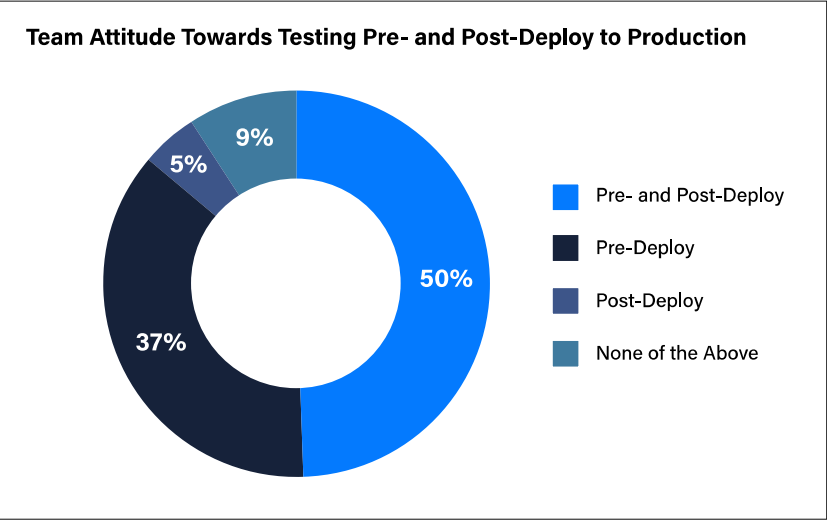
- ▶ Two-thirds of organizations are now utilizing CT, and just under a quarter are considering implementing it in the future.
- ▶ The majority of organizations have automation built into their performance, functional, unit, and integration tests.
- ▶ Of those pursuing CT, only one in five said their teams have achieved this form of testing, but more than half proclaimed they are well on their way.
- ▶ A majority of teams that have achieved CT claimed it has helped improve product quality and mitigate risk, but the overall benefits to the organization appear to be overblown.
- ▶ The leading challenges preventing teams from achieving CT are skills/talent shortages at their organizations, and figuring out which tests should be automated or manual.

The State of Continuous Testing

Before the emergence of CT, it was common practice to only test software after it had been deployed into production. With digital transformation requiring a constant rate of change within organizations across industries today, we wanted to get a better sense for how the attitude towards testing has changed.

We asked the DZone audience to share at which stage of the process (either pre- or post-deployment) their team or organization cared most about testing software. As it turns out, it is no longer wise to wait until after deployment (see Figure 1). More than a third of respondents now said their primary focus for testing is prior to production, while half of developers now say their teams place equal priority on testing pre- and post-deployment.

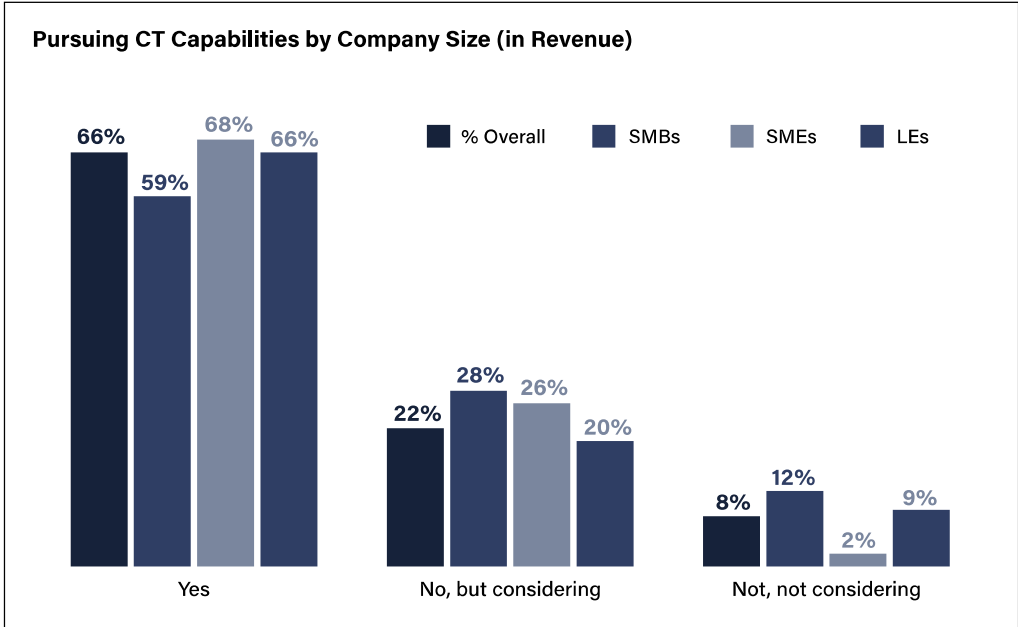
Figure 1



A SHIFT IN TESTING CULTURE

The general attitude towards testing has shifted as organizations seek feedback earlier. So we asked how many DZone members work for companies that use CT methodology. A majority (66%) use it in some capacity, while an additional 22% consider using it in the near future. Less than 10% reported their organizations weren't interested. These findings are consistent across company sizes, as technology enabling CT has been widely available and affordable (see Figure 2).

Figure 2



**TO AUTOMATE OR NOT TO AUTOMATE, THAT IS THE QUESTION!**

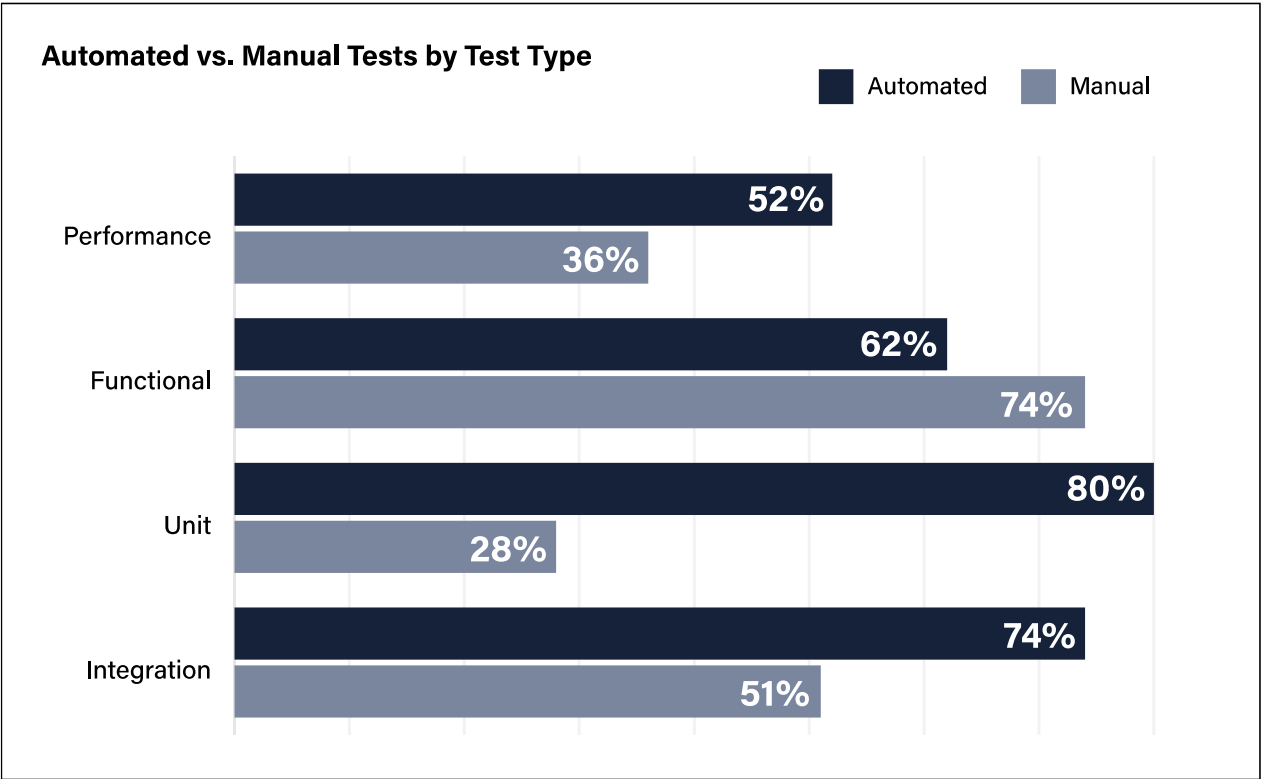
Building CT capabilities requires automation to be built into testing processes throughout the development lifecycle; however, it is not just about automating everything. In fact, [over-automating your tests](#) can limit your team’s ability to conduct exploratory testing. It can also impact teams’ ability to make changes to data or systems that don’t fit the traditional sequence of steps the automation was designed to handle.

Instead, CT is about figuring out what needs to and can be automated, while also running manual tests when necessary. The most important thing is creating a sequence of tests that can be conducted seamlessly and continually, without inhibiting the development process.

We asked respondents what tests they are automating, and which they are conducting manually as part of their CT pipeline. It turns out that more than half of organizations are automating at least some of their performance, functional, unit, and integration tests (see Figure 3).

Functional tests are the only test type that a larger percentage of respondents said that they are conducting manually as opposed to with automation. Unit tests, unsurprisingly, are quickly approaching complete automation, with less than 30% of respondents still running these tests by hand.

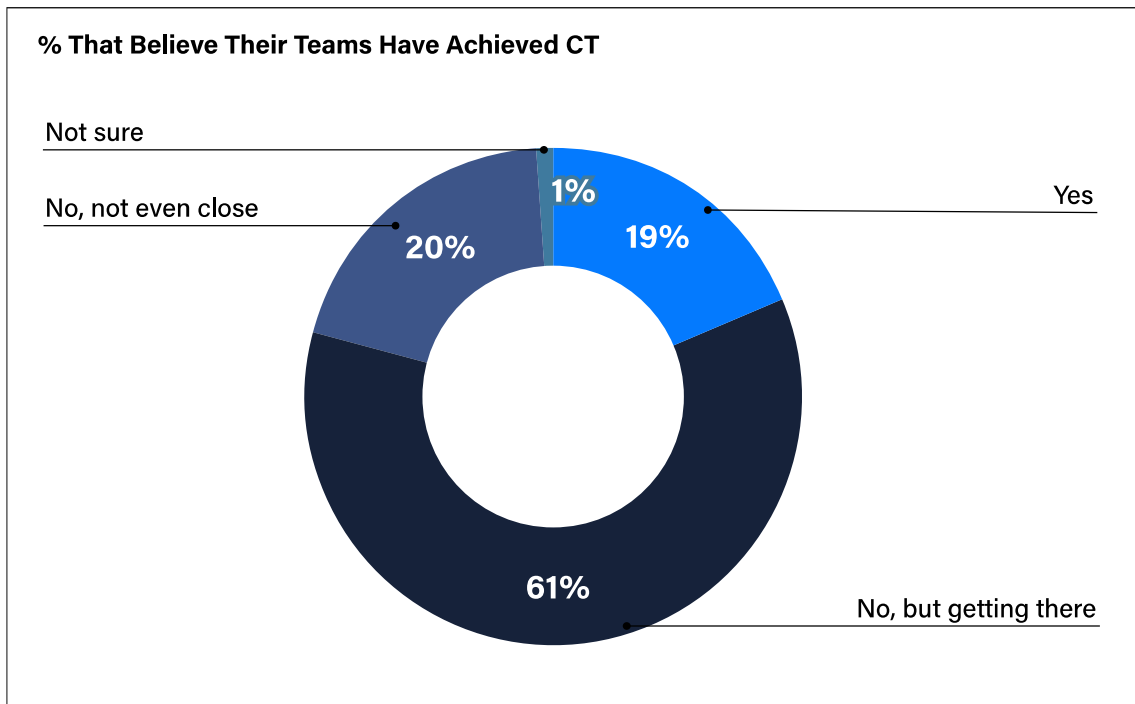
**Figure 3**



**CLOSING THE ACHIEVEMENT GAP IN CONTINUOUS TESTING**

With the surging popularity of CT, it leaves one wondering how many organizations have actually achieved this standard of testing, and how many still have work to do. Of those who are pursuing CT capabilities, only one in five said that their organization has fully achieved continuous testing, but the same amount also claimed that they aren’t anywhere close. However, just over 60% said they are well on their way to having these capabilities fully matured (see Figure 4).

**Figure 4**



Part of the challenge in achieving CT, as we will highlight later on, is figuring out what tests can and should be automated, and how to prioritize the order in which to automate them. While teams across organizations, industries, and company sizes will have differing reasons for how they go about building their pipelines, it may be helpful to see the order of different successfully automated tests compared to those that are not quite there yet.

Table 1 shows the order in which survey respondents said their teams prioritized test automation by test type. It then compares overall responses to teams that said they have achieved CT, those that said they are getting there, and those that said they aren't even close. According to the survey, teams that have successfully achieved CT have set themselves apart by taking a different approach than their peers (with the exception of automating unit tests first). The key difference for successful teams is that they tackled automating unit and functional tests first, before moving onto performance and integration tests.

**Table 1**

		Achieved CT		
Order of Test Automation by Test Type	Overall	Yes	No, getting there	No, not even close
Unit Tests	1	1	1	1
Integration Tests	2	4	2	3
Functional Tests	3	2	3	2
Performance Tests	4	3	4	4

### **Organizational and Technological Benefits of CT**

There are countless vendors in the market that promote the supposed benefits of CT and how their product helps you get there. To get a sense for what benefits organizations are actually experiencing, we asked survey respondents to check off what benefits they believe CT will bring to their teams from an organizational and technological perspective. We then compared answers from those who have achieved CT with those who have not yet gotten there.

From an organizational perspective, there seems to be a disparity between the benefits respondents believe CT brings to their organization overall, and the extent to which those benefits have been realized by teams that have achieved these capabilities (see Table 2).

Overall, more than three-quarters of those who have achieved CT said the methodology helps to improve the quality of products or apps. Greater than half said it helps the organization mitigate risk.

However, there seems to be a large gap between how many believe risk mitigation, improved team productivity, and “enabled time for innovation” will occur between those that have achieved CT and those who have not. Far fewer who have achieved CT said these benefits fit their reality compared to the expectations of teams working to get there.

**Table 2**

		Achieved CT		
Organizational Benefits of CT	Overall	Yes	Not Yet	% Difference
Improved app/product quality	78%	77%	85%	-8%
Risk mitigation	63%	56%	69%	-13%
Improved team productivity	56%	44%	63%	-19%
Enabled time/resources for innovation	41%	29%	50%	-21%
Reduced overhead costs	38%	38%	38%	0%

From a technological standpoint, the benefits promoted to respondents are largely consistent with what actually happens to organizations that have gotten there (see Table 3).

**Table 3**

		Achieved CT		
Technological Benefits of CT	Overall	Yes	Not Yet	% Difference
Reduced # of bugs post-deployment	71%	68%	77%	-9%
Shortened feedback loops	67%	62%	71%	-9%
Shortened development cycles	52%	59%	53%	6%
Eliminated/reduced testing bottlenecks	45%	50%	45%	5%
Reduced MTTR	32%	35%	39%	-4%

## Organizational and Technological Challenges of CT

On the flip side, there are certainly challenges that come with the transition to continuous testing. Just like the benefits we discussed above, we compared the challenges of those who have achieved CT with those who have not. The hope was to identify key barriers that stand in the way of organizations that have not been able to mature their CT capabilities and identify solutions to overcome them.



From an organizational perspective, the leading barrier by far, as is often the case with transitioning to new technologies and workflows, is changing team culture. This was consistent for nearly 70% of all survey respondents (see Table 4). However, the greatest difference for teams that have not yet achieved CT was a shortage of talent or skillsets that enable them to deploy this tactic.

**Table 4**

		Achieved CT		
Organizational Challenges of CT	Overall	Yes	Not Yet	% Difference
Changing team/org culture	68%	64%	70%	-6%
Facing a talent/skills shortage	44%	18%	48%	-30%
Developing/enforcing testing policies	37%	39%	41%	-2%
Gaining buy-in/support from management	32%	27%	32%	-5%
Aligning testing strategies across teams	31%	24%	36%	-12%
Managing start-up costs or investment	27%	18%	29%	-11%
Collaborating with teams about bug fixes	18%	30%	16%	14%

From a technological perspective, as we discussed earlier in this report, it can be challenging for teams to figure out what to automate and what should be conducted manually. That has so far been the greatest challenge for teams that have been unable to achieve CT (see Table 5).

**Table 5**

		Achieved CT		
Technological Challenges of CT	Overall	Yes	Not Yet	% Difference
Finding balance b/t automated/manual testing	43%	27%	52%	-25%
Transitioning out of old workflows	36%	27%	41%	-14%
Prioritizing which test to automate	35%	27%	32%	-5%
Keeping up with bug fixes	18%	18%	19%	-1%
Prioritizing bug fixes	16%	15%	12%	3%

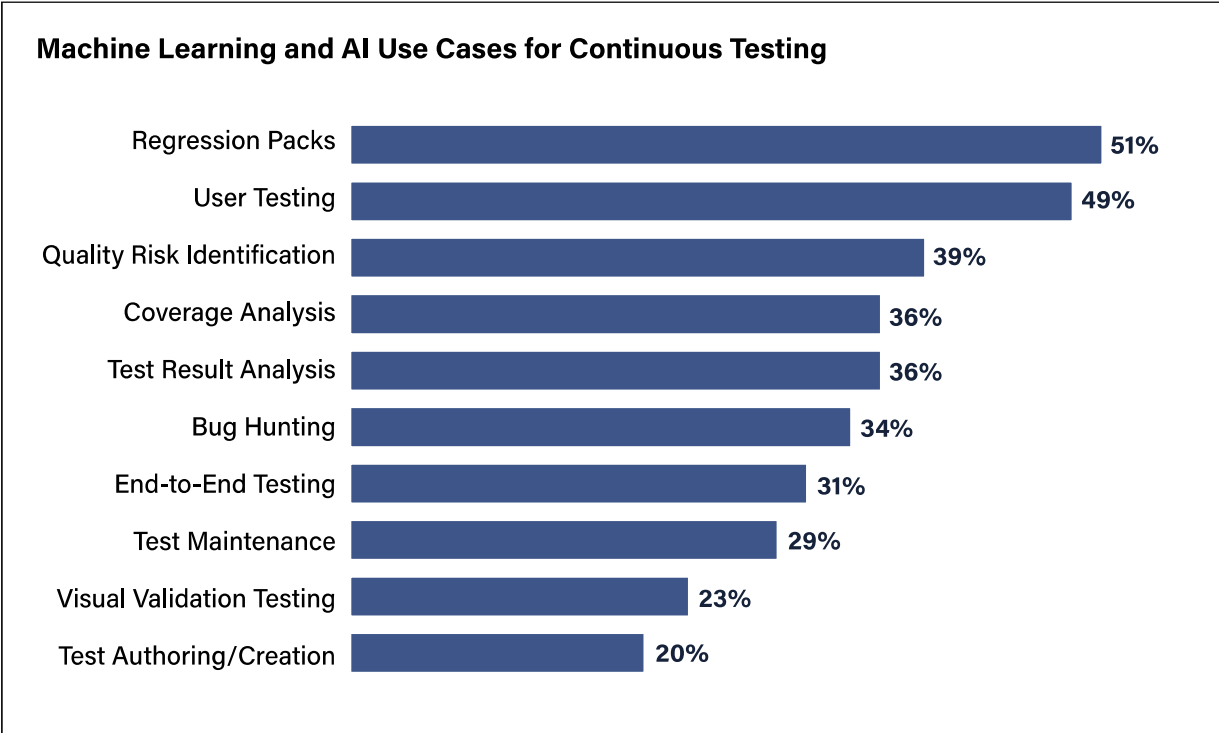
## The Future of Continuous Testing: AI and ML

With any conversation about emerging technologies and capabilities in software, it is impossible to escape conversations about AI or machine learning (ML). Continuous testing is one of those areas ripe for AI or ML interventions to bolster testing capacity. We wanted to know to what extent AI and ML are being deployed for CT, and for what purposes they are being put to use.

Surprisingly, a very small percentage of respondents (just 8%) said they are currently using AI or machine learning for CT. Only another quarter said that while they aren't currently utilizing these capabilities, they are considering acquiring them. However, it is important to note that 27% of those who claimed they have achieved CT and 6% of those who have not achieved CT said that they are currently leveraging AI or ML.

For those using, or considering using, AI or ML for continuous testing, regression packs and user testing were the leading use cases. Other notable ones included quality risk identification, coverage analysis, and test result analysis (see Figure 5).

Figure 5



Based on our research, one thing is certain: The attitude towards testing has shifted to testing “early and often”. Organizations across industries and company sizes have realized the upside of collecting feedback earlier, and continuously, throughout the development lifecycle. With so many organizations on the cusp of maturing their CT capabilities, it will be exciting to see what new innovations in products and applications emerge, and the approaches organizations take to get there. 🧩

# Leaders in Software Testing

## Alan Page, Renowned Author and Industry Expert, Discusses Modern Testing Trends

By **Lindsay Smith**, Publications Content Manager at DZone



**Alan Page**

Co-host of [AB Testing Podcast](#)  
Author of "[How We Test Software at Microsoft](#)" & "[The A Word](#)"

[@alanpage](#) / [angryweasel.com](#)

When it comes to adopting continuous testing methodologies, it's all about good design. Tests can no longer be an afterthought to development. The "write-first-test-later" mentality is dead. In order to achieve quality and continuous testing within your organization, you have to adopt a test-first mindset when writing code.

To better understand the technological and cultural shift organizations are experiencing around continuous testing, we spoke with Alan Page, renowned author, industry expert, previous Engineering Director at Microsoft, and current Director at Unity Technologies, to discuss the importance of test-driven code, common mistakes, and advice for implementing continuous testing principles.

### ALANS'S ADVICE FOR DEVELOPERS

- ▶ **Test-first mindset.** Once developers begin to think about testing as they're writing code, they tend to write the code so that it's more testable.
- ▶ **Hire testing specialists.** Instead of testers conducting tests, employ testing specialists who can teach and influence a testing culture across development teams.
- ▶ **Automate 100% of tests that can be automated.** This is the biggest test design challenge: Figuring out what to automate. Outside of UI tests, the vast majority of tests can be automated.
- ▶ **Avoid flakiness.** These are tests that provide inconsistent or unexpected test results.
- ▶ **Keep it simple.** "If it's difficult to make the test work, the chances are the code isn't very testable, and that's the real bug."
- ▶ **Establish responsibility for software quality.** Clear responsibility and accountability for developers allows teams to apply continuous improvement and continuous testing principles so that testing culture improves over time.

**One of the leading challenges preventing teams from achieving CT are skill and talent shortages within their organizations. How would you direct teams to handle talent shortages around testing?**

What I've done throughout the last six or seven years of my career is dissolved test teams. Not because I don't like testers — I've been a career tester. I find, especially in teams doing continuous delivery, which are generally services orgs, that they can be more efficient without the bottleneck of a dedicated tester. So what I've done is not eliminate but dissolve and reduce. I generally have a few really good testers on my team, but their job isn't to test; it's to help the developers do better testing themselves.

We need to deal with the shortage of testing specialization in general. We need to develop those skills in the generalists on the team, and so, on the teams I work with, the developers are responsible for their own quality and testing for at least 90 to 95% of it, as well as other aspects of project management and delivery.

So, the way I do that is I count on developers to expand their skills and do a large amount of that testing with some help from test specialists.

**Another common challenge reported was determining which tests to automate and which to keep manual. What's your approach to test automation and determining when to conduct manual tests?**

Something I started saying at least 10 years ago is that you should automate 100% of the tests that should be automated. And the topology there is the test design challenge: Figuring out what should be automated. You should automate some amount of UI tests. Those are generally the ones that are more difficult to automate, things at the UI level and at the web level.

I generally advise teams to have as few manual tests as possible. In order to do that, often it's a design choice when developing the software, creating the architecture in a way that makes it easy to test with automated tools. You can get yourself into quite a big hole if your software is designed in a way that a bulk of the verification can only be done through manual tests.

You want to design your system so that the vast majority of it can be verified through low-level tests. Additionally, there needs to be some checks and balances in place for things that have to be tested manually so that they can be done quickly and easily. What I've noticed and what I've coached now to thousands of developers is how to do good testing, and once developers begin to think about testing as they're writing the code, they tend to write the code so that it's more testable.

**Based on our findings, "change in test culture" proved a major obstacle for organizations. Any advice on how teams can overcome these cultural hurdles?**

It really comes down to having clear responsibility and accountability for what it is you do as a developer. What I do like to do with my few test specialists we have on the team is have them play a large part in responsibility for the quality and testing culture of the team.

We can help push it and drive it, but one thing I do with my development leads is I make sure they know they're responsible for quality and delivery of their software. I use the people we have in house with that specialty to help push that culture. And then we just try and apply continuous improvement and continuous, proven principles. Because of this, testing culture should get a little bit better all the time.

But I think you have to have an expectation and accountability of people owning aspects of quality in order to begin building a culture, and like with any other cultural thing, you celebrate the behaviors you want to see. And in doing that, I have gathered some momentum, creating a shared culture across the organization.

**Based on your experiences around automated testing and continuous testing, what are some of the biggest mistakes that you've seen teams make?**

1. **Trying to automate too much at the UI level.** This is still a problem today — in the web space especially. I know of major applications that are tested entirely with Selenium at the web level; it's just inefficient. Those tests take a long time; they're flaky; and teams spend time debugging broken tests all the time. It's just a really bad trap to fall into.
2. **Test flakiness.** I went to a test automation conference five or six years ago, and it seems like every single talk mentioned this concept of flaky tests — tests that are running that are giving you inconsistent or unexpected results.
3. **Test code trying to be too clever.** If it's difficult to make the test work, the chances are the code isn't very testable, and that's the real bug, trying to do gymnastics in your test. In order to test code that's far too complex, or far too difficult to process, in that investigation when writing the test, you've probably discovered the real bug: That code is never going

to be maintainable. It's important to watch out for things like that and not trying to bend over backwards to make your tests work.

**Surprisingly, a very small percentage of respondents (just 8%) said they are currently using AI or ML for CT. Regression packs and user testing were the leading use cases. Where do you think teams should expand their AI/ML implementation within testing?**

All of the tools that I know that use AI and ML to help with testing have actually done really well in that UI testing space I complained about earlier. One of the things that traditionally made those UI automation tests flaky is that the UI can move around, IDs can change, all kinds of stuff can change that break your test.

In the next five years, we can use AI and ML to generate a whole lot of test cases for us. Then to take it one step further, if you can collect actual customer usage patterns — lots of companies collect data on how customers use the software — and then feed that into an AI/ML to create variations, you will be able to create additional test cases or test runs. Testers are infatuated with the end-to-end user experience test, like how this user workflow works, which those tests tend to be flaky.

But again, if we can get rid of the flakiness in these new tools, and then combine that with the ability to generate those real and user test cases from customer data, they will begin to get something that's pretty valuable. In the end, if my evil plan goes well here, testers won't spend any time or very little time writing those UI tests. Writing takes up too much time, and they get a higher quality product at the end. So that's where I hope AI and ML goes.

**Where do you see continuous testing in the next 6-12 months? And what do you think is most important for teams, developers, and managers to keep in mind in order to be successful?**

Continuous testing, continuous integration, continuous everything is all about tightening up feedback loops. Getting to continuous deployment is all about making sure we can get feedback on the thing we built as soon as we can.

Continuous testing makes sure that for every step along the way, we're doing the right kinds of testing. In order to deliver whatever that is, whether it's CI, whether it's to customers, wherever, there's still a lot of improvement left to do there.

As ideal as it all sounds, we're not quite there yet. But continually selecting the right tests to run at the right time, whether those are manual or automated, and getting the tightening of that feedback for that entire system will strengthen our ability to get feedback at all phases faster and faster. 🧩

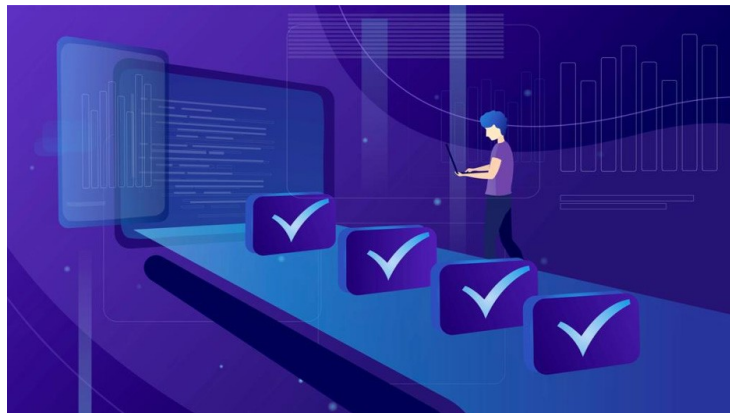
# To Automate or Not to Automate

## The Framework to Continuous Testing Success

By **Alireza Chegini**, Site Reliability Engineer

Images by **Raha Khademi**

When it comes to continuous software delivery, one of the most important steps is testing. You should be able to constantly test existing software and new changes to make sure not only these changes are working as expected, but current functionalities remain intact. Talking about software having many functionalities means not having to conduct all tests manually, and instead, you can automate the majority of tests, thus increasing the speed and quality of your release process.



Once teams shift to automated testing, testing can become a repetitive process in which you're constantly running various automated tests to ensure your application is working as expected. This is a process called continuous testing. In this article, we will discuss key challenges and benefits to adopting a continuous testing methodology with a primary focus on test automation, where to automate, and testing frameworks. Let's get started!

### The Importance of Continuous Testing in the SDLC

Continuous testing accelerates software delivery by making the entire testing process faster. It also ensures teams deliver high quality and reliable applications through rapid feedback in the software delivery pipeline. Additionally, the ability to conduct efficient testing can then reduce testing costs significantly for your organization.

Continuous testing starts at the beginning of each project and continues until the application is delivered to production as a final product. This facilitates quick feedback that helps to identify bugs and other problems in an application faster and earlier. Not only does continuous testing save time for the delivery team, but it also creates a reliable software delivery pipeline in which developers can deliver changes fast with little risk of losing functionality or breaking changes in production. There are plenty of tools out there to help adopt and implement continuous testing. Some of the most popular tools include Jenkins, Selenium, Appium, Katalon Studio, and Tosca. However, regardless of the tool you choose, it is crucial that you pick a tool that can integrate into existing Agile processes, CI/CD pipelines, and DevOps teams.



## Benefits of Test Automation

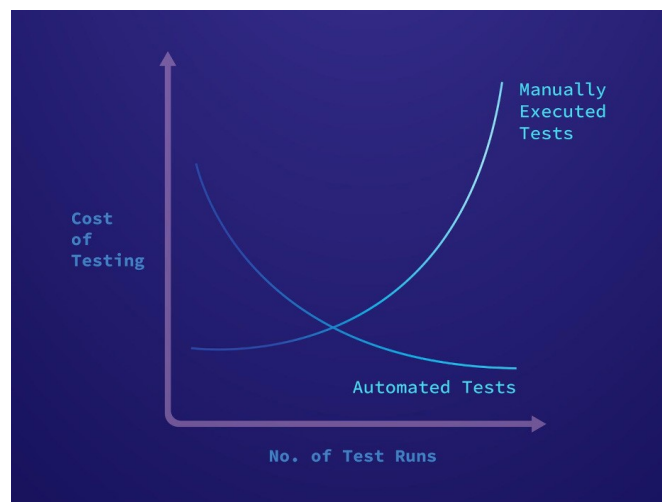
As mentioned above, the key element of continuous testing is test automation. Test automation brings a lot of benefits to your software delivery, which include:

- Rapid feedback
- Accurate testing
- High test coverage
- Faster detection
- Reusable tests
- Shorter release and delivery time
- Better adapted for DevSecOps
- Saved time and money

## Test Automation and ROI

Although test automation has a lot of benefits, it can be quite expensive in the beginning. Buying a new tool, training resources, and implementing automated tests take time and money. In the beginning, the ROI for automated testing is lower than manual testing, but once you are developing new features in your application, manual testing eventually becomes more expensive while automated testing becomes cheaper.

In the end, not everything should be automated — and not everything can be automated. So it is important to assess, research, and analyze your requirements carefully before deciding how to best implement test automation. Next, we will look at when to automate and when to not automate tests.



## When to Automate Tests

To build a successful continuous delivery pipeline, making the right choice at the right time is crucial. Nearly every software team is working on a time-sensitive project, which means there is never enough time to apply every best practice. This can also apply to test strategy since testing is not always a high priority for software teams. Having said that, you should carefully consider this section and try to make the right choices based on your application type, time frame, testing tools, and resources. Here are important test types for your team to consider automating:

### Unit Tests

This is a great place to start when first adopting test automation because unit tests only test a piece of code, with no dependency on other parts of the application. And the code is tested for its functionality. So it gives developers more visibility into the code functionalities. Due to modern testing culture, many teams are following the Test-Driven Development (TDD) methodology in which they start writing tests before writing code. In this way, both code and test quality are assured.

## High Priority Features

If you have dozens of features and limited time, you can list features and prioritize them based on those with a high possibility of failure. Therefore, features with a higher possibility of failure can be automated earlier.

## Regression and Integration Tests

Integration tests are used to determine if individual units in an application are working as a group, and regression tests ensure application functions are functioning as expected. These two tests are usually executed after an application is changed or improved, so testers are constantly running these tests. Having these tests automated saves a huge amount of time, giving time back to testers to execute other types of tests.

## Performance and Load Tests

For performance and load tests, there is no alternative method via manual testing since you need to simulate hundreds or thousands of users with different conditions, often from different browsers, locations, and platforms.

## Repetitive Test Cases

These are very important tests; that is why teams run them repeatedly as they have with login tests. For example, you do not want to lose login functionality since it is essential for application accessibility. Therefore, it is better to automate this test type and save testers' and developers' time.

## Basic Smoke Tests

Unlike other tests, smoke tests are not as complicated and are relatively easy to implement. But don't be fooled — getting these tests right is crucial. Smoke tests are the last hurdle, informing teams if the basic functionality of the application is working properly. These are used primarily when companies are delivering new functionalities and/or basic functionalities, such as:

- Checking whether the main page of the application is up or not.
- Are users able to log in? And are APIs accessible?
- Is the web application accessible from different locations?

## When to Not Automate Tests

When learning about test automation, and the benefits available, one might wonder why not automate all tests. This is a little bit tricky — while the automation mindset is a must for developers and DevOps teams, several factors should be considered before planning to automate a test. Below, we will discuss example tests and scenarios of when to NOT automate your tests:

## User Experience Tests

This aspect of application testing cannot be automated. Many aspects of UX design require manual, tedious rounds of testing. For example, if teams want to test how easily users can sign up to a website, or if they want to test whether various field compositions provide better visibility of users' profiles, these tests would all need to be conducted manually.

## Features in the Early Stages of Development

When a feature is still in the early stages of development, developers are constantly making changes to the code, and this can make it difficult to define a test. Therefore, it takes less time to test these features manually, so wait to start automating until you have a stable version.

## Low-Priority Features

Test automation costs time and effort, so not all features can be automated within the project's time frame. Because of this, teams should prioritize features and automate the important ones earlier. And they might have some low-priority or small features that can be tested manually in the meantime.



## Tests Without Foreseeable Results

When it comes to automation, teams need to know the expected result for each input. If they cannot find this for a test, it means automation will not provide the necessary proof that the feature is working. Therefore, this type of feature test cannot be automated.

## Tests Cannot Be Automated in Full

If teams automate half of a test and the other half remained manual, this would create complexity and unnecessary overhead since such a test is time-consuming and reliability could be difficult. It might be more cost-effective to continue testing these features manually until they can automate them fully.

## What Is an Automated Testing Framework?

In every software delivery team, the QA team is responsible for the design, implementation, and execution of tests. Therefore, for each test, there must be a predefined test case to establish guidelines, rules, and tools to conduct tests. A testing framework is a combination of these guidelines, tools, and practices, which helps test engineers accomplish test cases efficiently. There are different frameworks for different testing goals. Let's introduce some of the most popular types of automated testing frameworks:

- **Modular-Based Testing Framework:** The application is divided into separated modules and each module is tested in an isolated condition.
- **Linear Automation Framework:** Teams record what should be tested in order and then play back the test whenever it's needed.
- **Library Architecture Testing Framework:** This framework is created on top of the modular-based testing framework — the only difference being that it contains reusable functions.
- **Data-Driven Framework:** Test engineers can define a data source for the tests and collect data from different sources like SQL, Excel, CSV, or any other format of data.
- **Keyword-Driven Framework:** In this framework, test keywords are separated from technical code stored in an Excel sheet or other type of data. And every keyword corresponds to a related object and action in the test.
- **Hybrid Framework:** A hybrid framework is a combination of different testing frameworks. In this way, teams can make use of different testing frameworks at the same time and increase the efficiency of automated testing.

## Bottom Line

The most common goal across all software teams is to have a faster software delivery in which you can hand over a quality and reliable product. In order to procure a fast, efficient delivery process, continuous testing must be implemented where possible.

Automation is the key to making sure software can quickly pass all gates and deliver features to clients. However, this does not mean teams should invest all of their time and resources on test automation. Teams must only automate what should be automated. Making the right choices surrounding tests in the early stages of development matters. Don't forget: The goal is not achieving test automation; it's about delivering reliable and quality software. 🎯

# SAP's #1 Choice to Test SAP Apps

Lorem Ipsum

## Continuous Automation Built for Complex Packaged Apps and Beyond.

Keep critical business processes intact throughout the cycles of change with codeless, scalable automation for packaged apps and everything they touch. Discover the power of **Connective Automation** and learn why more leading global enterprises trust Worksoft to automate the full lifecycle of their business processes from discovery to testing to RPA.

Discovery > Testing > RPA

[worksoft.com](https://worksoft.com)

**WORKSOFT**  
Trust in Automation

# From Weeks to Hours



## JOHNSON MATTHEY EXPERIENCES THE VALUE OF AUTOMATION

### Challenge

Shahed Bashir, ERP Project Delivery and Solution Governance Manager for Johnson Matthey recently shared the company's automation journey at SAP TechEd Barcelona.

- **Global SAP Implementation** – deploying a global instance of SAP across 30 locations.
- **Aggressive Timeline** – heightened the need for expedited regression testing.
- **Ineffective Manual Testing** – operating in a manufacturing environment, defects have the potential to impact the availability of production lines. Testing practices were manual, time consuming and ineffective with defects being entered into production systems.
- **Change Management** – needed to achieve better change monitoring and control.

### CUSTOMER PROFILE

- Leading global chemical manufacturer
- FTSE 100 company with 14,800 employees worldwide
- Focused on science and technologies that make the world cleaner and healthier

### Solution

JM chose the Worksoft Continuous Automation Platform based on its ability to:

- **Improve Testing** – automated continuous testing to save time and improve QA.
- **Integrate with SAP** – SAP certified solution capable of conforming to SAP practices.
- **Support Different UIs** – a platform to cater to many different UIs across different use cases.
- **Expedite Adoption** – minimal training required, easy to use, implement and support.
- **Identify E2E Business Processes** – able to support "shift left" approach to automation.

# 90%

TIME SAVINGS VS.  
MANUAL EFFORT

# 75%

COST REDUCTION IN  
TEST DEVELOPMENT

### Results

#### Acceleration of SAP Deployments

"When you implement SAP it takes time. You need to define the processes and everything else. But with Worksoft Certify, we actually had the solution itself up and running within a week, because the platform is very lean in terms of infrastructure and is very rich in functionality....It allowed us to actually speed up the different SAP deployments and the waves across all our regions. Prior to that, we were looking for each deployment to take at least two months of manual regression testing. But now, we can have it done, tested within a week, signed off and approved, ready for deployment."

#### Test SAP and non-SAP

"As you know, SAP runs a lot of your processes, but there are still lots of other applications as well. So one of the advantages we found with Worksoft was that we could also apply the same platform, the same tools, the same capability to non-SAP systems as well. It really does give us a platform we can use to automate both SAP and non SAP."

#### Automate Discovery and Documentation

"We're able to actually use the tool to capture the processes and then automate them as part of the next cycle. And the real value of that is by actually shifting it to the left, we're able to catch those issues at an earlier stage before they impact production." And with Worksoft BPP, JM is able to document their detailed process to meet QA and quality assurance requirements.

# 300%

ACCELERATION OF  
BUSINESS PROCESS  
DISCOVERY

(972) 993-0400  
[info@worksoft.com](mailto:info@worksoft.com)  
[worksoft.com](http://worksoft.com)



# Strategic Continuous Testing

## Integrating Test Automation into the SDLC

By Priyanka Arunachalam, Data Analyst

*"Automation applied to an inefficient operation will magnify the inefficiency."*

– Bill Gates

Continuous testing has been a buzzword over the years, and it is no longer considered a new term in the world of software development. Rather, it is a software testing strategy that is used to evaluate the quality of the product, not only during the latter half of the software development life cycle (SDLC) but also early stages. The famous saying "prevention is better than cure" is applicable in the SDLC, too. Yes, testing the product early and often, and fixing the bugs before moving it to production, can actually bring a wide range of benefits. So how could this actually help in managing deployments?

### Why the SDLC Demands Continuous Testing

The primary factor in a successful continuous testing framework is incorporating automated feedbacks. Test automation reduces the stand-by time for developers by shift-left and shift-right mechanisms, which means testing takes place throughout the SDLC, from beginning to end.

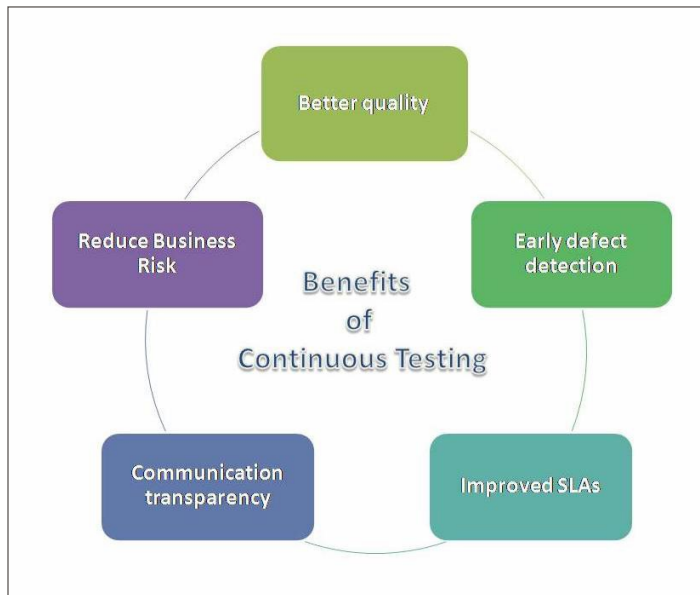
This is the essence of continuous testing. It acts as a catalyst for the effectual continuous integration and continuous delivery (CI/CD) processes, thus accelerating the SDLC schedules by providing continuous quality and improvement. The intent behind continuous testing is to drive velocity without impacting the business value and cost. Continuous testing is also vital to making sure the end-user experience is uninterrupted and consistent, thus bridging the gap between continuous delivery and reliable user experience.

### THE VALUE OF ADOPTING A CONTINUOUS TESTING METHODOLOGY

Realizing the key benefits of continuous testing in system development is mandatory for adoption of the right tools and successful implementation. CT is equipped with myriad key benefits, out of which a few are listed below:

- Accelerates delivery to production and the timeline for release with continuous feedback at each stage of SDLC.
- Triggers frequent product releases when integrated into CI/CD pipelines, resulting in quick user feedback and in turn, faster delivery of improvements to product features.
- Ensures products are released with better quality to meet and help exceed customer expectations.
- Increases testing speed as performance tests can run parallel. And with automated test cases, overall time spent testing is reduced, creating a more reliable testing process.
- Saves time as developers no longer need to wait on QA teams to complete tests, which can help avoid the effort of reworking code.
- Reduces the risk of business and costs by accessing the issues at an earlier stage before they develop into larger, more complex problems and create negative business impact.

**Figure 1:** Benefits of continuous testing



### **PAIN POINTS IN IMPLEMENTING CONTINUOUS TESTING**

Teams run various tests throughout the entire continuous integration and continuous delivery process, yet testing remains a bottleneck in the software development lifecycle. As teams begin to embrace continuous testing, they may encounter hurdles that will need to be tackled to ensure successful implementation of a CT methodology.

#### **1. Test Data Management**

Locating and managing the right test data already takes you half-way. Various surveys have spotted that most of the testing time is spent on managing test data because accurate testing of new functionality requires valid test data for each run.

#### **2. Smarter Testing**

The number of tests to perform is yet another invisible challenge, as it's often not a primary focus for teams. The extensive goal of continuous testing is to integrate a seamless feedback mechanism, which can lag if there are too many tests running for hours simultaneously. Hence, test smarter!

#### **3. Testing Analytics**

The volume of output generated by testing has become mind-boggling with the adoption of CT, making it tough to analyze and difficult for both the development and operations teams to gain clear visibility of testing analytics. Analysis is needed to check that the output can be fed back through testing loops, which enables developers to seamlessly observe user response.

#### **4. Access to Dependencies**

Not having access to the right environment and services could be yet another obstacle while implementing continuous testing, for which service virtualizations could be a helping hand. Using service virtualization, developers can simulate interaction with missing dependencies. This can ensure consistent performance across the test runs.

#### **5. Scalable Infrastructure**

At times, organizations realize that continuous testing can be tough as their infrastructure may not be scalable enough to adapt to uninterrupted testing. Focusing on business priorities and parallelizing tests with various application release automation (ARA) tools can help address this challenge.

Along with Agile and DevOps, continuous testing drives organizations to deliver quality software as quickly as possible, but it has its own set of roadblocks to cross over before achieving the victory point.

## Test Automation in the SDLC

In this competitive world, consumers are often waiting impatiently for the newest products but still expect them to be the highest quality. Meeting high market demands and user expectations creates a sense of urgency for development teams, whom work to shorten release cycles, in part, to meet the dynamic needs of their customers. Making this possible is the keystone of the SDLC — automation.

Agile methodology has long been integrated into software development and while automation continues to dominate today, you just can't say, "automate everything." Automating a process should ensure time reduction, cost cutting, accuracy, and quality. As a key part of a healthy, mature CI/CD pipeline, automated testing should uphold those principles just the same. So how can development teams integrate automation in their continuous testing efforts?

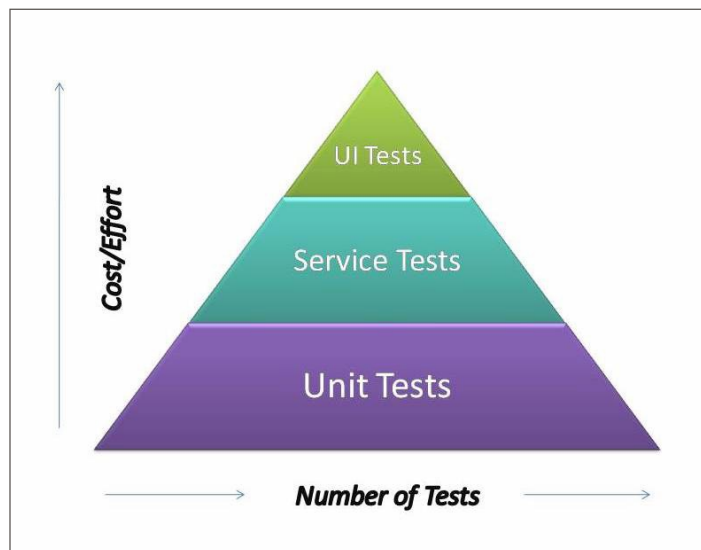
### TYPES OF TESTS TO AUTOMATE

Automation should kick start right from the design stage. Though test design widely remains a manual activity, automation can eliminate much of the effort required in areas such as **functional testing**, and it enhances overall testing quality. Test automation helps assure application quality and reduces the cost of production bugs. Also prime for automation are **unit tests**, for which an application is broken down into tiny, testable parts to identify bugs at an earlier stage in the SDLC.

The vital part of DevOps — **regression testing** — is one of the best candidates for automation for the sheer fact that managing manual regression tests for each new feature takes up an extensive amount of time. Automating regression tests is vital for faster feedback and quicker verification of new bugs that were raised due to recent change of functions.

Though there are differences of opinions about what to automate versus not automate in continuous testing, eliminating any level of manual effort will help increase the efficiency of the continuous testing process. The test pyramid below offers some hints for test automation. The split of layers is based on Return On Investment (ROI) proposed by automating each of them.

**Figure 2: Testing pyramid**




Automation is everywhere — many things now come with the click of a button. Test automation, in particular, should be prioritized followed by automating Development, which is considered the most significant part of the SDLC. This empowers developers, as they can spend less time reworking code, enabling them to concentrate more on high-quality, efficient deliveries that generate expected business value.

Some common areas where automation can be evaded are those that handle lower volumes of data, need less human interaction, and have less direct impact on ROI. Remember, automation in a wrong direction never allows you to progress.

### **Continuous Testing: A Journey, Not a Destination**

In the end, every organization wants their application to stand out as the top in the market to attract and retain more satisfied customers. Continuous testing performs well in this regard and also helps teams catch critical bugs with test cases that involve real-world use. Though developers may feel that manual tests are more effective for some efforts, integrating automation in continuous testing continues to improve processes and build trust with more developers.

The 'fail fast, fail often' methodology in test automation has helped to ensure quicker feedback loops and reduce the risk of bugs becoming complex and impacting dependent functions. Now, we're seeing the shift-left practices blooming with behavior driven development (BDD), service virtualization techniques, and more. As continuous testing becomes an integral part of CI/CD pipelines, adoption by many teams continue to be a challenge.

Today's digital world necessitates faster delivery without comprising on quality. While continuous testing will endure time, artificial intelligence will also progress and help address the vivacious needs of the user. From continuous testing to "Digital Testing," let's thrive together to adapt to the accelerating rates of change. 

# Functional Testing vs. Robotic Process Automation

## Same Car, Different Paint?

By Chris Kraus, VP Product Management Worksoft

When evaluating functional testing tools, specific functional requirements should be considered. Over the last 20 years, the concepts and requirements for functional testing have changed and evolved in the same ways that software architectures have evolved. During the early days of software, testing was manual and not well documented. Automated testing was first introduced by Mercury Interactive with TestDirector. Later, the industry moved away from the manual testing of user interfaces to automated functional testing of user interfaces, which later evolved to API, load, and performance testing (non-functional forms of testing) to service virtualization and security testing. It is important to remember the large history requirements and innovation that have led us to modern-day testing concepts and best practices for functional testing.

Unfortunately, marketing teams have become highly skilled at glomming onto industry trends and whitewashing their existing product, allowing them to freely say "Hey, we do that!" However, buyer requirements for functional testing are broad and feature-rich. For example, authoring in a code-free manor creates the desire for API tests to run functionally or as performance tests, operating systems to include desktops, servers, and mobile devices. More importantly, the core workflow of testing requires that test results be fully verbose as well as available in ALM tools and/or via a DevOps pipeline of choice.

RPA tools are rapidly entering popular conversation and the core requirements are new enough that buyers may not know exactly what they are looking for. It is easy for RPA vendors to bend an ear and place FUD out in the market as to what customers need. Overall, RPA buyers are looking for bots, security, and repeatable processes that can be easily accomplished by a bot.

So, the big question is: Are all RPA vendors whitewashing when they remind customers that they also provide testing? That seems to be a pretty big leap given the 20+ years of innovation and requirements testing vendors work hard to provide. Moving past the low-level technical bites (think features, stories, and tasks in Agile) to zero in on which epics and themes should be evaluated and provided as useful for testing vs RPA. Can RPA be a drop-in replacement for testing and start to take over?

## Testing vs. RPA: Key Differences

The following themes highlight many differences regarding the goals and purposes of test versus robotic processes, with some important overlaps.

### Access

#### TESTING

1. Every tester and developer needs access to the testing tool. Open source and free means everyone can help.



2. High-level, pass-fail statistics should be accessible across the development, operations, and business organization.
3. Sharing test assets across the organization is encouraged; it is just test data and test systems. It is easy to get access to a test system in the development organization because pre-production systems don't contain data considered sensitive by PII or GDPR.
4. The Agile Manifesto declares everyone should own quality.

## **RPA**

1. Specific people will have access to the authoring tool, because it is expensive, and no open-source options are available.
2. Development should be left out of reporting; the bot's throughput and performance is a key metric for operations and businesses to evaluate whether or not the bot is providing tangible value, or failing to do so, which then requires manual rework.
3. Sharing assets may not be as easy in production. Bot authors may not have access to all of the systems in production to see them and/or build against them.
4. Sarbanes Oxley and GDPR laws mean that not everyone should see production data and PII data.

## **Security**

### **TESTING**

1. Every tester and DevOps engineer should be able to run tests.
2. The scope of end-to-end tests and API tests can span different systems and blur organizational boundaries. The goal of testing is to see how systems work together (integration testing), and end-to-end workflows should span a complete set of user journeys.
3. Detailed reports, system, application logs, and screen shots of the process running must be easily accessible to developers and QA teams. When a test fails, it is common to determine if the failure was a change in the system, a functional failure, or a false failure. Access to all of the logged information must be shared with multiple roles to conduct the analysis.

## **RPA**

1. Only production operations people should be able to run or schedule the robotic processes. The bots are working with production systems and data.
2. The scope of a bot is drastically reduced and does not span multiple divisions of a company or even processes. Given a bot is doing the majority of the work, performing a single task or step in the business process is much more auditable, and by being limited, it won't have as much of a chance to go rogue.
3. Bot results, logs, and screens are highly secured and available in operations on a need-to-know basis. This is actually production data.

## **Workflow**

### **TESTING**

1. The workflow for testing can be a simple one-person job — development has a requirement change; test case documents and automated tests are created or updated to meet the new requirements; and the test is scheduled to run or is placed in the DevOps pipeline.
2. The purpose of testing is to find bugs, break the system, and vet unhappy paths through the application. Test cases should run a happy path through the application, and negative paths validating all of the fields will reject bad data.
3. It is a given that the system being tested is either changing or causing a change in another part of the system, which could have an upstream or downstream consequence. Testing lives for a dynamic environment.
4. Validate everything! In testing, you validate every field for error message text, required fields, numeric vs. alphabetic data, date validation, rendering across multiple browsers and devices, and whether it runs across various operating systems. A test case can complete an order in a system but flag errors on the page. For example, dates can be entered but the calendar picker can sometimes not render correctly; marketing images can not render properly,

vertical spacing in a table can be too short for text height, or the order just sent to the page could not pass verifications from a test perspective.

## **RPA**

1. With business process changes, the modification of a bot in production is a multi-actor workflow. A change to the process (or bot) is identified, discussion with change review board and users decide to stop bot or rebuild the bot and move through the change process. This involves business, bot developers, business approval, and operations for deployment.
2. The purpose of a robotic process is not to validate the system but to enter data into the system in the fastest, most efficient, and auditable manor. Bots run the happy path through the application and don't try to enter bad data on purpose.
3. Stable systems only — we don't want to change robotic processes weekly and go through change control.
4. "Validation, what's that for," said the bot? By definition, the robotic process is not written to validate every field on the page and report back findings. The robotic process does not care if an image rendered or not. If the data was entered and the style sheets are all wrong, but the data is successfully entered, the bot's job is done. There is no concept of a completed order aside from failing data validations or visual indications.

## **DevOps**

### **TESTING**

1. Integrate with every requirement and ALM tool — then provide reports back.
2. Provide a plugin for each CI tool and report back to the pipeline what happened during testing.
3. Rerun and restart the tests after failures.
4. Scale up the Continuous Testing cycle for full regression and load testing.

## **RPA**

1. What is ALM?
2. Operations runs bots in proprietary tools — there is no need to work in CI pipelines.
3. Run, fail, stop, and run next or run fail, and expect human intervention to fix what went wrong.
4. CT? RPA bots are in production doing the real work of entering orders across multiple systems that are not integrated or providing lookup services creating consolidated reports for auditors who need the full view of an order.

Many RPA companies claim they have a great bot with lots of prebuilt gizmos to make authoring really easy. In the testing world, we remain skeptical as we have enterprise-class testing platforms, code-based products, and open-source solutions with rich user communities and prebuilt environments.

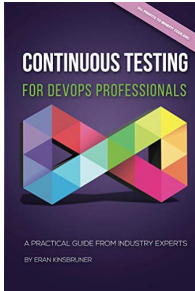
Give RPA bots due credit, they can type data into user interfaces or forms and conduct data input faster and more efficiently than humans. Some bots excel at OCR and read typed invoices, turning the visual into decipherable input for other systems. All seem to excel with opening mail, looking for data, and performing an action based on the received email. (If only that worked for my inbox!) The decisions they make are actually very rote and mundane, but many RPA vendors will claim that with help of AI and ML, the bot will make decisions and adapt in real time. In the current reality, that's just marketing spin! More than one futuristic movie has shown us what happens when bots start to think for themselves.

Personally, I have never heard a QA or business analyst say, "let's spend the weekend in the office manually testing," but our human side makes us leery to think a robotic process is intended to replace human jobs. It's not likely to be the case with testing tools. The art of designing tests to verify and break down software tools is just too human. More importantly, RPA vendors cannot suddenly produce 20+ years of functionality in testing tools and have the evolution of open source allow them to encroach upon the market.

The RPA market is still too new and fluid, and they may be whitewashing "we can do testing too" presuming that it is just another automation. A tenured expert in the testing profession knows better.

# Diving Deeper Into Continuous Testing

## Books



### Continuous Testing for DevOps Professionals

By Eran Kinsbruner

Continuous Testing for DevOps Professionals is the definitive guide for DevOps teams and covers the best practices required to excel at Continuous Testing (CT) at each step of the DevOps pipeline.



### Continuous Testing: A Complete Guide, 2020 Edition

By Gerardus Blokdyk

This complete guide to continuous testing explores modern testing practices through a self-assessment for tech professionals to better assess testing standards, processes, and best practices.

## Trend Reports

### The State of CI/CD: Evaluating Pipeline Maturity

DZone surveyed developers about how DevOps teams are building up and taking advantage of their engineering maturity. Read our original research, articles from DZone's most reputable contributors, and our exclusive interview with Gene Kim, renowned author and DevOps thought leader.

### Application Performance Monitoring

Readers will discover key findings from our original research and a DZone-exclusive interview with DevOps activist Andreas Grabner. Also included are new articles from DZone contributors, whom shared their insights into APM's impact on team culture and the end-user experience, AIOps, and considerations and priorities for growth-minded organizations.

## Refcards

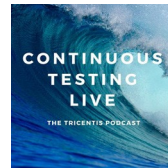
### Quality Assurance Patterns and Anti-Patterns

This Refcard walks through patterns and anti-patterns in quality assurance within the Lean- and Agile-forward implementations of one such methodology, the Scaled Agile Framework, also known as SAFe.

### End-to-End Testing Automation Essentials

This Refcard evaluates E2E testing automation, process, and key benefits, including how it provides the same level of functional validation performed by a computer but in a dramatically shorter amount of time.

## Podcasts



### Continuous Testing Live

This podcast shares conversations with thought leaders around continuous testing, Agile development, DevOps, digital transformation, and ways software testers can have an even bigger impact on the world around them.



### AB Testing

In this podcast, Alan Page and Brent Jensen interview thought leaders and talk all things modern testing — including Agile, Data, Leadership, and more.

## Zones

**DevOps** As a cultural movement supported by exciting new tools, DevOps aims to encourage close cooperation across development teams and IT operations. The DevOps Zone is your hot spot for news and resources about continuous delivery, Puppet, Jenkins, and more.

**Performance** Scalability and optimization are constant concerns for the Dev and Ops managers. The Performance Zone focuses on all things performance, covering everything from database optimization and garbage collection to tweaks for maintaining efficient code.



INTRODUCING THE

# DevOps Zone

DevOps encourages close collaboration within cross-functional teams of developers, IT operations, and system admins. Dive into the latest DevOps news and resources to learn how this movement is changing the way modern software is made.

Keep a pulse on the industry with topics such as:

- CI/CD pipeline management
- Automation across the SDLC
- DevOps culture and methodology
- Testing strategies and frameworks

Visit the Zone



TUTORIALS



CASE STUDIES



BEST PRACTICES



CODE SNIPPETS