DZone

DZONE'S 2019 GUIDE TO

Automated Testing

FACILITATING CONTINUOUS DELIVERY

BROUGHT TO YOU IN PARTICIPATION WITH

bitbar. Compuware Kobiton functionize

mabl Perfecto SAUCELABS

Dear Reader,

Welcome to DZone's latest guide to *Automated Testing: Facilitating Continuous Delivery*. We chose this subtitle not because automation is a strict prerequisite for practicing CD, but because it can make that process so much easier and accelerate your organization's development efforts. Many may think that this is all common knowledge, but we're seeing that more and more DZone readers are incorporating testing earlier in the development process. There has yet to be a plateau or decline in these numbers year over year.

Users are fortunate that this is the case. The world of software has officially moved to a subscription-based, software-as-a-service model. Pushing regular updates when your software is browser-based is now considered normal, but as updates are pushed, it's critical that there are no issues that make the user's life difficult, lest they start questioning why they spend a yearly fee instead of buying licenses and choosing to skip updates if they hear about any new issues they create, or move to a competitor. They may not cancel their subscription the first time, but their patience will wane with each subsequent problem, and it's your job to make sure that patience is not continuously tested.

Automated testing can alleviate these concerns and ensure that your team is shipping working software the first time, just as your customers expect. When your software doesn't work as intended, you may not even have days to fix these issues. This would be impossible if you're relying solely on manual testing to fix bugs and address performance concerns.

In this guide, we will cover the benefits of Test-Driven Development, integrating automated testing into your existing DevOps processes, mobile UI testing, parallelization in automated testing, and how Natural Language Processing could make testing easier. You'll also find research thanks to the efforts of readers like you that help us determine the state of automated testing in the software development industry and how it might move in the future.

Thank you for downloading and reading this guide. We hope these articles and insights help you understand why advancements in automated testing are still of paramount relevance and importance. For more information, you can find articles on automated testing at our DevOps zone.

Happy reading (and testing)!

WRITTEN BY MATT WERNER
PUBLICATIONS MANAGER, DEVADA

Table of Contents

DZone is...

Executive Summary

BY KARA PHELPS EDITORIAL PROJECT MANAGER, PUBLICATIONS, DEVADA

As we release this guide, the third annual Guide to Automated Testing, the world of software development continues its evolution toward true continuous testing and continuous delivery. If a software organization wants to scale, the current environment demands that it continually "shift left." Automated testing is a crucial part of this growth and change, and more and more organizations are beginning to understand this each year. We asked 498 tech professionals to share their thoughts on automated testing, the tools they use, and how their organizations are ensuring quality at every stage of the software development lifecycle.

An Earlier Start in the SDLC

DATA

Last year, 54% of those surveyed said they typically began automated testing in the development stage of the software development lifecycle (SDLC), and 31% said they began in the staging/QA/testing stage. In 2019, 66% of survey respondents said they begin automated testing in development, and 22% said they begin in staging/QA/testing.

IMPLICATIONS

Among survey respondents, we've seen a significant uptick in the number of organizations beginning automated testing in development. We've also seen a roughly equivalent decrease in the number of surveyed organizations leaving automated testing until the staging/QA/testing stage. Organizations are shifting tasks "left" in the product pipeline, completing them earlier in order to enter production faster.

RECOMMENDATIONS

Reliably, more and more organizations start the road to continuous testing and continuous delivery each year, which often necessitates automation. Shifting your automated testing further left is generally a smart move, regardless of where you are in your implementation of DevOps --- the quality of code improves the earlier it can be tested.

A Critical Priority for More Orgs

DATA

Only 28% of those surveyed in 2019 said they do not consider automated testing to be a critical priority. That's a decrease of 5% from last year, when 33% said they did not consider it to be critical.

IMPLICATIONS

This data suggests that fewer organizations among the survey takers have stolen focus away from any automated testing goals this year. It reflects a more general trend of automated testing becoming more central to the success of an organization's SDLC.

RECOMMENDATIONS

Manual testing is becoming a bigger bottleneck as the speed of the software market demands multiple releases a day --- especially in shifted-left organizations where the developers are also the testers. If you haven't yet been convinced to investigate your options for automated testing, now is the time. DZone's recent Refcard on Test Design Automation is one good place to start.

DevOps Teams Practice TDD

DATA

15% of survey respondents who use test-driven development (TDD) on all their projects also have a dedicated DevOps team in their organization. 7% of those who use TDD for everything do not have a DevOps team (using TDD with "only certain teams" proved just as common among organizations with their own dedicated DevOps teams as those without). Among survey respondents who reported that they work for orgs with dedicated DevOps teams, "test automation creation" was chosen the least often to be the "most challenging" pillar of continuous testing.

IMPLICATIONS

Organizations with DevOps teams are more likely to use TDD on all their projects. They are also the least likely to find test automation creation to be challenging.

RECOMMENDATIONS

TDD is an important part of DevOps, and teams focusing exclusively on DevOps implementation already recognize this. While it may not be suited to every use case, TDD offers a substantial advantage particularly to developers of modern enterprise software applications. For more details on the practice of writing tests first, take a look at the article on the benefits of TDD later in this guide.

Key Research Findings

BY JORDAN BAKER PUBLICATIONS ASSOCIATE, DEVADA

Demographics

For this year's Automated Testing Survey, we polled 498 IT professionals, with a survey completion rating of 73%. Below is some of the basic demographic information based on the responses of these survey takers.

- On average, respondents have 17 years of experience in the field.
- Respondents work for organizations based in two main geographic regions — the USA (41%) and Europe (32%) — while they live in three main geographic regions — Europe (34%), the USA (27%), and South Central Asia (14%).
- Respondents tend to work for enterprise-level organizations:
  - 23% work for orgs sized 1,000-9,999
  - 22% work for organizations sized 10,000+
  - 20% work for organization sized 100-499
  - 60% of respondents work on teams of 10 people or fewer.
- Respondents are engaged in developing three main types of software:
  - 82% develop web applications
  - 46% are working on enterprise business applications
  - 25% develop native mobile apps
- Survey takers tend to perform one of three main roles:
  - 34% are developers/engineers
  - 20% are developer team leads
  - 16% work as architects
- Three main programming language ecosystems are used by respondents' organizations:
  - 78% use Java
  - 69% use client-side JavaScript
  - 40% use Python
- The ratio of respondents who work at an org with a dedicated DevOps team is 50:50.

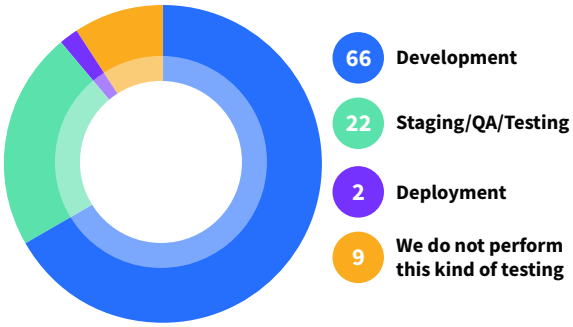
Automated and Manual Testing: A Continued Shift Left

Even though the shift-left mentality and methodology has been around for several years, it continues to gain momentum. In 2018, when we asked respondents at what stage in the SDLC do they typically begin automated testing, 54% reported development and 31% said staging/QA/testing. In this year's survey, 66% of respondents told us that they begin automated testing on their software during development, and 22% begin in staging/QA/testing. Interestingly, while automated testing continues its rather seismic shift to the left, manual testing methodologies saw little to no change over last year's survey. In 2018, 51% reported beginning manual testing in development, while 40% reported beginning in staging/QA/testing. This year, 49% reported performing manual testing in development and 44% claimed to do so in staging/QA/testing. But, again, neither of these year-over-year changes are statistically significant enough to signify a large shift in manual testing's place in the SDLC.

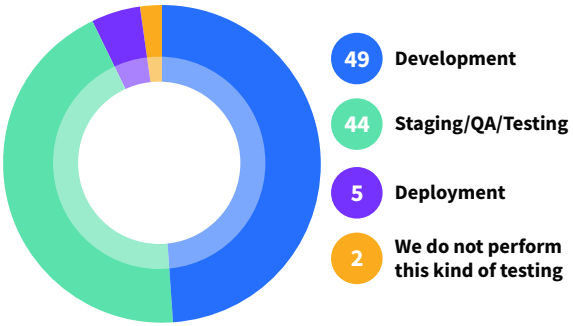
While the above highlights the usage patterns of those developers who have adopted either automated or manual testing, we also gathered data around why certain respondents' organizations opted against one or the other of these testing strategies. Among those survey takers whose organizations/teams do not perform automated testing, the most common reason for doing so, reported by 28%, is that it is not considered a critical priority. In 2018, 33% reported not considering automated testing a critical priority. This matches up nicely with the continued adoption and leftward shifting of automated testing discussed above. When it comes to those whose organizations do not perform manual

SURVEY RESPONSES

At which stage does your organization usually begin automatically testing software?



At which stage does your organization usually begin manually testing software?



testing, one statistic jumps of the page. In 2018, 7% of respondents reported the main reason for not adopting manual testing was the lack of a QA/testing team. In 2019, 43% told us they had no dedicated QA/testing team. This seems a troubling stat. One possible answer to this large year-over-year change is that, as testing is continuously shifted left into the development phase of the SDLC, developers are becoming de facto testers on top of their roles of software engineers.

Now that we've covered where automated and manual tests occur in the SDLC at a high-level, let's drill down a bit to examine the types of tests that are automated and performed manually. Among those respondents who perform automated testing, four main tests were reported: integration, component, performance, and story-level. Comparing the data we received around these tests to our historical data from 2018, we find that the percentage of those using automated integration tests stayed stable (77% in 2018 vs. 78% in 2019), while automated story-level tests (36% in 2018 vs. 46% in 2019) and component tests (52% in 2018 vs. 59% in 2019) both grew in popularity. Automated performance tests, however, fell slightly year-over-year, dropping from a 51% adoption rate in 2018 to 46% in 2019. Among those performing manual tests, four types of tests proved predominant: user acceptance tests, usability tests, story-level tests, and post-deployment tests. Interestingly, despite the seemingly stagnant state of manual testing outlined above, we found that all four of these manual tests grew year-over-year in terms of adoption rates. Here's a quick breakdown of this historical data:

- User Acceptance Tests
  - 2018: 60%
  - 2019: 67%
- Usability tests
  - 2018: 50%
  - 2019: 58%
- Story-level tests
  - 2018: 41%
  - 2019: 47%
- Post-deployment tests
  - 2018: 36%
  - 2019: 40%

THE ROLE OF DEVOPS IN AUTOMATED TESTING

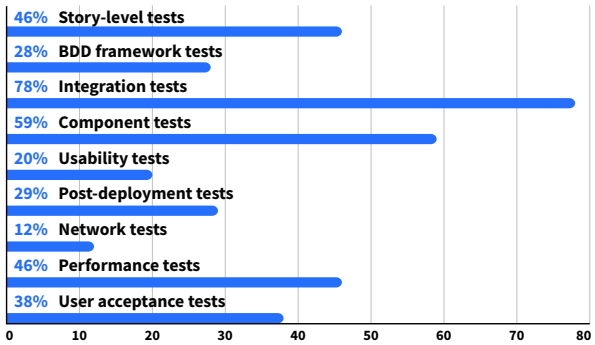
THE EVOLVING ROLE OF DEVOPS

As noted in the Demographics section at the beginning of this report, the ratio of respondents' organizations who have and don't have a dedicated DevOps team is an even 50-50 split, which didn't change significantly from last year (in 2018 48% of respondents reported having a DevOps team). But for those who do have dedicated DevOps teams, the role of these teams is gravitating more to a cultural role than a technological one. In our 2018 Automated Testing survey, 67% of survey takers reported that the main goal of their DevOps team was to help the organization adopt the best Continuous Delivery tools; in 2019, this rose to 74%. Additionally, last year 46% of respondents told us that their DevOps team worked to increase the collaboration and break down silos between Dev and Ops; this year, that figure rose to 53%. All the other, more technical, DevOps team functions either did not grow or shrank in importance. The most shocking, however, was the decreased impetus to develop and deliver software across the entire stack. In 2018, 52% reported this a main goal of their DevOps team; this year, 42% said this type of software delivery was a main goal for their DevOps team. Also, improving frequency of deployments (62% in 2018 vs. 63% in 2019) and introducing automation across the SDLC (56% in both 2018 and 2019) remained of equal importance year-over-year.

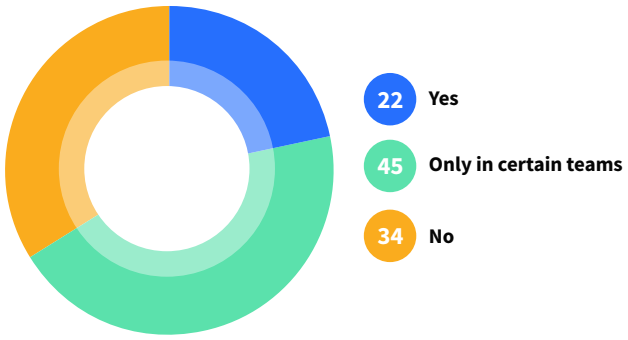
Despite the growing cultural role of DevOps in enterprise software organizations, it still has a large technological part to play. If we compare the data on those respondents whose organizations have dedicated DevOps teams to the stats we gathered on respondents' software build processes, DevOps's continued role in automation becomes clear. Of the 58% of respondents who told us they break their builds up into stages, 34% have a dedicated DevOps team, and of the 46% who include automatic checks to proceed, 21% have a DevOps team. Additionally, while a total of 34% of respondents have security issue detection built into their SDLC, 21% of those respondents have a DevOps team in their

SURVEY RESPONSES

Which tests in your organization's pipeline(s) are currently automated?



Does your organization practice test-driven-development (TDD)?



organizations create agility and automation throughout their build process and implement proper testing and security practices.

TESTING STRATEGIES

Much like the role of DevOps, the role of testing within the SDLC seems to be undergoing an evolution. In 2018, 28% of respondents reported to practice test-driven development (TDD) across all their development efforts. In 2019, this number fell to 22%, while the percentage of respondents who do not practice TDD at all remained stable (36% in 2018 vs. 34% in 2019). The number of respondents using TDD selectively, however, saw a year-over-over increase. In 2018, 36% of survey takers told us they use TDD "only in certain teams." In this year's survey, that number rose to 45%. If we compare these numbers to our data on organizations with DevOps teams, we find that those orgs who have dedicated DevOps teams are more likely to practice TDD across the board than those that don't. 15% of those respondents who use TDD on all their projects also have a dedicated DevOps team, versus 7% who have no DevOps team, but still adhere to TDD for all their development efforts. Interestingly, TDD adoption among "only certain teams" proved equally popular among organizations with and without DevOps teams.

Despite this year-over-year shift in the frequency of test-driven development efforts, the benefits that respondents reported to derive from TDD remained the same. Just as in our 2018 survey, the top four benefits of TDD were: improved quality of code (80%); less time spend debugging (59%); easily maintainable code (49%); the ability to use tests as documentation (45%). Intriguingly, after running these numbers against our stats on DevOps adoption, only one of these benefits seems to occur more frequently in organizations that had dedicated DevOps teams. Among those who reported seeing improved code quality due to TDD, 54% work for organizations with DevOps teams, while 46% do not.

Moving from TDD to the continuous testing paradigm, however, we see DevOps have a more considerable effect. When we asked survey takers which of the four continuous testing pillars they saw as the most challenging, we saw the following results (number one receiving the most votes for "most challenging," with number four receiving the least votes for "most challenging"):

1. Maintaining a stable and up-to-date test lab
2. Test automation creation
3. Test execution and orchestration
4. Test report analysis

Among those respondents who reported to work in an organization with a dedicated DevOps team, we see these results turned almost upside down. If we look at the four pillars of continuous testing through this DevOps lens (again, with number one receiving the most votes for "most challenging," with number four receiving the least votes for "most challenging"), we see the following:

1. Test execution and orchestration
2. Test report and analysis
3. Maintaining a stable and up-to-date test lab
4. Test automation creation

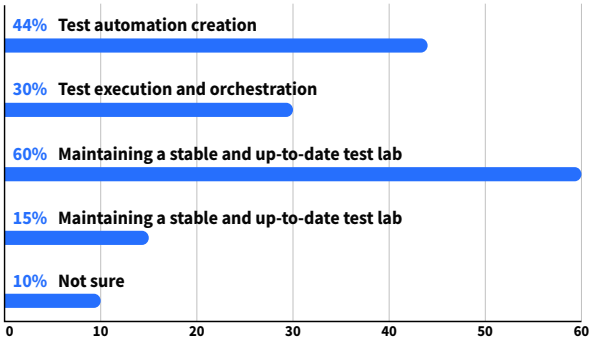
Tellingly, those survey takers who work for companies with a DevOps team found test automation the least challenging of the four pillars.

TECHNOLOGY ADOPTION

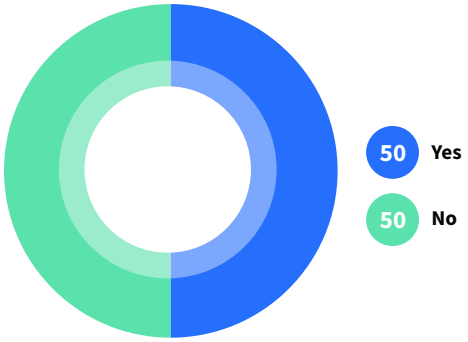
In order to continue to make strides in the areas delineated above, organizations have begun a shift toward the use of microservices and containers. And while these technologies do not necessitate the use of a DevOps paradigm, DevOps teams have found them a boon to their efforts. In terms of microservices adoption, among the general survey population 35% reported adopting a microservice architecture for some projects. But, of even more consequence, we found that 20% of survey takers have both had a dedicated DevOps team and had adopted microservices for some projects. Thus, well over half of respondents who have adopted microservices on certain projects also work in a DevOps development environment. When we look to containers, we see much the same pattern play out. Among the general population, 54% reported to have adopted container technology, while 32% of all respondents both had adopted containers and had a dedicated DevOps team. Again, over half of the users of this technology work in a DevOps-enabled organization. Additionally, when we asked respondents in what environments they use containers, 49% reported to use containers in a DevOps environment.

SURVEY RESPONSES

Which of the four containers testing pillars is the most challenging for you?



Do you have an officially designated DevOps team in your organization?



Diving Deeper Into Automated Testing

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.

Web Dev dzone.com/webdev
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.

Twitter

 @RealGeneKim

 @bridgetkromhout

 @martinfowler

 @bitandbang

 @testertested

 @techgirl1908

 @jezhumble

 @editingemily

 @MaritvanDijk77

 @kriscorbus

Refcardz

Continuous Testing
In this Refcard, you'll learn about what continuous testing really involves, why continuous testing is crucial for effective Agile and DevOps teams, and what the five-level path to continuous testing looks like.

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

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

Books

Agile Testing: A Practical Guide for Testers and Agile Teams
Learn about Agile testing, the tester's role in a software testing team, seven key traits of successful Agile testing, and more.

Just Enough Software Test Automation
Learn how to design, implement, and manage software test automation.

Implementing Automated Software Testing
Review the components, capabilities, and limitations of automated software testing (AST), and work through the entire AST implementation process.

Podcasts

Test Talks
Dive into all things from the world of software test automation, from recent news and book reviews to inspirational test automation thought leadership.

The Testing Show
Learn about myriad aspects of software testing, like advanced Agile with DevOps, automation struggles, continuous delivery, and more.

AB Testing
Explore the modern testing world from the perspective of Agile, data, leadership, and more.

The Benefits of TDD

BY JUSTIN ALBANO
SOFTWARE ENGINEER AT CATALOGIC SOFTWARE

QUICK VIEW

01. Test-Driven Development (TDD) is a development technique that starts with writing tests and requires only enough code to make the tests pass.

02. TDD provides developers with instant feedback but can lead to a false sense of security if expectations are not tempered.

03. TDD also leads to simpler and more concise code with 100% test coverage.

04. Not many studies exist on the effectiveness of those using TDD (compared to projects not using TDD), but the most prominent studies agree that TDD leads to less defective code but requires more upfront development time.

Test-Driven Development (TDD) has been a mainstay of the software industry since 1999, but despite its wide acceptance, its effectiveness has been hotly contested. While the idea of starting with tests makes sense to some, others find the technique slow and cumbersome. In practice, TDD is an effective technique if applied prudently and correctly, but the devil is in the details. The first step in reaping the rewards of this long-standing development strategy is understanding what it is — and just as important — what it is not.

What is TDD?

At its heart, TDD can be broken down in five steps:

1. Write a concise, automated test
2. Run the test and watch it fail
3. Write just enough code to pass the test
4. Run the test and watch it pass
5. Refactor the code, ensuring that all tests continue to pass

Steps 1 through 4 are required, but step 5 is optional. In some cases, the code we write is so simple that no refactoring is necessary. If so, we simply skip the refactoring step. This process repeats, building up more code and tests as we progress.

In practice, TDD means writing an automated unit, such as a JUnit test, for a method that does not exist and running the test in an editor, such as Eclipse or IDEA. When the test fails — because the method does not exist — and we see the unpleasant red bar, we then write *just enough* code to get the test pass. We rerun the test and enjoy the accomplishment of seeing the green bar signaling that our tests have passed. We continue this process until we have enough code to perform the desired task (i.e. we meet the desired specification).

How Can TDD Be Beneficial?

This process may seem primitive, but there are important benefits that come from writing code in this disciplined manner.

The first is that we have instant feedback. One of the most fundamental questions when writing code is: *Does my code work?* When writing code first, there is no real means of knowing whether the code we wrote is correct. Instead, we simply run the code and attempt to spot check it. With TDD, if we write tests that effectively represent the specification for our design, we instantly see a binary red (fail) or green (pass) signal that lets us know whether our code is fulfilling its intended purpose. What's more, we can refactor this code and know that we have not broken any existing functionality so long as our tests continue to pass.

The second benefit is that our code is simpler and more concise. Many times, we can write code that does much more than what is necessary. With TDD, unless there is a test that corresponds to a specification, we do not have to write code for it. This means that unless a use case or edge case is important enough to warrant a test case, it is not

important enough to warrant code. This often produces minimal code that strictly meets our specifications, and nothing more.

A third byproduct of TDD is that the test coverage is often much higher for code developed with TDD than through traditional methods. When starting with TDD, code is only written in order to pass our test cases, which leads to code that is comprehensively covered by the tests. When using traditional methods, we often attempt to backfill our code with unit tests, hoping to eventually reach 100% coverage. Due to the complexity of the code and our human nature (i.e. "this piece of code is so simple that I don't need to bother writing a test for it"), we usually end up short of that goal. Unless we start with tests, we usually do not end up with them — or at least not enough of them.

How Can TDD Be Detrimental?

While TDD can be a beneficial technique for almost all development situations, there are some important pitfalls that can adversely affect a software project.

One of the major disadvantages of TDD is that it often increases development time. Even the most ardent supporters of TDD will usually concede that code written using TDD takes longer on average to write than code developed using traditional methods. While this is inescapable, the development time only represents one side of the coin: the development time is a trade-off for code quality.

Due to the increased test coverage and reduced complexity, the code produced through TDD is often more sound and effective than code produced otherwise. This means that teams are trading time for greater quality. In most cases, this is a fair trade, but there may be times when this trade-off is not worth it. For example, when developing a prototype or proof of concept, time may be more valuable than robustness, and TDD may not be the best option.

A second issue with TDD is that it can create a false sense of security — or even arrogance — when not properly tempered. While 100% test coverage and green feedback (passing tests) create a sense of accomplishment and motivation, code created through TDD is not necessarily defect free. What if we wrote a test incorrectly? What if our tests are misaligned with our specification? What if we missed an edge case in our design and do not have a test case to represent it? These questions that are not specific to TDD, but sometimes, developers can be lulled into thinking that since they see green, the code is flawless. We can be draw into a false sense of security and let our guard down. It is important to remember that TDD is not magic and that our code is only as good as we make it — which means it will inevitably have bugs.

As with all techniques and tools, developers should be wise and prudent in applying TDD to their projects, ensuring that it is used only when the cost is worth the reward.

> "In practice, TDD is an effective technique if applied prudently and correctly, but the devil is in the details."

What Do the Numbers Show?

While costs and benefits are all essential, it is important to understand how TDD affects *real* projects. Unfortunately, many of the prominent studies done on the effectiveness of TDD are dated (usually from 2007 to 2010), but they do reveal some common results: TDD requires more time but usually leads to better code.

In *Realizing Quality Improvement Through Test Driven Development*, Microsoft and IBM found that its defect density (the number of defects per lines of code) decreased anywhere from 40-90% relative to similar, non-TDD projects, but the development time increased by 15-35%.

Likewise, in *A Comparative Case Study on the Impact of Test-Driven Development on Program Design and Test Coverage*, researchers found that while TDD did not have a substantial effect on the design of programs (in terms of metrics such as coupling and cohesion), it did have a noticeable, positive impact on the test coverage of the code produced.

Conclusion

While TDD is not a silver-bullet, it is generally results in well-tested, less defective code, at the cost of higher development time. While not perfect, even skeptics should consider using TDD, as the benefits — including quicker feedback, simpler code, and higher test coverage — will likely outweigh the costs.

JUSTIN ALBANO is a Software Engineer at Catalogic Software, Inc. responsible for building distributed catalog, backup, and recovery solutions for Fortune 50 clients, focusing on Spring-based REST API and MongoDB development. When not working or writing, he can be found skating, playing or watching hockey, drawing, or reading. LinkedIn Twitter

PAINLESS
CONTINUOUS TESTING

The way we build, test and deliver software has fundamentally changed. Continuous Testing is still a challenge for most companies. Only agile, flexible and efficient New Software Organizations deliver great digital experiences to customers in the shortest time. Just imagine you have:

NO STRINGS ATTACHED

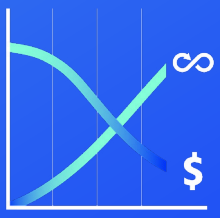
Use your existing processes, containers, and frameworks. **No vendor lock-in**

NO RIGID SOLUTIONS

Unlimited users and unlimited concurrency on thousands of real devices and browsers.

NO WORK IN SILOS

Native support for any DevOps environment: Jenkins, Gradle, JIRA, TeamCity, Travis, or integrate via REST API.



NO WASTE OF RESOURCES

Save up to 60% on your development budget and reduce testing time from months to hours.

THE TOOL OF CHOICE FOR THE NEW SOFTWARE ORGANIZATION.

bitbar.

bitbar.com

Containers Multiply Test Automation Efficiency

One of the most profound changes in the past few years in software development is the transition from manually configured development, testing, and deployment environments to code-configured environments using containers. Coded configurations can be stored in the version control, next to the actual application code.

This significantly boosts developer productivity since software developers, test automation engineers, build engineers, and operations engineers are relieved from manual tasks. The ultimate value comes from the time and resources saved from setting up exactly the same environment over and over again. With programmatically defined environments there is no troubleshooting of differently configured environments.

Benefits of Containers in Test Automation Environments

There are multiple benefits of moving your mobile and web testing frameworks to containerized environments.

MANAGEABILITY

A container-based architecture helps you manage changes in the test environment more efficiently. Any change can be rolled out to any and all environments, from development to production, in a controlled and repeatable manner. Typically, each step can also be automated to bring additional benefits.

CUSTOMIZABILITY

With a container-based architecture, you get shell-level access to the virtualized Linux/Mac server where you are executing your tests, and everything is described in a simple and easily readable Dockerfile. Modifying Dockerfiles for various purposes is straightforward and can be done in isolation of any shared environment.

PERFORMANCE AND SCALABILITY

Using containers for your mobile test environments makes it easy to move your test execution to a cloud service supporting containers (such as Bitbar). No matter how customized and complex your test container is, running it in the cloud allows you to execute and scale in parallel.

Additionally, as everything is executed in the cloud (instead of remotely running over the Internet), the container-based execution is multiple times faster compared to traditional remote Webdriver sessions. Docker-based test executions also store results in the cloud making it great for sharing results between teams.

Summary

The simple interface of modern container technologies, combined with a disciplined process for managing and versioning container images across the DevOps lifecycle, provides a fantastic opportunity to improve the repeatability, efficiency, and scale of your automated testing efforts with minimum changes to your existing workflows.

Test environments utilizing containers enable you to move from manually managing local environments, to running tests on your dynamically defined environments at scale, in cloud services. Thus, freeing up valuable developer time from configuration, environment customization, and ad hoc troubleshooting to more value-adding development tasks such as creating new features for your applications.

WRITTEN BY JOUKO KAASILA
CO-FOUNDER AND CHIEF OPERATING OFFICER, BITBAR

PARTNER SPOTLIGHT

Bitbar

The most flexible cloud-based web and mobile app testing platform. Designed for the new software organization with a DevOps mindset.

Category Automated and manual testing cloud, AI-based codeless automation

New Release Continuous Deployment, SaaS or on-premise

Open Source? Support for any open source framework

Case Study

Mozilla's Firefox browser testing teams are deriving significant benefit from Bitbar Cloud's support for containers. Their test framework utilizes an open source web driver based on the Marionette framework and they are able to wrap the whole framework, including all dependencies and test scripts, into a Dockerfile that is passed to Bitbar.

This way Mozilla can enjoy the efficiency-boosting scalability, test parallelism, load balancing, and centralized test result dashboards without any additional work on their side. The cloud environment is always identical to Mozilla's local environment which reduces significantly unnecessary debugging and saves developer time for more value-adding tasks.

Strengths

- **Full flexibility:** Use your existing processes, containers, and frameworks. No vendor lock-in
- **Unparalleled scalability**: Unlimited users and unlimited concurrency on thousands of real devices and browsers
- **Born Agile:** Native support for any DevOps environment: Jenkins, Gradle, JIRA, TeamCity, Travis, or integrate via REST API
- **Forward-thinking technology:** With Cloud-Side Execution, enjoy zero configuration, as well as faster and more stable tests
- **Industry-first:** Support for Dockerfile-defined test environments

Notable Customers

- IBM
- Deutsche Telekom
- CBS Interactive
- Square
- PayPal

Website bitbar.com

Twitter @bitbar

Blog bitbar.com/blog

Saving Time, Money, and Headaches with Automated Mobile UI Testing

QUICK VIEW

01. Manual testing is still a crucial part of the SDLC due to unexpected issues that could occur such as unstable Internet connections.

02. Automating mobile UI testing allows organizations to only resort to manual testing for extreme edge cases.

03. The Arrange Act Assert Pattern is one of the best ways to organize your tests.

BY BRANDON MINNICK
MOBILE APP DEVELOPER, MICROSOFT

Testing mobile applications is time-consuming. It requires us to test each edge-case, new feature, and existing feature, then repeat those tests dozens of times on multiple devices across multiple mobile operating systems.

To accomplish this, many teams manually follow step-by-step documentation specifying exactly how to interact with the UI to ensure everything is working nominally. But as more mobile devices and new versions of iOS and Android are released, the time required for testing grows exponentially.

Automated mobile UI testing reduces this burden, enabling us to run automated repeatable tests on any mobile device. The automated tests are written in code to mimics a user's interactions within an app; anything a user can do, such as entering text, swiping, or tapping buttons, can be performed via UI test.

There are many different frameworks available to perform UI testing. Some of the popular frameworks are XCUITest, Espresso, Xamarin.UITest, and Appium. Each framework appeals to different developer groups, e.g. Swift developers, Java Developers, and C# developers; and all of them allow us to write robust UI tests.

These automated tests can (and should!) run before pushing a new release to our users, giving us confidence that both new and existing features work as expected and ensuring we didn't accidentally inject new bugs into our code. But automated testing can't test every scenario.

Unexpected scenarios outside of the control of our app, like an unstable Internet connection or a phone call interruption, will still need to be tested manually because UI tests cannot modify the mobile device settings or hardware. That being said, automated UI tests still allow us to automate up to 80-90% of our test suite, allowing manual testing to focus on these extreme edge tests. This upscales the manual tests from tapping buttons for a regression test to stress-testing the app using creative scenarios, and also shortens the overall time required for testing.

Writing UI Tests

The most common UI test architecture is the Page Object Architecture. This architecture creates a class for each UI page (e.g an iOS UIViewController or an Android Activity/Fragment), and the class contains a function for each action that a user can perform. For example, if a page contains a submit button, its associated page-class will contain a public function called `TapSubmitButton`. This page-class then allows us to reuse the same functions for our test scenarios, avoiding the need to rewrite it for each test.

Each function should also trigger a screenshot that captures an image of the device's screen. When a test fails, having a series of screenshots helps us identify the cause of the failure, because when the test fails it can be difficult to intuit exactly what went wrong. But screenshots give us the ability to review the UI as it appeared during the test to better understand what failed and why.

To identify the UI control used by each function, it's recommended to set unique identifiers for each UI control.

iOS provides a property for each UI control for this called `accessibilityIdentifier`. Here is how iOS defines it:

An identifier can be used to uniquely identify an element in the scripts we write using the UI Automation interfaces. Using an identifier allows us to avoid inappropriately setting or accessing an element's accessibility label.

Android doesn't provide a specific property for UI testing, but `contentDescription` can be used. It's important to note, however, that `contentDescription` is also used by screen readers to identify UI controls for vision-impaired users. Here is how Android defines it:

A content description briefly describes the view and is primarily used for accessibility support to determine how a view should be presented to the user. In the case of a view with no textual representation, such as `ImageButton`, a useful content description explains what the view does. For example, an image button with a phone icon that is used to place a call may use "Call" as its content description. An image of a floppy disk that is used to save a file may use "Save."

Once the page-classes have been completed we can begin writing the tests.

Each UI test should follow a user journey. For example, if a user can create a new account, we should have a UI test that successfully creates a new account. We should also test the failure paths where the user submits incomplete or invalid data and the test verifies that our app gracefully handles those failure scenarios.

The Arrange Act Assert Pattern is recommend for each test. For this pattern, we first *arrange* values to be used in the test. We then *act* by executing the UI test. Then once the testing steps have completed, we *assert* that the onscreen behavior is correct and data was changed as expected.

Each test should also start with a fresh installation of the app. This ensures that all data from the previous test has been cleared and that the test always begins with the app in a known configuration. Most UI testing frameworks uninstall and re-install the app before each test by default, but it is important to double-check that our framework is following this best practice.

Now that the code for the UI tests have been written, we'll need to test our app on real devices. There are thousands of different mobile devices on the market today each with a unique combination of screen size, pixel density, CPU chipset, and operating system version; we need to ensure our app runs properly on each permutation. To avoid purchasing and maintaining dozens of devices, we can leverage existing device labs like App Center and AWS Device Farm. These platforms give us the ability to run our UI tests on thousands of real iOS and Android devices to ensure our app functions properly, helping to catch bugs before shipping the app to our users.

Integrating Automated UI Tests into Existing Continuous Integration Workflows

Like unit tests, it's important to run UI tests each time the code has been updated. Here are some recommendations to integrate UI testing into our existing continuous integration workflow.

The easiest way to integrate UI testing into our team's existing workflow is to run the UI test suite every night by scheduling a continuous integration service to build and run the full test suite on the latest version of the app each night. It won't stop bugs from being added to the code base, however, it will catch the bugs before the next day.

To avoid bugs from making it into the code base, it is recommended to run the UI tests each time a pull request is opened. This will allow the reviewer to ensure all tests are passing before approving the pull request.

Finally, to help developers catch bugs even more quickly, we can configure our CI tool to run UI tests on each code commit. To avoid utilizing too many resources, a subset of UI tests can be selected in lieu of running the entire test suite by selecting specific test fixtures.

Conclusion

Automated mobile UI testing allows us to iterate quickly and be confident that the app will work as expected across the vast array of mobile devices. By integrating the testing into our existing continuous integration workflows, tests will execute automatically and the results will be ready for review before we push any updates to our users.

It also reduces the time and cost of manually testing. Manual testers are free to perform more exploratory tests and stress tests to push the extreme edge cases. And services like App Center and AWS Device Farm reduces the need to purchase, upgrade, and maintain mobile devices in our own test labs.

It's never too late to start UI testing your app. Get started today, and happy testing!

BRANDON MINNICK is a mobile app developer, Florida Gator, and world traveler who has been creating iOS, Android, and UWP apps since joining Xamarin in 2015. His journey to becoming a mobile app developer started with a Computer Hardware Engineering degree from the University of Florida, where he fell in love with writing code. His career continued to evolve as he was designing end-to-end satellite network architectures for Harris Corporation, and it was during this period that his passions re-aligned and he completed the Professional MBA Program at UF. As he saw mobile becoming the wave of the future, he moved across the country to join Xamarin and be a part of the mobile revolution. LinkedIn Twitter

Signs Your Test Automation is Failing You and What to Do About It

In the world of automated testing, failed tests happen all the time. One small thing changes and suddenly you're drowning in red. And when you're testing continuously, small issues can easily grow into huge ones — which is why you need visibility.

When something breaks, you need to know why. You need visibility and fast feedback so you can fix what's not working.

At Perfecto, we've found that 80 percent of issues have a pattern. That means that teams can easily overcome the majority of their issues just by identifying the root cause behind them.

And that's where smart analytics are key.

Through Perfecto's smart test reporting and analytics, we've discovered the top four most common test automation issues teams encounter.

Leading the pack, the most common sign your test automation is failing is if you encounter issues with test scripts or test frameworks. In fact, these account for 40% of all issues that DevOps teams face.

Why?

Test script and test framework issues stem from problems with skillsets, culture and processes, and an overall lack of communication between testers and developers.

Particularly problematic to DevOps teams are objects. With object identifiers, too often teams lack the knowledge to define the right object being used. This is especially true when developers design pages featuring multiple objects with the same ID. Two similar objects on the same script are sure to cause issues in automation.

Learn more about the top issues in test automation and how to overcome them. Read the full article: perfecto.io/test-automation-failing.

WRITTEN BY ERAN KINSBRUNER
CHIEF EVANGELIST, PERFECTO

PARTNER SPOTLIGHT

Perfecto

Accelerate automated testing with Perfecto's continuous testing platform. With Perfecto, you'll test confidently and continuously at DevOps speed.

Category Continuous Testing Platform

New Release Cadence Continuous **Open Source?** No

Case Study

Before working with Perfecto, Anthem performed all testing manually. Release cycles varied from 1-6 months, making it extremely difficult to keep up with market demands. To compensate, they hired additional staff, which was costly and unscalable.

Perfecto provided Anthem with an easy to use, scalable, and secure testing solution that met all industry standards. Perfecto helped the client add automation throughout the entire release cycle, including build validation tests, regression tests, and more.

Now Anthem has cut their release cycle down to 2 weeks. This has allowed them to meet market demands and focus on producing features while spending less time resolving bugs. Better yet, adding automation has allowed them to triple their coverage.

Strengths

- Unified solution provides full end-to-end support for continuous testing.
- AI-backed smart insights help you cut through the noise to focus on what matters.
- We're an enterprise-grade platform for scalable and secure testing.
- Perfecto is the only cloud-based platform for mobile, web, AND IoT testing.
- Test on real mobile devices and leading web browsers in an always-on, always-available lab.

Notable Customers

- Chase Bank
- Wells Fargo
- AT&T
- Allstate
- IBM
- Ford
- Comcast

Website perfecto.io

Twitter @perfectomobile

Blog blog.perfecto.io

Integrating Automated Testing into Mainframe Workflows

BY SAM KNUTSON
VICE PRESIDENT OF PRODUCT MANAGEMENT AT COMPUWARE

QUICK VIEW

01. Manual testing steals precious time from value-added innovation that enables companies to compete effectively in the digital age.

02. Technology exists to automate the creation and execution of unit, functional, integration, and regression tests for mainframe code.

03. Companies that have automated testing of mainframe code have improved code quality while significantly increasing developer productivity.

If you work for a large enterprise, it's more than likely a few mainframes serve as the mission-critical back-end for your business. Work for a software vendor? It's likely some of your customers — large banks, insurance companies, airlines, retailers, etc. — rely on mainframes for the same reason: it's the most reliable, available, and securable platform for managing competitively differentiating applications and data.

The first thing you need to understand about the mainframe is just how advantageous and irreplaceable it is, for the reasons above. The second thing you need to understand is that the mainframe is an extremely modern platform — but, unfortunately, significant aspects of its culture, processes, and tools aren't modern at all. Testing, in particular, has become a principle area of concern.

One company recently told us their developers spend up to 60% of their time testing mainframe code. While mainframe programs are large and complex — and the more complex a program is, the more time you spend testing it — this is no excuse for spending the majority of your time testing.

In fact, the main reason this protracted process steals time from value-added innovation is not necessarily program complexity. Rather, it's because mainframe application testing has traditionally relied upon manual processes, such as writing programs and scripts, eating up precious developer and quality assurance (QA) time. At least, this has been the case until recently.

Technology now exists to automate the creation and execution of unit, functional, integration, and regression tests for mainframe code. Just like distributed tools, this technology allows developers and testers to validate code changes immediately as well as continuously manage code quality and support continuous integration/continuous delivery (CI/CD) pipelines through REST APIs and integrations with cross-platform DevOps tools.

Companies using this automation technology are substantially reducing the time their developers and testers spend on testing. In turn, they're accelerating the rate and frequency at which they deliver innovation. Overall, they're increasing application quality along with development velocity and efficiency.

Large UK Bank

One of the UK's largest banks has been driving a cross-platform, mainframe-inclusive DevOps strategy to digitally transform its banking services and align closer with the expectations of customers. Core to enabling this modernization was ensuring the bank's mainframe — which powers competitively differentiating products like an automated debit-card issuing service — was able to integrate into that strategy and be managed with the same Agile and DevOps processes as other decentralized systems.

The bank realized automating the creation and execution of unit tests to find low-level bugs would be essential to increasing code quality, development efficiency, and delivery velocity on the mainframe. Within 12 months of implementing automated testing, the bank had increased developer productivity by 400 percent.

An integration between the automated testing technology and the bank's CI tool allows for the automation of repeatable tasks. This enabled the bank to adopt a "shift left" approach, where developers are able to unit test as they code, reducing the testing time of a large application from two weeks to five minutes.

Finally, the bank reduced the time developers spend writing and checking code from about 40 hours to just seven hours per sprint through another integration with a popular open-source tool that highlights syntax errors as developers write code as well as monitors technical debt to help developers prevent it from increasing.

Standard Bank

Standard Bank, one of the largest banks in South Africa, has also seen great success with automated testing on the mainframe. Rather than spearheading its efforts with a formal testing tool, it focused instead on implementing a modern source code management (SCM) tool that helps support an automated testing framework.

Prior to implementing its current modern SCM tool, the mainframe team was outsourcing testing. Upon committing code back into its (former) SCM repository, test coverage, and code quality analysis were constantly inadequate, ultimately allowing risk to leak into the delivery pipeline.

Additionally, deployment times were prolonged up to six days because mainframe code deployments could only be made into one environment per day, with each instance requiring a separate testing environment.

By adopting a modern SCM as well as an IDE that productized development, testing, and delivery best practices and integrated with their debugging tool, Standard Bank automated the manual steps in their pipeline to ultimately ease the process of compiling, promoting, and deploying programs for testing.

Like the bank in the UK, they've also integrated this technology with a popular open source solution for on-the-fly QA that includes a dashboard for tracking defects, complexity, and technical debt.

What Your Tool Should Include

As forward-thinking mainframe shops like these two banks have proven, automated testing on the mainframe is critical. To accomplish it, you need the right framework of tools and processes within a DevOps culture of continuous improvement. You can't buy processes or culture, but you can adopt the right technology to support them. So, if you're going to implement automated testing on the mainframe, which you should, it needs to include the following capabilities.

AUTOMATED CREATION AND EXECUTION

It may seem obvious that the technology you implement should automatically create and execute tests, but not every "automated testing" tool can claim this. If you're still manually collecting and entering data, manually compiling and linking test cases — it's not truly automated testing.

What's more, automation should span unit, functional, integration and regression testing with support for QSAM, VSAM, Db2, IMS, and CICS on the mainframe.

You should be able to accomplish everything through one intuitive user interface and integrate into a CI/CD pipeline that automatically triggers tests.

DATA VIRTUALIZATION FOR UNIT TESTS

Virtualization, also known as stubbing, eliminates the need for access to "live" subprograms, files and subsystems, allowing you to focus a unit test on one specific program and run the test consistently by eliminating the need for external dependencies.

When you re-execute those tests, you don't have to concern yourself with tedious setup work, because you're reusing data you've already captured. Not so, if you have to manually collect and input this data each time.

Data virtualization should also extend to functional testing, allowing you to virtualize the program being tested without executing a test case against it, but rather the program stubs created. To do this, you really need a tool that enables test driven development, which essentially allows you to design test cases, before developing programs, based on program specifications.

TEST SUITES AND TEST CASES

Being able to organize test scenarios into logical test suites used to regression test an application when a change occurs is critical to ensuring your tests include a consistent set of test cases you want to run.

We saw the effectiveness of this in play at another company that had generated several automated unit tests before being hit with changes a week later. All of their testing had to be re-done, but they were fortunate in that their automated testing technology allowed them to simply alter a few data values and the data's input stub to quickly test the new cases. They said without automated testing on the mainframe, this one-day job would have taken one month.

Automate, Automate, Automate

Mainframe teams can't afford to rely on manual testing processes in our rapidly accelerating digital economy. As web, mobile and cloud innovation continue to proliferate while maintaining high reliance on mainframe applications and data, mainframe workloads are going to continue to grow — as mainframe experts continue to retire.

Businesses must ensure they have a strategy in place that enables the stewardship of the mainframe platform. Attracting next-gen programmers to the mainframe is critical, but automation is just as important.

Testing should be one of the focus areas where you implement automation technology. By doing so, you can dramatically transform the quality, velocity, and efficiency with which developers and testers carry out this critical task.

SAM KNUTSON is Vice President of Product Management at Compuware. His past positions include Vice President of Product Management for Mainframe Performance Solutions at CA Technologies, System z Team Leader for GEICO and Senior Developer at Landmark Systems. Sam has served on the board of directors for SHARE, the world's first and longest-running user computer group. LinkedIn Twitter

Kobiton
kobiton.com

A perfect mobile experience starts with quality.

Mobile Device Testing Starts Here.

01 Codeless Appium Scripting

02 Tame Device Chaos

03 Accelerate App Delivery

04 Deliver the Perfect CX

A Perfect Mobile Experience Begins With Quality

The approach to quality has evolved – from involving QA earlier in the release cycle ("shift-left"), placing more focus on the user experience, and the introduction of AI into the testing process, the demands on QA are higher than ever, as are the expectations. It's no longer about just finding bugs. Testers are becoming the new product managers in the organization and test professionals need new, innovative approaches to driving quality throughout the product release cycle.

Mobile app testing in particular has traditionally been fraught with challenges – from needing to test on the thousands of different device types used by customers to the dearth of robust cross-platform automation and testing frameworks. And while Appium is revolutionizing the mobile automation and testing landscape, its nascency poses significant challenges to testers. Fragile element identification, brittle test scripts, and the technical expertise needed continue to slow the pace of Appium test script creation, causing QA to unintentionally be the bottleneck in the release cycle.

Driving mobile app quality requires:

1. Introducing automation into your mobile dev and test cycle to overcome the QA bottleneck
2. Introducing autonomous testing to overcome the technical challenges inherent in mobile app testing
3. Recognizing that AI should be used to empower testers, not replace them.

It's an exciting time to be a quality engineer. As we move into a new era of applying AI into the testing space, it will be necessary for testers to sift through the jargon emanating from vendors, and leverage solutions that truly drive value into the testing continuum.

WRITTEN BY CHRIS DU TOIT
CMO, KOBITON

PARTNER SPOTLIGHT

Kobiton

Mobile Device Testing Starts Here

Category Mobile Device Testing

New Release Continuous, Saas Delivery

Open Source? No

Case Study

A leading IT security company aimed to improve their mobile app quality in response to declining app store ratings. The company observed a correlating drop in customer satisfaction, brand reputation and revenue. Their goal was to leverage a mobile testing cloud to optimize the manual and automated testing efforts of Dev and QA.

The solution comprised 24/7 automation runs on a hybrid deployment of dedicated private devices, internally hosted local devices and public devices hosted on the Kobiton cloud.

Over the course of 30 days they produced 3 years worth of testing across 25 iOS and Android devices. This comprehensive test and device coverage gave them the confidence necessary to release their app at unprecedented speed without compromising quality.

Strengths

- Less than 200 milliseconds of latency
- Flexible cloud or on-premise deployment options
- Codeless Appium Script Generation & Remediation
- ⅓ of the cost compared to leading competitors
- Fast, efficient & effective customer service

Notable Customers

- Q2ebanking
- GreenSky
- Guitar Center
- Wolters Kluwer
- Amerisource
- Office Depot

Website
kobiton.com

Twitter
@kobitonmobile

Blog
kobiton.com/blog

Plain English is the New Language of Testing

QUICK VIEW

01. Natural Language Processing (NLP) can reconnect you with end-users.

02. NLP has reached the stage where you can use plain English to write your tests.

03. If you can couple NLP with other forms of AI, you create a revolutionary intelligent test agent.

BY TAMAS CSER
FOUNDER AND CEO, FUNCTIONIZE

Summary

Imagine if you could use plain English to write all your automated tests. What would that mean for the software delivery lifecycle? Here, we explore how advances in natural language processing allow you to write functional automated tests in plain English. The result is everyone from product managers to business analysts can contribute to the SDLC process. By replacing test scripts with tests written in English, you reconnect the product and testing teams. Now, it is easy for product managers to write tests that match the real user journeys in your application. Furthermore, the gap between testing and the end customers is narrowed. Traditional test automation focuses on how the test works over what the test is trying to achieve. With NLP you are able to focus on what the test should do.

Background

One of the great problems faced by many companies is to avoid building software which is broken. This is the job of the product and QA teams. These teams need to work closely together to ensure all necessary testing is being done, and to keep the focus on the most important aspects of the system.

In the traditional software development lifecycle, Business Analysts from the product team collaborated with Quality Engineers to define a set of test plans. The BA gave a rough plan or user journey to the QE, which was refined into a detailed set of test steps. These test plans were used by manual testers to verify the product was working as expected. Other manual testers would then try to "break" the product through exploratory testing (trying to find broken paths through the system that users might stumble upon).

The Problem With Test Automation

All this changed about a decade ago when test automation hit the mainstream. Nowadays, the aim is that all the routine functional testing of your UI should be done using automated tests. The BA still gives a user journey or test outline to a QE. But now the QE refines this into a detailed test plan and then works to convert that plan into a test script. This script is probably written in Selenese (the language of Selenium) and one or more other scripting languages (e.g. JavaScript, Python, etc.).

Writing these scripts is notoriously hard, especially if they have to be converted to work cross-browser. Usually, the process is incremental and slow. The QE writes a few steps, checks that they work, debugs them and then writes some more. During this process, the QE gets distracted from the intended functionality and concentrates instead on creating a working script. But because it is a script, the BA has no way to verify that it accurately reflects what was in the plan.

A Different Approach

The problem is that automated test scripts are written in a language designed for computers to understand. Computers are dumb, so we humans have always had to translate our language into simplified programming languages the computer can understand. But there is another way to do things. Natural language processing is a specialized field of artificial intelligence. In NLP, the aim is to teach a computer to understand human language. So, you don't need to learn to speak to the computer, the computer learns how to understand you. NLP has reached the stage where you can use plain English to write your tests.

HOW DOES THIS WORK?

The starting point should be a test plan written in plain English. The easiest is to have a structured plan such as the ones produced by test management software. Here, each line of the script is a single action to be taken on the UI and you can pass in data and state the expected outcomes. Many UI tests share common elements. So, you can define some

keywords like VERIFY, SCROLL DOWN, CLICK, and ENTER. However, as we will see later, NLP has actually already gotten more advanced than this and will understand even unstructured text.

CREATING AN INTELLIGENT TEST AGENT

The next step is to create an intelligent test agent that can take this plan and compare it against your UI. The aim is to use the test plan to train the intelligent agent to understand how your system functions. This intelligent agent will be based on forms of machine learning, but it also requires other AI techniques. These include computer vision (recognizing icons, buttons, etc.), OCR (reading buttons that aren't text-based), and template recognition (understanding the relationship between UI elements).

You need to train the intelligent test agent to understand your whole system. So, you need to process as many test plans as possible. Effectively, during this training stage, these plans act as instruction manuals to your system. This means it pays to ensure your plans cover all realistic user journeys. Once you have trained the model you have a set of tests that can be run automatically.

The intelligent test agent then takes these test plans and formally converts them into tests. Once a test has run, the results can be displayed graphically. A screenshot for each step of the test is shown, color-coded to indicate any errors. This can be done using computer vision to compare screenshots from the current test with the results of the training. Clearly, this implies the need for significant storage and, coupled with the need for AI, means this sort of system really has to be cloud-based.

The Benefits NLP Brings to Your Users

So, what does the use of natural language processing mean for your end users? Why is it worth going through all this trouble?

CLOSING THE DISCONNECT BETWEEN THE PRODUCT AND QUALITY TEAMS

As we said above, traditionally, there was always a close connection between the quality and product teams. But the growth of test automation had consigned this to history. Nowadays, there is a real disconnect between quality engineers and business analysts. Once upon a time, quality engineers were temperamentally closer to the product team than the development team. After all, their job was ensuring the product works. But modern test automation engineers are much more likely to be developers that have moved across to testing.

NLP-based testing offers you a way to reconnect. It provides a way to bridge the gap in understanding between QEs and BAs. Once again, the product team can be sure that the test plans accurately representing all the user journeys through your system. They can even go into the system and visually see that the tests were doing the right thing.

EMPOWERING YOUR WHOLE TEAM

One of the key implications is that NLP-based testing empowers every single person in your team to be involved in the software development lifecycle process. Now, anyone from product managers and business analysts, to UX designers and even C-suite executives can understand your tests. More importantly, all these people can contribute to new tests. Until now, the people that actually designed your product had to rely on a chain of others to ensure that the product being released was working. Now, they can actually help define the tests.

This is important because these people are the ones that best understand the end users. They are the ones that came up with the user stories and customer journeys. They are the UX experts that worked with focus groups to come up with a perfectly-designed product. So, why not let them get involved in the testing process that checks the system is delivering what was promised? That way you reconnect the people that understand your users with the users themselves.

RECONNECTING WITH YOUR CUSTOMERS AND USERS

Traditional test automation approaches like Selenium brought about an absurd situation where what matters is tests that work, rather than tests that are correct. Even worse, Selenium tests are so brittle that they break any time you change your code. In the worst cases, a simple CSS change can break all your tests. One of the upshots of this has been the birth of a new breed of developer — the developer in test. This solves one problem (poorly-written test scripts), but it totally ignores the elephant in the room: testing should be about connecting with your customers and users, but test automation achieves the opposite.

With NLP-based testing, you can start to reconnect with your customers and users. Almost all test teams still have manual testers whose job is to understand how the user might interact with the system. This may just be second-guessing the inventive ways in which they will try to break your system, but to do that requires a deep understanding of the user.

"A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools." - Douglas Adams, Mostly Harmless.

In many cases, these QEs have remained as manual testers because they lack the skills to become test automation engineers. Now, they are once again empowered to use their customer-focused test skills and create better automated tests.

Conclusions

NLP-based testing has the potential to transform automated testing. If you can couple NLP with other forms of AI, you create a revolutionary intelligent test agent. This can reconnect your product and testing teams, removing the obfuscation introduced by test scripts. Most significantly, it can reconnect you with your customers. In turn, this means you will be able to produce a product that works reliably and actually delivers what your users want.

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.
LinkedIn Twitter

Force Multiplier: \fo(e)rs \məl-tə-plī(ə)r n:
A tool that dramatically amplifies
your effectiveness.

73% of customer-facing apps are **highly dependent** on the mainframe. Yet **2 out of 3** lost mainframe positions **remain unfilled**, putting quality, velocity and efficiency at risk.

You need **Compuware Topaz** as your Force Multiplier to:

- Build and deploy with agility
- Understand complex applications and data
- Automate unit, functional and integration testing

Learn more at **compuware.com/force-multiplier**

compuware.com | @compuware | linkedin.com/company/compuware

Compuware | The Mainframe Software Partner For The Next 50 Years

UK Bank Proves You Can Deliver DevOps on the Mainframe with Compuware Topaz

A large UK bank evaluated the mainframe software market and quickly identified Compuware Topaz as an option to help them modernize mainframe development and delivery with DevOps.

The key capabilities that attracted the bank to the Topaz suite included:

- **Topaz for Total Test**: enables the bank to fully automate the process of creating and executing unit tests. Tests can be stored in GitHub for reuse, enabling developers to automatically run "Java-like" unit tests to find low-level bugs.
- **Topaz for Program Analysis**: creates an instant, static visual summary of a program, or a dynamic visualization that provides a clear and accurate understanding of a program's runtime behavior.
- **Topaz Workbench**: provides a modern Eclipse-based IDE through which developers can access Compuware tools alongside tools like Jenkins and SonarSource SonarQube—in a single environment.

Taking the Pain Out of Testing and Development

Topaz is enabling the bank to set up automated unit tests and "shift left" with testing so that it takes place in parallel to new code development, rather than at the end of a sprint. This has allowed the bank to automate system testing, reducing execution time from two weeks to just five minutes by removing the manual effort from the process with repeatable tasks in Jenkins.

Additional Results

The bank has seen significant developer productivity increases. Over a 12-month period, the adoption of Topaz has increased flow by 400 percent on the mainframe, virtually eliminating defect leakage, saving on rework and test timescales.

The bank has also reduced the time it spends writing and checking code during each sprint from an average of 40 hours to just seven hours. Additional code quality checks have been incorporated, with SonarSource SonarLint highlighting syntax errors or necessary improvements while developers are working on code. Plus, the integration with SonarQube confirms compliance with the bank's coding standards and monitors technical debt to ensure it does not increase.

"We're innovating faster on the mainframe than we are in our digital channels, which is just incredible." – Engineering Lead

Additionally, Topaz for Program Analysis has provided visibility into application structure that has been built over decades enabling new developers to understand the implications of changes they are making.

"Topaz has allowed us to really see what's going on inside our mainframe code. A typical project used to take anywhere from six to nine months, but we're now hitting a stride where we can deliver in a third of that time. We're innovating faster on the mainframe than we are in our digital channels, which is just incredible," said the bank's engineering lead.

WRITTEN BY STEEN BRAHE
PRODUCT MANAGER, COMPUWARE

How to Test What Matters

QUICK VIEW

01. Test coverage tools can only measure quantity, not quality

02. Making tests a requirement can create the wrong incentives

03. Designing a program to be testable improves its overall quality

BY TIM LEBEL
SENIOR SDE, SMARTSHEET

It is universally understood that having good automated tests is a requirement for any serious enterprise software in the modern era. Less universally understood is what exactly constitutes "good automated testing." Managers of software engineers will typically dictate a vague requirement that the code should "have tests," but determining whether these tests are sufficient is where things start to get fuzzy. Contemporary Agile management trends dictate that these things must be quantified — how is the team to know whether they have fulfilled the requirement of having adequate tests, without some precise measurement of test quality?

This leads us to the dubious notion of "coverage," in which some external program is employed to analyze the source code and its tests in order to make some determination about which lines of code are tested or not. This is intended both to communicate something about the quality of the tests for a particular bit of code as well as to provide an incentive for the developer to write more tests and watch the coverage number go up. Unfortunately, the technology does not exist for coverage tools to tell us how *good* these tests are, therefore the incentive they create is for the developer to write scores of meaningless tests to produce the desired metric, rather than those that would actually improve code quality.

Most managers are savvy enough to realize that these coverage numbers are not an adequate measure of test quality. However, managers also struggle with the realization that there *cannot be* a pure, accurate quantification of test quality, except perhaps the error rate of the actual code when it eventually ships. Therefore, they are typically resigned to simply requiring that developers write *any tests at all*, resulting in awkward statements like "I estimate 4 story points for completing the feature, and 2 for writing the tests." This sort of thinking misses the point.

Tests are not some product requirement to be imposed externally, nor is their purpose to "cover" all of the code in the eyes of some external system. Tests are a tool the developer employs in order to write correct code. In order to test what matters, we must shift our thinking from testing as a requirement to testing as a natural part of the programming process. What follows are some techniques for writing meaningful test cases while avoiding wasted effort.

Use Real-World Data

If the system you are testing accepts user data or data from some system beyond your control, it is essential, if you can, to obtain

some real examples of that data and use them in your testing. You can save these data to files and use them as input to your tests to simulate data coming from the external system. Once the test harness is set up to do this, you can add further examples as your code inevitably misbehaves or crashes in production when it receives external data you weren't expecting.

Don't Test Your Dependencies

When tests are thought of as a requirement, it is typical to write them for even the smallest bit of logic. However, most modern enterprise applications do not consist of complex, original algorithms. Most of the code we write in these situations is simple glue code that is stringing together calls to various libraries and external APIs. It is important to keep in mind that these systems likely already have their own tests — at least we should hope they do, if we are building our application on top of them!

Consider this real-world example: a test for a React component that injects some data into its state model, then checks whether that data is being displayed. This test is probably unnecessary; React likely already contains tests to confirm that it displays data from a component's state model in its template — that is its entire function. Often the response is that it "doesn't hurt" to include the perfunctory test. I would argue that it does: the future developer reading these tests will have to sift through this useless cruft to identify what really matters.

Reduce Logic to Pure Functions

A pure function is a function which does not modify any internal or external state and will always have the same output for a given input. Writing pure functions in a modern application is hard — it is much more intuitive to mix calls to external systems like databases with our business logic, because that is how we visualize the problem in our heads — each step happening sequentially in a single context. Class-based languages like Java attempt to remediate this by forcing developers to think in terms of "objects" instead of splatting out their logic in a long string of statements. This attempt backfires because objects by their very nature can contain state, and their instance methods cannot be assumed to be pure functions.

Instead of designing code into an object model, think of which parts can be modularized into pure functions. In doing this you may realize that the pure functions actually encapsulate your business logic, the "special sauce" that your application is actually providing. What is left can be considered "glue code," simple calls to other systems or libraries which are likely already covered by tests you have the good fortune of not having to write. In thinking this way you have not only separated your code into what is easy to test and what isn't — you've also separated it into what's *worth* testing and what isn't.

Be Truly Test-Driven

The term "test-driven development" has been so overused that its meaning has been somewhat lost. This approach is so prescriptive and severe that many developers dismiss it as being impractical; it's much easier to just spin up MySQL, memcached, and whatever else in Docker containers, hand-test everything, then write a few integration tests after the fact to make your coverage numbers. However, once you reduce your business logic to pure functions, the power of this approach becomes clearer

When you've already conceived of a piece of business logic as having only inputs and outputs, you can set up a test case that provides an ostensibly correct set of inputs *before* your implementation is capable of producing the correct outputs. Then, you can write your implementation and watch the failing test change to a passing one! If you think coverage numbers are satisfying, you ought to try this — it'll knock your socks off.

Be Honest With Yourself

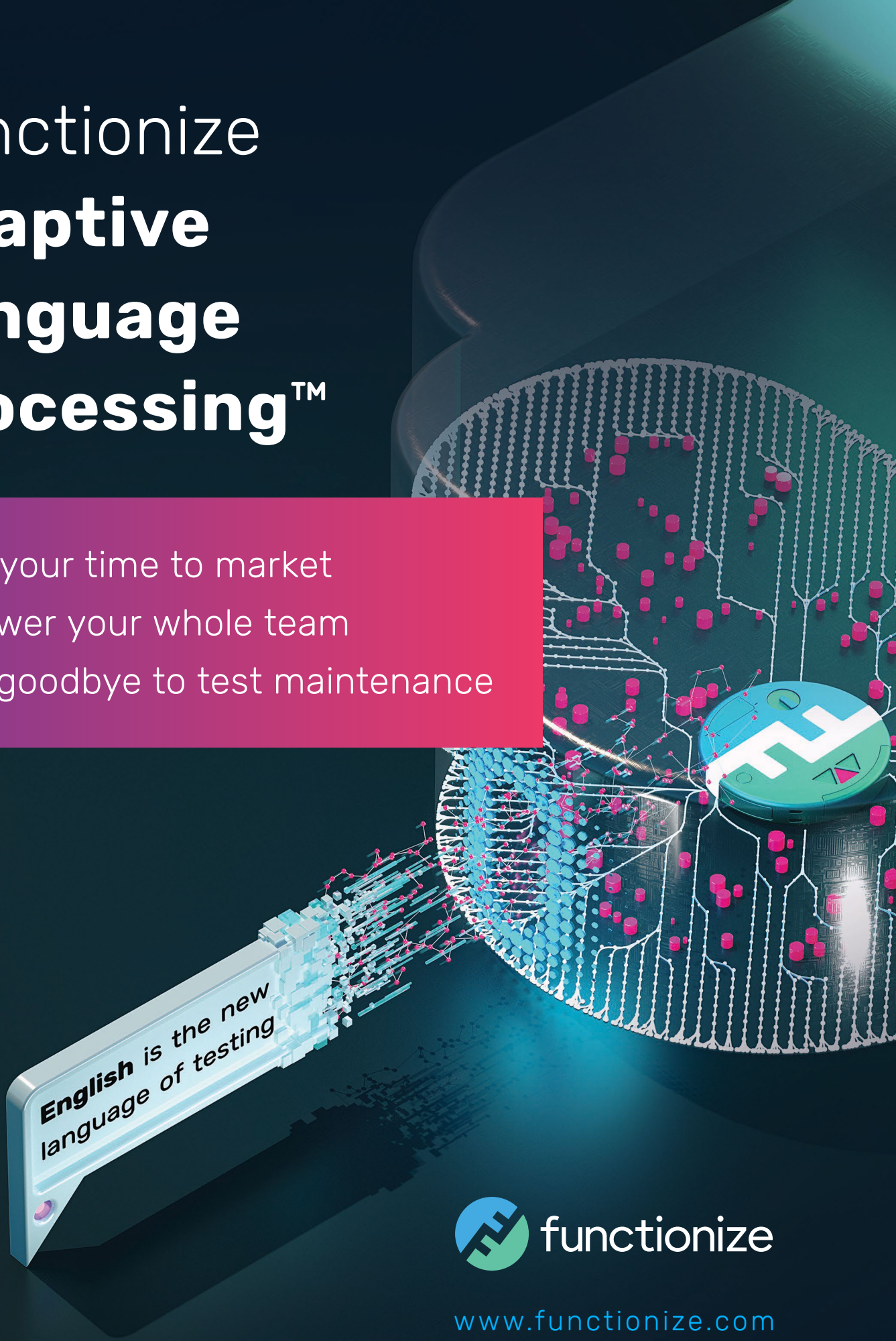
Testing application code properly can be disheartening. Once you separate glue code from business logic, and stop testing the former, you may arrive at the humbling realization that your genius application code isn't actually doing as much work as you thought it was. Use this as a motivating factor rather than a demoralizing one. Going forward, you can actually test smarter, not more, and the saved time will result in you being able to produce far more original programming logic than you would have otherwise.

As a software engineer, it is your responsibility to advocate for best practices and correctness — not your manager's. Hours spent writing tests may not be reflected on the agile board, and that's okay. Tests are a necessary part of implementing any given feature, as long as they're the kind that actually make a difference.

TIM LEBEL is a Senior Software Developer and Engineer, working on development environments, release pipelines, and everything in between at Smartsheet, a work planning and automation tool. LinkedIn Twitter

Functionize
Adaptive Language Processing™

- Slash your time to market
- Empower your whole team
- Wave goodbye to test maintenance



functionize

www.functionize.com

An Intelligent Use of AI in Test Automation

Suddenly, every test automation company is claiming their product has AI. How many of these so-called AI systems are really using AI? Using ML or computer vision doesn't necessarily make your system artificially intelligent. Certainly, it's not an intelligent use of AI.

What do we mean by the intelligent use of AI? Well, there are a few important factors:

1. Don't just repackage an existing model unless you are certain how it will perform in all cases.
2. Be sure that the problem is really suitable for AI. If you don't have a large dataset, ML will never work well. Equally, if you are looking for really obvious features, AI may just be unnecessary.
3. Make sure the use of AI will give better outcomes than an existing alternative. Sometimes, old-fashioned statistical approaches for clustering and feature extraction may be better.
4. Consider using hybrid approaches where you combine several different forms of AI or combine AI with traditional statistical approaches.

Here at Functionize, we make a lot of noise about our intelligent test agent. But, some of you may ask, how do you justify calling it intelligent? How do we know you're not just another of those companies claiming to do AI? Well, let's look at our checklist above and compare it with our approach:

1. None of the Functionize models are re-packaged from other services. For instance, our Adaptive Event Analysis™ engine uses machine learning to fingerprint and identify elements in your UI so your tests won't break from release to release.
2. We have spent several years collecting quality data to construct advanced metal models to improve our training data. This has allowed us to launch our revolutionary ALPTM engine. Users simply write in plain English and functional tests are generated.
3. We are not dumbly wedded to AI. There are several aspects of our system where we use other techniques. A good example is our autonomous template recognition.
4. Our entire system is a hybrid of multiple AI techniques alongside more traditional statistical methods. We use NLP, several types of machine learning, image recognition, reinforcement learning, and various clustering algorithms to name but a few.

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

PARTNER SPOTLIGHT

Functionize

Functionize increases QA efficiency by 6x

Category Testing Automation, AI, cross-browser testing

New Release Continuous, Saas Delivery

Open Source? No

Case Study

Brazil-based TOTVS is the biggest enterprise software house in Latin America, providing mission critical industry software for about 100,000 enterprise customers.

TOTVS Labs is responsible for introducing cutting-edge technology that can be fed into the rest of the business. One of the most successful outputs of the labs is Carol – a Data and Machine Learning Platform. Testing Carol with conventional QA tools was near impossible. TOTVS Labs started using Functionize in early in 2017 and saw an immediate change. "Using Functionize I was able to create new tests in minutes instead of hours", said Wilson Souza, TOTVS Labs lead QA Engineer. This has amazed Vicente Goetten, Executive Director of the TOTVS Labs. Not only has the QA team managed to remain productive on their own, but perhaps most strikingly, Functionize has been able to completely remove QA's reliance on help from the backend team. As Vicente puts it: "Without Functionize, we would have had to grow our test team significantly, and also the team who supports QA. When you combine this with the reduced impact on the development team as a whole, this marks at least 5-6 fold increase in efficiency."

Strengths

- Tests created by simply writing in plain English
- Tests that don't break from release to release
- 6x increase in overall QA efficiency
- Release product 3x faster

Notable Customers

- Salesforce
- Cisco
- Mastercard
- Zenefits
- TOTVS

Website
functionize.com

Twitter
@functionize

Blog
functionize.com/blog

The Four Keys to Achieving Parallelization in Automated Testing

QUICK VIEW

01. At some point in their automated testing journeys, most organizations encounter the same problem: as the number of tests in a suite starts to grow, the suite starts taking too long to run.

02. Leveraging parallelization is one of the best ways to successfully implement automated testing.

03. Make sure you run atomic tests, keep tests autonomous, correctly manage test data, and avoid the Static keyword in WebDriver.

BY NIKOLAY ADVOLODKIN
SOLUTIONS ARCHITECT, SAUCE LABS

Agile development has reached a crossroads. Keenly aware of the need to deliver high-quality web and mobile applications at speed, most organizations have adopted modern agile development methodologies with the expectation that doing so will drive faster release cycles, improve functional quality, and, ultimately, lead to better customer experiences.

Yet, for many organizations, release velocity has stalled, if not declined. According to a recent report from Forrester, the percentage of organizations releasing software on an at least a monthly basis declined from 36% in 2017 to 27% in 2018.[1] Our collective efforts to deliver better software faster have hit a roadblock, and automated testing is often the reason why. In order to deliver quality applications at speed, organizations must overcome their mounting automated testing challenges.

Parallelization: The Only Path Forward

At some point in their automated testing journeys, most organizations encounter the same problem: as the number of tests in a suite starts to grow, the suite starts taking too long to run. Testing is the foundation of agile development, so once your suites start taking too long to run, your efforts to deliver quality at speed eventually stall out.

Fortunately, there's a clear solution: parallelization. Take the hypothetical example of a suite of 100 tests, each of which takes two minutes to run. If you can run those tests in parallel, you'll complete your entire suite of tests in just two minutes, enabling your developers to quickly get the feedback they need. If, on the other hand, you're unable to leverage parallelization, that same suite of tests will take more than three hours to run, while your developers wait unproductively.

Leveraging parallelization is the only way to successfully implement automated testing. But most organizations immediately run into a number of hurdles as they attempt to scale their automation efforts through parallel testing. Let's take a look at how to overcome those hurdles by examining the four mandatory requirements for effective parallel testing.

Requirement #1: Run Atomic Tests

The most powerful and effective strategy for achieving parallelization is to run atomic tests. Atomic tests asses just one single application feature. So, rather than scripting a single test to validate that the login page loads, the user name field displays, the password field displays, the logout displays, and items can be added to a cart, a team leveraging atomicity would design five separate tests that validate each of those functions individually.

The benefits of atomic tests are many. When I conduct automated testing workshops, I often open by asking attendees to guess which of the

following test suites, each covering the same exact application features, will execute faster: a test suite featuring 18 long-flow, end-to-end tests, or a suite using 180 atomic tests. The answer, to the surprise of most, is the suite featuring 180 atomic tests. In fact, when I run live demos using this exact scenario, the suite featuring 180 atomic tests typically executes 8 times faster than the suite with 18 long-flow tests!

Most organizations mistakenly assume that running longer tests in smaller quantities is a faster approach than running atomic tests in larger quantities. Thus, they attempt to combat longer-than-desired test run times by adding more validations to their tests. Doing so only adds fuel to the fire. No matter how much parallelization you apply, your test execution time will only be as fast as the slowest test in your suite. So, if you have a suite of 30 tests, 29 of which take 2 minutes to execute, and one of which takes 30 minutes to execute, you'll have to wait the full 30 minutes to get the results of every test in that suite.

ATOMICITY DRIVES STABILITY

Suites that leverage atomic tests are far more stable than those that don't. After all, every single validation request you add to a test is another chance for something to go wrong. Atomic tests are also far easier to debug when a test does fail. Because atomic tests execute much faster than longer tests, developers are getting feedback on code they literally just wrote, making it exponentially easier to go back and fix.

Moreover, since atomic tests focus on just one specific piece of application functionality, there's no ambiguity about what's broken in the event of a failed test. It can only be that one thing, and developers don't have to waste precious time rooting around for the source of the failure. By comparison, when a non-atomic test fails, developers get no feedback on features beyond the point of the failure. So, if you're testing 50 different elements of functionality within a single test, and the failure occurs at element 10, the remaining 40 elements go untested.

Requirement #2: Keep Tests Autonomous

An autonomous test is one that can run completely independent of all the other tests in your suite. Many teams make the mistake of designing a multi-threaded process in which one test cannot successfully execute until its predecessor has done the same. This means, for example, that in order to execute a test validating the efficacy of your checkout function, you first have to successfully execute tests for all of the functions that precede it in the application workflow. It also means that once one test fails, all of the other dependent tests will fail as well. This type of co-dependent approach is a non-starter for effective parallel testing. Instead, make sure you design your tests such that they can all run entirely on their own and in any order necessary.

Requirement #3: Correctly Manage Test Data

To effectively implement parallel testing, you must be able to control your test data. This is difficult to do even in the best of circumstances, as it depends on more than just your automation engineers to execute. But it's especially difficult to do if you're relying on traditional hard-coded test data, the static nature of which is a poor fit for the dynamic nature of automated testing. Instead, the best strategy is to leverage what's known as just-in-time data. With a just-in-time approach, you create test data, utilize it for a given automated test suite, and then destroy it at the end of the test. This cuts down on complexity and ensures that data from a previously executed test doesn't muddy the results of your current test.

Requirement #4: Avoid the Static Keyword

The final mandatory requirement for effective parallel testing is to avoid applying the "static" keyword to the WebDriver instance you're managing in your test scripts. Using the "static" keyword is the fastest way to kill your parallelization dreams.

Identifying the WebDriver as "static" effectively ties the WebDriver instance to the **definition** of the class, not **instances** of the class, which means that there can only be one instance of WebDriver attached to (and shared between) **all** of the tests in your suite. It's like telling all the cars in the world that they have to share a single steering wheel!

Some testers will work around this roadblock by people choosing to "fork" the JVM process for each test in their suite, thus creating a new instance of the entire test suite for every test. To extend the car metaphor, this is like recognizing that your cars all share the same steering wheel and compensating for it by creating a new planet Earth for each car you want to put on the road. Not exactly the most efficient workaround.

Though use of the "static" keyword is technically acceptable in certain situations, as a general rule, the best approach is to simply avoid it.

Putting it All Together: Excellence Through Automation

As you move through implementation of each of these critical requirements, it's important to always keep the end goal in mind. That end goal, of course, is happy customers. In the modern age of digital business, the best and fastest way to turn prospects into customers (and customers into loyal advocates) is by rapidly delivering high-quality web and mobile applications. If you can achieve parallelization in automated testing, that end goal is well within reach.

NIKOLAY ADVOLODKIN is a Solutions Architect at Sauce Labs. He has an extensive background in software testing, quality assurance and test automation as the CEO and Test Automation Instructor at UltimateQA.com, a training site full of videos and resources covering the gamut of testing topics and technologies. LinkedIn Twitter

A brief history of web and mobile app testing.

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

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

SAUCELABS
Testing at the speed of awesome.

Automated Testing is Essential to DevOps and Continuous Delivery

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

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

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

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

WRITTEN BY LUBOS PAROBEK
VP OF PRODUCTS, SAUCE LABS

PARTNER SPOTLIGHT

Automated Testing Platform

SAUCE LABS

Sauce Labs ensures your favorite mobile apps and websites work flawlessly on every browser, operating system, and device.

Category Automated Testing Infrastructure

New Release Daily

Open Source? Yes

Case Study

With leading brands such as AOL, HuffPost, Yahoo Finance, Yahoo Sports, TechCrunch, and others, Verizon Media is home to media, tech, and communication products that more than a billion people love and trust. Its engineers develop and manage web, mobile, and server apps spanning tens of thousands of projects. With most of those teams relying almost solely on manual testing, Verizon Media needed an automated testing solution that would scale to their volume and run seamlessly alongside their CI/CD platform. They chose Sauce Labs, and the results have been transformational. By enabling engineers to programmatically access all the latest browser/OS combinations and run tests in parallel, at scale, Verizon Media now runs more than 2.7 million tests a month. Tasks that used to take 45 minutes to complete manually are now completed in five minutes or less. With Sauce Labs running in tandem with their proprietary CI/CD solution, Verizon Media saves a total of 4,500 hours per year in staffing and maintenance, making their investment in automated testing and continuous integration a worthwhile one.

Strengths

- Enterprise-grade cloud-based test infrastructure provides instant access to more than 900+ desktop browser/OS combinations, ~200 mobile emulators and simulators, and 2,000+ real devices
- Highly scalable, on-demand platform reduces testing time from hours to minutes when tests are run in parallel
- Optimized for CI/CD workflows, testing frameworks tools, and services
- Improve developer productivity with Saucer Headless & Sauce Performance

Notable Customers

- Salesforce
- Walmart Labs
- VISA
- Slack
- Starbucks
- Yahoo!
- American Express
- Verizon Media

Website
saucelabs.com

Twitter
@saucelabs

Blog
saucelabs.com/blog

Executive Insights on the State of Automated Testing

QUICK VIEW

01. The best practices for using automated testing to improve speed, quality, and testing vary wildly across responses, reflecting a lack of standards and best practices.

02. Automated testing is being integrated into the DevOps methodology and the CI/CD pipeline.

03. Developers should work with engineers and security professionals, and practice test-driven development to ensure their applications perform well with automated testing.

BY TOM SMITH
RESEARCH ANALYST AT DEVADA

To understand the current and future state of automated testing, we spoke to 14 IT professionals intimately familiar with automated testing.

Here's who shared their insights:

- Drew Horn, Senior Director of Automation, Applause
- Angie Jones, Senior Developer Advocate, Applitools
- Isa Vilacides, Director of Engineering, CloudBees
- Himanshu Dwivedi, CEO, Data Theorem
- Antony Edwards, COO, Eggplant
- Kevin Fealey, Senior Manager Application and Product Security, EY
- Hans Buwalda, CTO, LogiGear
- Malcolm Isaacs, Senior Solutions Manager, Micro Focus
- Madan Mohan, Global Head of Travel and Transportation, NIIT Technologies
- Jared Go, CEO, OhmniLabs
- Derek Choy, CIO, Rainforest QA
- Nancy Kastl, Executive Director of Testing Services, SPR
- Rishikesh Palve, Integration Product Manager, TIBCO
- Ray Wu, CEO, Wynd

Key Findings

1. The keys to using automated testing to improve speed, quality, and testing are all over the place, which reflects the lack of standards and best practices that are hindering widespread adoption. While most respondents see the value of automated testing to improve scale, speed, and quality, there is little agreement on the steps companies need to take to be successful.

You need a model-based approach to auto-generate tests, get impact analysis, self-healing, and focus on what really matters. Identify the tools necessary to design and run the tests and test applications as they are being developed. Decouple components and make tests small. Consider what tests need to be automated and those that do not.

2. The most significant changes to automated testing are its integration into the DevOps methodology and CI/CD pipeline, how its embedded in the SDLC, and the shift from UI to API testing.

As we have adopted an Agile methodology of development, we have to adopt an agile testing methodology. Continuously improve automated tests and integrate them into the CI/CD pipeline. Get fast feedback by running tests in the development cycle to find and fix defects as early in the SDLC as possible. A focus on DevOps will lead to security being integrated at the beginning of the process.

Automated tests are becoming embedded into the developer process and has a more critical role in the software development process.

There has also been a shift from UI to API automation. With the movement from monoliths to distributed systems, it is more common to focus on testing contracts between APIs. Testing via REST APIs is getting more attention as well.

3. Use cases for automated testing revolve around increasing speed, security, and scale in financial services, telecommunications, retail, healthcare, travel, and automotive.

Citibank was able to move their mobile banking app rating from 3.0 to 4.5 with massive increases in speed of time to market and in-sprint performance testing.

A large telecom provider leveraged out-of-the-box CI/CD tooling and testing to reduce development cycles. They now release every few weeks and are able to monitor the health of their applications from tablets while out of the office.

Automated testing improves time-to-market and quality-to-market, and enables testing to scale to test more cycles more quickly to identify potential failures.

4. The most common failures affecting continuous testing are the lack of proper planning and strategy and the slow speed of automated test execution. Customers know they need to do something about testing, but they do not have a plan to deploy and use technology. There's a lack of catering to the specific needs of a given business, and automation is frequently being done after the fact. A lack of design and structure of tests leads to fragmented test cases with many low-level steps that can make maintenance difficult. People tend to underestimate the complexity, effort, and skills required to build and maintain automated tests.

Other automated testing failures are high automated test creation and maintenance time, low speed of automated test execution, and automation that's unable to detect and prevent leaks.

5. The future of automated testing is AI/ML and all of the ancillary benefits of the technology being used to improve the process. There are a lot of opportunities to leverage AI/ML to learn from past test executions and optimize automated tests with autogenerated test cases, identify root causes, self-healing test, test case creation, and optimal coverage. Also look for cognitive automate, cognitive vision, voice automation, and robotics to be used as automated testing becomes more advanced.

6. The biggest concerns around automated testing today are the slow uptake and the lack of skilled people to drive greater uptake, success, and ROI. Even though everyone wants automated testing, only 40% of testing is automated. Advancements in automated testing are not keeping pace with the rest of IT, and it is still treated as a side show. While there are a lot of tools available, the biggest challenge is going to be finding staff who know how to use these tools effectively.

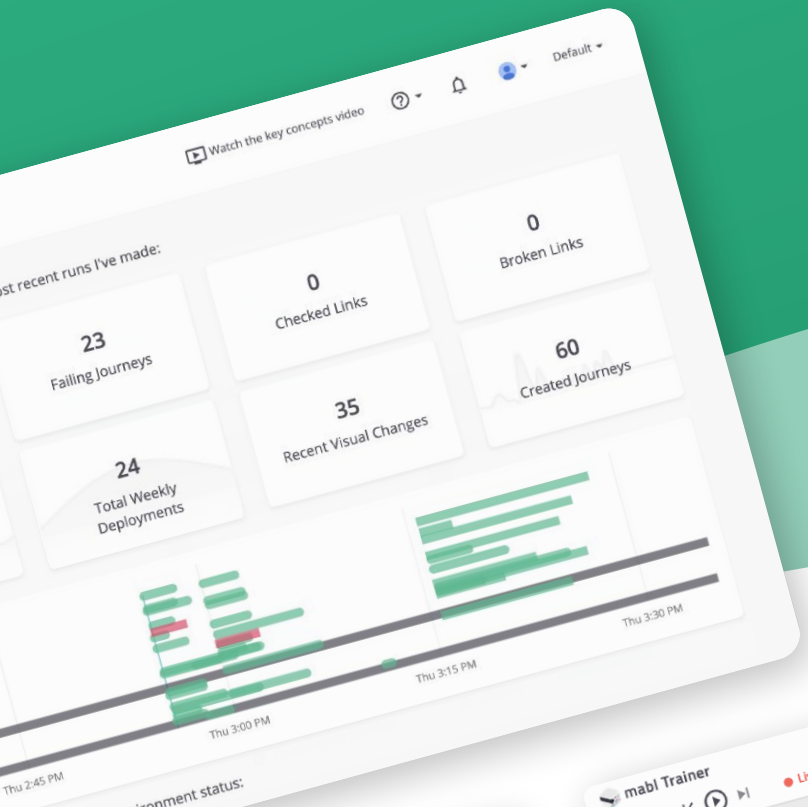
The demand for test automation professionals is higher than the supply. There is such a need for people with test automation skills, and there are not enough people trained to fill this void. There is also a lack understanding of security vulnerabilities.

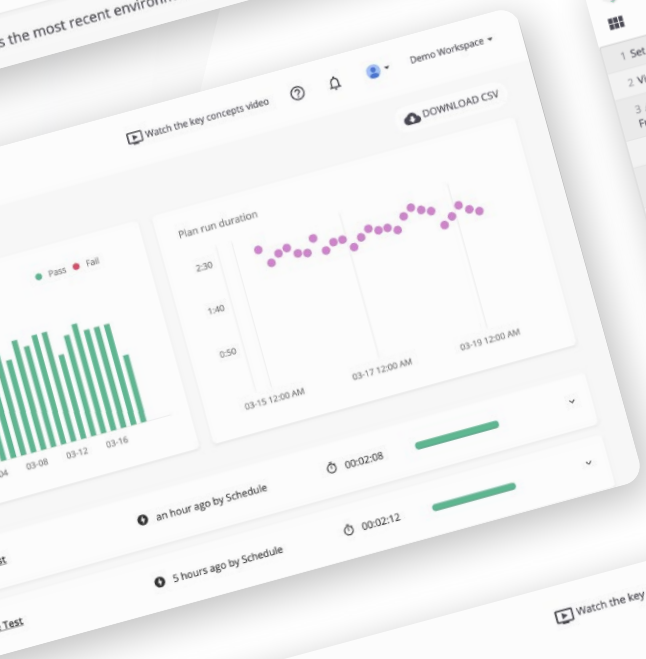
7. Developers need to work with engineers and security, and practice test-driven development principles to ensure their code and applications perform well with automated testing. Follow coding best practices and focus on test-driven development. Start running automated tests before you create a build and send for quality analysis. Design code with clean interfaces that can lend themselves to automated tests. Proactively work with security and engineering from the beginning to identify what can go wrong. Design tests up front in addition to manufacturing. Think about building a more testable product with tagging elements. Work with automation engineers to learn what a tester may need.

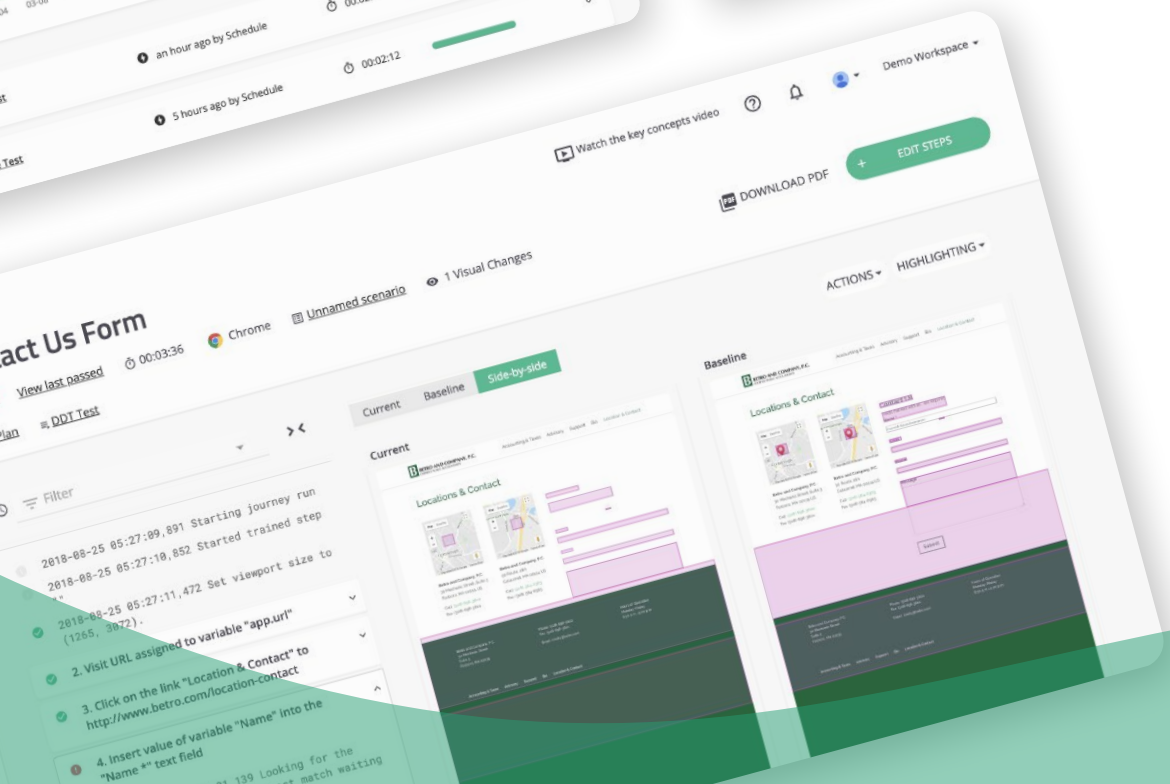
TOM SMITH is a Research Analyst at Devada who excels at gathering insights from analytics—both quantitative and qualitative—to drive business results. His passion is sharing information of value to help people succeed. In his spare time, you can find him either eating at Chipotle or working out at the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

mabl

Bringing testing joy to a DevOps world







+ Scriptless testing
+ Cross-browser
+ Auto-healing tests
+ Visual regression testing
+ ML-driven insights
+ Infinite parallel test runs
+ All in one platform

Free to try now at
app.mabl.com/signup

TRUSTED BY

actian activate your data · Betterview · HARVARD BUSINESS SCHOOL · splunk> · Honeywell · shi · Symantec

www.mabl.com

Testing at the Speed of DevOps

Innovations in software delivery processes over the last 10 years have enabled engineers to build and ship code faster, and system admins and operators to more quickly release code to production, monitor it, and manage it. But where's the innovation for QA? We've focused our attention on dev and ops, but QA has been left with cumbersome manual processes and antiquated test automation tooling that was built for a world where software changed infrequently and functionality was highly specified and documented.

But this won't remain the case for long. According to Forrester, 68% of agile teams now include testers, and quality has become by far the most important success metric that dev teams use. QA is finally making its comeback, being spearheaded by next-generation QA solutions like mabl.

The New Era of Test Automation Solutions

New-age QA solutions have the challenge of enabling continuous testing in DevOps — no small feat given these requirements for success:

1. TESTS MUST ADAPT TO CHANGE

Tests that break due to app changes that don't affect UX causes notification fatigue and eliminates the value those tests could provide. Auto-healing tests update the test script whenever element locators change that don't affect assertions or the intent of the test.

2. TESTS MUST RUN IN THE CLOUD

Testing requires its own infrastructure, which takes time to provision, operate and scale. The cloud enables flexible scheduling and burst concurrency with on-demand containers that can shut down when no longer in use, eliminating waste, resource bottlenecks, and operational overhead.

3. TESTS MUST BE PART OF THE DEVELOPMENT PIPELINE

The pace of development today is such that testing needs to be automated in the delivery pipeline. Gartner notes: "Traditional quality assurance (QA) practices are designed around finding defects through testing, which is inefficient and leads to cost escalation. Digital approaches require a continuous approach to quality that includes automated tests combined with a quality focus delivered through practices." By running tests after every build, we multiply the capacity of the QA automation team with more test coverage and better analytics on the root cause of issues.

4. TESTS MUST AUTOMATICALLY SURFACE INSIGHTS FROM OUR TEST DATA.

Machine learning is an integral part of mabl's approach in bringing QA to the modern era. With ML, we can analyze big data generated from test executions in production environments and automatically discover trends that would take incredible amounts of effort for humans to find manually. Gartner agrees on the importance of ML: "AI and machine learning (ML) will have a massive impact on how apps are being developed and tested. Applying AI and ML to move from predominantly manual to automated QA will help determine and increase the value of automation for QA and testing. AI-assisted automation can support a new culture of working, such as DevOps."

Welcoming the Era of Intelligent QA

SELF-HEALING TESTS REDUCE TEST SCRIPT MAINTENANCE

mabl is built upon a proprietary framework that uses sophisticated, durable methods of simulating user behavior in automated tests, and therefore does not suffer from the brittleness associated with existing automation frameworks like Selenium. Tests aren't tightly bound to individual front-end elements which change frequently. Instead, mabl uses ML to create and maintain models of tests, enabling test adaptation rather than failures when faced changed element locators.

TEST INFRASTRUCTURE RESOURCES IS ON-DEMAND

Teams struggle with the performance, cost, and operational overhead of on-premises testing systems. mabl takes advantage of on-demand cloud computing resources to execute tests faster and more efficiently. Tapping into powerful data processing and analytics services, mabl analyzes test results and delivers insights upon every test execution. mabl is delivered completely as a service, offloading the operational burden from your team.

QA IS PART OF THE DELIVERY PIPELINE

mabl can be tightly integrated into automated CI/CD pipelines. Tests can be triggered automatically after new builds and on-demand, and team members can be notified over Slack or email when there are potential issues so that they can be addressed before they affect end-users or accrue technical debt. A dashboard gives QA and dev teams visibility into test coverage and results across builds and environments.

THE DEFINITION OF QUALITY HAS EVOLVED

Beyond validating specific assertions in tests, mabl puts an emphasis on detecting and highlighting regressions outside the scope of the test suite assessing performance regressions and unusual visual changes from one test run to the next. This holistic approach helps QA understand the extent to which changes improve or degrade the user experience, putting a stronger focus on product coverage over code coverage.

mabl incorporates cutting-edge machine intelligence, is delivered from the cloud, and embeds into the modern development workflow. These ingredients together will drastically improve the effectiveness of QA, ushering in a new era of efficiency and innovation across the software industry.

WRITTEN BY CHOU YANG
PRODUCT MARKETING, MABL

Automated Testing Solutions Directory

This directory of automated testing, mobile testing, QA tools, and DevOps platforms 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	Free tier available	applitools.com
Appvance	Appvance IQ	AI-driven, unified test automation system	Demo available by request	appvance.com/appvance-iq
Atlassian	Bamboo	CI, CD, & automated testing	30 days	atlassian.com/software/bamboo
Atomist	Atomist	Cloud-native app delivery	Free tier available	atomist.com
Bitbar	Bitbar	Cloud-based mobile testing	14 days	bitbar.com
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	CI & CD	Free tier available	circleci.com
CloudBees	CloudBees Core	Governed & flexible	Available by request	cloudbees.com/products/cloudbees-core

Company	Product	Product Type	Free Trial	Website
CollabNet	VersionOne	Acceptance & regression testing	Available by request	collab.net
Compuware	Compuware	Agile mainframe maintenance	N/A	compuware.com
Cucumber	Cucumber	Automated rails testing	Open source	cucumber.io
DevExpress	TestCafe Studio	End-to-end web testing	30 days	testcafe.devexpress.com
Eggplant	Eggplant	Digital performance mgmt & 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
GE Digital	Predix Platform	IIoT application optimization	Demo available by request	ge.com/digital/iiot-platform
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	Free tier available	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	CI & build mgmt	Available by request	developer.ibm.com/urbancode/products/urbancode-build
InfluxData	InfluxData	Stream processing & analytics	Open-source components	influxdata.com/products
Jenkins	Jenkins	CI & CD	Open source	jenkins.io
JetBrains	TeamCity	CI & 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

Company	Product	Product Type	Free Trial	Website
Kobiton	Kobiton	Mobile app testing	120 minutes	kobiton.com
Loadster	Loadster	Web app load testing	Free tier available	loadsterperformance.com
mabl	mabl	ML-driven automation testing	14 days	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 & mgmt platform	Available by request	software.microfocus.com/en-us/products/quality-center-quality-management/overview
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 mgmt, exploratory, & browser testing	Free tier available	visualstudio.microsoft.com/vs/test-professional
Mobile Labs	GigaFox	App development & continuous testing	Available by request	mobilelabsinc.com
Neotys	NeoLoad	Automated performance testing	Free tier available	neotys.com/neoload/overview
NowSecure	NowSecure	Mobile app security	30 days	nowsecure.com
NUnit	NUnit	.NET unit testing framework	Open source	nunit.org
Panaya	Panaya Test Center	Continuous testing	14 days	panaya.com
Parasoft	dotTEST	Test automation platform	Available by request	parasoft.com/product/dottest
Parasoft	SOAtest	Functional & load testing for distributed apps	Available by request	parasoft.com/product/soatest
Perfecto Mobile	CQ Lab	Automated web & mobile testing	Available by request	perfectomobile.com/solutions/perfecto-test-automation

Company	Product	Product Type	Free Trial	Website
PHPUnit	PHPUnit	PHP unit testing framework	Open source	phpunit.de
Plutora	Plutora	Release & test mgmt platform	Demo available by request	plutora.com
Practitest	Practitest	QA & test mgmt	Available by request	practitest.com
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 JS testing	30 days	sencha.com/products/test
Site24x7	Site24x7	Performance monitoring for DevOps & IT Ops	30 days	site24x7.com
SmartBear Software	Cross Browser Testing	Automated mobile & web testing	7 days	crossbrowsertesting.com
SmartBear Software	LoadUI	API load testing	14 days	soapui.org/professional/loadui-pro.html

Company	Product	Product Type	Free Trial	Website
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
Sonatype	Nexus	Repository mgmt	Available by request	sonatype.com/nexus-repository-sonatype
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	Free tier available	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	CI & CD	Open-source version available	gocd.org
Travis CI	Travis CI	Continuous integration	Open source	travis-ci.org
Tricentis	Tosca	Continuous testing platform	14 days	tricentis.com/products/automate-continuous-testing-tosca
Turnkey Solutions	cFactory	Automated mobile & web testing	Demo available by request	turnkeysolutions.com
Ubertesters	Ubertesters	Automated mobile testing	14 days	ubertesters.com
Venafi	Venafi Platform	Machine identity protection	N/A	venafi.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	N/A	worksoft.com
xUnit	xUnit.net	Unit testing framework	Open source	xunit.github.io
Zephyr	Zephyr	Real-time test mgmt	7-30 days	getzephyr.com

VISIT THE

Performance Zone

Scalability and optimization are constant concerns for the Developer and Operations manager. The Performance Zone focuses on all things performance, covering everything from database optimization to garbage collection to tweaks to keep your code as efficient as possible.

Keep a pulse on the industry with topics such as:

- Java unit testing best practices
- Software usage analytics
- Log management: From basics to expert tips
- Integration testing in Spring Boot

Visit the Zone

TUTORIALS

CASE STUDIES

BEST PRACTICES

CODE SNIPPETS