



# The Definitive Guide to Continuous Testing

2018



# How to Succeed in Your Continuous Testing Journey

## CHAPTER 1

## WHY CONTINUOUS TESTING IS IMPERATIVE NOW

As the pace of business continues to quicken, companies are starting to recognize that to stay competitive the process of developing and releasing software needs to change. Release cadence has greatly accelerated. There is no occasion anymore for a 6- to 18-month find-and-fix turnaround in which the customer will find the delay acceptable. Things need to move faster, and they need to be ready and perfect faster.

A sea change is occurring in which the responsibility for testing is now being shifted leftward, closer to the beginning of the software development lifecycle and then all the way along it. This is knocking down the wall that has traditionally separated developers and testers, making quality everyone's responsibility. Testing has become more imperative because the consequences of not doing it properly have become more visible.

This is a technical change that in turn demands a cultural change. As testing evolves and spreads out across the SDLC, companies and their IT departments must find a way to change the entire culture of how software is developed and tested.

Maybe most simply: We must stop thinking about testing as an event. It is not something to be done at a specific point. Instead, it must be done by everyone all the time, even before development starts. Defects and problems must be nipped in the bud, not caught/missed — as the product comes out the chute into the customer's lap.

DevOps is about breaking down those walls between development and operations to ensure that software, hopefully built with agile methodology, can be deployed quickly to the users as soon as it is ready, without sacrificing quality. But even where DevOps exists, companies continue to experience a trade-off between quality and speed.

In current testing culture, 63% of software development delays occur in test-QA practices across the lifecycle, and 70% of testing is still manual. But this is no longer practical. You cannot just run tests manually, and there are additional problems to consider:

**56%** *of critical dependencies are unavailable*

**50%** *of time is spent looking for test data*

**64%** *of defects occur in the requirements phase*

Failures, especially those that reach the public, are damaging to data, business processes, customer adoption, retention and brands. It is not possible to turn software around with both speed and quality using outdated waterfall or even agile processes.

Something more comprehensive, dynamic and fluid is required to ensure software is developed and deployed with the quality built in and errors detected from the very start, rather than relying on assigning verification at the end of the lifecycle. This is why we need to embark on a continuous testing journey.

## WHAT IS A CONTINUOUS TESTING JOURNEY?

Continuous testing is the practice of testing across every activity in the SDLC to uncover and fix unexpected behaviors as soon as they are injected.

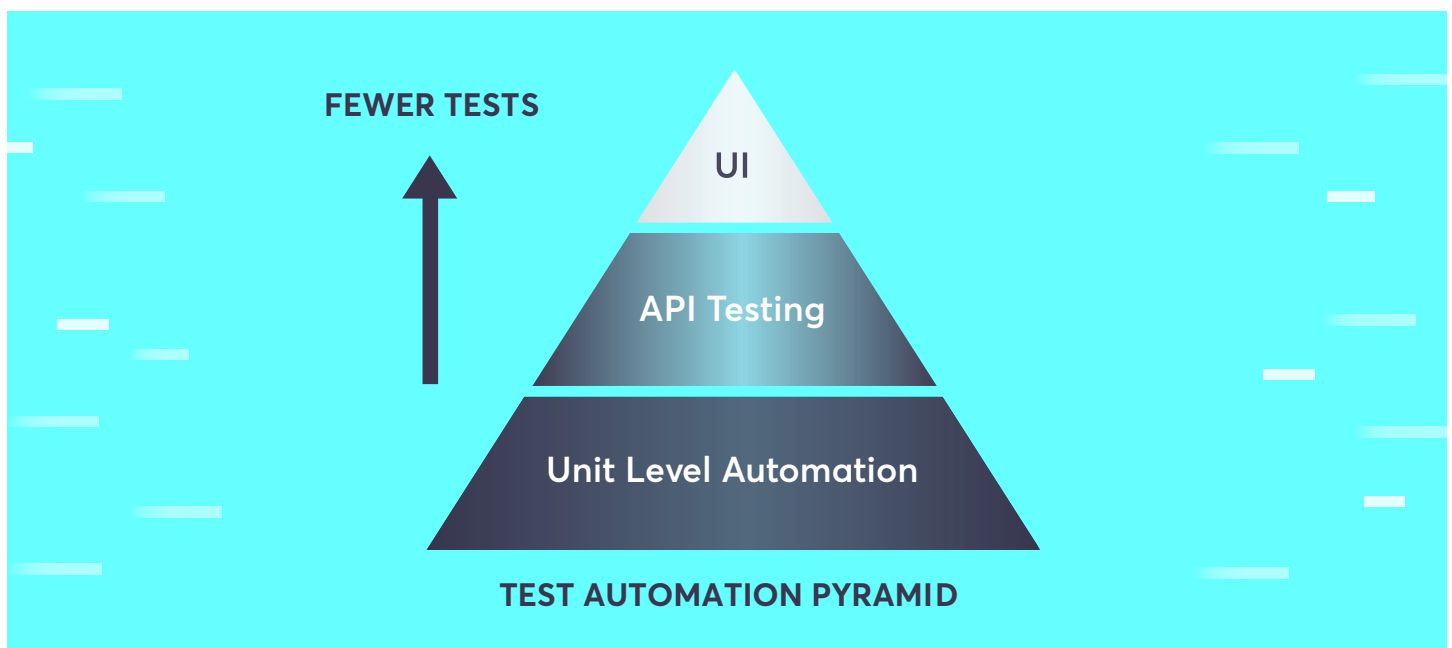
Continuous testing is the embedding of testing as a fundamental and ongoing aspect of every activity through the application lifecycle, from requirements through production, to ensure the business value is being achieved as expected.

Life has never been easy for IT. It is a department filled with craftspeople and dedicated engineers, striving to develop and build a product that works according to the specs — specs that might not always match up to the expectations and limitations of its human end-users. The roles of software have changed. The products must be accessible on a wide range of platforms and must deliver a constantly shifting collection of services.

Quality suffers when we put walls between the people needed to test, develop and create the requirements for the software. IT should no longer be seen as back office work. It must hold its own seat at the decision-making table.

Quality also suffers when unrealistic deadlines dominate, and when capabilities are squeezed into a release without using adequate metrics to delineate quality.

Testing has traditionally operated in a silo, which has resulted in an underwhelming degree of robustness. Testers and the business ask: "How good is my testing? What coverage do I have in terms of functional coverage? What is my functional coverage, not just my code coverage? What are my acceptance criteria? Am I actually testing for them effectively?" Many testers don't know.

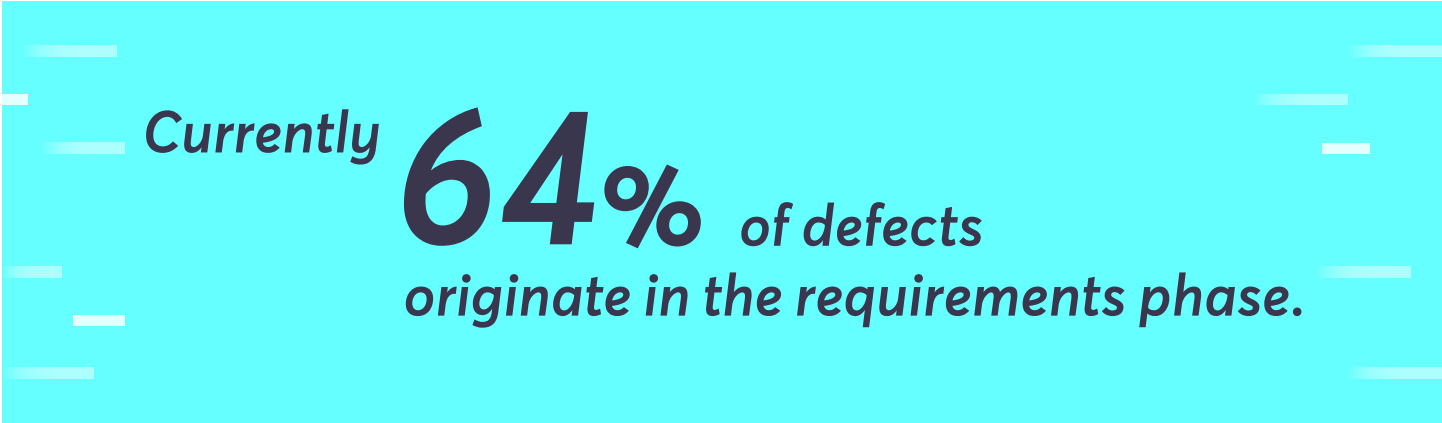


To ensure quality at speed, testers need to aim for a large number of tests at unit level, a smaller number at API level, and an even smaller number at UI. Ultimately, testing should become parallelized or concurrent (test functional, performance and security at the same time), and then shifted left to ensure it all happens continuously, from the beginning of the lifecycle.

Too often, people spend time in non-value activities, testing the wrong things or the wrong set. A team of eight people might say "we're great," but typically 40% of the work must be retested and fixed, meaning a team of eight is doing four people's worth of work, often a couple of months behind. They need to perform a value stream analysis to ensure testing is being done the right way. Value stream analysis is a method of analyzing current processes and identifying a better, more effective one, using lean management ideology.

The tools used in most organizations are "legacy." They sufficed for waterfall, but they don't work well for testing to shift left. They are an impediment to testing at the speed of agile. They don't support automation in the right way for a continuous integration environment and they are tools that developers refuse to use. There needs to be a new set of tools that won't get in the way of speed and quality; tools that enable quality to be built into the process and not "just" test the application as in the past.

But it's not just testing automation that has to change, the requirements definition process must be improved as well. Requirements must be unambiguous, complete and testable.



**Currently 64% of defects originate in the requirements phase.**

Very often requirements are specified in a high-level fashion that is good enough for a developer and a tester that have the domain subject matter expertise to make their interpretation and fill in the gaps in the requirement. However, that leads to team members interpreting the same thing differently, introducing defects in the lifecycle before a single line of code is written. On the flip side, requirements can be specified in an extremely detailed fashion, dozens and dozens of pages long. In that case, requirements take too long to be specified and are very hard to maintain over time. Not to mention developers and testers find it hard to grasp so much information bundled together in a giant requirement. A better way for specifying requirements must be used, one that works for agile teams adopting DevOps.

Quality assurance must also be approached differently. It must start with senior leadership, who must keep the focus on quality and not just delivery. This is because delivery is now about the total user/customer experience, not just the act of deploying or releasing software. This sets the stage for delivering the right set of requirements that help developers and testers understand what they are building. Now, with releases happening every month, proper and efficient feedback loops are critical. QA must no longer be seen as, or act as, a second-class citizen simply because it has, to date, been identified as a shared services property.

To accomplish this change, cultural and organizational changes must occur. This is a human issue rather than a technical one, but it is essential to a successful continuous testing journey. With any technological or process change such as continuous testing or shift left, some people can handle the change and others cannot. The change also threatens the Centers of Excellence (CoE), as testers question how "testing" can be handed over to people who are not trained.

This is chiefly why the "Center of Excellence" CoE must transform into a "Center of Enablement" CoE. Those who must now take on the responsibility of testing all along the SDLC must know the proper processes and have the tools to make it easier, and the testers with the most skill and experience must enable them. This needs change management support from the top down as well as from the bottom up.

As testing shifts left, so too must performance and security. They should no longer be treated as "non-functional." They must be treated as equals to functional testing. There is currently inadequate security testing before

deployment in production, and it is time to treat security as a first-class citizen, incorporating specialized security testing practices into the continuous testing process.

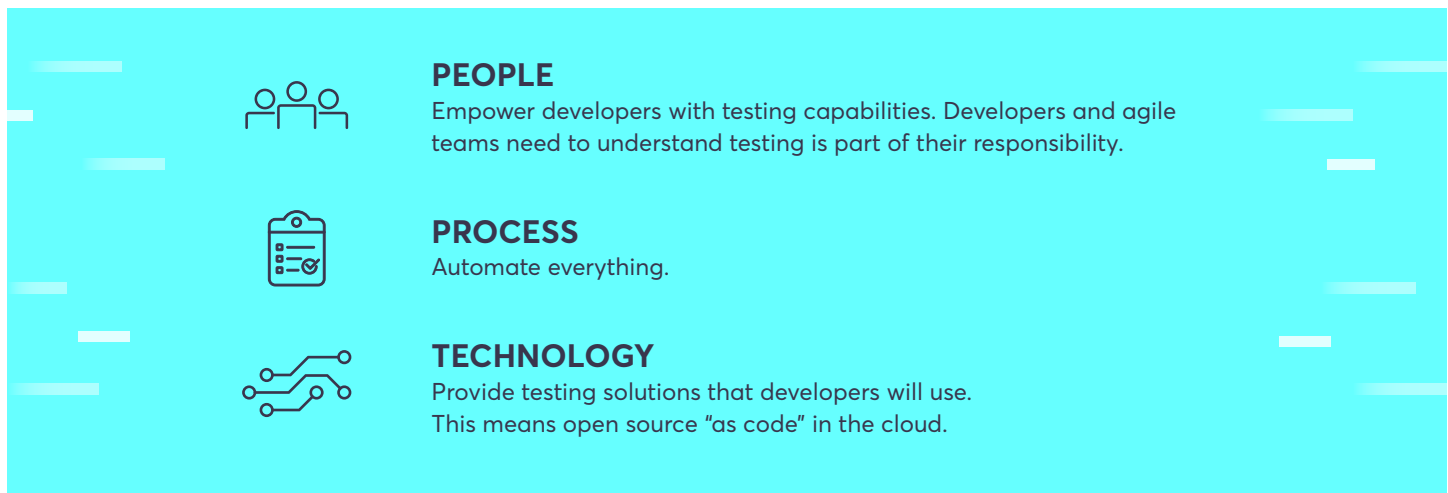
Once an application is in production it may turn out that it is not performing correctly under actual user traffic. More horsepower might seem like an adequate quick fix, but this is only achievable within a cloud-based infrastructure. Consequently, there is a need to ensure application performance is tested prior to production, starting all the way to the left, at the unit level by the developers, and then incrementally added to the performance testing scope as units of code start forming larger components that will realize stories, features and epics, always leveraging the smaller performance tests that have been built previously. Companies need to use the proper tools and processes to ensure this happens.

A change in culture demands top-down and bottom-up change initiatives paired with great leadership. Overall, there needs to be a continuous testing culture. Executives and IT managers must talk to testers and developers and focus on building a plan together. Make them part of the solution by involving them in the initiative. Give them the chance to fail and learn from it.

In continuous testing, everybody codes and tests. The team is individually and collectively responsible for engineering and delivering high-quality software. This highlights an evolution from waterfall to scrum fall to agile to the merging of agile and DevOps to business agility, enabled by continuous testing.

Collective responsibility uses retrospectives to inspect and adapt the process and its outcomes. This redefines the definition of done. Creating a continuous testing mindset means providing developers with quick feedback to ensure their code is working so that it can move through the release pipeline quickly. It means creating or finding data to feed all the functional and non-functional tests running before production and in production.

The three key items that constitute continuous testing are:



An organization must ensure it has a universally understood definition of continuous testing. It is still relatively new and like many other areas of software development, definitions and practices may vary.

***Continuous testing is the practice of testing across every activity in the SDLC to uncover and fix unexpected behaviors as soon as they are injected.***

A good analogy is to think of an automated software assembly line that includes testing at every step, beginning at requirements gathering, automatically triggered, from first code check-in all the way to production. It includes all aspects of testing all the way up to integration and performance testing and monitoring in production. Continuous testing should be an effective and efficient application-centric test regime that is end-to-end in scope — across CI, CD and production. Its goal is to be as efficient as humanly possible to getting changes into production in the shortest lead time with highest quality and top customer experience.

Accelerating and improving a code release process through continuous testing requires a set of disciplines to ensure that quality keeps pace. To embed quality in applications, teams need to add more modern testing practices in the form of small quality checks performed throughout the application pipeline. This would enable small sections of code to be constantly tested.

Test automation is not always the first thing to be tackled. The reason is because even after API tests and GUI tests are automated and integrated into a continuous integration agent, those tests have dependencies — test data, interfaces and environments — that have to be satisfied before they can be executed. You must:

- Remove environmental constraints and free up developers and testers to do their thing, even if manually. Virtualize any and all interfaces you don't own or control, or don't want to test.
- Utilize ephemeral testing environments.
- Automate test data provisioning and management to feed your tests, manual or automated, on-demand.
- Automate your tests so they can run against your virtualized interfaces using the appropriate test data.
- Have your pipeline orchestration engine set up in such a way that there is no manual intervention on the steps above.
- Next, focus on the complementary disciplines.

### 11 DISCIPLINES OF CONTINUOUS TESTING

#### 1. VIRTUALIZED ENVIRONMENTS

Continuous Testing means testing more frequently. That means you will be hitting multiple environments more frequently. And that represents a problem, a bottleneck, if those environments are not available all the time. Some environments are accessible through APIs, others through various messaging interfaces. Those environments could be built with modern architectures, while others are monolithic legacy client/server or mainframe systems. So how do you coordinate testing through multiple environment owners that may not always keep their environments up and running for you to test against? Virtualizing those environments allows you to test your code without having to worry with areas you're not changing (i.e., other systems and environments). Making those environments ephemeral, available on-demand through virtualization, removes that constraint from your development lifecycle as they'll always be available. You control them — you can spin them up as needed.

#### 2. TEST DATA MANAGEMENT

You must have the right data and the appropriate diversity of data for positive and negative scenarios. You won't find all that diversity in production — that is critical. Therefore, synthetic data generation is the way to go to achieve continuous testing with the highest levels of confidence that no personally identifiable information is at risk in the data. Also, you won't be able to bring data from production, condition it and provision it at the speed your application pipeline requires.

#### 3. TEST AUTOMATION

In a fully orchestrated application pipeline, today's scripts will fail due to things not related to the application code. So, no one will trust the results. You need reliable test automation to achieve continuous testing.



#### 4. PIPELINE ORCHESTRATION

The backbone. Everything is tied to it. It must be integrated with your automation suite. You must understand how it works, how to interpret results and how to make it scalable. This sets up continuous testing in a manner where you integrate continuous testing attributes within a pipeline. Make sure it's transparent and everyone has full visibility into what's being run through the pipeline. It's an automated workflow tool that will run all of your automated tests and fully integrate with code deployment activities as it moves through the pipeline. As part of any DevOps adoption initiative, it is unrealistic to expect to get to continuous testing without the reliability and speed of a standardized and automated pipeline.

#### 5. API TESTING

This will enable alignment of the testing pyramid. Organizations should strive to test as much at the unit and API levels as possible and minimize reliance on UI testing. To achieve continuous testing, you have to embrace the concept of the testing pyramid properly, strengthen unit and API testing, and reduce reliance on UI testing, especially for business logic testing.

#### 6. PERFORMANCE/LOAD TESTING

Even when applications are functionally good, you must radically change the way you think about performance testing. As you shift testing to the left, you're testing earlier in the lifecycle, with less application and infrastructure components, which means the tests are smaller by nature. But they're also much greater in volume. Make that capability accessible to everyone across all agile teams. That means all developers and testers alike have to be able to create a performance test and run it by themselves, without having to send any requests to anyone.

#### 7. SECURITY TESTING

Developers need to be clear on what the expectations are around the state of the code in order to be allowed or denied to be released. They must receive proper training and tooling to measure how secure their code is. Education of developers must be ongoing, as the application security area is always evolving. The application pipeline must be set up to automatically run static analysis, dynamic analysis and software composition analysis, so that any security issues are caught and highlighted, then those must be used as an opportunity to make developers better. Security testing needs to happen at every step of the lifecycle.

#### 8. ACCEPTANCE OF TEST-DRIVEN DEVELOPMENT AND BEHAVIOR-DRIVEN DEVELOPMENT

Take the acceptance criteria for each user story and create the tests to ensure those were met. This keeps testing focused within the sprints and ensures developers develop what the business expects. Over time, teams will define much more detailed acceptance criteria, which requires tests to also be more comprehensive.

#### 9. AUTOMATED TEST GENERATION

The activity of manually designing and writing manual tests or automated test scripts is a bottleneck. Automatically generate the new acceptance-level tests for manual and automated testing in the beginning of the sprint, so when developers start checking in and merging the first working pieces of their code, we can start running our tests in an automated fashion. This gives the ability for developers to get immediate feedback on the quality of their code as they check it into source control.

#### 10. REQUIREMENTS ENGINEERING

You must include all SDLC stakeholders into the continuous testing journey. That means finding a better way to communicate and collaborate on the requirements. The earlier you include requirements engineering into your continuous testing initiative, the smoother the continuous testing journey will be, because team members will be working with the same understanding from the beginning.

#### 11. FEEDBACK LOOPS

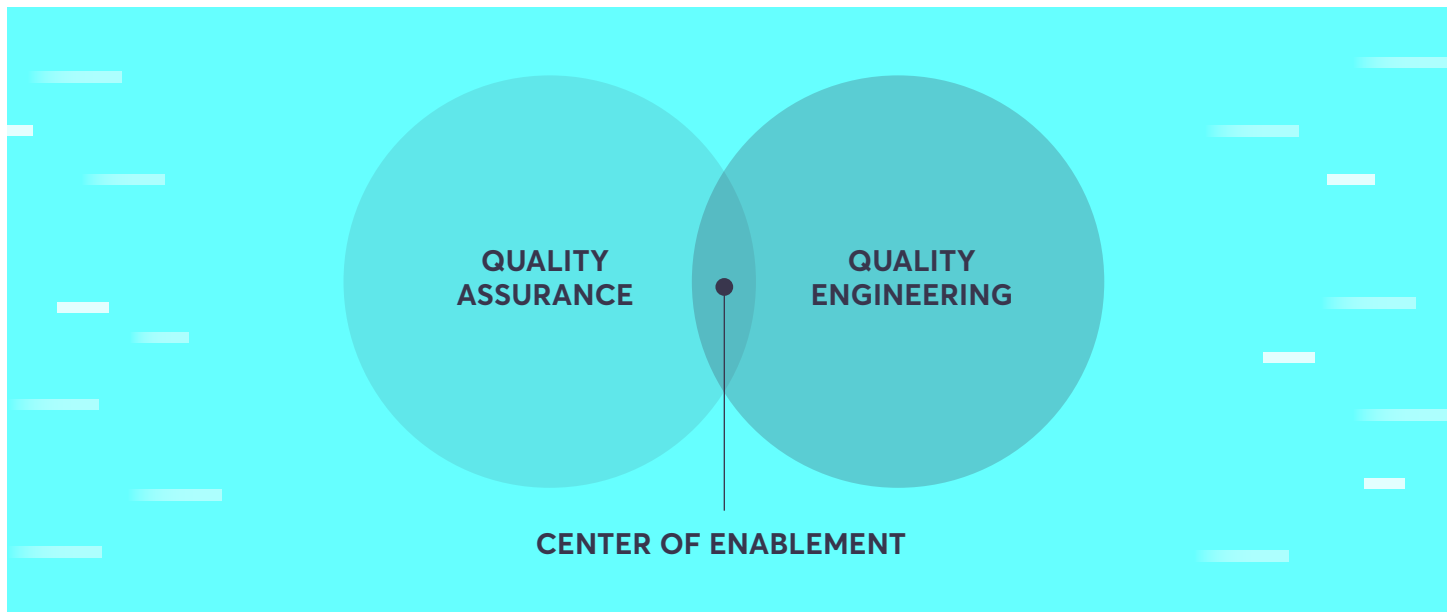
These are critical to successful continuous testing. Dashboards in real time. Must be automatic and entire team must have access. Feedback loops across the entire SDLC, not just production, should be used as a compass to help you navigate your continuous testing transformation.

Moving toward a continuous testing culture is a critical component to enabling DevOps, but it means committing to a new area of knowledge and practice. This requires some getting used to, but it is backed by solid science and practical knowledge. Underpinning the urgency of this type of improvement initiative is the provable fact that customers, in both the retail and B2B sectors, have an extremely low tolerance for poor experiences and will influence a company's success much more quickly and visibly.

Some of these new practices may naturally run counter to the existing culture, but such is the nature of change and innovation. Teams need to be introduced to these new practices and trained or assisted in managing the transition.

For example, when coding for a feature, there should be release notes embedded in that code, so that anyone can read it. Just as in the world of project management, create a plan that is good enough for someone else to take over as needed.

Quality assurance (QA) should evolve into quality engineering as a Center of Enablement. This is a place where teams build into the software the knowledge of how to test, deploy, monitor, and even heal itself so that application teams can leverage that within their scrums, without having to depend on external staff.



Everything should be treated as code so that it can be controlled through CI in a standard and scalable way. This requires teams to think about how it will be tested to ensure that, if the code is to be integrated into the source code repository, it will do what it is supposed to do and will not cause adverse effects in other parts of the application.

The people involved, developers and testers, must become part of this new culture. There should be a DevOps organization set up to hire or retrain people into more well-rounded engineers who can wear many hats. This could include a matrixed organization that allows the app development team to hire or use the skills it needs as it needs them. Change management with humans is as vital as the physical transformation to continuous testing.

From a platform standpoint, cloud becomes an enormous success factor. A continuous testing journey cannot be achieved with on-premises systems and legacy systems alone.

There comes a point early in the continuous testing journey where the leader and stakeholders must pause and ask themselves, “Where am I along the deployment timeline?” It is vital to first understand the imperative for continuous testing and then to adequately define it, but in preparation for launch, one must ask where the team currently stands.

## TWO CRITICAL POINTS TO BEAR IN MIND AT THIS POINT

- To always put the customer at the center of the change process and its resulting continuous testing activities; and
- To determine how to measure quality appropriately.

There are new techniques to adopt at this point, and in addition, there are barriers to achievement to identify and eliminate.

## TECHNIQUES TO ADOPT

### 1. IMPROVE THE RELATIONSHIP BETWEEN TESTER AND EACH DEVELOPER

Keep the teams small, encourage inter-team collaboration, and make reports easy to access and share online.

### 2. MAKE AUTOMATION A PRIORITY

Do not remove all manual testing tasks, but adopt an “automated first” mindset. Focus that on areas that will be run repeatedly. In essence, set up the plumbing first so that you can let the water flow later without any worry or concern about leaks. Map out your SDLC and identify automation opportunities.

### 3. GET SMALL

Break work down into smaller increments that are easier to test after coding and design. This makes it easier to automate the tests, and it also makes things much more easily deployable.

### 4. KEEP TRACK OF EVERYTHING

Use metrics and set pass-fail criteria. Continuous testing is all about immediately identifying if things are working or not, so make sure you can set that up easily.

### 5. GET THE RIGHT CONTINUOUS TESTING TOOLS

Find the tools that will help you to develop, test and analyze continuously. Pick the best-of-breed tools that work together in a way that incorporates easily into the working environment. Choose tools that have community-based support where everyone shares their challenges, solutions and interesting use cases.

### 6. DEVELOP A SYSTEM TO DISPLAY YOUR RESULTS

Do deep dives into results, because that’s how you will know if your code is working and where the gaps are. Define your KPIs and acceptance criteria and make them quantifiable. Create dashboards to track the KPIs, including a baseline and subsequent changes.

The transition to continuous testing can appear onerous, but is best achieved by starting with one small project, bringing it to successful fruition, and then building on other small projects. It's an iterative process where going for small wins and building upon them generates momentum and knowledge.

Failures will happen, but when they do, teams and leaders should be prepared to fail small and fail forward — take the time to learn from the experience.

Senior leadership driving accountability to the organization will make a substantial difference. An individual evangelist can only connect to 20%–30% of the population. For the rest of the organization, it's up to leadership to set expectations and hold the course.

The continuous testing transition will have its barriers, and these must be identified individually. They may include the inability to access and use open source. The work culture — not just people, but practices, too — might not be sufficiently prepared. There will likely not be enough time to start the implementation, training and transition toward continuous testing. This may be paired with inefficient leadership, and the existence of legacy applications.

The ultimate goal at this stage is to develop and deploy a continuous testing maturity model. That means establishing the right mindset in terms of attitude and aptitude. Build up test automation at the unit and API levels for functional, performance and security testing, and keep test automation at the UI level to a minimum, as scripts are very brittle as UI changes. Add a feedback loop that includes performance and user monitoring tools in pre-production environments as part of the pipeline. Full telemetry accelerates defect resolution.

All these capabilities must be fully integrated, collaborative and not siloed in any way. Companies must be able to accept and understand open source, since it allows people to experiment and fail faster without having to justify expenses. They should also ensure adequate access to test data, since this is one of the most significant barriers to completion.

Every project requires metrics to determine progress and quality, and to identify business value. This is no different for continuous testing. In fact, continuous testing could be called continuous validation, because you are constantly verifying that your app or service will provide (or is providing) the business value expected. Some of the overarching benchmarks include customer adoption, engagement and revenue. Teams should seek to understand and quantify the sentiment of end users and add them to a continuous feedback loop back to developers and the business.

You must also be able to track how many problems you have — how many defects in pre-production and production — and pair this against the speed at which you can deploy. As people write increasingly better code, there should be a ration — one that sees the number of defects drop against a constant release cadence, or better yet, against an increasing release cadence.

Take code coverage and required coverage (functionality coverage) and combine them then to track the overall functionality coverage.

Make sure to understand what quality means for your organization and set a baseline. If you don't know where you are starting from, you can't tell if you are improving. Use a KPI that establishes production quality.

Learn what kind of coverage is needed across unit tests and functional tests, and performance and security. It should no longer be thought of as functional and non-functional testing.

Measure the overarching time to production. The point of continuous testing is to reduce timelines to release higher quality releases that provide business value. Focus on increased releases while decreasing rate of production defects and condensing the time to market through automation.

Change people's minds to look beyond pass-fail by developing an aligned definition of done. Defects must be tracked in order to use the analysis to help with improvements.

You can measure continuous testing success by testing in the lower test environments, which can be done quickly because the tests are smaller. Recognize that testing can keep up with development and you can ship code at any point, not just at the end of the sprint.

In assessing and measuring quality versus quantity, factor in the following:



**TIME TO MARKET**



**VALUE STREAM ANALYSIS**



**TIMELINES & COSTS OF MANUAL TEST EXECUTION**



**REGRESSION TESTING**

Start shifting left to developers by democratizing testing tools and evolving testers into engineers.

Every leader could and should have their own view of what constitutes the most valuable continuous testing metrics. Ideal metrics to track include:



As you move from waterfall to agile, adoption requires test efficiency, ideally gained from automation and shift left. As you move toward increasingly shorter releases in your continuous testing journey, it can't just be about getting more automation, more shift left — it's about thinking very differently about how you test.

This calls for a competency around continuous testing that isn't something you do between CI and CD, but something that is a competency in that CI and CD live, enabled by a set of tools that enable these different ways of testing and different ways of thinking about testing.

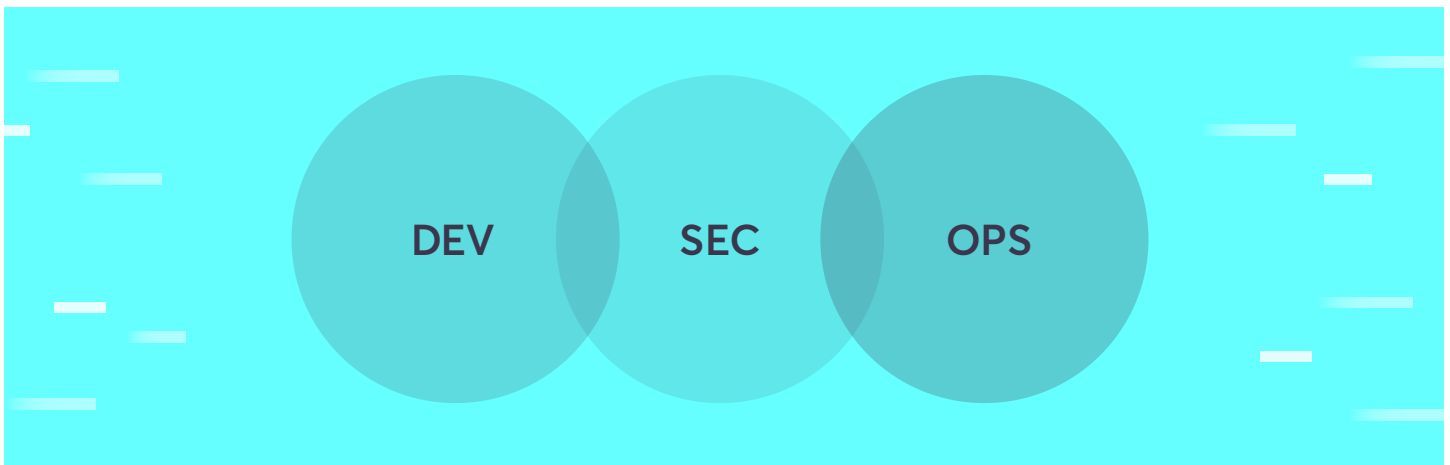
You need tools that help with planning, test plans and test automation, that then get implemented across the CI and CD implementations tool chains.

When strategizing a continuous testing transition, it is vital to factor in security, front and center. Security specialists will quite correctly point out that security has always been treated as something of an “end-of-the-line” concern as well as running in opposition to developers. As much as developers wish to get their product out the door, it is always the security people who say, “Not so fast.”

Security deserves to be placed in the continuous testing stream at the very start. It is a primary concern when identifying and confirming all aspects of the value chain, from inception to customer. Feedback loops will help to confirm whether the product is making the grade and enable the team to find and fix security-related issues as they arise.

Developers often unknowingly create security vulnerabilities. Continuously testing security is not just about policing those vulnerabilities; it's also about finding out what training they can bring to bear on making the developers better able to understand and become aware of the security of their code and not create those vulnerabilities in the first place. Security must become a partner, a trusted advisor and expert, giving developers training and tooling that can be measured downstream.

Security testing needs to happen at every step, so developers need to run security scans in order to find problems when they happen. Currently, security and testing are sitting on the wrong side of the wall from the developers. See this not as a conflict between security and quality, but more of a partnership of trying to figure this out together.



Clarity must be given in the requirements for what constitutes sufficient security. Security is everyone's job, but we need to determine when we're done. The development team must be enabled with the training to understand and the tools to measure against the requirements. Testing with these tools should be instrumented into our pipelines so that we assess ourselves continuously. As in any good DevOps process, security scanning is feedback to be used to measure current state and shift training left to upskill the team over time. In addition to more secure outcomes, there is much velocity to be gained by teaching developers to write it correctly the first time.

Accountability for security must also rest with a company's higher levels. From there, the questions and solutions cascade all the way down to the lowest levels of the organization all by themselves. If you just start at the lowest levels, you may get traction across a handful of teams, but to get real adoption across the company, you need executive sponsorship.

Finally, the security team itself must evolve its thinking to fulfill the mission of ensuring its members correctly see themselves as an integral part of IT.

Ultimately, a continuous testing plan must include future proofing to exist in an era of unabated change and speed. To future proof, you must have a strategy for all aspects of development. This means from coding to testing, including functional, performance and security. It also includes environment management, from test data generation to service virtualization.

It's imperative to maintain a clear line of sight to ensure tools used for continuous testing are going in that same pattern. The approach must align with where the company is going from a business strategy perspective, and IT needs to be part of that business strategy, not just playing a supporting role.

A company needs dedicated resources to get this done. It should not be given completely over to a third party, but nor should third parties be excluded. It is fine to use GSIs and GSPs to accomplish goals, but the company should have its own personnel involved in the process. Build upon your own expertise, but examine and solidify your relationships with third parties, because they might have a solution today — or tomorrow — that can be integrated into the pipeline.

Identify and assemble dedicated teams to build out the platforms that support the application pipeline and the core metrics aligned to continuous testing. This means growing from an organization that has some manual testers and a few engineers to one that has more capacity to build.

The continuous testing journey means the amount of black box testing will start decreasing while white box testing will become more prominent and relevant, and testers will need to be able to do both. This will demand a change in a CoE from being a Center of Excellence to a Center of Enablement, and this will further demand change in the makeup of the teams and the testers.

Big data will be critical for testing, especially getting refinement in code coverage for all types of tests. The market is exploding regarding coverage of the regression suite — especially in terms of developing a scientific way to know our regression tests achieve the appropriate levels of code coverage.

Companies must be prepared for change and be willing to go with the change. Some of the key milestones here include:

### AI PLAYING A LARGER ROLE

- Automating the automation, autogenerating testing on the fly will replace regression suites of tests, which will no longer be maintainable.
- Real-time observation of users gaining prominence, and consequently tests will be required to validate the observations.
- Record and replay being a thing of the past.
- There will also be a continued improved focus on getting the requirements correct and on new ways of analyzing them and identifying what's missing.

The executive leadership team must recognize how fast the world around them is changing. Work is being done with mobile technologies and cloud-based applications, and speed is its lifeblood. Everything is moving toward a seamless end-to-end customer experience. Cloud, DevOps, agile, big data — the end product for all these things is a seamless customer experience. You must decide how to apply that to testing. This is where the two worlds meet.



The user/consumer must perceive value that the business intended. This is where continuous testing becomes an innovation enabler.

The Internet of Things means devices are connected in all kinds of ways. Right now, continuous testing is challenging for browsers and APIs, but when you talk about physical and mobile devices and the Internet of Things, integration hubs that allow continuous testing will be the drivers of change. And wherever it is lacking, people will start to notice.

You must always be thinking three steps ahead in terms of things such as predictive analytics.

Top notch QA and QE folks are going to evolve quickly. Their skill levels will get higher and the skillset needed for continuous testing will be very different than today. Companies must put mobile and omnichannel first.

The most nimble companies stand to not only win market share through their adoption of a continuous testing culture; they also stand to attract the best people to help them there.

## SUMMARY

Transforming to continuous testing is a journey, just like agile and DevOps was and still is. Companies and their IT teams will encounter setbacks, which must be anticipated and accepted. This is why planning is essential at every step in the continuous testing journey.

Continuous testing is as much a culture as it is a technique. The entire team can learn from setbacks in a "fail-forward" approach — advanced from the walled silos of the past. All great developments and personal victories stand firm on a foundation in which knowledge binds with experience.

Although continuous testing has technology as its tools, it has human beings at its core. Once they become comfortable and confident, they will bring forth the necessary energy, vision and sheer practical ability.

From these achievements, momentum will be gathered, and this will provide organizations with sufficient lift to travel successfully across this new global economy.

