

THE FUTURE IS QUALITY ENGINEERING



WHITEPAPER

NEW AGE BUSINESSES NEED NEW AGE QUALITY ASSURANCE

Today's digital age has magnified the quality assurance challenge of what to test, and how much to test. Digital systems have added a layer of complexity to the quality assurance process as product strategy shifts from building standalone products to creating connected ecosystems, opening up opportunities as well as vulnerabilities.

What seems evident is that the digital era requires us to test for experience and not just functions or features. The focus has also shifted from the deliverable to “delivery” as continuous delivery and integration gain importance. There is use of feedback loop in learning from past cycles and building models using machine learning algorithms to manage these new aspects of testing.

Software delivery capability today is more about speed to cater to the release windows of days and hours, as against weeks and months. With Agile and DevOps setting in firmly, enterprises worldwide are making use of these methodologies for software development. DevOps builds software products in iterations, covering the end-to-end phases of the development lifecycle. This supports frequent delivery of system updates and software alike, enabling quality engineers to interfere at any point of time, and thereby work to improve the product quality. With frequent builds and quicker delivery, enterprises are witnessing a significant increase of productivity and enhanced customer experiences.

This paper discusses how the widespread adoption of digital technology has broken down legacy structures of software development and changed the paradigms of quality assurance, bringing in and relying more and more on Quality Engineering.

Introduction

The role of QA is evolving from assuring IT to assuring the business, and is accountable for end-to-end business process performance

With constant changes in the technology landscape and increasing competition, Digital Transformation has become the norm for enterprises worldwide. Organizations are investing a lot of effort and resources into digitalization, even as the need for new age Quality Assurance (QA) grows. From the typical three to four releases a year, some organizations are contemplating daily releases (think Amazon), which demands an entirely different ecosystem. Customers want software products to be delivered with first time readiness. Software quality is thus a must in times where a broken application may result in a loss of business opportunity. This demand, hence, also necessitates the need for continuous testing. The waterfall model has already become a thing of yesterday and organizations are rapidly moving towards Agile and DevOps adopting a continuous testing approach.

The role of QA is evolving from assuring IT to assuring the business, and is accountable for end-to-end business process performance rather than parts of the stack. Traditional quality management models have been built on the independence between development and testing, leading to an industry structure characterized by large horizontal shared services. The integration between development, QA, and IT operations has the potential to change how QA processes are designed, with acute downstream implications for people and technology.

The impact of Digital

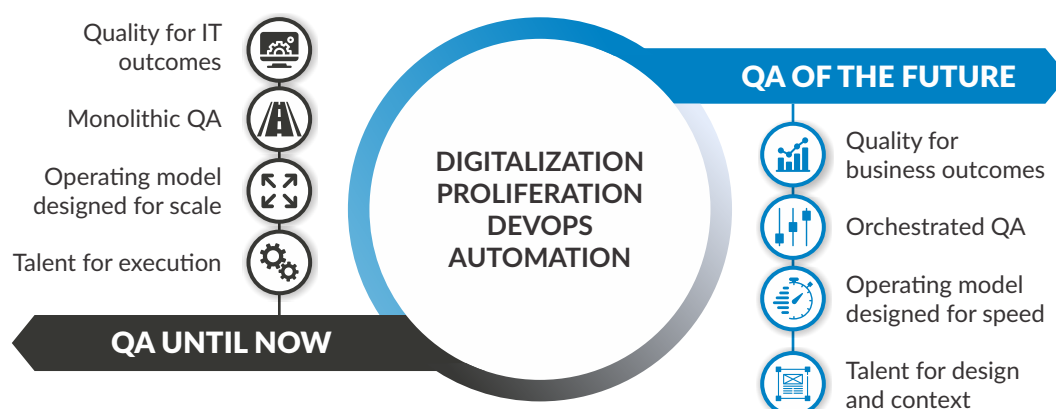
As digital platforms take center stage, IT teams and infrastructures have become critical aspects of business success. IT teams have been scrambling to meet the business's operational needs. Assuring the availability, performance and usability of digital assets — infrastructure, operations and customer-facing applications — is now a strategic concern for C-level executives across industries. QA is now more about ensuring positive business outcomes and improving stakeholder satisfaction, and less about testing the individual IT

What's Really Changed

components. This shift is largely due to the failure of traditional QA approaches to take into account the business processes and goals associated with digital transformation. To avoid this scenario and ensure that new solutions and services meet the quality standards driven by digital, enterprises must turn their attention to Quality Engineering.

The past decade has been a period of intense disruption for development and testing. Agile and DevOps approaches have merged, and QA has evolved into Quality Engineering (QE). Customer expectations demand a new level of disruption in software development. Therefore, quality as a function cannot simply be limited to ensuring that the software works but also ensuring that it provides differentiated user experiences in highly competitive, always-on digital markets.

To add to this, new technologies are rapidly evolving and becoming more complex. To adapt, QE must ensure continuous monitoring as cutting-edge industrial technologies are introduced with increasingly complex layers. Programmers need the agility to adapt to frequent and sudden shifts in core technologies, and monitoring code configuration to keep it up to date. The demand clearly is for tighter alignment and collaboration than traditionally has been seen between the business teams. To achieve seamless collaboration, enterprise customers are adopting a DevOps approach.



Bringing DevOps Into Play

Testing techniques have changed a lot with the advent of inclusive methodologies such as Agile and DevOps. Today, testing is no more an afterthought, but is an integral part of the Development Cycle, and a continuous process to ensure quality at speed, leaving behind the waterfall model.

With multiple releases happening almost every week, the quality engineer can no longer continue with a 'test after development' approach. QE professionals are leveraging DevOps and Agile methodology to ensure availability of all that it takes to ensure timely release of builds - be it devices, tools, data, hardware, performance related readiness, cloud solutions, Open source solution setups if any etc.

QE professionals are leveraging DevOps & Agile methodology to ensure availability of all that it takes to ensure timely release of builds.

AUTOMATION

- Integration with testing solutions so that the results appear automatically without a human trigger
- Test automation script development happens in parallel to feature development
- Stronger collaboration with the development team, especially with respect to feature specifics and object properties.
- The QE professional must also have experience in programming and should be able to write software if needed
- Optimization of test cases, and improving automation efficiency
- Evaluating and implementing new tools and technologies
- Creating custom automation solutions to address app specific use cases
- Create frameworks and accelerators that help scale quality engineers across multiple channels

TDD & BDD

- Test Driven Development(TDD) and Behavior Driven Development (BDD) urge the developer to unit test the code before committing for quality checks
- Quality engineer has to assure quality via API validation against UI driven test cases to ensure speed
- In the BDD approach, QA is integrated for acceptance outcomes
- BDD and TDD models demand closer interaction and collaboration of the QE function with the development and business teams to align goals and objectives

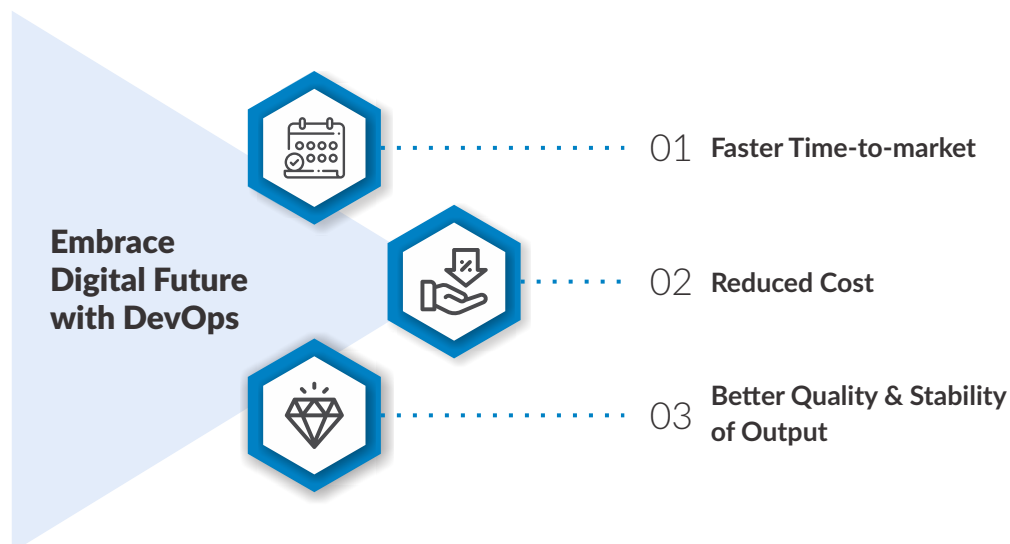
USER EXPERIENCE

- Any glitch or error in a product or service is met harshly by customers
- Digital platforms enable channels for instant and continuous feedback about the application quality, and enterprise offerings
- The QE team's focus this shifts to user experience (UX) in turn shifts the focus completely on User Experience (UX), which has become as important as the application functionality
- UX is a combination of factors such as UI, performance, process flow, ease of interaction with the MMI, device capabilities, device state and networks, accessibility and usability, etc.
- QE team should carefully look at user feedback and comments in to improve the quality of the software and the experience, because every error has the potential to directly impact revenue and reputation
- The QE teams must have a solid understanding of monitoring tools, application architecture, enterprise subsystems etc., and provide insights about the problem/ root cause, and the fastest way to solve it.

Adopting DevOps

A natural extension of the Agile software development revolution, DevOps is meant for any enterprise embracing their digital future and needing faster time-to-market, reduced costs, and better quality and stability of output.

DevOps has been the buzzword for the last few years and enterprises have only recently begun adopting it for continuous integration (CI) and continuous delivery (CD). DevOps takes a holistic view of the entire software delivery chain or the product life cycle. This involves using a set of tools that automate testing and streamline the development process thus helping teams work more quickly and efficiently. A natural extension of the Agile software development revolution, DevOps is meant for any enterprise embracing their digital future and needing faster time-to-market, reduced costs, and better quality and stability of output.



DevOps allows the unit tested code to be integrated to the repository continuously, while verifying the quality and completeness of the integrated code by performing build, test case execution, code quality analysis. The merged code is then continuously deployed to the environment automatically at pre-defined periods. This ensures that quality code is available in the repository throughout the development lifecycle, and that the latest available code is deployed to the testing environment.

DevOps practices encourage quality teams to collaborate, communicate and embrace automation methodologies with the sole purpose of deploying enhanced products to the consumer continually. DevOps product teams don't separate quality and deployment processes from the development process, but focus on a deep integration.

But watch out for the flakiness

CI requires all programmers to merge their working copies of a code to a shared pipeline several times a day. A flaky test – a phrase made popular by Google to describe possibly erroneous tests for checking a code – can block/delay development until it is spotted and resolved. The challenge lies in figuring out if you caused the test failure or if the test itself is flaky. Flaky tests are useless because they don't provide a reliable signal on the particular code being tested. Excessive flakiness can cause engineers to start losing faith in the entire CI process and starting pushing red builds anyways, in the hope that the build was red just due to flaky tests.

When a test fails and it's not an app issue, the QA team wants to dive into execution details of how the test was run. In particular, flaky tests are those that fail for no obvious reason – so the failure is usually assigned to the testing infrastructure. In order to deal with them, you should somehow record all the tests that are flaky. Upon a failure, you have to gather all related data. Logs, memory dumps, system current state or even screenshots in UI tests, that can help you investigate later what went wrong. To better understand these failures, extended debugging provides additional information to help users debug the tests they run. This helps them understand app issues and pinpoint the root cause of test failure by providing JavaScript Console log and networking logs with each automated test. With these data, developers gain insights into browser and networking issues to help them more quickly identify the root causes of test failures.

Why tests go flaky

Here are some common reasons a test could be flaky:

- **Concurrency**
Does your test run standalone or are there other threads that could affect the flow? In integration tests, maybe some batch jobs running in parallel or a background thread could also disrupt your test under execution. If you run your test against a live system, there could be other external requests affect the test.
- **Caching**
Do you cache data? We should consider caches during test development. Sometimes due to time manipulation, cache evictions or stale data, the outcome of the test may become unpredictable.
- **Cleanup state**
Are you keeping things clean? A good test should always setup its expected environment and always cleaning up any custom behavior to a vanilla state. This is one of most difficult flaky test to identify since it is not the one that fails, but consecutive tests could get affected.
- **Dynamic content**
Though you may want your tests to run fast, a test might need to wait for dynamic content to load first. Some asynchronous calls to load data can impose a delay. Keep in mind that tests would run much faster rather in case of human interaction.
- **Time bombs**
Does your test request for the current time in the same time zone? You should not make assumptions that your test will always run in the same time zone as developed. You need to measure time intervals accurately and keep in mind all factors that can cause anomalies.
- **Infrastructure issues**
Sometimes, it is not your test that is flaky. Your test might fail for external reasons. A bug in the testing framework, Selenium driver or a problem with that browser version could waste you a lot of time while trying to figure out what is wrong with your test. Other random incidents, like Continuous Integration (CI) node failures, network issues, database outage etc. are usually easier to spot.
- **Third-party systems**
Integration tests that do not run against a stubbed external environment, inevitably depend on third-party systems. You also need to verify external systems' correctness and stub all external systems when you check the integrity of your system.

The Sustainable Solution

As a software organization grows, well-written test suites are the first line of defense for maintaining product quality.

The long-term solution is to either fix or replace the flaky tests. If one developer cannot fix them, another one should try. If a test cannot be fixed, it should be deleted and written from scratch, preferably by somebody who was not involved in creating or using the flaky test. As a software organization grows, well-written test suites are the first line of defense for maintaining product quality. They document and enforce what the expected behavior of code is, which prevents one engineer from accidentally messing up another's work, or his own.

Test coverage tools can be used as a kind of a safety net, showing if some tests have been deleted without being adequately replaced. Of course, deleting and fixing flaky tests is a pretty aggressive measure, and rewriting tests can be time consuming. But not taking care of flaky tests leads to certain long-term test suite degradation.

Not being able to develop stable tests for some part of the code usually means one of these two things — either that something is wrong with the test and/or the testing approach, or that something is wrong with the code being tested. If we are reasonably certain that the tests are fine, it's time to take a deeper look at the code itself.

Conclusion

With continuous delivery, need for agility and short lead times, the pressure on quality engineers is huge as the focus shifts continuous testing. But with legacy systems and their complex interaction with new digital technologies, enhancing testing for speed and continuous delivery is easier said than done. Supporting digital innovation is the key agenda of the quality team today. In order to deliver value beyond the initial labor arbitrage and the incremental process improvement, enterprise QE needs to innovate at a pace that keeps up with rapid technology disruption and changing business models.

This calls for an approach that addresses critical components towards the delivery of a product. The next-generation of business technology solutions need a shorter time-to-market, are increasingly customized, and are created and delivered in shorter cycles using Agile methods. This is why QE makes sense and why focus should be on building quality into the process, right from the start.


Through engineering excellence, QE ensures that the methods employed in delivering software are consistent with the quality outcomes. The goal is zero defects in every unit of output well before the quality team steps in for testing. This involves the strict use of architecture quality attributes, rigorous test coverage, continuous validation, integration and deployment methods, and TDD/BDD approaches.

Enterprises willing to provide high-quality products and services to satisfy customer expectations must establish, manage and monitor the end-to-end quality and the integrity of their systems. QE begins at product inception and follows its entire lifecycle until the time its retired. Throughout the development and continuous improvement phases, it becomes critical for companies to maintain the kind of quality their users expect. It constitutes critical quality systems, and is an integral part of quality assurance.


About the author


Sujatha Sugumaran Head of Quality Engineering at Zuci Systems

Sujatha is an old hand in testing with more than 15 years of experience and numerous accolades in her name. She manages quality for variety of products in different phases of product life cycle across different domains and technology stack. Her expertise include test automation and performance testing as well as management of an extremely proficient quality engineering team at Zuci Systems.

 Headquarters – Chennai, India
+91 (44) 49525020

 www.zucisystems.com

 Office – Chicago, U.S.
+1 (214) 230 9824

 sales@zucisystems.com

