ZUCI SYSTEMS

WHITE PAPER

# A COMPLETE GUIDE TO CONTINUOUS TESTING

A customer of ours recently said,

“it ain't about only test automation and its value for QA but what we want is value for everyone from it”

Continuous testing a.k.a shift-left testing, involves running tests frequently, as needed and early in the software lifecycle.

Done right, continuous testing can provide quick and useful feedback on the health of the application under test which can be used to determine if the application can move forward in the delivery pipeline.

**This whitepaper provides a detailed insight into continuous testing and how to achieve it.**

# CONTINUOUS TESTING AN INTRODUCTION

The idea behind continuous testing is to perform tests more early and more often throughout the CI/CD pipeline.

Developers start the Continuous Integration process by checking code into a shared repository many times during the day. Each check-in is then verified by an automated build process that shares immediate feedback to the developers.

In consequence, they run regression tests each night to make sure any changes made during the day did not break something else.

Continuous delivery is a further extension of CI. Continuous delivery means that once all tests have passed, updates can be pushed directly to production with confidence.

The goal of continuous testing in a delivery pipeline is always the same: to prove by successive levels of testing that the code is of release- quality by applying right level of automation at each stage.

## IS TEST AUTOMATION AND CONTINUOUS TESTING THE SAME?

While one cannot implement Continuous Testing without Test Automation, either of them is not synonyms. Test Automation is the use of special software designed to produce a set of pass/fail data points correlated to user stories or application requirements.

Continuous testing, on the other hand, is the process of executing automated tests as part of the software delivery pipeline to obtain immediate feedback on the business risks associated with a software release candidate." So, what is the relation between the two?

In Test Automation, once the code gets merged with the repository, then automation test scripts are written and developed. The binaries are tested against these scripts automatically using the automation test suites.

Whereas in Continuous Testing, teams have a stable automation cycle (test scripts are written already) even before the coding begins. So, after the code gets integrated, aka continuous integration, the automation tests are automatically run one after another leading to production – hence the term Continuous Testing.


The primary advantage of continuous testing over automated testing is that as soon as any code is merged to the central repository, the process of integrating and validating begins, and feedback is received quickly. Thus, no breaks exist between integration, testing, and feedback.

With organizations adopting DevOps and Agile mindsets more than ever today, test effectiveness becomes a crucial factor. The practice of running regression test suites manually each time the team gets a new build will no more be sustainable for a successful release. The only way to deliver quality software at speed is by applying the right level of automation at each stage of software development.

To accomplish this, teams need a combination of Test Automation and manual exploratory testing, both running in a continuous pattern.

# HOW TO GET STARTED WITH CONTINUOUS TESTING?

## A THE BASICS

### TEST AUTOMATION

To begin with, automating all tests is not feasible, and the cost of trying to do so eventually counterbalance the advantages. It is vital to identify automatable tests, the ones where it,

- Has diverse issues
- Went through multiple regressions
- Picked up many customer complaints
- The probability of failure is high

The breaking down of these right sets of test cases should involve an entire team from business to development, to test, to operations, and support. Each will bring a perspective on where things could go wrong and where to implement test automation. For instance, it could be applied for complex data sets, integrations with a third-party system, business logic validation cases.

Primarily, Unit tests, Functional tests, Performance tests & Security tests suit well for automation.

## SHIFT-LEFT

Over the period, people realized that the costs involved in finding defects at the end of SDLC are significant than finding it earlier. And, the time taken on fixing it becomes a blockade in launching the product on time.

What had come as a result of this realization is "Shift-Left" testing; test earlier and more often.

Imagine a situation where functional integration and Performance delays contribute to the highest number of defects in the application.

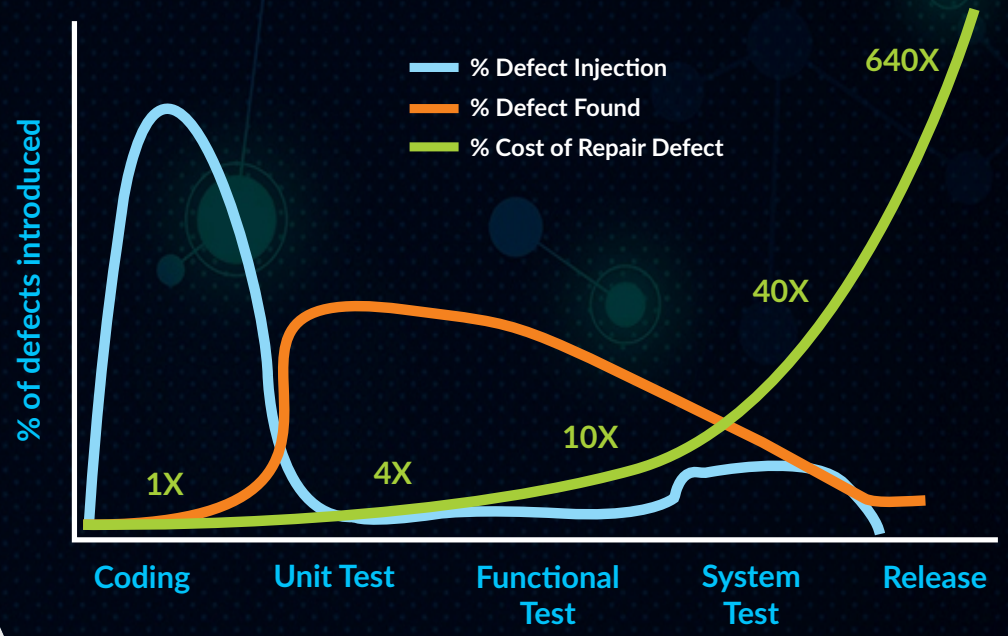
As the first step to shift-left, teams will do a thorough analysis of the defect data. Subsequently, they collaborate and work together to perform these tests earlier, right after the code is built and not wait until just before it passes through the deployment pipeline and reach production. This way, they can contain defects much early in the SDLC.

For the most part, unit tests get along well with this shift-left continuous testing. Still, shifting left the execution of later-stage functional tests is often difficult due to external system dependencies.

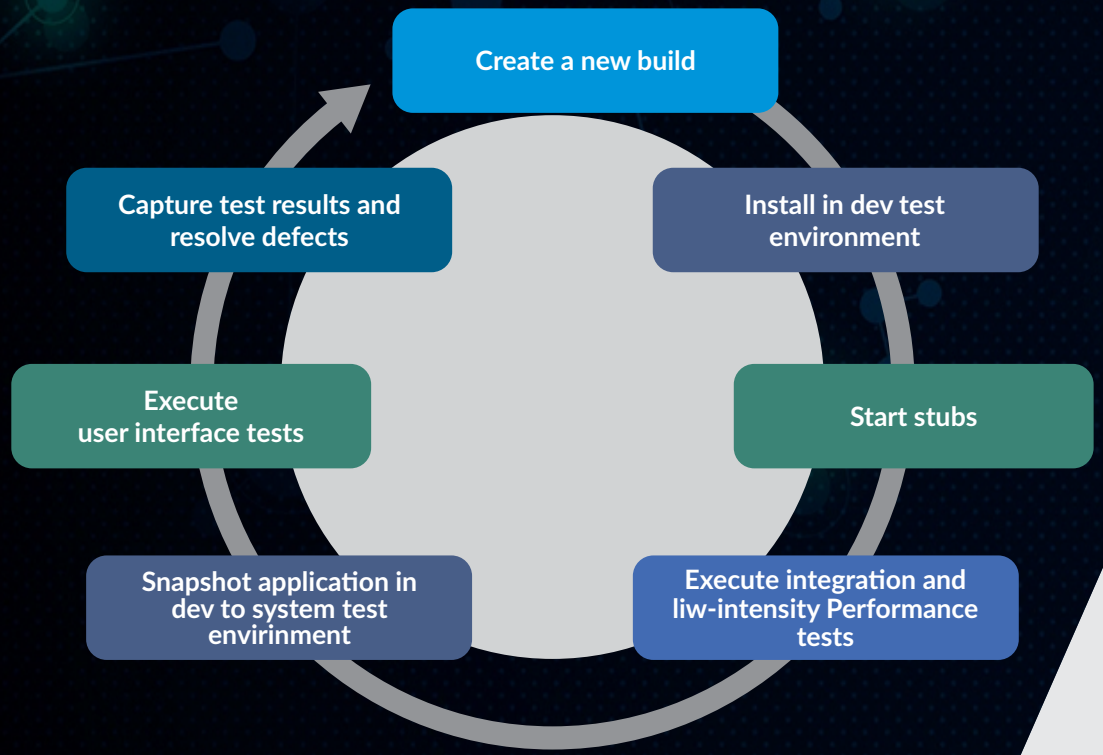
This is where teams can leverage service virtualization to enable continuous testing.

Jones, capers

**Applied Software Measurement:**
**Global Analysis of Productivity and Quality**



Series of Shift-left Test Execution using Service Virtualization




## SERVICE VIRTUALIZATION

Based on the interfaces, developers and testers can create virtual services or stubs that might have limited availability, such as mainframes, third-party services. The virtually created stubs simulate other systems and allow the developers and testers to shift the test execution to left i.e., development desk – as soon as the code is built.

The teams then can make fair use of test automation to be performed on these functional integration tests, and performance tests, since those are where most of the significant production defects came from.

In the context of performance testing, service virtualization allows teams to run all kinds of what-if scenarios and push the system as hard as possible and beyond and do it as early as possible.

The figure shows a series of shift-left test execution using service virtualization.

Similarly, we can test systems for security-related issues, also in a continuous pattern. The next part of the paper discusses running tests in continuous testing mode..

# B WHAT TESTS CAN RUN AS CONTINUOUS TESTS

## SECURITY TESTING

Recent security fails have driven businesses and governments to wake up to the seriousness of such impacts, which resulted in treating them as a national emergency. A few include,

- Capital one's recent data breach costing $100-150 million – hackers gained access to 140,000 Social Security numbers, 1 million Canadian Social Insurance numbers and 80,000 bank account numbers
- Reports say Russian hackers allegedly hacked into the Democratic National Convention (DNC) and manipulated the election in favor of Donald Trump

A substantial proportion of security tests are necessarily checks that known vulnerabilities have not been introduced in the system, and most of them lend themselves superbly to automation and can be run in a continuous fashion.

Examples include verifying security features such as authentication and logout and, testing known weaknesses such as lack of the HttpOnly flag on session cookies, or use of known weak SSL suites and ciphers.

Any testing framework can be used to run these tests, but in the true spirit of DevSecOps, teams should choose where they feel comfortable using and one that easily integrates with CI/CD server. For example, look at this security testing requirement from https://devops.com using BDD language type called Gherkin, which is as much as saying it in plain english:

```
Scenario: The application should not contain SQL Injection vulnerabilities
Meta: @id scan_sql_injection
Given a scanner with all policies disabled
And the URL regular expressions listed in the file: tables/exclude_urls.table are excluded from the scanner
And the SQL-Injection policy is enabled
And the attack strength is set to High
And the alert threshold is set to Medium
When the scanner is run
And false positives described in: tables/false_positives.table are removed
Then no Medium or higher risk vulnerabilities should be present
```

Example of non-functional security requirements that can be identified early and incorporated in the Continuous Delivery pipeline, automated and run as Continuous Tests.

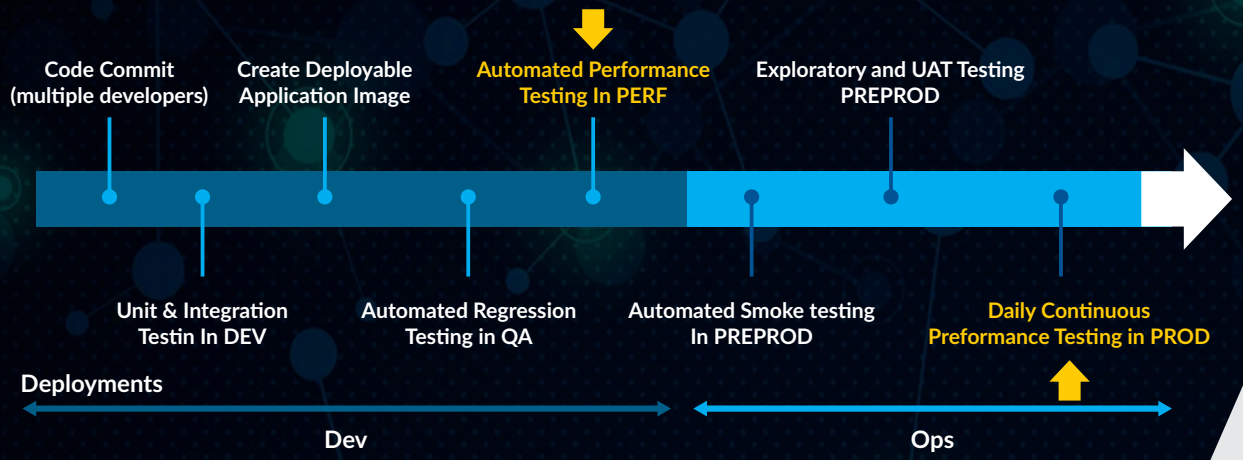
More than half of all companies are implementing continuous integration and continuous delivery practices, direct adoption of DevOps, but without continuous performance testing, their success could be limited.

NASA has shown us once and for all how slicing and dicing of each component and running performance tests on each of them help them be a forerunner in their field. By the time a system is ready for launch, engineers would get a complete picture of the system's ability to perform under different environments.

The same can be correlated with the continuous performance testing of software systems.

Continuous performance testing involves the deployment of low volume load tests on developer's check-ins in a DevOps CI/CD pipeline and capture performance metrics of the critical application user flows, API endpoints, and database queries of the systems. They are run daily in a customer-facing environment or similar environments.

## A representation of a continuous performance testing in a CI/CD pipeline




## PERFORMANCE TESTING

### Three steps to Continuous performance testing:

- Bridge the gap between development and operations; this way, the pipeline becomes more efficient, giving everyone in the team quantifiable insights into what is happening in every stage leading up to production.
- Break and test as often and early as possible. This should be done at the API layer so that different services are tested at the same time but independently of one another.
- Service virtualization to more easily test what-if scenarios for dependent services.

Continuous performance testing makes room for key metrics such as latency, throughput, bandwidth, Time to First Byte (TTFB), application Http response codes (404, 500), CPU, Memory, Disk I/O.

## C WHAT TOOLS SHOULD I USE

As mentioned in the paper earlier, continuous testing implies that automation is integrated into the CI/CD pipeline. CI/CD tools enable continuous testing right from the design phase until the last stages of production monitoring.

There are a lot of CI/CD and automation tools in the market to choose from. Let's take an example of a Continuous Integration tool, Circle CI.

**Test team at Zuci, in the recent past leveraged Circle CI, for one of our customers and helped them ship quality code at speed.**

### VCS Integration

CircleCI integrates with GitHub, GitHub Enterprise, and Bitbucket.
Every time you commit code, CircleCI creates a build

### Automated Testing

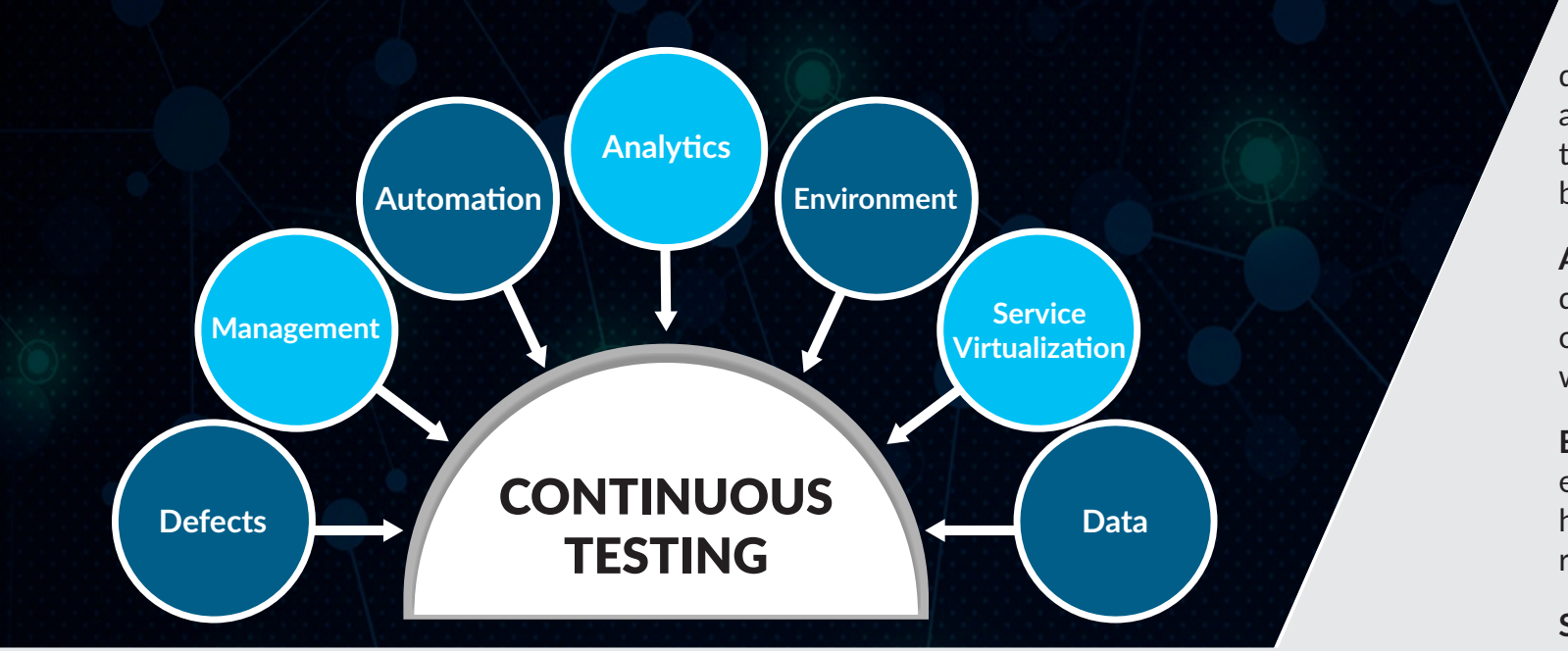
CircleCI automatically runs your build in a clean container or virtual machine, allowing you to test every commit.

### Notifications

Your team is notified if a build fails so issues can be fixed quickly.

### Automated Deployment

Passing builds are deployed to various environments so your product goes to market faster.




## D A FRAMEWORKFOR CONTINUOUS TESTING

**The continuous testing framework is made of the integral components viz., people, process, tools, and time**

**Defects:** Many a time, the defect lies not just in the code itself, but with the requirements, design, or even the test. Occasionally, defects stem also from the test environment, test data, test script itself, or some combination of these things. Thereupon, having an efficient defect tracking system in place is essential in any SDLC.

**Management:** Test management, when done right, helps both teams and management understand at a glance if tests are passing, failing, or blocked. Compelling test management practices are much similar to running tests in PODSCORB fashion.

**Automation:** Successful creation of a robust test automation framework involves shared vision, requirements, design, even coding in some instances, and finally validation, much like developing a software development project. However, if not effectively managed, automation can jeopardize the quality and increase risk. Firms must track the right quality metrics to ensure these risks don't negate the benefits of automation.

**Analytics:** Test analytics and insights expands testers and developers' vision on code change impact analysis and regression changes. It answers questions like: Which tests should we run, and when? Why are we running them?

**Environment:** Automatic creation and configuration of test environment cut short the time spent on testing new builds from hours to minutes. Getting hold of a well-managed test environment reduces false positives a great deal.

**Service Virtualization**: By creating virtual services from known and agreed-upon interfaces, developers and testers write code and test against the same interface, even when that dependent system is not available. It allows testing of scenarios that might not be readily tested with a live system – for example: **Exceptions and errors I Missing data I Delayed response times I Large volumes of data or users**

**Data:** Testing efforts and effectiveness would become result in vain if there isn't the right set of data. Adopting data that is as production-like possible provides better coverage with more scenarios.

Working together, a set of effective testing practices enables whole teams, not just testers, to improve quality and decrease the time it takes to deliver new capabilities. These practices remove dependencies & bottlenecks to allow testing earlier in the lifecycle and enable continuous testing - without the prohibitive investments typically required by traditional testing environments.

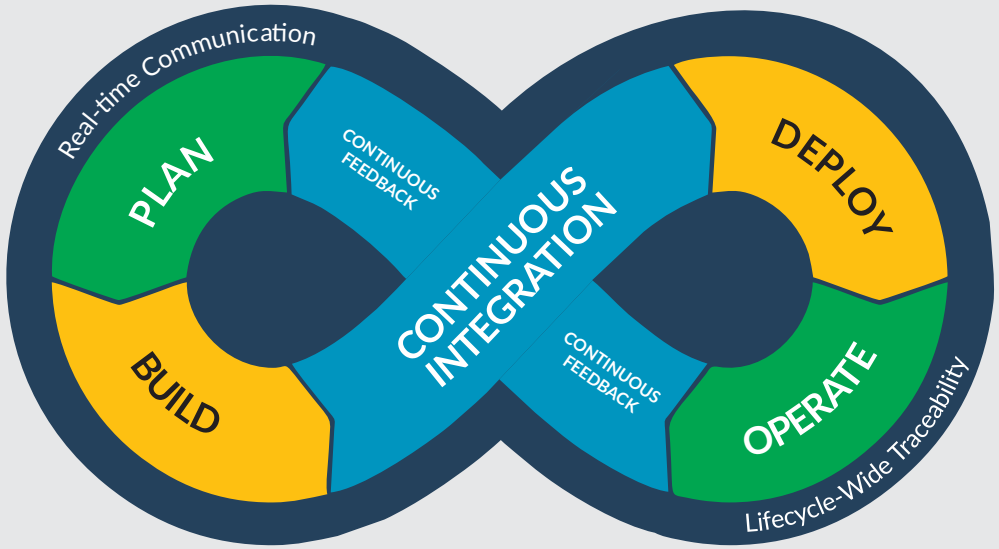
# CONTINUOUS TESTING & DevOps WHAT'S THE CONNECTION

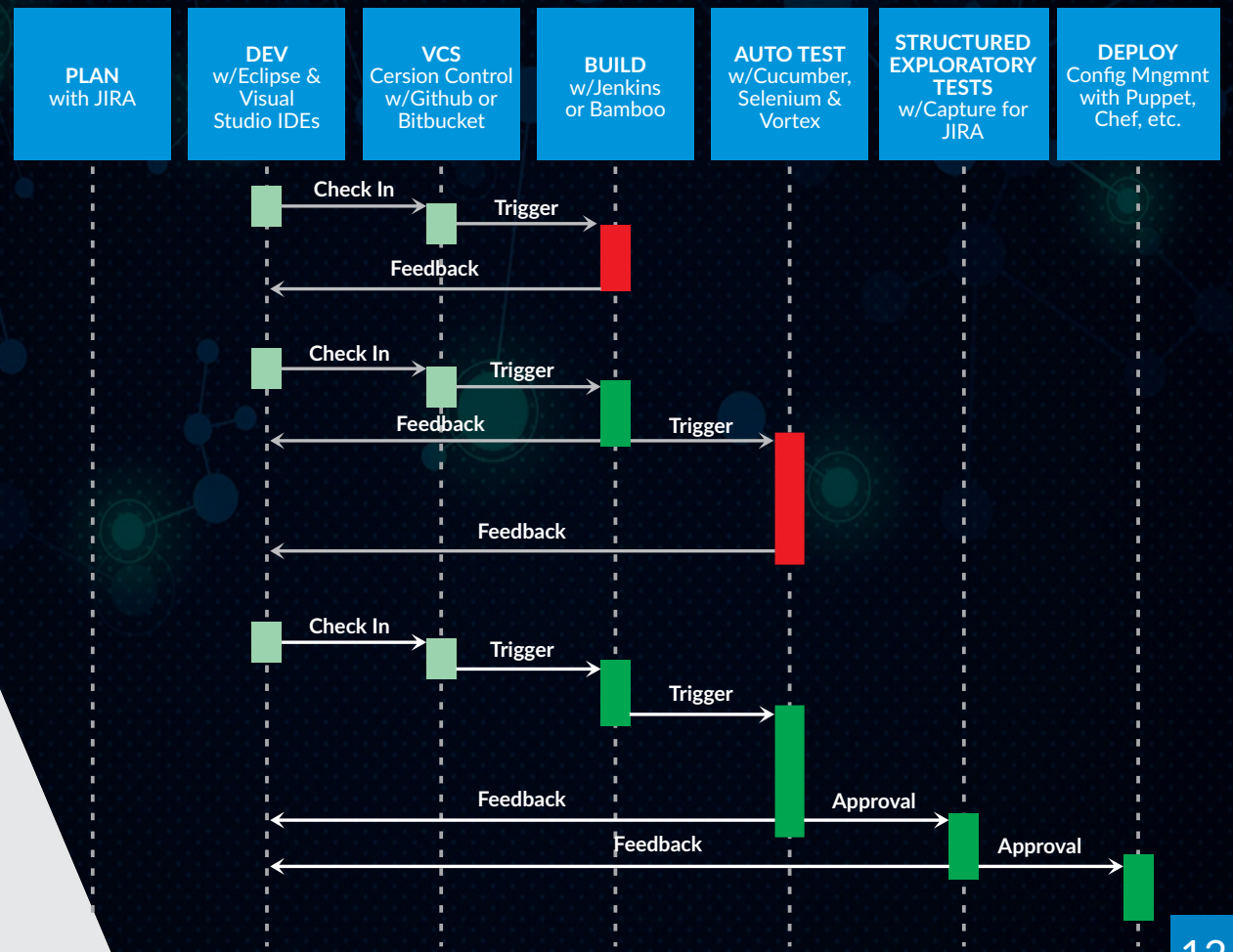
DevOps is more than adopting the right set of tools; it's a mindset that breeds testing at every stage of the SDLC.

Continuous testing is the key to making this happen because it weaves testing activities into every part of the software design, development, and deployment processes.

With the advent of DevOps,  came the continuous everything; Continuous Integration, Continuous Delivery, and Continuous Testing.

The image below shows an example DevOps pipeline that incorporates continuous testing during check-ins, continuous integration, and continuous delivery.



Forrester reports reveals that continuous testing, done properly, is a key differentiator between DevOps leaders and laggards.

### Five core practices separate successful DevOps leaders from laggards

1. Allocate proper testing budgets and focus on upgrading their testing skills
2. Implement Continuous Testing to meet the demands of release frequency and support Continuous Delivery
3. Include testers as part of their integrated delivery teams
4. Automate end-to-end functional testing
5. Shift-left testing to earlier in the development life cycle

## CLOSING THOUGHTS

Given the pace of modern application delivery, Organizations need to move away from traditional operating siloes and learn from the fail-forward approach to provide customers with quality digital experiences faster with Continuous Testing.

Successful adoption of continuous testing requires changes in process and approach. Companies and their IT teams will have to foresee setbacks along the way and accept it. This is why having a reliable continuous testing approach and framework is essential at every step of the software development journey.

Headquarters – Chennai, India
+91 (44) 49525020

Office – Chicago, U.S.
+1 (331) 903 – 5007

www.zucisystems.com

sales@zucisystems.com

ZUCI SYSTEMS