Automated Testing in DevOps

Rajesh Sachdeva

Optum technology Inc.

Rajesh_sachdeva@optum.com

Abstract

Can you test your application without it being deployed to a static environment like development, test, stage, etc. and execute all your tests (no matter what the size) as many times you want in a day? **The answer is *Yes.*** In a DevOps or Continuous Delivery Model (CDM), automated tests (unit, smoke, regression, API, business logic tests, E2E, etc.) are hooked up with an application build deployment pipeline and are executed in the background by a Continuous Integration (CI) server. The developers or quality engineers write a test first**,** but are not forced to execute the tests manually. The tests are run automatically by a continuous integration tool. If builds are happening 50 times a day, then the CI server can execute your regression suite of any size 50 times a day**,** and promote the code to higher environments automatically.

At Optum Technology Inc., for one of our programs we execute more than 1.5 million tests per day. To enable this, our QC (Quality Control) team has transformed into a ***quality engineering*** team. We are now capable of more frequent releases to production with no testing phase. This paper details how our testing pyramid differs from the legacy waterfall model and what advanced technology and tools are available in the market that helped us to create the test environment on the fly, in the cloud or a transient environment, before deploying to static environments. Docker and Mesos are such examples which provide this flexibility; they can spin up an app server, DB server and deploy the code to the transient environment within a few seconds. You can run thousands of tests in minutes for each build. This technique potentially reduces the time wasted resulting from manual test execution in physical environments, by eliminating the entire testing phase in a software development cycle in scaled agile delivery. The testing feedback is given to the developer in the same build cycle, without having to deploy the code to the development environment.

Biography

Rajesh is working as a Senior Project Manager at Optum Technology Inc and is responsible for real-time test automation in DevOps, Microservices API Certification and testing in a transient environment (Docker, Mesos, Selenium Grid), Agile Transformation, Acceptance Test Driven Development (ATDD) ATDD, and 100% Real Test Automation (Spock, Geb, and Groovy). He is involved in designing solutions/strategies for testing consultancy and services, QA cost optimization and quality Index Implementations, test strategy development, regulatory compliance roadmap formulation, and current state testing assessments. He has extensive experience in testing of Healthcare Payer Systems applications & Products, Web Portal Testing, Regression Optimization, Test Automation; Test Optimization Technique Implementations like OATS, RBT, DTT, etc., and Test Automation of Mobile and Cross Browser Testing. His initiatives around real-time automation reducing QA cost, improving productivity, and optimizing resources, has greatly helped quality improvements in United Health Group IT deliveries.

Rajesh has a Bachelor degree in Economics (Honors) in 1998 and Masters in Computer Applications from Kurukshetra University, Kurukshetra, India (2001). He has 14 Years of professional experience in software quality assurance and engineering for banking and healthcare domain organizations.

1 Introduction

IT in the healthcare industry is complex, especially when you are working on claims adjudication systems. Currently, most of the IT delivery is done in releases (monthly, quarterly, or longer) due to the nature of complex healthcare systems. Our application team is tasked with a mission to develop a new healthcare administration platform with advanced software capabilities. On Day 1, the goal for the project team is to support daily releases. In addition, the project should follow a Continuous Integration & Continuous Delivery (CI/CD) model, move away from silo divisions of development, QA, SA & Operations and implement DevOps.

For this program, there is no separate testing phase. Testing effort is spread across the entire development cycle, not isolated to a single test phase. Our testing is done at build level on a cloud environment using automated build/deploy/test; with 100% automated testing of all test phases, and regression testing around the clock. A healthy test pyramid is developed with no testing on static environments.

Our quality assurance team transformed itself by adapting engineering practices (Figure 1) over the last 18 months. To achieve the business goals, the IT team had to completely redefine our current software engineering techniques, process, and skill sets.

Below are the key business objectives met by our software development team:

1) Innovate and modernize the technology
2) Increase speed to market through frequent releases (quarterly to bi-weekly)
3) Reduce technology cost by almost 50%
4) Improve customer experience

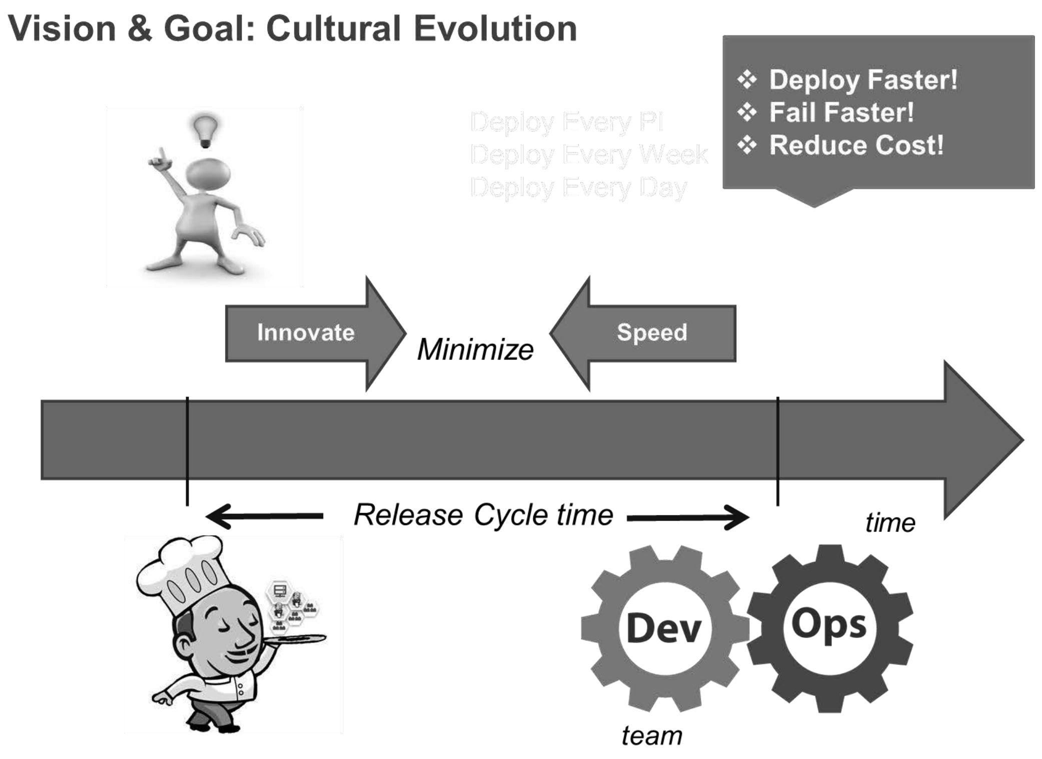


Figure: 1

2 Transformation Journey

We started the DevOps journey in early 2015. The Quality Assurance (QA) team members had experience in automated UI testing in the healthcare domain. Keeping in mind the goal, the first thing we changed was our hiring strategy. We started looking for *quality engineers* (rather than manual testers), who possessed programming and coding skills, and understand continuous integration process. Acceptance Testing Driven Development (ATDD) was chosen as a delivery model. The existing team started evaluating the automated testing frameworks available in the organization. The initial assessment criterion was that the automated test framework should support CI/CD and any testing artifacts (usually scripts) should be well integrated with application code. This kind of framework allows tests to be executed with each build and deployment.

Having manual testers on the team, we decided first to choose an automated test framework making script writing easy. FitNesse with Selenium was the test framework selected for this project for UI testing. It is Open Source, and integrates well with CI servers. The team developed almost 1000 tests and we were able to execute them with each build. But due to certain limitations in the framework such as an inability to write conditional statements, inability to loop, we decided to retire this framework and look for alternate code driven frameworks to enable writing more robust tests and provide CI/CD delivery support.

The development team was already using the Spock framework for unit tests which lent itself well for business logic and UI testing. A POC was done with GEB (a browser automation solution) + Selenium + Spock framework, designed to support our fast delivery requirements.

In the next six months, the team was able to create more than 40K+ unit tests, 8K+ UI tests, and 1000+ API tests. We moved to bi-weekly releases to production and were able to execute all the tests in the Dockerize environment by using Mesos in under 30 minutes. But still we were missing a thick layer of business logic integration tests. With more than 200 engineers constantly changing the application code, the corresponding UI tests were very difficult to maintain.

As a team, we decided to move away from UI tests and focus more on building the integration tests framework to test complex business scenarios which added much value to the project compared to the UI focused testing. At the same time, a non-functional testing strategy was also finalized and we started in-sprint automation and performance testing.

Our software development team continued to evolve, adapting new software techniques and maintaining agility. This has enabled our business to compete well in the market place.

The below Figure- 2 represents our transformation journey to CI/CD and DevOps.

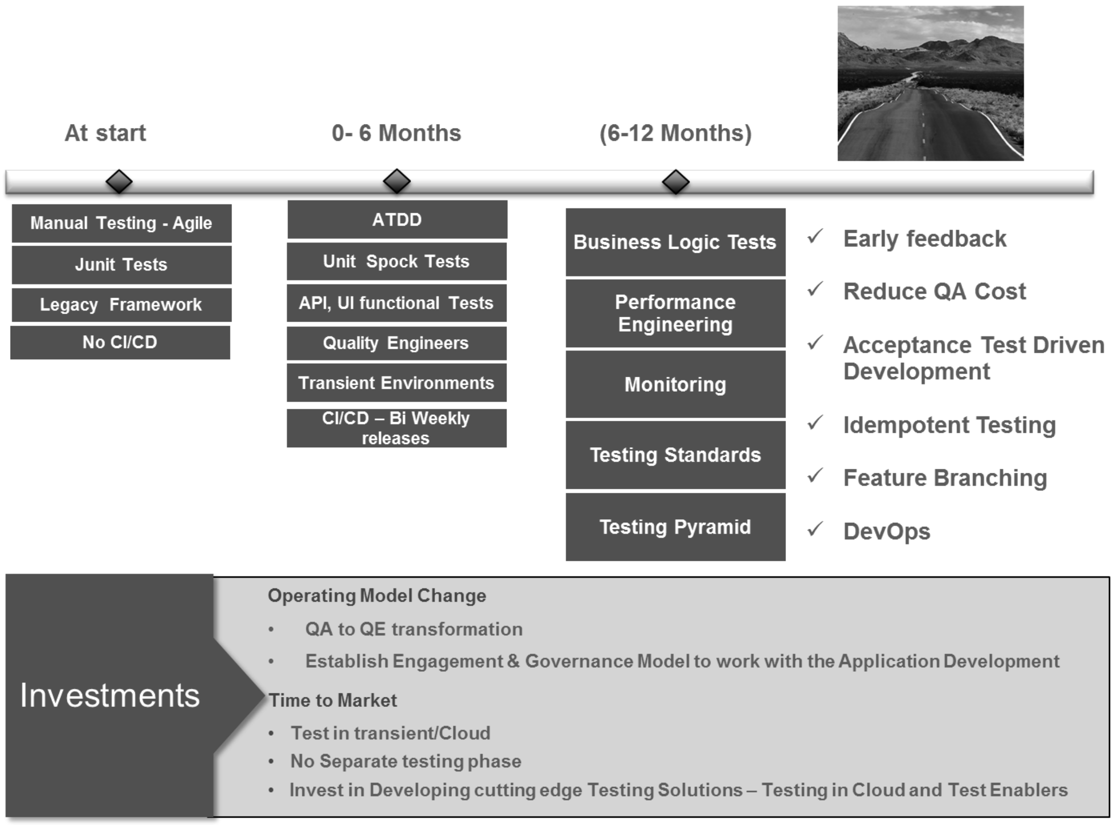


Figure: 2

3 Quality Control to Quality Engineering

A software development team cannot attain the DevOps delivery model without reliable test automation. Test automation done in the legacy world is completely different from the CI/CD model. In the latter model, automated tests have to be written at the same time code is developed. The testing artifacts need to not only be part of the application code, but be well integrated into it. Code coverage is measured at each build level and we used the Cobertura tool to measure it.

We established an engagement and governance model to work with the application development team. We had to change the culture and mindset of our existing team. The goal for the Quality Engineering (QE) team was to start writing the automated tests within the same sprint. Figure 3 explains the key reasons behind our team's move from a quality control mindset to adapting engineering best practices.

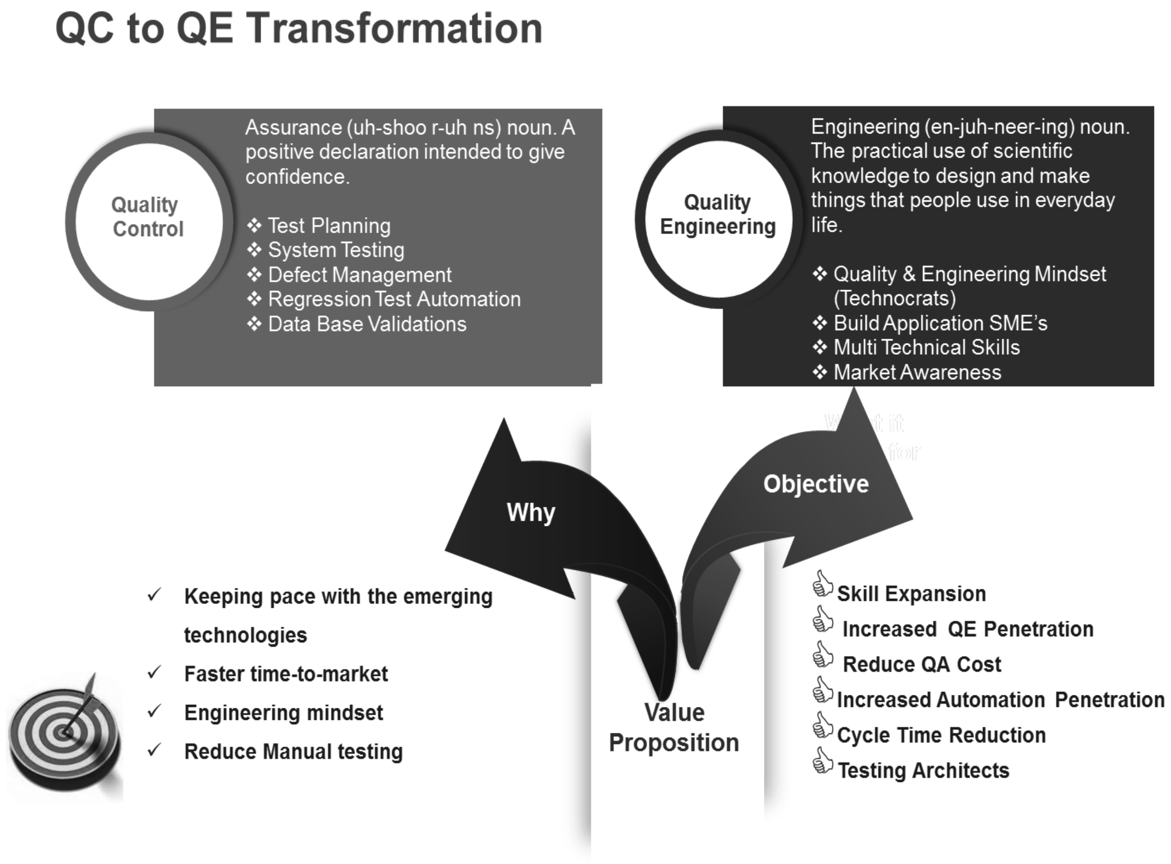

QC to QE Transformation
Quality Control
Assurance (uh-shoo r-uh ns) noun. A positive declaration intended to give confidence.
❖ Test Planning
❖ System Testing
❖ Defect Management
❖ Regression Test Automation
❖ Data Base Validations
Quality Engineering
Engineering (en-juh-neer-ing) noun. The practical use of scientific knowledge to design and make things that people use in everyday life.
❖ Quality & Engineering Mindset (Technocrats)
❖ Build Application SME's
❖ Multi Technical Skills
❖ Market Awareness
Why
Objective
Value Proposition
✓ Keeping pace with the emerging technologies
✓ Faster time-to-market
✓ Engineering mindset
✓ Reduce Manual testing
Skill Expansion
Increased QE Penetration
Reduce QA Cost
Increased Automation Penetration
Cycle Time Reduction
Testing Architects

Figure: 3

4 Role Transformation

In a DevOps delivery model, the QA roles need to be redefined. The QA team has to partner with the development and business organizations, leveraging people, process and technology to drive speed and efficiency through a new role and with a new automation framework.

ATDD CI adoption has introduced a need to reinvent the QA role from quality assurance tester to quality engineer. This paves the way to introduce the new Quality Engineer/Software Development Engineer in Test (QE/SDET) role into the mix making QA empowered, enriched, and deeply ingrained in the software development process. The QE/SDET role is vital to deliver maximum value and customer satisfaction more efficiently and effectively.

SDETs are development testers that can drive better collaboration with the developers and business stakeholders within scrum teams. They can code, design, develop, and maintain test code and automation frameworks. The SDETs develop code and build tools to test product code and are instrumental in driving continuous integration with ATDD.

SDETs help organizations shift QA to the left in the development cycle. They also enable fully automated testing from day one, by adding development expertise to complement existing Subject Matter Experts

(SMEs) in QA. They accomplish this by partnering with the developers in driving test automation at the UI and API levels.

Testing roles are categorized into four areas**:**

- Automation Engineer - an agile, entry level role, academy (university) resourced providing automated testing under the guidance of senior engineers
- Software Developer in Test - review any automation frameworks that may already exist, setup acceptance test frameworks, and work closely with the SME and developer in support of ATDD
- Quality Engineer/Subject Matter Expert – understands business goals, application constraints, integrations, E2E scenarios; works with SDETs and Business Analyst (BAs on achieving quality goals
- Quality Engineer in Non Functional Testing (NFR) addresses non-functional requirements and works with SME and SDET to integrate within ATDD-CI framework to validate NFRs.

We followed a multi-pronged approach to develop the community of Quality Engineers:

- ✓ Initiated a monthly Quality Engineers “Open Forum” meeting, open to all QEs in the organization to share knowledge and best practices
- ✓ We hired Quality Engineers for most open requisitions to inorganically seed this talent into high visibility programs implementing ATTD
- ✓ Identified existing resources who were already playing the role of QEs and formalized their role under an QE CoE.
- ✓ Trained over 200 QA resources, from our existing staff, on Quality Engineering skills such as programming languages, Selenium, Gatling, Groovy, CI/CD concepts, data analytics and skills required to become SDETs.

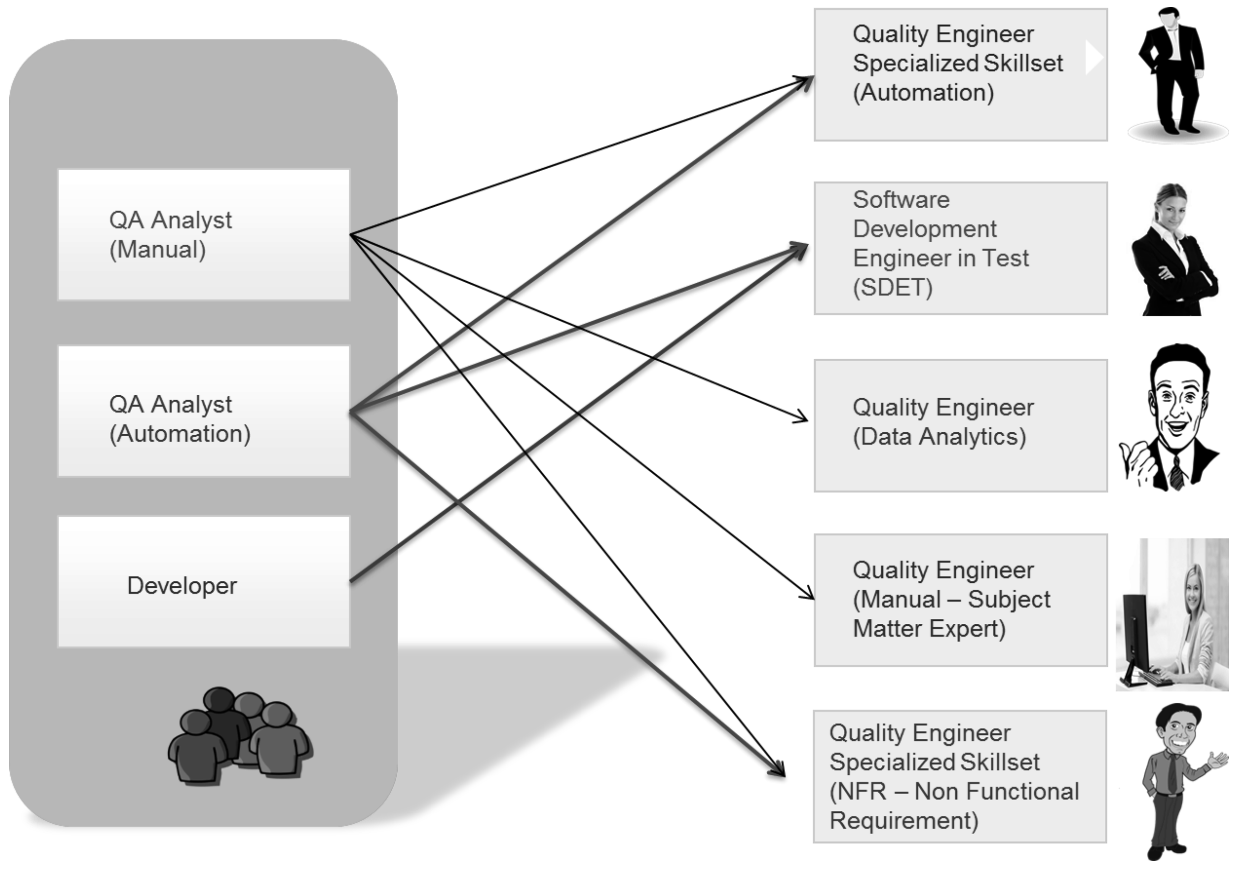

QA Analyst (Manual)
QA Analyst (Automation)
Developer
Quality Engineer Specialized Skillset (Automation)
Software Development Engineer in Test (SDET)
Quality Engineer (Data Analytics)
Quality Engineer (Manual – Subject Matter Expert)
Quality Engineer Specialized Skillset (NFR – Non Functional Requirement)

Figure: 4

5 Testing Pyramid

In order to do real continuous delivery, our applications must have fully automated black box functional / integration / system tests which cover everything necessary to verify that a particular build is ready for release to customers. All of those tests must pass in order for the build to proceed to the next stage in the continuous delivery pipeline. From that standpoint, the order of test case execution is irrelevant. However, from a practical standpoint, it helps to divide test cases into separate suites based on the likelihood of finding defects. In other words, if a recent code commit could have potentially introduced a defect, you want that failure to be detected as early in the testing cycle as possible so that the build system can alert the developers rather than waiting hours for the entire test suite to finish. This helps to compress the development cycle and prevent building up a long queue of commits.

The tests should be arranged in a pyramid with just a few tests in the first suite at the top. That way if a build is completely broken, the developers will find out within minutes. The lower levels of the pyramid contain progressively more test cases and take longer to execute. For example, it might look something like this:

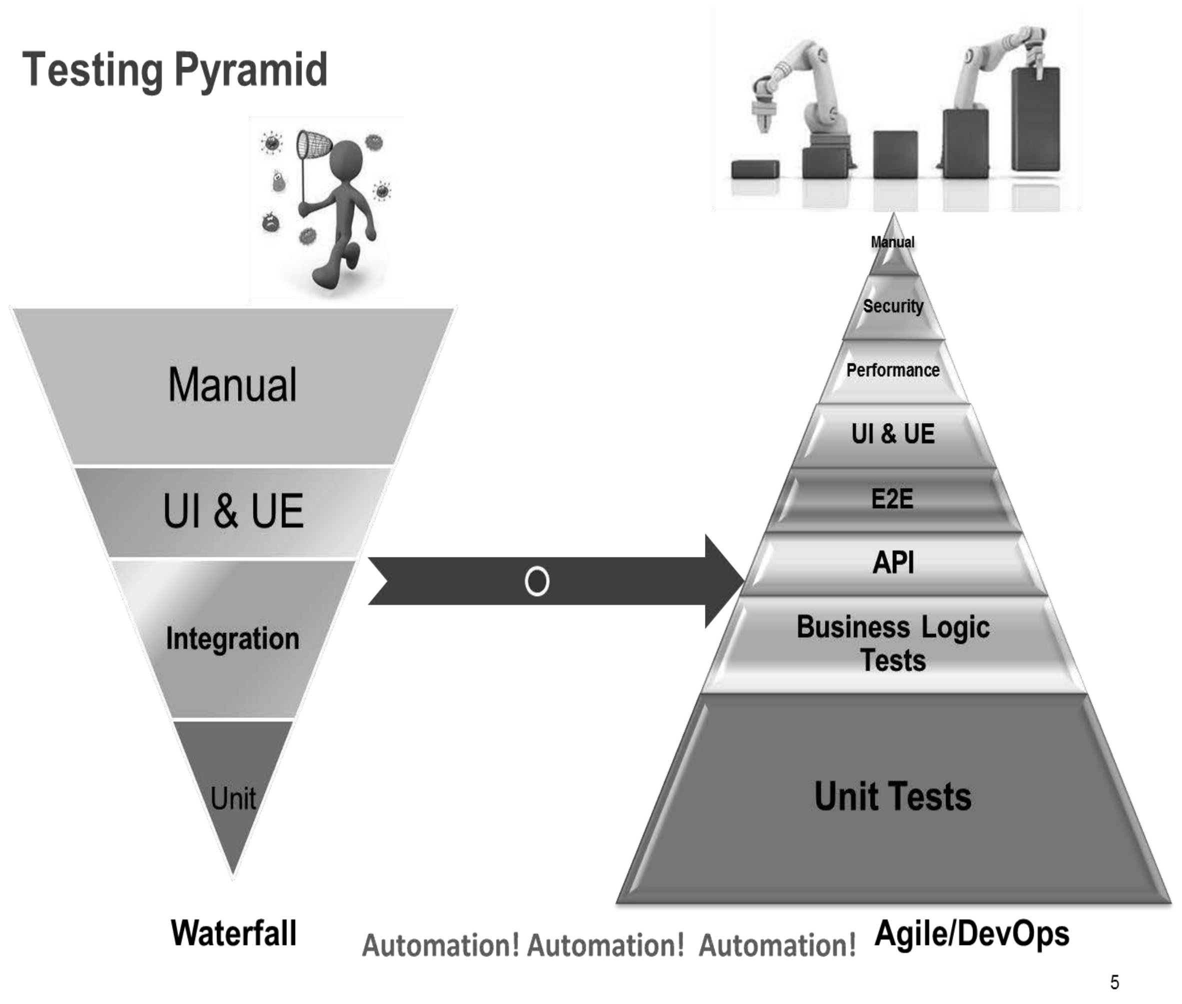


Figure: 5

6 Testing types

Gone are the days of ginormous test cycles running thousands of large, multifaceted test cases spanning days to weeks to complete in a single run. In are the "speed of agile", targeted sampling by OS, platform, etc., parallel execution, multi-technique tests**.**

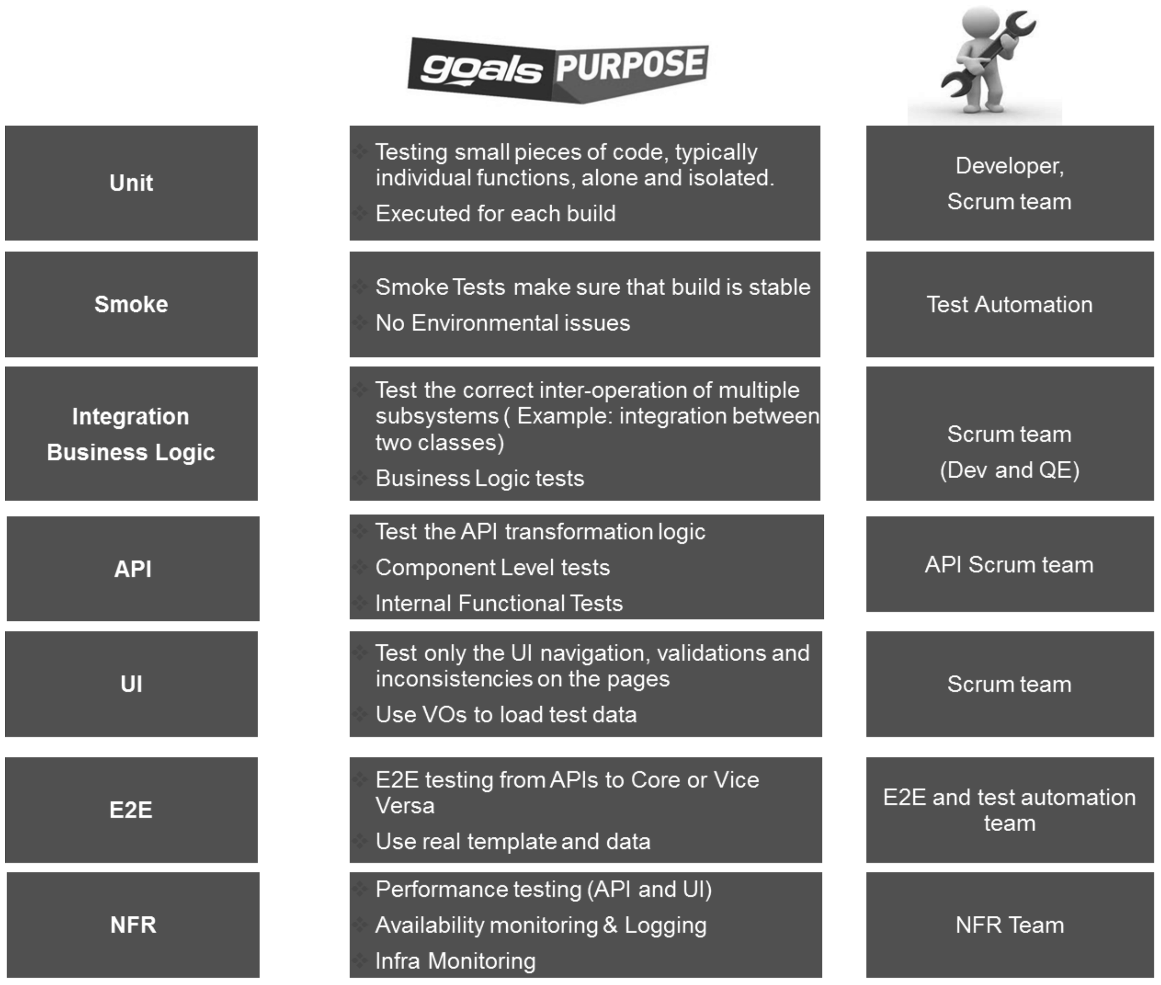
In the legacy world, more focus is on manual functional testing and if automation is there, it is mainly for functional regression, UI or a few select web-services tests. But that approach does not work in CI/CD delivery.

In CI/CD delivery, each test phase needs to be defined with goals and purpose of testing. As there are different layers like unit, component, integration, regression, NFR, etc., it is very important to draw the lines and define what should be covered in each layer. The overall motive should be to reduce the duplicate effort spent in subsequent layers. Suppose if a unit test covered the basic level of tests or validations, that validation should not be covered in subsequent layers. UI tests should test only the UI part; all functional testing should be moved to the component or integration level.

UI tests are very slow to execute and are error prone. The more UI tests there are, the more waiting time for developers to receive feedback. The turn-around-time is increased when you have more UI tests and these are very brutal because of slow execution, are seen as a low value addition and can delay the project delivery of the branches and builds.

The integration tests should cover all the vital business functions and complex engine scenarios; which are very fast to execute if done through services like REST (Representational State Transfer).

The NFR, performance, scalability and security types of testing need to be done in sprints with clear goals and defined purpose.



	goals PURPOSE	
Unit	Testing small pieces of code, typically individual functions, alone and isolated. Executed for each build	Developer, Scrum team
Smoke	Smoke Tests make sure that build is stable No Environmental issues	Test Automation
Integration Business Logic	Test the correct inter-operation of multiple subsystems (Example: integration between two classes) Business Logic tests	Scrum team (Dev and QE)
API	Test the API transformation logic Component Level tests Internal Functional Tests	API Scrum team
UI	Test only the UI navigation, validations and inconsistencies on the pages Use VOs to load test data	Scrum team
E2E	E2E testing from APIs to Core or Vice Versa Use real template and data	E2E and test automation team
NFR	Performance testing (API and UI) Availability monitoring & Logging Infra Monitoring	NFR Team

Figure: 6

7 Build Life Cycle

In a typical waterfall project, the “build cycle” means: a set of developers ready with code changes, merge these with the master repository, write and execute few unit tests, and deploy the code to the static environments. Feedback is provided to developers in the system or regression test phase which could be days after the code is committed.

On the other hand, in CI/CD delivery, the “build cycle” means: develop functionality and at the same time, write and run all types of automated tests (unit, UI, component, integration, regression, NFR etc.), providing feedback to the developers in minutes rather than days.

If any of the tests fail in building the software, these need to be fixed immediately, before deploying the code to the master branch. All tests must pass. This is a mandatory step in the build pipeline cycle.

The below figure shows the build life cycle in DevOps delivery:

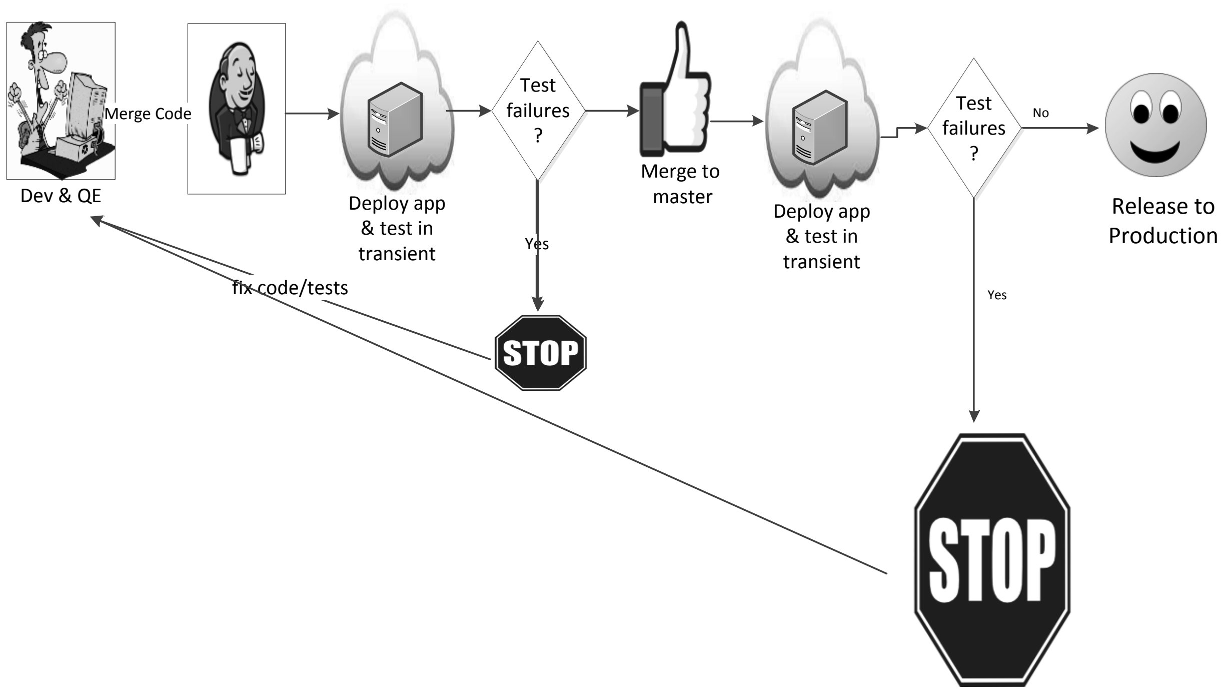


Figure: 7

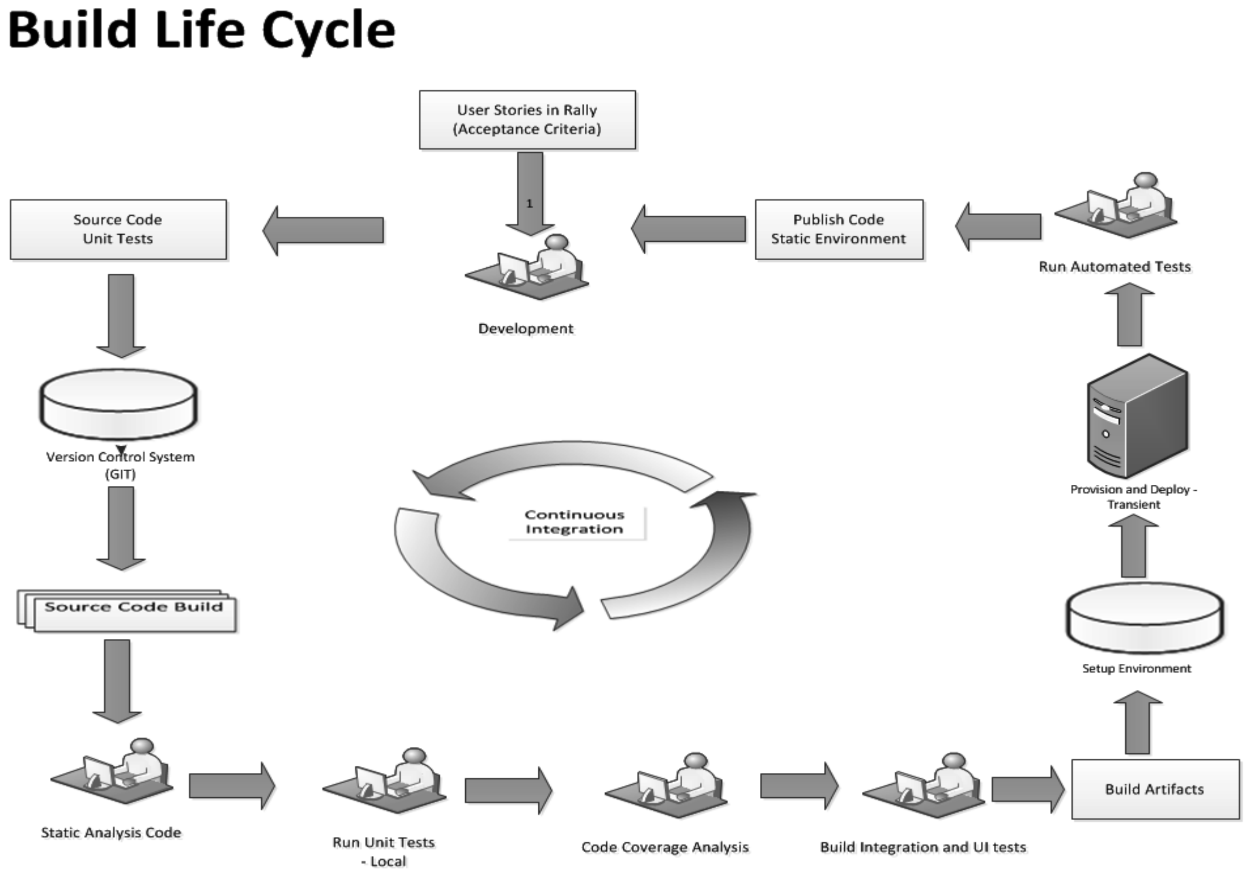


Figure: 8

8 Test Development

In a typical CI/CD environment and where ATDD is implemented, the Product Owner writes the requirements in a tool such as Rally. The Quality Engineers work together with the product owners and define the acceptance criteria for a user story or set of features. As soon as the acceptance criteria and timelines are defined for implementation, developers take the user story and start building the functionality at the same time that Quality Engineers start writing the automated tests. Developers are more focused on functionality development and writing unit tests to cover over 80% of the code. While Quality Engineers are more focused on writing integration and UI tests. As both are done with their development, they test the functionally in the same code branch by executing all unit, integration and UI tests. If all the tests pass for the user story, the code is checked into the main branch for build deployment.

From there, continuous servers like Jenkins/Anthipro take the latest code and merge it, building the software and deploying it to transient environments or any other static environment defined by the project build pipeline, according to the implemented technical stack.

The intent is to develop all test types combining unit, integration and UI testing in advance to be included in the build process.

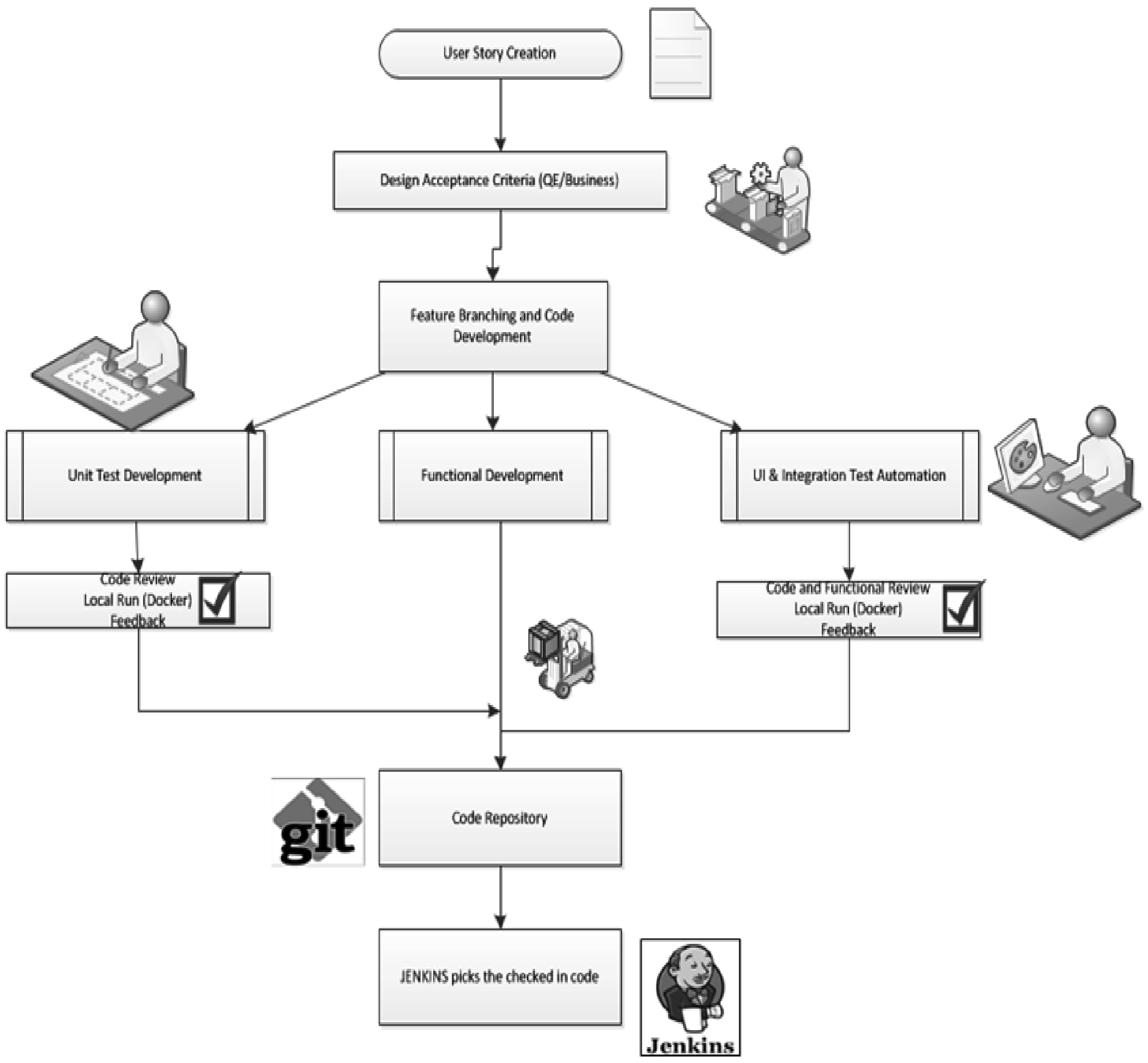

User Story Creation
Design Acceptance Criteria (QE/Business)
Feature Branching and Code Development
Unit Test Development
Functional Development
UI & Integration Test Automation
Code Review
Local Run (Docker)
Feedback
Code and Functional Review
Local Run (Docker)
Feedback
git
Code Repository
JENKINS picks the checked in code
Jenkins

Figure: 9

9 Test Environments

Inception of the Cloud has given flexibility to application development teams to build and test application software in transient environments. The software is available in the market to spin up environments in a few minutes, allowing developers to write and test software without it being deployed to static environments.

We have used Open Shift Environment (OSE), Docker and Mesos in our technology stack to build the environment on the fly and test the software. The code is not moved to the static environment until it passes all the toll gates of different types of testing. All tests must be passed before code moves to the static environments. Only smoke and E2E tests should be run on the static environments in a large complex program or project.

A transient environment gives the opportunity to design the idempotent and repeatable tests and make sure every run gives the same results.

The below figure shows the test environment grid in the CI/CD type of software delivery model.

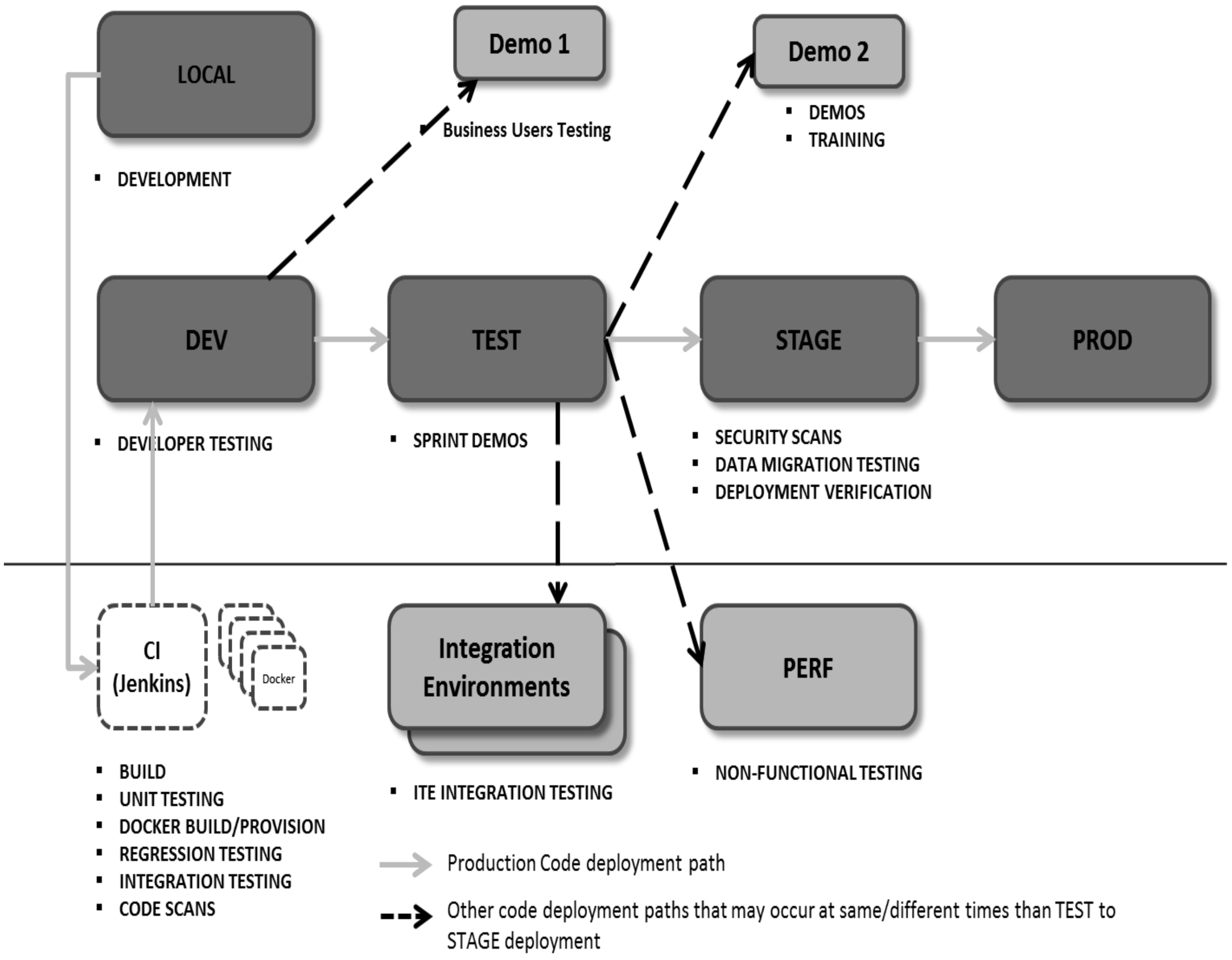


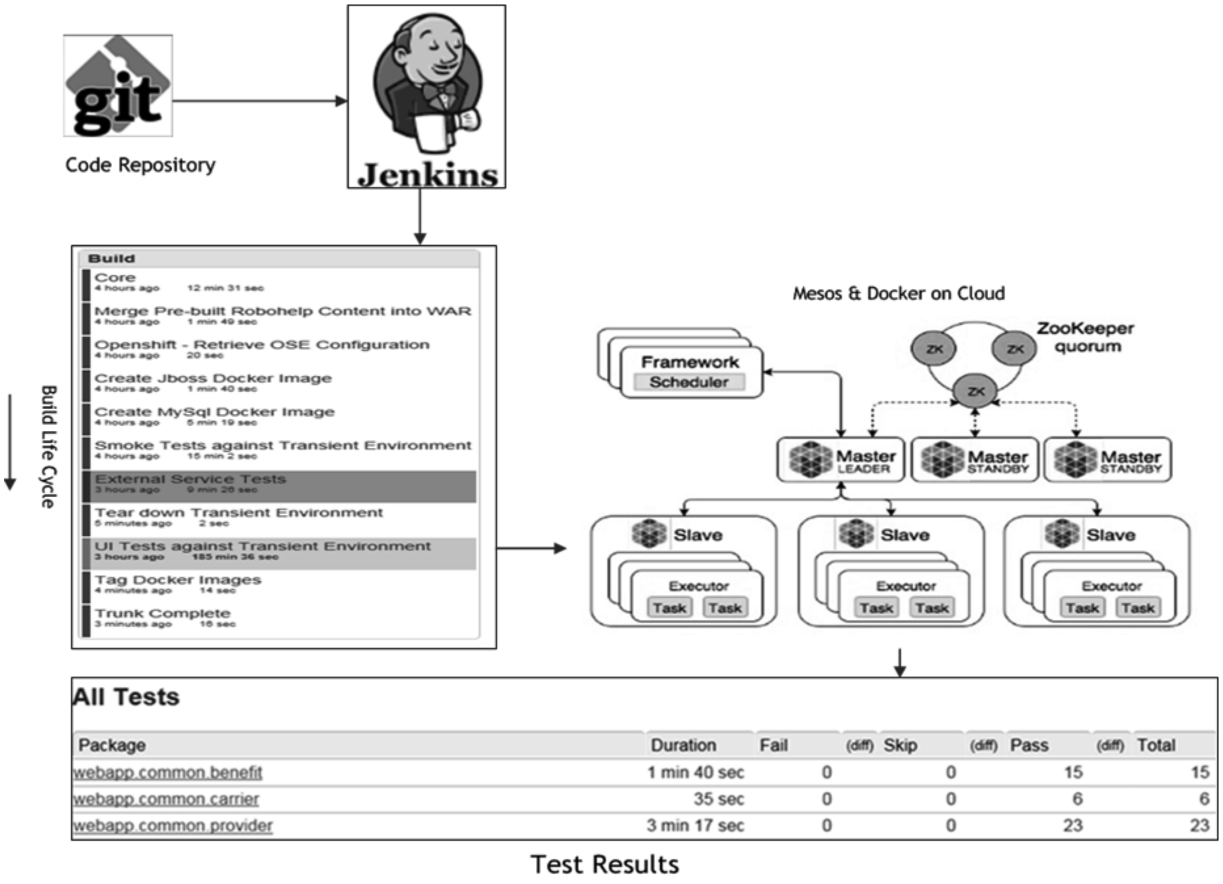
Figure: 10

10 Test Execution

In the legacy development life cycle, test execution is done on the static environment. In a DevOps or Continuous Delivery model, the automated tests (unit, smoke, regression, API, business logic tests & E2E, etc.) are hooked up with an application build deployment pipeline and executed in the background by the CI servers. The developer or Quality Engineer writes tests first but they are not expected to execute the tests manually.

For test execution purposes, Docker and Mesos can be used to create the environment and run all the tests as needed. We are executing almost 50K+ tests in under 30 minutes with 32 nodes in the Mesos environment. The entire build cycle finishes in 25 to 30 minutes. A normal day would see us complete on an average 30 builds

The test results can be obtained from the CI servers with a screenshot facility.



Build

- Core
  4 hours ago 12 min 31 sec
- Merge Pre-built Robohelp Content into WAR
  4 hours ago 1 min 49 sec
- Openshift - Retrieve OSE Configuration
  4 hours ago 20 sec
- Create Jboss Docker Image
  4 hours ago 1 min 40 sec
- Create MySql Docker Image
  4 hours ago 5 min 19 sec
- Smoke Tests against Transient Environment
  4 hours ago 15 min 2 sec
- External Service Tests
  3 hours ago 9 min 26 sec
- Tear down Transient Environment
  5 minutes ago 2 sec
- UI Tests against Transient Environment
  3 hours ago 185 min 36 sec
- Tag Docker Images
  4 minutes ago 14 sec
- Trunk Complete
  3 minutes ago 16 sec

All Tests

Package	Duration	Fail	(diff)	Skip	(diff)	Pass	(diff)	Total
webapp.common.benefit	1 min 40 sec	0		0		15		15
webapp.common.carrier	35 sec	0		0		6		6
webapp.common.provider	3 min 17 sec	0		0		23		23

Figure: 11

11 Test Data Management

Parallel test execution is required in DevOps delivery for a faster feedback loop to the developers. But managing test data in parallel to test execution is difficult. The team has to be very careful in sharing the data among various tests being executed simultaneously. One test can create the data but another can update or delete it and later the same data is of no use to other tests.

To address this issue, we have adopted a different approach in managing test data. An automated test should be capable enough to create the data for its own need. We are deploying application code to a transient environment seeded with static and reference data that is never going to change. Also, the automated tests have been designed in such a way, that they create data for the functional scenario itself, use the data, and later neatly dispose of it. All tests are isolated from each other, from the test *data* point of view. No tests use test data created by other tests.

Hence, this approach helps in reducing test data and environment related defects. The following figure depicts a successful test data management technique which supports parallel and idempotent testing.

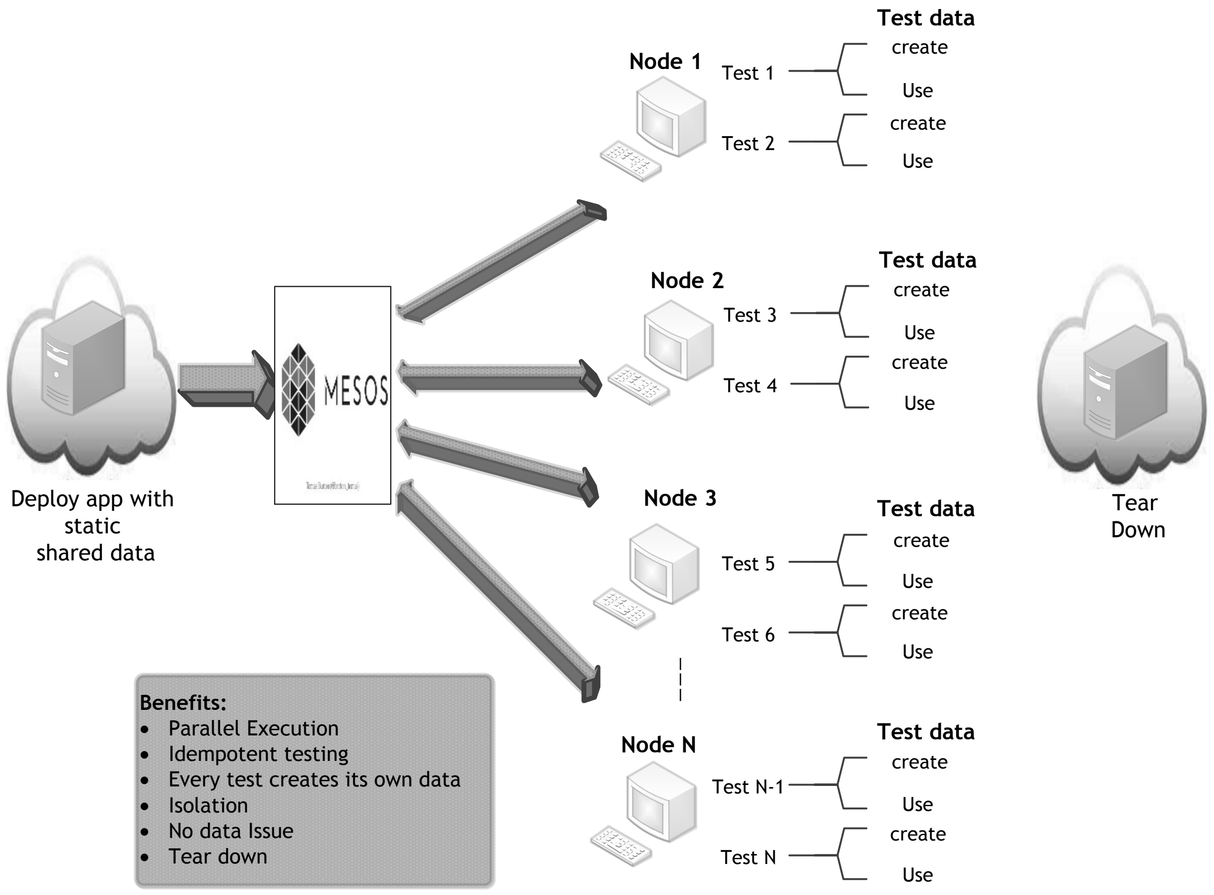


Figure: 12

12 Benefits: Transient Environment testing

There are several benefits of using a transient environment in the software project development cycle. The project team can achieve real time automation and up to 100% penetration of test automation.

With cloud infrastructure costs being very cheap (compared to physical servers) and application teams able to create the environment on the fly, this gives the flexibility to use the latest technology stack providing the fastest test execution! QA costs have been observed to be much lower in comparison to legacy testing where separate phases are needed to test the software.

With this implementation, we are able to maintain project QA costs less than 10% and automation penetration at 95%.

The below figure shows the QA cost difference and benefits of automated testing in the DevOps delivery cycle.

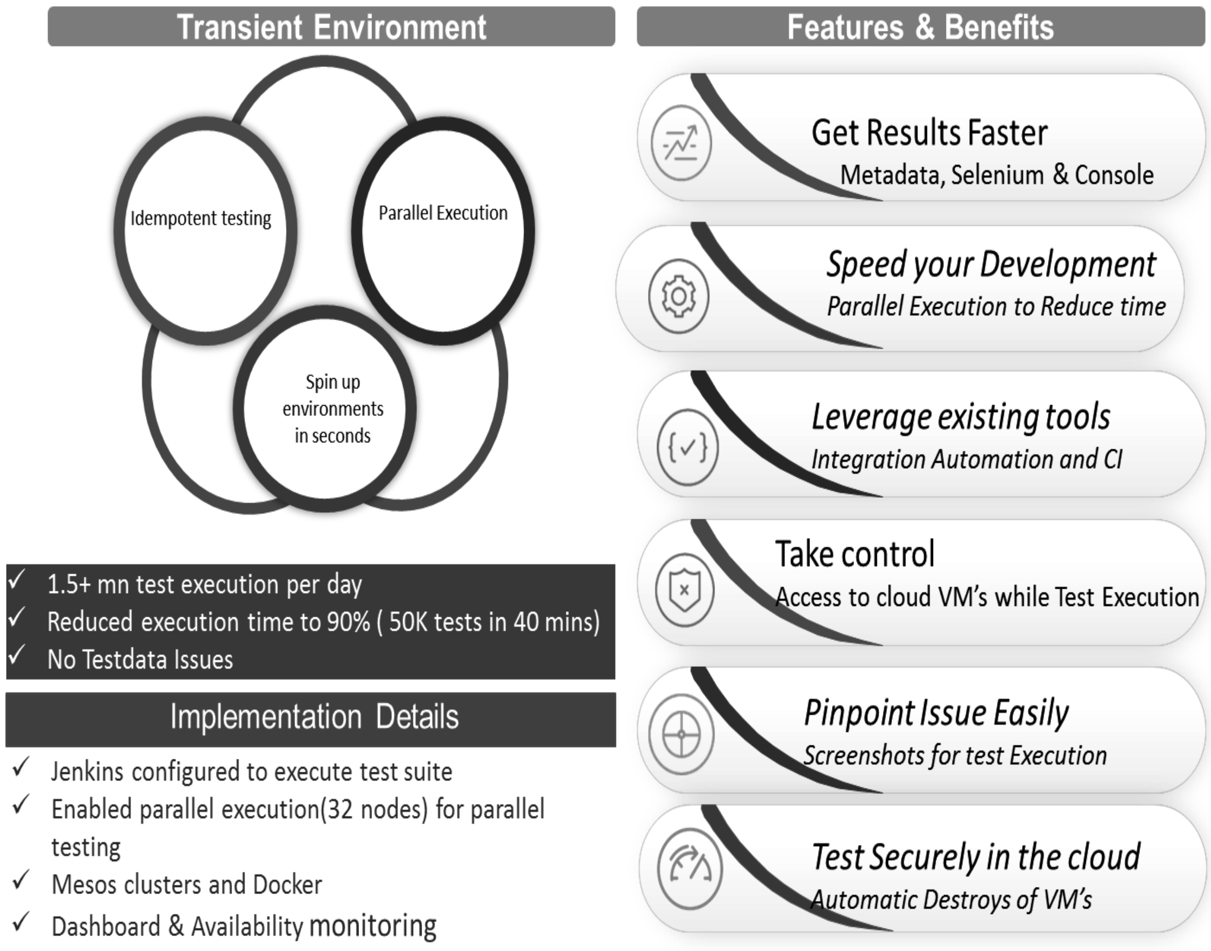

Transient Environment
Idempotent testing
Parallel Execution
Spin up environments in seconds
✓ 1.5+ mn test execution per day
✓ Reduced execution time to 90% (50K tests in 40 mins)
✓ No Testdata Issues
Implementation Details
✓ Jenkins configured to execute test suite
✓ Enabled parallel execution(32 nodes) for parallel testing
✓ Mesos clusters and Docker
✓ Dashboard & Availability monitoring
Features & Benefits
Get Results Faster
Metadata, Selenium & Console
Speed your Development
Parallel Execution to Reduce time
Leverage existing tools
Integration Automation and CI
Take control
Access to cloud VM's while Test Execution
Pinpoint Issue Easily
Screenshots for test Execution
Test Securely in the cloud
Automatic Destroys of VM's

Figure: 13

References

This is all based on practical experience and under the guidance of the program leadership team.

1) https://www.docker.com/
2) http://mesos.apache.org/
3) http://www.gebish.org/
4) https://jenkins.io/