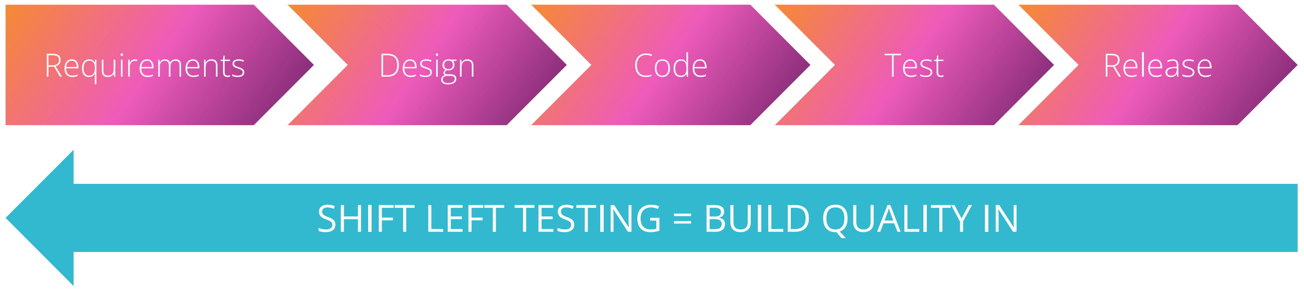
Introduction to Shift Left testing

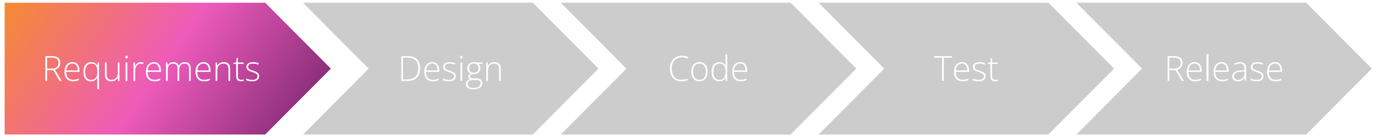
Introduction to ‘Shift Left’ Testing



Why Shift Left?

- ➔ Cost of Defect Fixing is high when defects are caught at the end of a big bang development
- ➔ Incremental testing enables faster delivery of business value
- ➔ Quality is ‘*built-in*’ and not tailored at the end
- ➔ Increases delivery team confidence

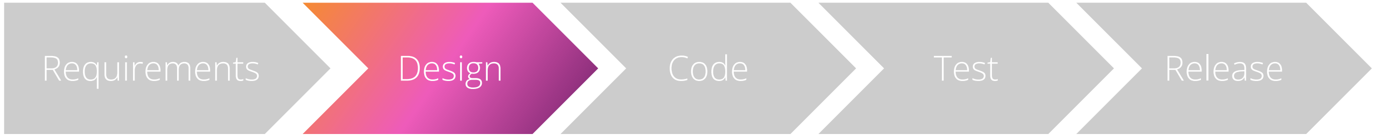
‘Shift Left’ Testing - QA’s role



Why should QA be in this phase?

- ➔ To understand customer landscape / business value of the functionality.
- ➔ Understand and provoke questions around integrations with different systems
- ➔ Technical know-how of Data migrations and DB structures
- ➔ Understanding and designing test scenarios as per Cross functional requirements.
- ➔ Planning right toolings for testing right systems
- ➔ Establishing collaboration with existing dev and QA teams

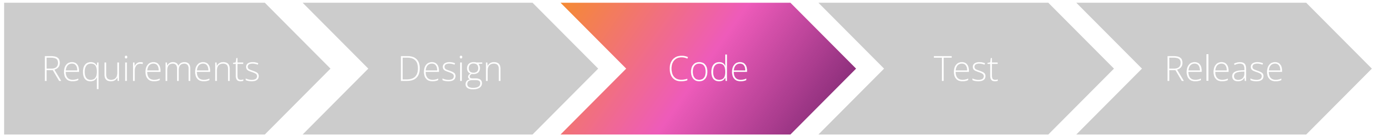
‘Shift Left’ Testing - QA’s role



Why should QA be in this phase?

- ➔ Understands customer experience needs
- ➔ Understands user personas to include them in testing scenarios
- ➔ Exploring various needs of customer that design have to cater to like different browsers, usability factors into testing strategy
- ➔ Capturing all test scenarios at a story level including functional and design categories

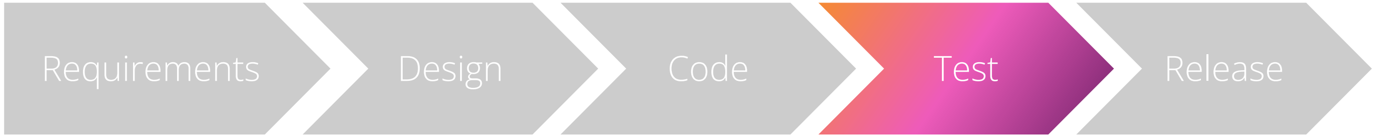
‘Shift Left’ Testing - QA’s role



Why should QA be in this phase?

- ➔ Communication / collaboration with the devs on the different edge cases to be covered for each story
- ➔ Collaborate with devs on the distribution of tests across multiple layers
- ➔ Build automated tests ready to pass in CI once the development is complete
- ➔ Aim to understand customer test data in required test environments

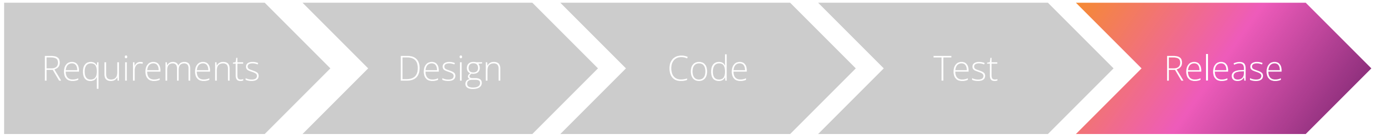
‘Shift Left’ Testing - QA’s role



QA’s during testing -

- ➔ Manual exploratory testing of all scenarios and edge cases in an integrated environment with customer relevant data setup
- ➔ Increase automated testing of edge cases when needed
- ➔ Cover cross functional aspects of testing at a story level scenario testing
- ➔ Collaborate in defect fixing

‘Shift Left’ Testing - QA’s role



QA’s during Release -

- ➔ Release testing is done in an end to end integrated environment with the actual configurations and existing data migrations for the required customer.
- ➔ Ideally all builds are release ready. This phase is mainly to make sure the end to end flows with production like setup works fine before saying ‘Go-Live’
- ➔ User acceptance testing (UAT) is performed by Customer Product owners on this integrated UAT environment once the release testing is done by the QAs.

A day in a QA's life

- ➔ Stand-Ups
- ➔ Dev / Team Huddles
- ➔ Story Kick off
- ➔ Dev Box Testing
- ➔ Story Testing
  - ◆ Exploratory Testing
  - ◆ Integration Testing
  - ◆ DB Testing
  - ◆ Security scenarios
  - ◆ Test scenarios documentation
- ➔ Defect Management
- ➔ Automated Testing
  - ◆ Reports
  - ◆ Failure Analysis
  - ◆ New Tests
- ➔ Regression Testing
  - ◆ Bug Bashes
  - ◆ Nightly Regression
- ➔ Story Review in IPMs
- ➔ Showcases
- ➔ User Acceptance testing

Shift Left testing - Case study

BEFORE

- Long regression phase before release
- Production issues
- Fear to refactor code
- QA capacity issue
- Delayed quality feedback

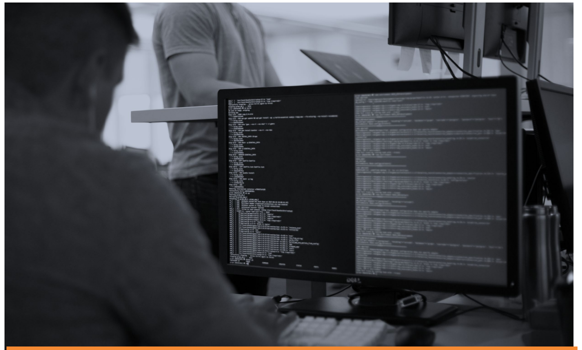
AFTER

- Short regression phase
- Very minimal to no production issues
- Increased client trust
- Code refactored to fit in the latest technologies
- QA capacity issue disappeared
- Early feedback

EXECUTIVE SUMMARY

CULTURE

- Frequent releases
- Did not manage tech debt



CODE

- Legacy code not unit testable
- Logic within sql
- No single code coverage report
- Code duplication
- Duplicate tests in various layers of test pyramid
- Redundant/Obsolete Tests

integration [integration-test]
backend-functional [integration-test]
backend [deploy-dev]
client [deploy-dev]
backend [smoke-test-dev]
both [functional-test]
client [deploy-showcase]
backend [deploy-showcase]
backend [smoke-test-showcase]

PROCESSES

- No TDD
- No estimation for test
- Branch based development
- Distorted test pyramid
- Conflict within team -who is responsible for quality

HOW WE TURNED THINGS AROUND

ITERATION PLANNING

- Estimated QA effort
- An iteration clearing the backlog
- ‘Write API tests’ as acceptance criteria

STORY KICKOFF

- Capture potential tests early

DEV & TESTING

- Run tests in devbox
- Collective Functional testing
- Introduced code coverage tool
- Maintain Code Quality
- Created automation backlog board

DEPLOY & TEST

- Beyond functional testing (Perf/Sec)
- Frequent Reports to all Stakeholders
- Bug Bash
- Remodel regression

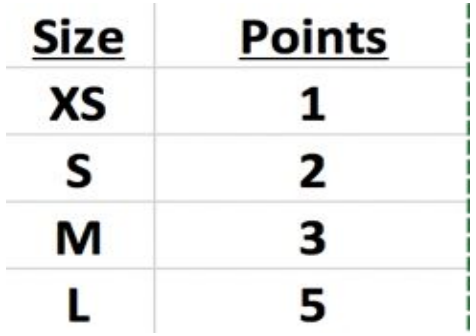
SIGN OFF

- Automation signoff before feature signoff

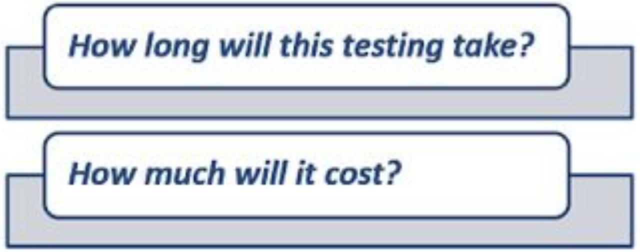
INCLUDING QA EFFORT IN STORY & AUTOMATION ESTIMATES

STEPS TAKEN

- Change in testing approach
- QAs pushed for inclusion in effort estimation efforts
- Huge impact by a small feature
- Effort to write tests
    - T-shirt sizing
- Additional data setup requirements



Size	Points
XS	1
S	2
M	3
L	5



How long will this testing take?

How much will it cost?

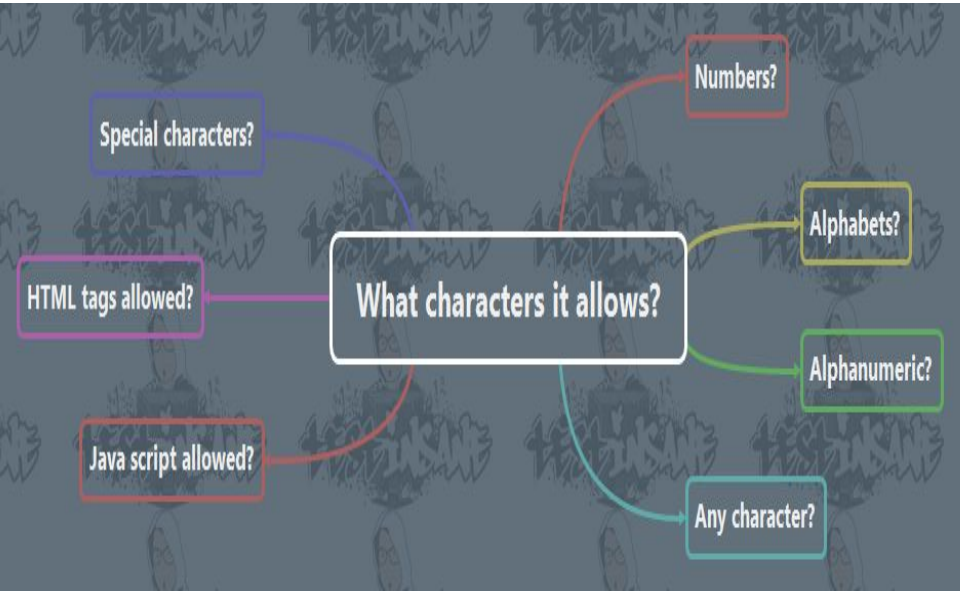
ADDED ITERATION TO CLEAR AUTOMATION BACKLOG

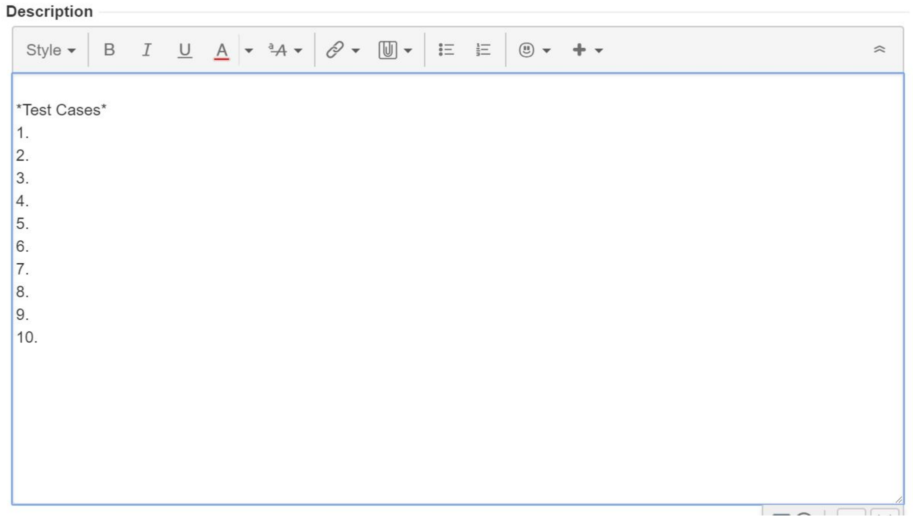
STEPS TAKEN

- Choose a certain number of backlog cards to automate per iteration.
- Prioritize of cards based on impact
- Track ROI of time saved while seeking approvals for clearing the existing backlog.

CAPTURE POTENTIAL TESTS EARLY

STORY KICKOFF





Test Cases
1.
2.
3.
4.
5.
6.
7.
8.
9.
10.

RUN TESTS IN DEVBOX

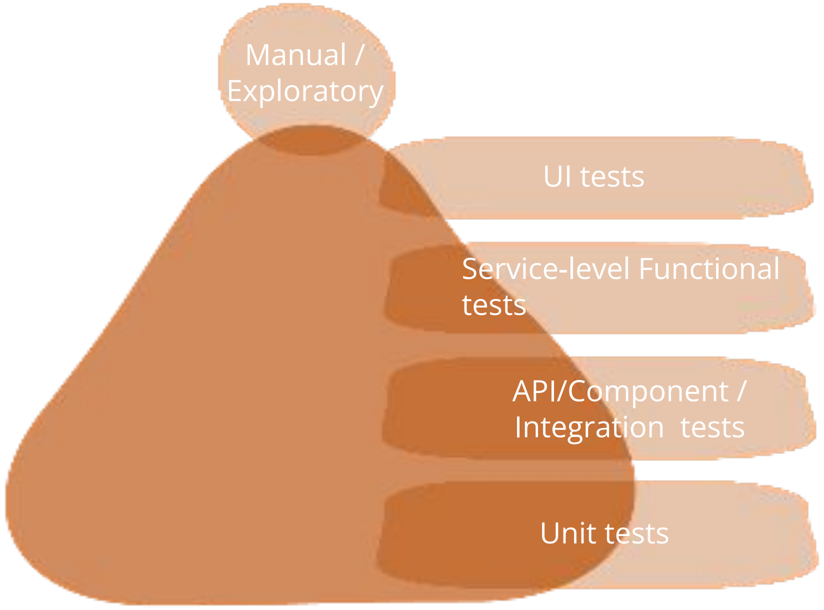
STEPS TAKEN

- Guarantee that all functionalities are working fine
- Data setup done correctly
- Edge cases are covered
- Assertions are right.
- Discuss if the tests have been written in the appropriate layers of the test pyramid

FUNCTIONAL TESTS – A COLLECTIVE RESPONSIBILITY

STEPS TAKEN

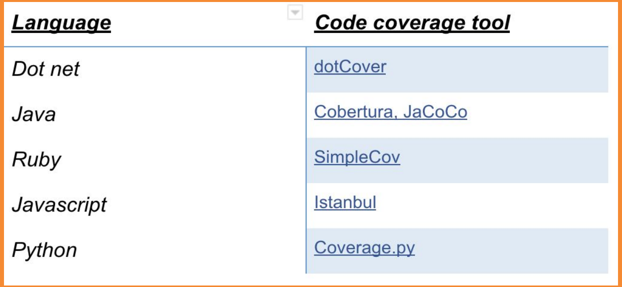
- Introduce the test pyramid
- Developers should write unit tests
- Developers should be encouraged to write API functional tests
- Follow BDD so that BAs can also write functional tests
- QAs can be trained to write unit tests
- Choose the right tool and technology
- Maintain the same tech stack across all layers
- QAs and Devs can pair to bring up the functional testing framework


Manual / Exploratory
UI tests
Service-level Functional tests
API/Component / Integration tests
Unit tests

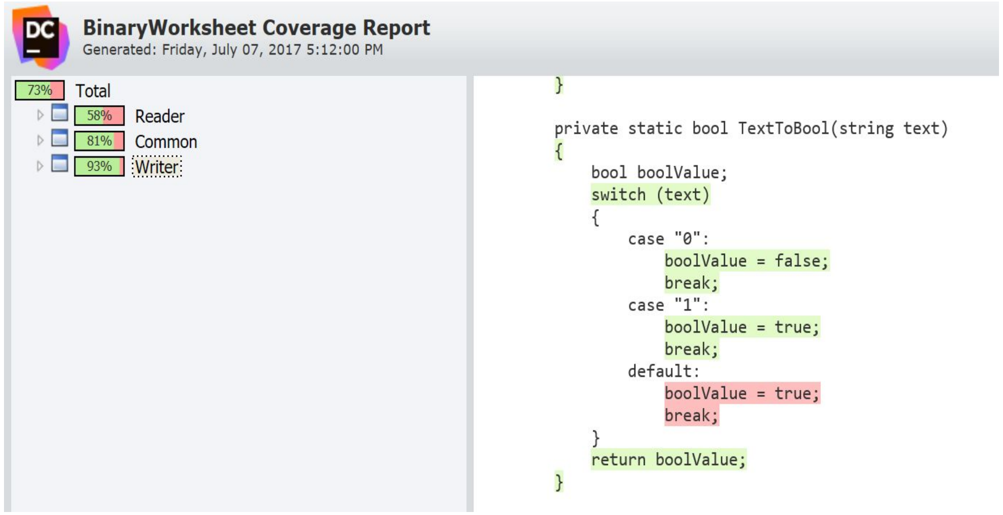
INTRODUCE A CODE COVERAGE TOOL

STEPS TAKEN

- Publish the code coverage for all the test suites - unit, integration and functional tests
- Make it part of the build pipeline
- Set a benchmark
- Limit the scope of manual regression cycle



Language	***Code coverage tool***
Dot net	dotCover
Java	Cobertura, JaCoCo
Ruby	SimpleCov
Javascript	Istanbul
Python	Coverage.py



BinaryWorksheet Coverage Report
Generated: Friday, July 07, 2017 5:12:00 PM

73%	Total
58%	Reader
81%	Common
93%	Writer

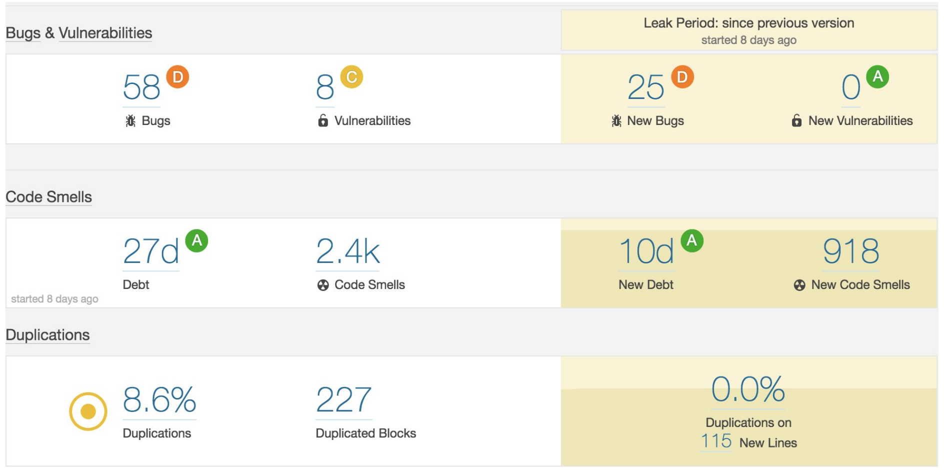
```
}

private static bool TextToBool(string text)
{
    bool boolValue;
    switch (text)
    {
        case "0":
            boolValue = false;
            break;
        case "1":
            boolValue = true;
            break;
        default:
            boolValue = true;
            break;
    }
    return boolValue;
}
```

MAINTAIN CODE QUALITY

STEPS TAKEN

- Introduce Code quality analysis tool
  - SonarQube etc
- Pair programming
- Code review
- Code should be self explanatory - no comments should be required
- Testable code
- Run tests on each commit


Bugs & Vulnerabilities

Leak Period: since previous version
started 8 days ago

58 D Bugs	8 C Vulnerabilities	25 D New Bugs	0 A New Vulnerabilities

Code Smells

27d A Debt	2.4k Code Smells	10d A New Debt	918 New Code Smells

started 8 days ago

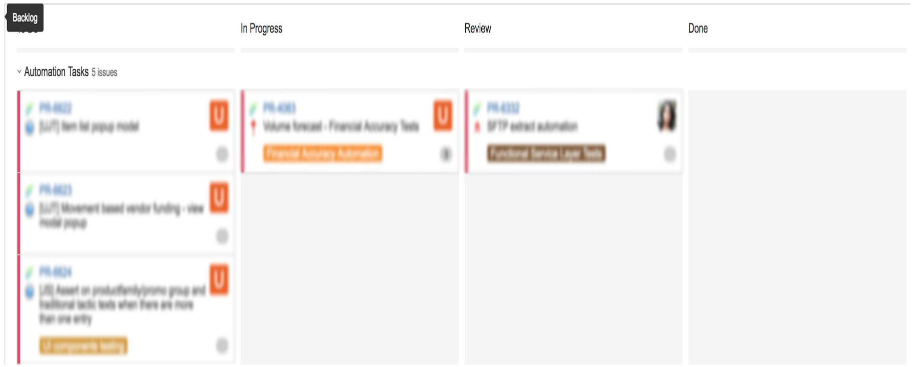
Duplications

8.6% Duplications	227 Duplicated Blocks	0.0% Duplications on 115 New Lines

CREATE AN AUTOMATION BACKLOG BOARD

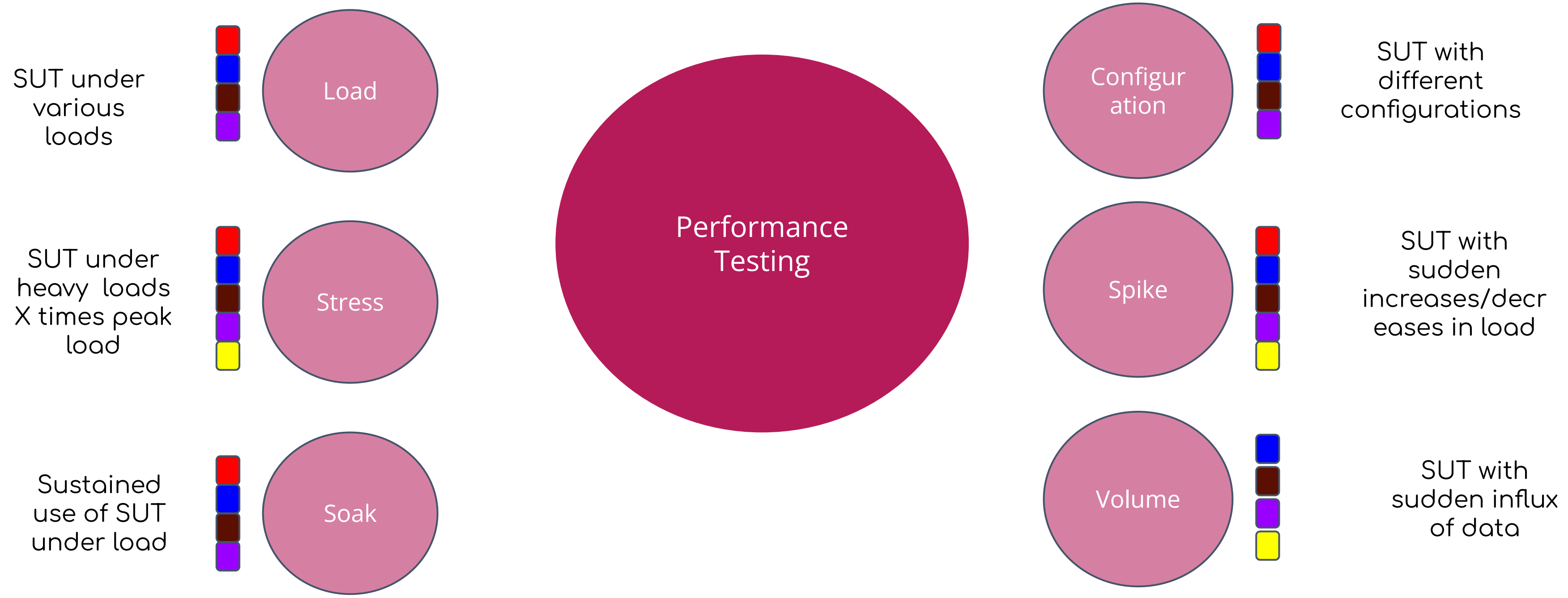
STEPS TAKEN

- Analyze corresponding functionalities from the code coverage report
- Visibility on the amount of work that needs to be done to ensure product quality

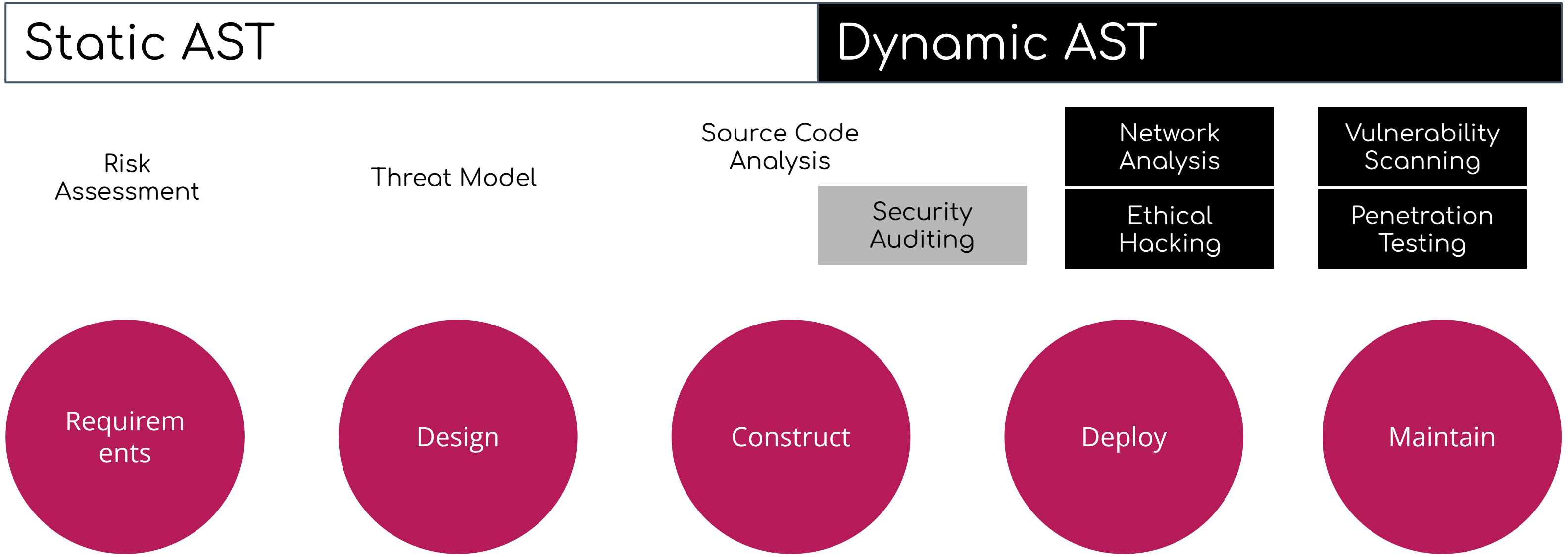


BEYOND FUNCTIONAL TESTING - PERFORMANCE

🟥 Responsiveness 🟦 Stability 🟫 Resource Usage 🟨 Scalability 🟪 Reliability

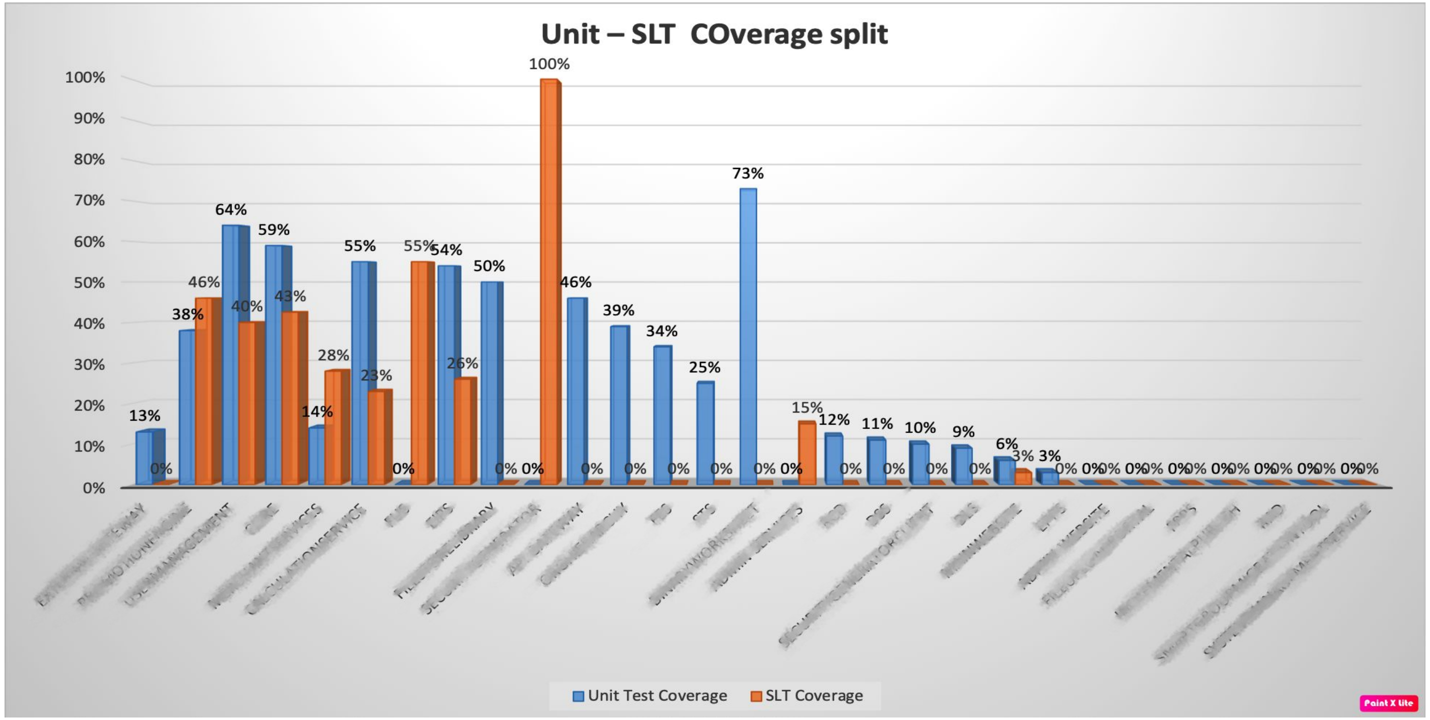

SUT under various loads
🟥🟦🟫🟪
Load
SUT under heavy loads X times peak load
🟥🟦🟫🟪🟨
Stress
Sustained use of SUT under load
🟥🟦🟫🟪
Soak
Performance Testing
Configuration
🟥🟦🟫🟪
SUT with different configurations
Spike
🟥🟦🟫🟪🟨
SUT with sudden increases/decreases in load
Volume
🟦🟫🟪🟨
SUT with sudden influx of data

BEYOND FUNCTIONAL TESTING - SECURITY



FREQUENT REPORTING TO ALL STAKEHOLDERS

DEPLOY & TEST



BUG BASH

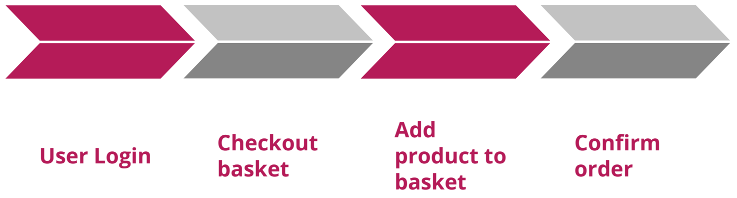
DEPLOY & TEST

MANUAL REGRESSION TEST CASES TO USER STORIES

DEPLOY & TEST

STEPS TAKEN

- Separate out non automated test cases per feature
- Create meaningful user journeys from the test cases
- Reduce the number of manual regression cards
- Easy to plan and execute

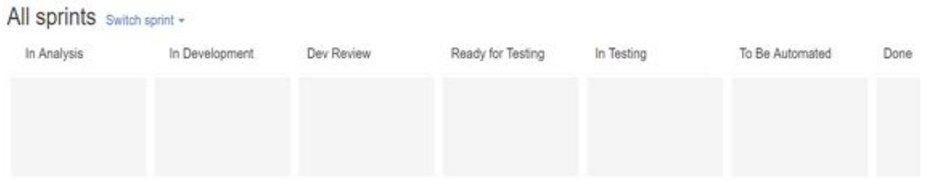


AUTOMATION SIGNOFF BEFORE FEATURE SIGNOFF

SIGN OFF

STEPS TAKEN

- After regular manual testing
- Automation testing get a sense of importance
- Capacity issues might end up causing a card accumulation on the new lane and hinder delivery



Thank You