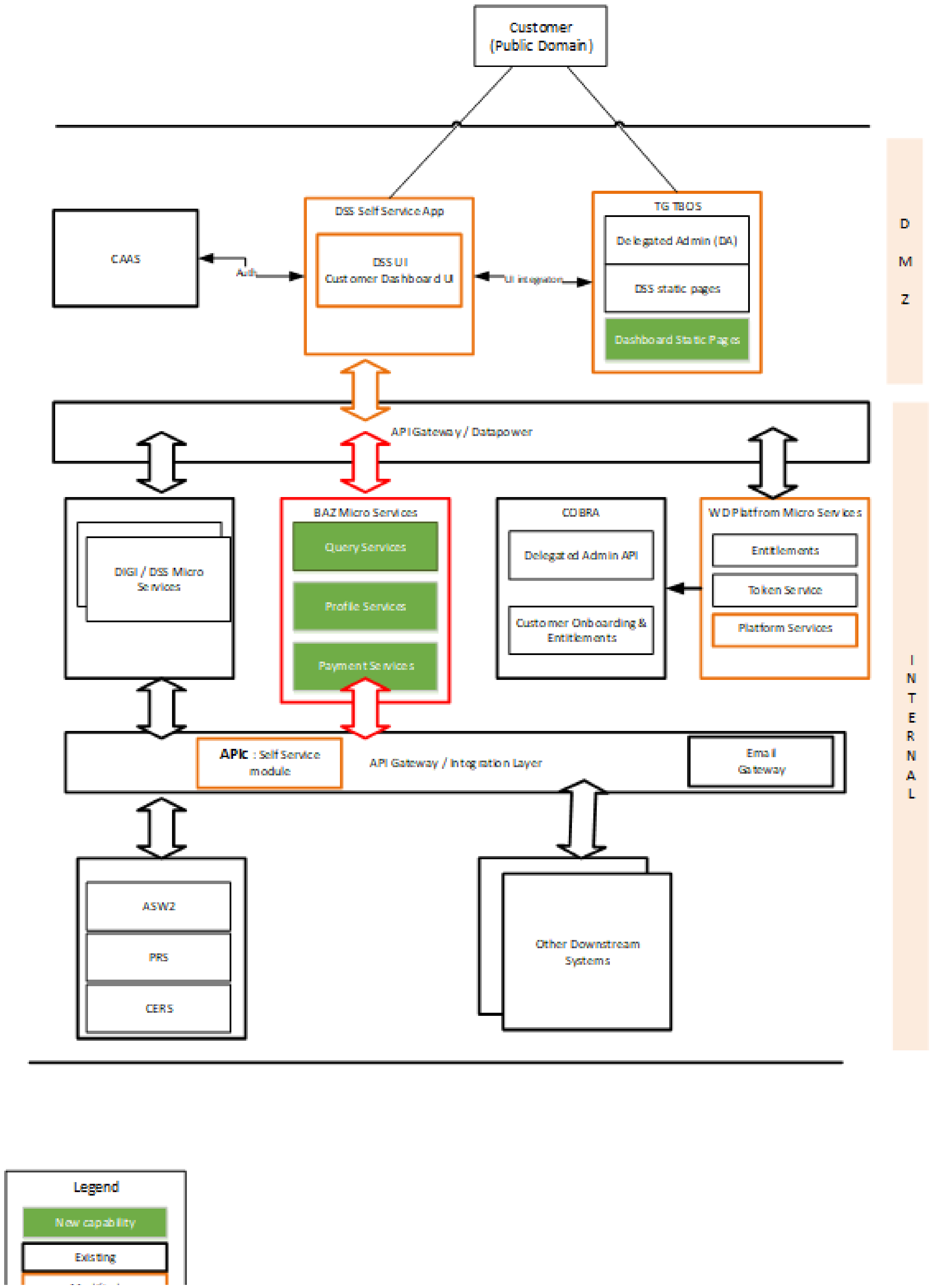
Testing Approach

Initiative Overview

As part of automation and enhancing customer experience for support and queries, customer dashboard has been setup fronting the user access to dashboard application via Omni channel. Various functions are automated in iterative delivery of functions.

The project leverages enterprise standard platforms such as Omni channel / CAAS / COBRA to provide secure, consistent and standardized mechanisms of channel access while the functions built using Openshift platform providing neater, simpler, faster and better maintainable systems that is both cohesive and as well as scalable and independently administered.

Digital Self Service is an hosted app on TG and the UI layer of this DSS app is being leveraged and extended to provide UI for customer dashboard

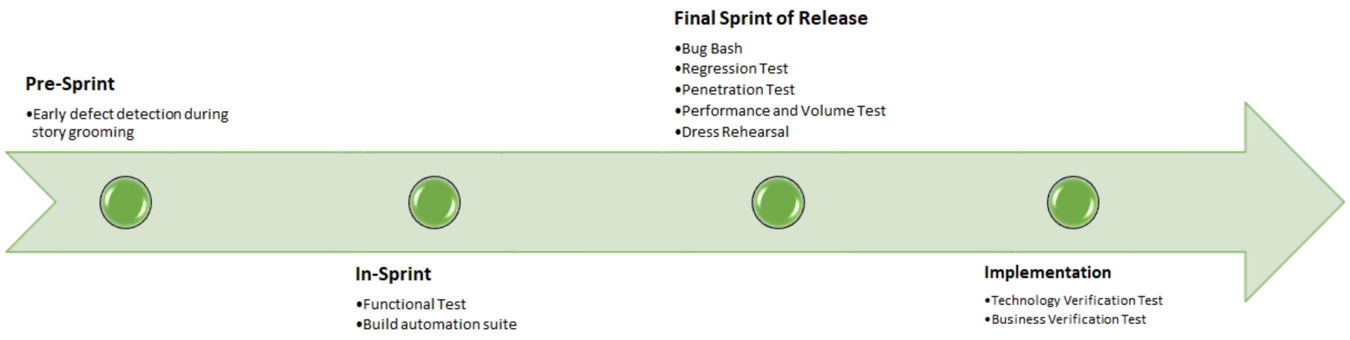

Customer (Public Domain)
D M Z
CAAS
Auth
DSS Self Service App
DSS UI Customer Dashboard UI
UI integration
TG TBOS
Delegated Admin (DA)
DSS static pages
Dashboard Static Pages
API Gateway / Datapower
DIGI / DSS Micro Services
BAZ Micro Services
Query Services
Profile Services
Payment Services
COBRA
Delegated Admin API
Customer Onboarding & Entitlements
WD Platfrom Micro Services
Entitlements
Token Service
Platform Services
I N T E R N A L
APIc : Self Service module
API Gateway / Integration Layer
Email Gateway
ASW2
PRS
CERS
Other Downstream Systems
Legend
New capability
Existing

Tools

Purpose	Tool	Remarks
Test Management	JIRA	With Zephyr plugin
Defect Management	JIRA	
UI Automation	Puppeteer, Node JS, Cucumber	
API Automation	Postman	
Service Virtualization	Inhouse tool built with Java (SIT) Montebank (PnV)	
Performance and Volume Testing	JMeter	
Log Monitoring Tool	Splunk	

Testing Approach

Digital Self Service Squad follows Agile based test delivery model. The following picture depicts the flow of testing during a release with majority of functional test is conducted during the sprint



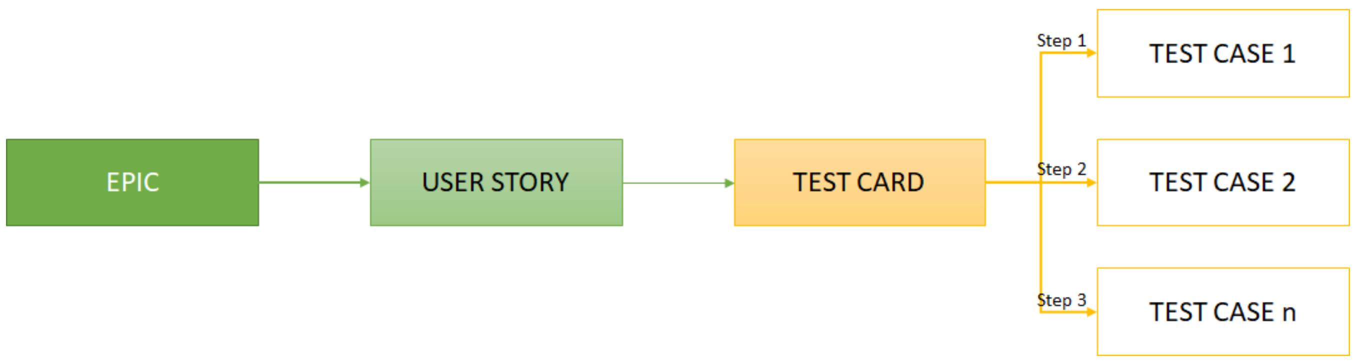
The following sections details the approach taken for each test phase for all of Releases (Major / Minor)

In-Sprint Testing (ST/SIT)

Test Design

Story grooming session is primarily used by testers to understand and decide the scope of testing for the card. During the session, appropriate questions are raised to get the story updated to hold as much detail as possible to make test design easy. JIRA is used as the primary tool for test management activity. The tool has a Zephyr / XRAY plugin that enables creation of ISSUE TYPE as TEST. Once a story is groomed and is moved into a sprint, a TEST type card will be linked to the STORY.

All the test cases associated to the story card would be written against this test type card. Each test case would be written as a TEST STEP in the card and hence a single test card represent 1 or more test cases. Test cases would cover all the possible scenarios associated to the user story along with any negative validation that need to be performed.



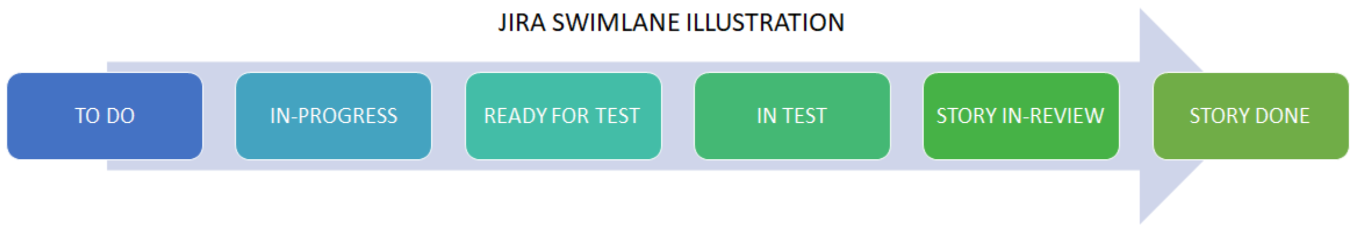
For each test type card the following fields are updated in JIRA

- Project - DT - Flex 1 - Automation (DTFAU)
- Issue Type - Test
- Summary - <*A short description of the test (mostly same as the name of the story with a test prefixed to it*>
- Reporter - <*Test Designer Name*>
- Label - SIT / Regression
- Epic Link - <*Epic card in Jira*>
- Sprint - Is left blank as the testing is tracked via Story card in the sprint. The test card is used behind the scene for scheduling and running test
- Affected Version - <*Release # where the story is built*>
- Linked Issues - always updated as "is a test for"
- Issue - <*Story card #*>
- Zephyr Teststep - <Test Cases>

Once the test card is ready for execution, its is scheduled to a release in Test Cycle

Test Execution

JIRA swimlane is set-up in a way to easily identify the cards that are ready for testing. Once the build and unit test is complete, the card would be moved into READY FOR TEST queue. The cards associated to changes in Websphere Liberty platform will be tagged with a Artefactory version number and the deployment would be done by the testers using UDeploy before they start testing. The changes to openshift gets deployed by the build engineer before assigning the card for testing.



In-sprint testing is performed in a connected environment. Once the card is moved to "READY FOR TEST" lane testing, it will be available for any tester who have bandwidth to pick on it.

Entry Criteria

- Stories are groomed and estimated
- Build is complete
- Back-end changes are deployed in Openshift
- UI card has a build version tagged to it for testers to deploy prior to start of execution

Once a tester picks the card, its moved to IN TEST queue and testing would be performed on 2 folds

1. Testing of the story per the acceptance criteria
2. Test Automation for UI changes in Mock UI environment

The test card associated to the user story will be scheduled to a Release and Test Cycle in JIRA for execution. All test cases associated to the test type card will be run to confirm if the story can be moved to REVIEW.

In event of any failures the test card is marked as FAIL and associated defects are logged against the story card and assigned back to developer for fix. The process of Defect logging and tracking is detailed under Defect Management section in this document

Exit Criteria

1. Test Type card is executed fully and all steps are marked PASS / DE-SCOPED as applicable
2. Test card is executed against the Release and Cycle in JIRA
3. Pass Logs are uploaded against the Test Cycle run
4. Pass Logs are loaded against the Story card for Analyst Review
5. Automation code is checked in git
6. Story reviewed by Analyst and marked as DONE

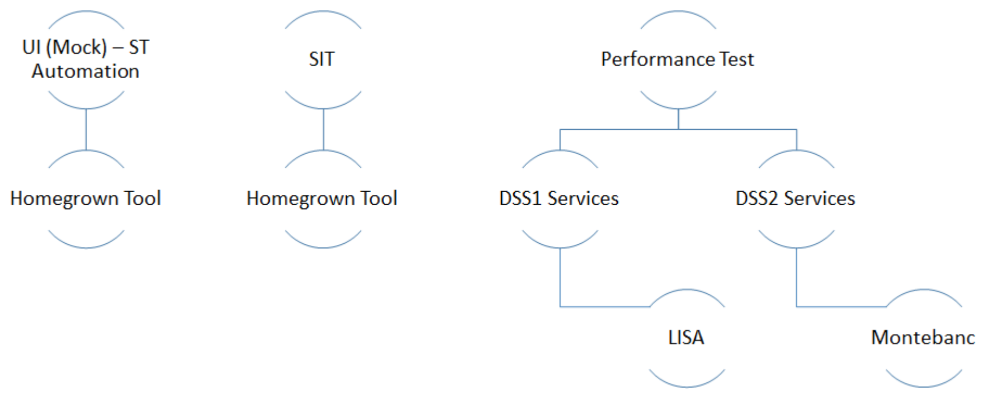
Also once testing of a end to end feature is completed test Automation for the service would be build to run in SIT environment (Regression) which would then be included for daily runs

Service Virtualization

Services are virtualized for the following purposes

1. Negative tests (especially when it comes to validating technical errors from downstream application)
2. Data variations which are hard to simulate at downstream systems
3. Reduce downstream dependency on integration which are stable and not changing to speed up testing
4. Performance Testing

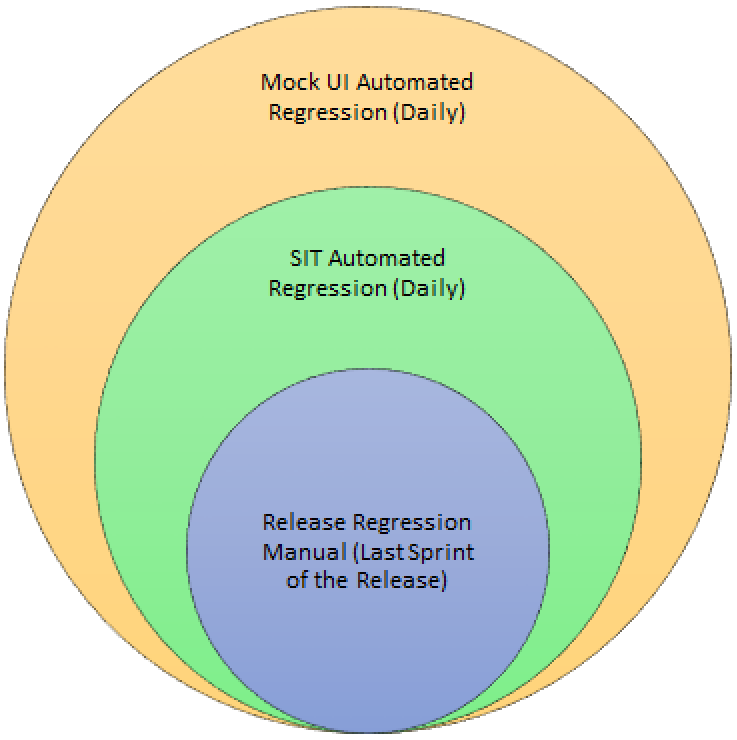
Service virtualization is delivered by a combination of multiple solutions and is evolving as the volume of work grows. The intention is to consolidate all of the stubs into Homegrown tools than reliant on external tools



Regression Testing

With automation being the focus the intention is to run as much regression upfront as possible. Regression is done in 2 stages

1. Daily automated run (UI-ST and E2E-SIT)
2. Release Regression - Last sprint of the release
    a. A clean run automated regression with over 90% of success rate
    b. Manual Regression covering regression items that are not automated yet and failures from last automated runs

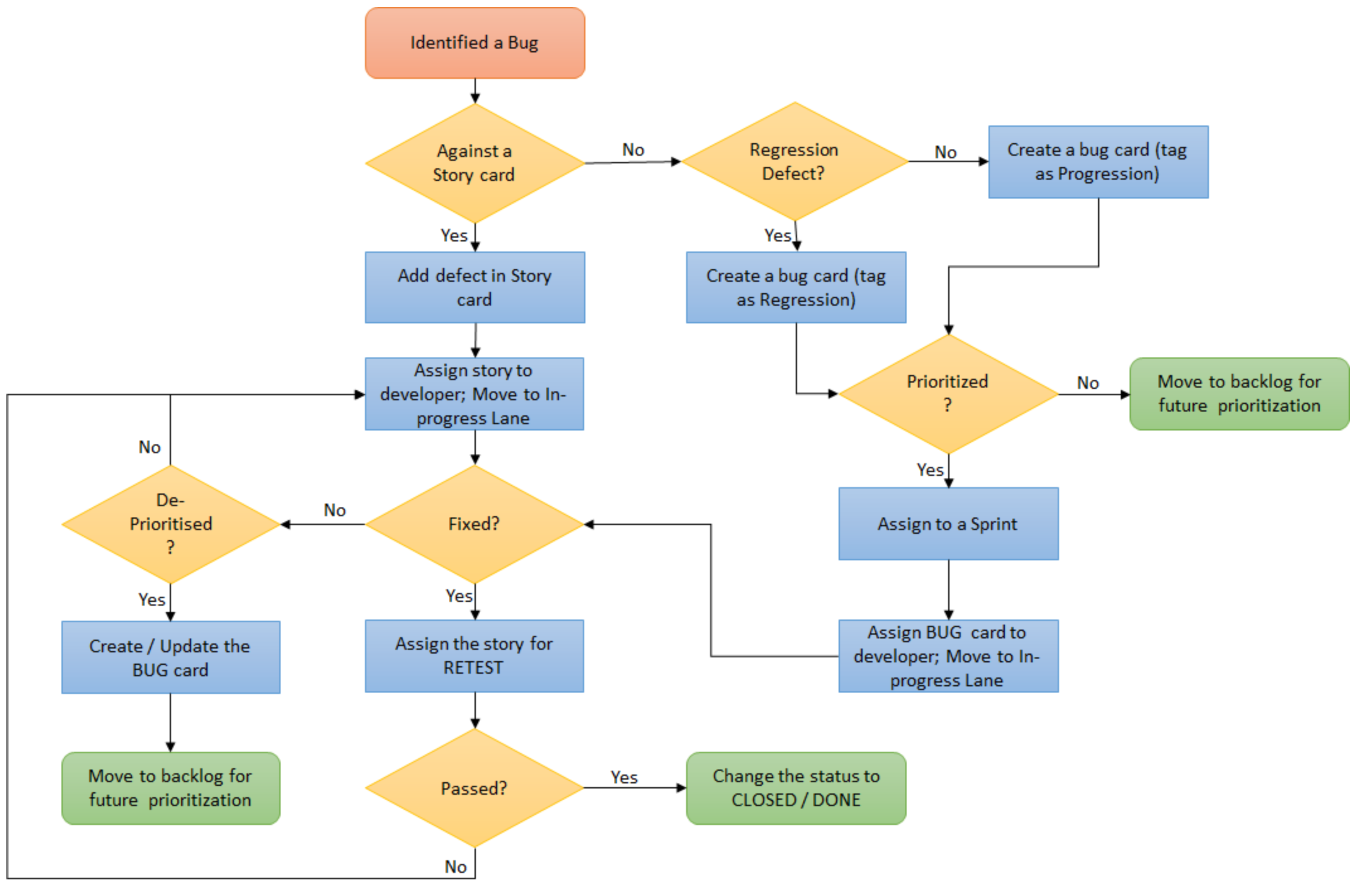

Mock UI Automated Regression (Daily)
SIT Automated Regression (Daily)
Release Regression Manual (Last Sprint of the Release)

Regression test cases are in JIRA which gets incrementally added for each release based on the changes delivered. The same is scheduled under Regression Test Cycle in JIRA and executed at the end of the release

Defect Management

Testing is performed based on User Stories and hence majority of the defects are managed within the User Story. During test execution, if any issue is identified against the story card under test the issue is recorded against the DESCRIPTION field as a table at the bottom documenting the issue #, Description, Status and Other Comments. Once all the testing is completed against the card, the story card is assigned back to the developer and moved to IN-PROGRESS lane in JIRA for defect fixes. Upon fixing the issue, the story card would then be moved back to READY FOR TEST / IN-TEST for retest. Any issues that are not fixed within the story card and are being carried over will be logged in and tracked as a BUG type cards

If there are any defects identified that are outside of a story card under the sprint (for eg: Regression), a BUG type issue will be created in JIRA and would be prioritized with Product Owner for a fix. The following exhibit covers the defect management flow for DSS


Identified a Bug
Against a Story card
No
Regression Defect?
No
Create a bug card (tag as Progression)
Yes
Add defect in Story card
Yes
Create a bug card (tag as Regression)
Assign story to developer; Move to In-progress Lane
Prioritized ?
No
Move to backlog for future prioritization
No
Yes
De-Prioritised ?
No
Fixed?
Assign to a Sprint
Yes
Yes
Create / Update the BUG card
Assign the story for RETEST
Assign BUG card to developer; Move to In-progress Lane
Move to backlog for future prioritization
Passed?
Yes
Change the status to CLOSED / DONE
No

Test Automation

Test automation is performed at various levels within DSS

1. UI Test Automation
2. End to End Test Automation
3. API Test Automation

Refer the page Test Automation for details

User Acceptance Testing

The application is for external large corporate customers and they do not have access to non-production environment, hence User Acceptance Testing is not in scope for DSS. Instead Business Acceptance of the solution is primarily considered as a lieu in place of a end user acceptance test.

Since majority of the activity is done within the squad, there is no separate User Acceptance Test team involved in this activity. The acceptance is split into 3 levels

Level 1: At individual Story

Once testing is completed on a individual story card, its moved to STORY IN REVIEW lane and assigned to the analyst who authored the story. Analyst performs a validation on the testing conducted against the acceptance criteria and perform a basic check on the functionality before marking the card as STORY DONE

Level 2: At Release Scope

At the end of the release (in the last sprint) a bug bash is organised with the Analysts in the team. Its scheduled for 30 mins to 1 hr depending on the scope of the release. The intention is for the analyst to use the window to perform exploratory test on the new and existing functions and identify any bugs. Issues identified during this process are logged in a BUG cards or if its a change request as STORY CARDs

Level 3: Sign-off on Squad Test outcome

Test Summary Report produced at the end of the release is reviewed and signed-off by the Business Product Owner confirming the test coverage and the outstanding issues / risks raised within

Penetration Testing

<TO BE UPDATED>

Test Environment

Refer to the page Test Environment for details. The below table can be used as reference to access to appropriate child pages

Topic	Link
Environment Links	Test Environment
Decision on Environment Usage	Environment Strategy
Integration Apps PoC	DSS Integration Partners - Point of contact
DownTime Tracker	SIT - Environment Downtime

Test Data

Most of

- Login User Details - Test Data / User Details
- Transaction Details - Transaction Test data
- Data for Automation runs - Automation Test Data

Performance and Volume Testing

Performance and Volume Test is performed for each DSS Release. The detailed can be found in Performance and Volume Testing page

Risk and Issues