Test strategy for SALT Application

Contents

Purpose

The purpose of this document is to provide a strategy for Regression testing of SALT Application. This document is created to solve below existing problem in SALT Application regression testing:

1. Currently Regression testing cycle requires minimum 300 hours of manual effort, how to reduce manual effort during regression?
2. How to effectively use manual testing resources
3. How to track & manage automation test cases to increase confidence before a release?
4. How to effectively use automation tools and test cases?
5. How to do identify test cases that are needed to be executed for a particular release?
6. How to manage regression test cases (create cases for new features/update existing test case in application/delete cases if not adding any value)?
7. How to manage traceability matrix and reports to share with different stakeholders?
8. Resource allocation in feature team (QA who will be part of feature sprint team) and standalone testing team (QA who will do testing activity like maintenance of Regression/Acceptance/Smoke suits and exploratory testing of the application).
9. How to establish collaboration between feature team standalone testing team?
10. How to increase compatibility to use specFlow BDD feature effectively and increase product knowledge?
11. How to move feature automation in feature development sprint (from n-1 sprint to n sprint)?

Improve product knowledge of testing team

No strategy or policy will help until testers have sound knowledge of the application. As of now most of the testers are new in SALT team and they are dependent on 1 or 2 experienced resources for functional review of their test cases (automation script / Manual test case). This is hampering the productivity of the testing team. As SALT is a big application so we should do the below mentioned activities to increase product knowledge of the testing team and thus the throughput of the team:

1. Divide the Product in Logical modules (maybe 4 or 5 Modules).
2. Identify SME's for each module or group of modules.
3. Do KT session between BA's or product owner focusing on modules.
4. Assign **one tester to one module** for long duration eg. 2-3 releases.
5. Involve testers in all release planning meetings
6. Encourage them to talk of project experience and different exploratory talk on project.
7. Create module wise documentation and assessment. All testers must go through documentation and complete the assessment.

8. Include this product training and assessment as part of your on-boarding process of the new joinee in testing team.
9. Conduct frequent on going KT session by Identified SME's. Minimum 2 hrs session/week for each tester should be planned for Product Knowledge.
10. During new feature development and existing feature enhancement, testing team should come with questionnaire to be answered by product owner. This questionnaire should contain question which can help to understand the new feature to testing team. e.g.
    - Why this feature is needed?
    - How this feature will benefit to customer and to business?
    - What is the impact area of this feature?
    - How this feature will be used by end user?
    - How should this feature look like?
    - What is expected load on testing for this feature?

Improve technical knowledge of testing team

Most of the automation test engineers in SALT are new in automation testing itself or new to this automation testing environment (C#, .net framework , SpecFlow, Selenium Webdriver, Visual Studio and Page Object Model, bitbucket , git commands). Because of lack of knowledge, team velocity is less in automating new feature in development sprint and during writing module regression cases. To improve technical knowledge we need to below activities:

1. Conduct training (online eg. pluralsight or in house training) and practice sessions for team. And if possible do assessment and certification of learned skill
2. Follow some best practices for writing automation resource like
    - Creating Page Object
    - Creating Steps
    - Creating Feature files
3. SME's of related module should do functional review of the cases for its truthiness.
4. Automation tester should create pull request in bitbucket once functional review is completed for a scenario / feature (functional review can also be done after pull request created).
5. Technical SME should do code review before the code get merged in master branch. Code review should be completed within 24 hours. And once code is accepted it should be merged in master branch in next 12 hours.

Structuring Testing Team

Currently in SALT we have 2 level of automation testing work. 1) Automating feature development stories 2) Automating module level Regression backlog. To do this we will divide automation test engineers in 2 teams:

1. **<u>Feature team</u>**: Each feature team should have atleast 2 automation test engineers. One out of this test engineer should be SME of related module. This team will be responsible for:
   a. Doing acceptance testing, exploratory testing and edge scenario testing of new development.
   b. Updating/deleting automation /manual test case for new feature once release is completed.
   c. Add story in Regression automation backlog for new sprint feature with details like impact on other area of the application by feature, existing regression scenario name which can be updated to accommodate this feature for regression run.
   d. Maintaining Smoke automation test suite.
   e. Should also automate bugs found during feature testing
   f. Execute new feature related testcase every day till regression cycle start.
   g. During regression cycle:
      i. Identify entry / exit criteria of regression cycle
      ii. Identify the testsuite/ testcase to be executed .
      iii. Manually execute testcase which got failed during automation execution.
      iv. Coordinate with standalone team to plan and run Manual regression execution.
      v. Verify bug reported and set priority and severity for that. Track the bug status. And make sure bug got verified and closed once bug got fixed in application.
      vi. Plan and conduct timebound (1hr) exploratory testing session (include all testers, developers and managers and assign small -2 module or feature to test) and focus should be report as many bugs as you can in excel sheet. Once bug all bug reported then compile the bug list and report unique bugs in JIRA with priority and severity . Repeat this exercise 1-2 time.

2. **<u>Standalone Testing Team:</u>** All remaining automation test engineers should be part of this team. This team will be mainly responsible for:
   a. Prioritize the regression automation backlog with help of SME's, Product Owner and release backlog.
   b. Automating test case as per Automation Regression backlog.

c. Run Acceptance test cases as per agreed frequency (alternate night or every night or twice in a week).

d. Maintain the existing automated acceptance test suite. To improve effectiveness, run it for different supported configurations (e.g. Browsers, localization, and other configuration). It will help to have automation acceptance suite up to date and increase confidence to release the application.

e. Plan and do mock regression suite execution once or twice before release regression run with the purpose of finding early bugs and fixing the automation suite for changes in application during development.

f. During regression cycle:

   i. Standalone team should be responsible for all acceptance cases execution (no maintenance of acceptance suite) and should share analysis of execution result with feature team.

   ii. If some test case (acceptance/regression) are broken and need to be fixed then those should be verified manually by feature team to complete regression. And should be added in regression backlog for updating with high priority.

   iii. Should report bug found in regression cycle and assign to feature team.

Categories of automation test suites

Automation Test Cases should be divided in below 3 categories:

<u>Functional Test cases</u>: The purpose of functional tests is to explore the behaviour of specific business functions and highlight corner cases. The automation cases that contain happy path and edge cases need to be run frequently (alternate night or every night or twice in a week) to ensure unbroken basic workflow of the application and maintaining the acceptance suite . Add tag '@Acceptance' for such scenarios.

<u>Smoke Test cases :</u> The automation cases that need to be run before accepting the build for further testing , smoke test suite should not take more than 1 hour to complete execution. Add tag '@Smoke' for such scenario

<u>Regression Test cases:</u> The purpose of regression testing is to give an overview of the entire system. The automation cases that need to run for once per release, these cases need to cover impact on all areas of application. Add tag '@Regression' for such scenario.

<u>Note:</u> Regression test suite!= sum(Functional test suites)

Process of authoring automation test cases

Functional Test cases: In an ideal BDD context, functional tests are used to prove that the team has delivered the functionality that was planned for a sprint and the development of the functional test cases is part of the sprint in which the functionality is developed. The functional test cases thus become a basis for the product owner to accept the new functionality.

The above approach works well when your application GUI is stable and new development is more towards functionality that GUI development and testing team have already achieved 90% + automation for existing functionality.

In case the above criteria are not met then recommendation is to automate functional tests cases for previous sprints (n-1). In this case in development sprint ('n' sprint) tester will :

- Author acceptance scenario in feature file and add tag '@manual'
- Test this functionality manually for acceptance criteria and for edge cases.

And in next sprint tester will:

- Write the automation code for scenario created in previous sprint to make this scenario executable and replace tag '@Manual' with '@Functional'.
- This scenario will be added to acceptance suite and need to run more frequently on build changes.

In case of SALT application it seems that GUI of the application is stable but automation coverage is very less (it is because for old automation test case migration in BDD format). So till automation coverage reaches 90 % +, new automation acceptance cases should be written in n-1 sprint.

Once SALT have 90%+ automation coverage and have well tested domain language steps repository then new feature acceptance automation should move to feature development sprint (i.e sprint n).

Smoke Test cases: Write a smoke test case at page level, should be written and updated by feature team. A smoke case should check basic things like existence of the new feature, for button click action, for text field some input action . It will not verify and business logic.

Regression Test cases: Regression test cases can be developed in sprint by the feature team or out of sprint by standalone team.

As of now SALT automation coverage is very less, only 20% modules have been automated yet. And 80% module are in regression automation backlog, so in this situation regression cases should be written and maintained out of sprint by standalone team. The process of regression automation should be as follows:

- In sprint planning meeting, the area of impact on application of the each story should be discussed and communicated to all testing stakeholders.

- Automation tester in feature team will author acceptance criteria and create a story item in regression backlog of out sprint (standalone automation team) for regression case to be added or updated.
- Standalone team will have a separate sprint focusing on Regression backlog and prioritizing backlog items as per impact analysis shared by product owner in feature sprint planning. Standalone team will also prioritize item on basis of:
    1. Critical path of the application
    2. Area which have more bugs in production or in past execution
    3. Area which have high risk
    4. Covering requirement gaps
    5. Cases need to be updated
- Some functional tests can be upgraded to regression tests, when they cover certain business functions better than existing tests. The existing tests can then be taken out of the regression test suite.

When SALT application will achieve 90% + automation coverage then it will have a robust and reliable domain language step repository and thus effort required for writing regression cases will be very less.

Then we can start writing/updating regression cases as part of feature sprint. As Regression test scripts are updated in the sprint when necessary. This will become part of the User Story and should be taken into account when estimating.

Process of maintaining automation test cases

Acceptance cases:

- After each sprint created acceptance cases and automated bugs should be re visited, if they do not add any value in acceptance and regression suite then should be deleted.
- If Acceptance case or automated bug can be merged with an existing case, then should be merged.
- If Acceptance case can be used for regression purpose, then tag '@Functional' should be replaced with '@Regression' and test case should be moved to regression suite

Smoke cases: Keep this suite as small as possible, it should have only stable cases. Add @ignore tag for unstable feature or scenario. And fix those case as high priority.

Regression case: Always keep regression suite up to date , for this we need to run regression suite once or twice before regression cycle to do health checkup regression suite. If regression case does not add any value (Regression testcase never find any bug because that module is very stable and new development does not impact it by any means) then add tag '@ignore' to remove them from regression suite (do not remove them physically from regression suite).

Process of identify automation test cases for regression cycle

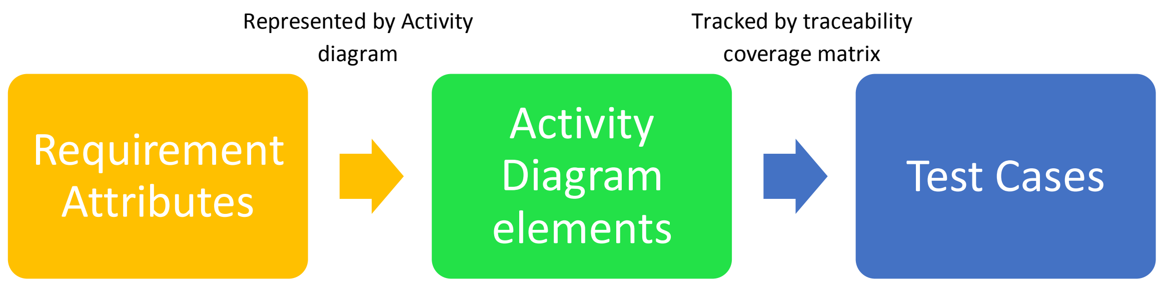
Identification of regression test case to be used can be done by two ways:

1. **Full Regression run:** In this case all test cases (manual + automation) will be executed. The modules which have regression automation coverage (currently 20%) will be validated using automation execution and for remaining module (80%) test execution will be done manually.
   And exploratory testing should be performed for impacted modules along with previously mentioned scripted testing (manual + automation)

2. **Target based regression run:** In this approach Impact area analysis need to be done at two level:
   1. **Code level Impact analysis**: Use code coverage tool eg. OpenCover to get visibility into automation testing coverage. Once your code gets modified then select regression test cases which are doing coverage of that related code.
      OpenCover can be set up and used by referring article: "https://www.codeproject.com/Articles/677691/Getting-code-coverage-from-your-NET-testing-using". Traceability matrix should be created and maintained for impact code analysis.
   2. **Functional level Impact analysis:** Do functional impact analysis and select test case and test suite for impacted area. Requirement Traceability matrix needs to be created and maintained to use this technique effectively.

   In regression suite all test cases identified by above two techniques should be added for regression run

3. **Specification-based regression run:** In this technique the test cases are distinguished into two types- target cases and the safety cases. The cases which validate the affected requirement attributes are called as target cases. The cases that are selected to reach the predefined coverage target are called as safety targets. These are incorporated on the basis of risk analysis.
   There are a number of steps required to select target test cases. The first step involves creation of a traceability matrix. Traceability specifies which requirement attribute is exercised by which test case.


Represented by Activity diagram
Tracked by traceability coverage matrix
Requirement Attributes
Activity Diagram elements
Test Cases

If a program code is modified, the specifications may change. In the next step, the activity diagram is traversed and all the nodes and edges affected due to modification in the program are recognised. Then in the next step, all those test cases which validate those edges are selected by using the traceability matrix created. These test cases are known as target test cases.

Then, safety test cases are selected which also involves a few steps. The first step is calculation of the cost of each case. The next step involves the calculation of severity probability which is achieved by multiplying total defects and the average severity of defects. The next step involves the calculation of risk exposure which is done by the multiplication of cost and severity probability. The result of this is considered to be the risk. The last and the final step involved the selection of those test cases which have a higher value of risk.

In safety test cases also add test case covering the below condition:

a. Ensure Critical workflow works
b. All bug found have been resolved
c. And some amount on negative testing.
d. Verify critical bugs which got fixed in previous releases.

In safety test case also add all integration test cases and complex test cases.

Executing Regression test suite

Once the regression test cases are identified and added in the regression suite based on the criteria, then below activity should start. In this activity feature team and standalone team will share responsibility as mentioned in section 'structuring Testing Team '.

1. In QAC make a test set for the release. Add all identify test case in test set.
2. Run all identified automation regression test cases outside of QAC and update result against each case (either manually or using some API call from automation framework). Attach automation result file for release test set in QAC.
3. Run all identified Manual regression cases using QAC and update all result for that run.

Traceability Matrix:

In SALT we need following traceability matrices:

1. **Requirement Traceability matrix:** It should specify which requirement attribute is exercised by which test case.
2. **Automation Traceability matrix:** It should specify corresponding manual test case/cases for each automation test case.
3. **Testcase code coverage Matrix:** This matrix will be created with the help of a code coverage tool. It will help to identify the test cases w.r.t changes in the code.

4. **Matrix to trace bugs found in application module:** More matrix can be added as per the need in the future.

   In SALT application, there is a need to write/update manual test cases (Acceptance and Regression) in identified test management tool. Then the flag/status needs to be updated as automated, blocked, in progress or manual. In the Note or description field of the test case, name of automated feature file and scenario should be updated. Scenario may have testcase id as tag for reporting purpose.

   If BDD is implemented at a later point of time, tags can be used for execution of desired test case and tracing purpose.

Using QAC for Test Management:

1. Manual cases should have a flag or attribute like tracking Jira id of the story.
2. Manual test cases should be marked as 'Automated/Manual'
3. A separate template TestSet should be created that contains all regression cases, acceptance cases and smoke cases.
4. For each release New TestSet should be created based on existing template and once execution has been completed then status should be updated for each test case and should be attached to the Specflow report for that Testset.