Impacted Tests Based On Function Coverage

Shagufta, Software Engineer
Shubhi Gupta, Software Engineer
Rakesh Jha, Quality Engineering Manager

Adobe Systems India Pvt Ltd.
Abstract

Rapid Release model of Software Development demands an expeditious shipment of product to market. The continuous integration process in turn requires frequent certification of the product build which necessitates the execution of a large number of tests. An optimization to this can be attained by an intelligent selection of a test suite for execution. Code changes can serve as reliable indicator for identifying the tests which need to be run for a particular build of the product. Assessment of the probability of failure can be done by running only the tests which invokes THE FUNCTIONS THAT HAVE BEEN CHANGED in the code (based on Code Coverage). This leads to a shorter test execution window and saving of computing resources. So, the extent of testing / the set of test cases that are to be chosen for a particular requirement are the ones which have their test requirements impacted by the function changes.

Background

Two years back Acrobat team switched to continuous release model and one of the challenge was to setup continuous integration process. There was a system in place to build and test but it required several changes to make it a rapid integration system. The changes were required in versioning and build process which production engineering team owned. There were changes required in test automation execution process as well. The changes were required to achieve following objectives and Automation Team had to come up with a solution.

1. Automation tests should be able to certify every change going into build.
2. Shorter build creation and test execution window so that developer gets build feedback as soon as possible.
3. Block bad changes into system so that daily builds are stable all the time.
The older system

Before we talk about the solution, it is important to understand the system which existed at that point of time. The Acrobat desktop team has 100+ developers, all contributing to same code base managed in a code repository. To ease up the maintenance and change conflicts, team devised a multi-layered integration system which is divided into branches at following three levels.

Feature Branch

This is the lowest level branch and developers submit any code change to these branches only. There are around 20 such branches in Acrobat divided as per features like PDF creation, PDF viewing etc. There are many code submissions to feature branch everyday by multiple developers. Teams working on features ultimately integrate to the next level branches namely the “Integration Branches” once their feature is stable.

Integration Branch

There are five integration branches into which feature branches are merged and ultimately these integration branches integrate into the uppermost shipment branch. The integration branches are owned by subgroups and typically belong to one engineering head. The one person from the subgroup is assigned as the branch custodian. The custodian of the integration branch scrutinizes every change from feature branches and merges only the good ones into his branch which finally integrates into the shipment branch. Every branch custodian is allocated one day in a week for integrating his branch into the shipment branch.

Shipment Branch

This is our top level daily build branch and it produces one build every day. This is managed by production engineering team and considered stable for sharing build with other teams. All other branches are derived from the shipment branch and ultimately merges into it.
The problem with older system

Acrobat desktop has 8K+ automated feature tests and one full cycle of execution take around 24 hours on our testing machines. If all tests have to be executed on all branches (20 + 5 + 1) and for all submissions then,

1. Developer has to wait for a minimum of 24 hours before he gets to know if he made a bad submission.
2. Most of the tests in feature branches are not impacted by the change so executing all tests is waste of resources and time.
3. We didn’t have the infra to run all the tests for all branches.

To counter these problems sometime back, we created test collections based on the features. Post this change, only the shipment branch was configured to execute all the tests and other branches were configured to execute collections corresponding to their features. This reduced the turnaround time for automated test execution to approximately 6 hours but still we had the following problems to solve:

1. In order to rollback a bad change we needed a turnaround time of less than an hour.
2. Still too many tests (~500) were executed and were consuming computing resources.
3. There were changes impacting across features e.g. a change in Forms area impacting the functionality in Commenting area. These were caught late when feature reached to shipment branch.

The solution

The Automation team proposed and developed a system (called IMPACTED) based on code coverage which identifies the tests covering functions modified in a submission. This list of impacted tests is then passed to test automation framework for the execution, instead of running full tests or collections. This system has two parts, first the “Test Profiler” and second the “Test Provider”. Below is the brief description of each of these parts.
**Test Profiler**

The job of profiler is to create the functional profile of a test which is actually a list of functions covered by a given test. It works in following way:

1. Execute each test with code coverage build (Code Coverage Tool).
2. Parse the code coverage file of a test to get the list of functions covered.
3. Store the information of functions covered for a test in database.

**Test Provider**

This provides list of tests which cover the functions changed in a given submission. It works in following way:

1. This system takes a change-list as input. The change-list is an entity which contains list of files submitted for a change.
2. Parse each file in change-list and using code difference tool to identify the lines which have been changed.
3. Use CTAG library for each changed line to identify the function which contains changed line.
4. Create a list of all the functions which changed in the given CL.
5. Use the changed functions list to query test profiler database to identify the tests which cover these functions.

**Putting it all together**

The moment any change goes into feature branches, a build is triggered and as soon as build is available a request is send to our IMPACTED tool to provide the list of impacted tests. The IMPACTED tool based on test profiler database returns the list of tests to be executed.

**How did it make a difference to the business or organization?**

The Test Automation Framework now executes only the tests identified by the IMPACTED tool for a given submission. Since changes in feature branches are smaller in quantum, now on an average 150-300 tests are executed in these branches for a submission and it takes around an hour.
This helped us to achieve the following objectives for continuous integration system:

1. All the tests which are impacted by the change are executed without wastage of computing resources and saving money in terms of dollars spent on computation resources.
2. The shorter test execution windows enabled us to rollback a bad change immediately. So now a submission is rolled back even if a single test fails and team has to fix it before proceeding.
3. Developers get the response within an hour and they can fix the problem when context is still fresh in their minds.

There are other benefits as well. The team gets to know about the failure as soon as it reaches into builds and rollback enforces quick fix of failures. This way we now get stable build at each level allowing us to do frequent Beta releases.
References & Appendix

https://stackify.com/code-coverage-tools/


Author Biography


The first co-author is Shubhi Gupta, Software Engineer in Acrobat Team. Involved into developing Automation Tools and writing Automation Tests.

The second co-author is Rakesh Jha, Quality Engineering Manager in Acrobat Team. Involved majorly in Acrobat security, commenting and Automation projects.

We are collectively responsible for developing Impacted Tests Based On Function Coverage.
THANK YOU!