Case Study

Behavior Driven Development (BDD)

Karan Vaid
Associate Lead
Think Future Technologies

About the client

Company is a stock photography agency headquartered in New York City, New York and United States. It maintains a library of millions of royalty-free stock photos, vector graphics, and illustrations, and sizeable video clips for license.

Industry

Photography

Business needs

Client wanted to reduce the release cycle by reducing the time taken by regression tests.
A special requirement was that the upper management wanted transparency in what was being tested.

Solution

Over a period of time, helped them arrive with QA service that grew and evolved with Company’s needs.

Scope

This paper is an attempt to outline the benefits of BDD based approach in automation testing. This will also make it clear that just using Cucumber or similar tools to write your automated tests, is not actually a BDD.

Resources

- Manual Testing Team - These folks were responsible for coordinating with Subject Matter Experts and design manual tests
- Framework Development and Test Automation - Resources on this team were assigned the task to develop framework in order to support automation team.

Using Cucumber to Write Automated Tests:

- The purpose of using a BDD tool like Cucumber in an automated testing scenario is to boost readability and promote the involvement of non-technical/business people into what is being tested. Other than Ruby (which cucumber itself is written in), implementations for cucumber exist for Java, JavaScript and Python.
- A Cucumber setup has a standard directory structure which makes it easily adaptable for someone who has prior experience on Cucumber easy to understand the test framework.

Process/Learnings:

- After a feature has been developed, automated tests using Cucumber were written. The steps inside a scenario were written usually by tester/developer.
- When writing the scenarios, we always spent some time to check and search for and reuse any existing step definitions.
- A separate Git repository was used for pushing automated tests
- The tests were run on multiple environment including Dev and QA. On every Jenkins deployment in the Dev and QA envs., the tests were run in the same environment. This was done by a trigger after the deployment finished. The tests were run on a grid of systems to minimize overall execution time of the test suite.
- To further minimize the execution time, the scenarios requiring a logged-in user or a new user were made to hit an internal route with appropriate data which automatically created the user in the backend, saving us so much of time we’d have wasted in going through the entire sign-up process on the website repeatedly for every test.
- Cucumber comes with a runner which generates beautiful reports, so after the tests were run, the failing scenarios could be easily be read by a non-technical person.

- The reports also included the information on which step the scenario failed. This is very important as it makes the entire staff including the upper management independent in finding out what broke.

Behavior Driven Development v/s Using Cucumber in Automated Tests

- BDD in real sense is development of features by defining their behavior. In BDD user scenarios are written followed by their step definitions (tests). The scenarios are then run to ensure that they fail. Lastly the application code is written/refactored to make the scenarios pass.
- So the behavior drives the development in true sense.
- Using BDD tools like cucumber to write automated tests is very different as it starts after the feature has been developed, in other words we start writing our tests after the application code has been written. So there really is no development driven by behavior.
- The only thing that makes it look like BDD are the Gherkin steps which are mapped to the test code. Using BDD tools like cucumber to write tests has its own benefits though.

Declarative v/s Imperative style scenarios

This is the design part of the cucumber scenarios. This is what shows your expertise in designing cucumber scenarios.
To start with a basic idea of what these styles are. Declarative and imperative only differ in the level of abstraction.
Imperative Gherkin is less abstract, more concrete, whereas Declarative Gherkin is more abstract. But that difference is fundamental.
Let’s consider an example of each of them:

- Imperative:
  Scenario: Redirect user to originally requested page after logging in
  Given a User "dave" exists with password "secret"
  And I am not logged in
  When I navigate to the home page
  Then I am redirected to the login form
  When I fill in "Username" with "dave"
  And I fill in "Password" with "secret"
  And I press "Login"
  Then I should be on the home page

- Declarative:
  Scenario: Redirect user to originally requested page after logging in
  Given I am an unauthenticated guest
  And I have a valid user account
  And I attempt to access restricted content
  When I log in
  Then I have access to the restricted content

As shown above the Declarative style is much more readable and sounds relevant with the details all abstracted away.
The Imperative style on the other hand is detailed, smells of UI, not-useful information but well parameterized and more re-usable.
Choosing between these 2 styles not an easy thing. In short, we can use imperative style for small scenarios and declarative style for long scenarios. Not forgetting to use the imperative steps inside the declarative step definitions to reuse existing code leading to better design.
Though this is a basic sense of where to use what style, the actual use is always context dependent, designing the scenarios so as to keep them readable, relevant and still promote re-usability with effective parameterization is and art what an expert would possess.

For more Information

Think Future Technologies
1st Floor, AIHP TOWER,
249G,UDYOG VIHAR,
PHASE-IV, GURGAON-122015
www.tftus.com
Email: info@tftus.com
Phone: +91 124 2807000