**Following are the main milestones involved during the development of hybrid test framework:**

**Step 1:**

1. Created maven project in eclipse. Added selenium and junit dependencies as of now in pom.xml.
2. Created overall folder structure for the hybrid framework.
3. For starters, wrote one basic acceptance test case (E2E\_Tests.java) using basic selenium syntax.

**Step 2:**

1. Converted the above written Selenium test into Cucumber BDD style test. During that process did following things:
   1. Updated pom.xml with Cucumber related dependencies.
   2. Created corresponding feature file using given, when, then gherkin keywords.
   3. Created step definition file.
   4. Created TestRunner file.
   5. Executed test as JUnit test.
   6. Deleted selenium file E2E\_Tests.java since BDD setup done.

**Step 3:**

1. Implemented Page object design pattern by creating page object file using Selenium PageFactory in Cucumber.
   1. In page class, used @findby annotation to initialize web elements and used them in writing individual java methods for appropriate actions.
   2. Updated step definition file accordingly.

**Step 4:**

1. Created a Page Object Manager class to manage the instances of page objects.
2. This was in line with implementing Singleton design pattern. This was to avoid creating object of Pages again and again in future when there will be multiple step definition files.

**Step 5:**

1. Created a configuration properties file to store hard coded values in the project.
2. Created a configuration file reader file to read those values.

**Step 6:**

1. Created File Reader Manager in line with Singleton design pattern to manage all types of file readers.
2. As of now to manage configuration file reader but in future to manage may be json file reader.

**Step 7:**

Created Web Driver Manager to manage the chrome/firefox/internet explorer web driver. In the process created or updated following files:

1. Created enums for specifying driver and environment type.
2. Updated configuration properties file with the related hard coded values and updated configuration file reader file accordingly.
3. In Web Driver Manager, wrote logic for creating, getting the already created one and closing the driver.
4. Updated step definition.

**Step 8:**

1. In future step definition files will be many. So created Test Context to implement dependency injection. Basically, to help instantiate step definition classes and write them properly.
2. Updated pom.xml with PicoContainer (DI) dependency.

**Step 9:**

Created Hooks to make a provision to do activities before and after each test scenario.

**Reporting : Step 10:**

Updated Test Runner to accommodate reporting options like 1. Pretty 2. Usage 3. html 4. json 5. Xml

**Reporting : Step 11:**

To implement advance reporting framework of Extent reports did the following:

1. Added Extent Report and Cucumber Extent Reporter dependencies in pom.xml.
2. Added Extent Configuration via an xml file to the project.
3. Updated the configuration properties file with the above mentioned xml file details and updated configuration file reader file accordingly.
4. Modified the Test Runner and Hooks file. Also updated Hooks file to accommodate screenshot capture mechanism in case of test case failure.

**Following are some of the main highlights in writing acceptance tests using above designed hybrid framework:**

1. Created two feature files. Details of which are as follows:
   1. **aaExtractExchangeRates.feature:** 
      1. The test scenarios in this are focused on capturing exchange rates for multiple currencies from the “browse all currencies” page.
      2. The idea behind this was to store live exchange rates for multiple currencies and then use it for assertions later with the ones used on the “Currency Converter” page during actual conversion.
      3. Wisely documented scenario outline for the same using country initials, from and to country name columns. Used background to avoid repetency of test steps.
      4. Implemented the same in corresponding step definitions using hybrid framework.
      5. Used assertions at each step for validations.
   2. **bbConversionRates.feature:** 
      1. The test scenarios in this are focused on testing the currency conversion logic for multiple currencies.
      2. The idea behind this was to store the converted amount based on exchange rates for multiple currencies.
      3. The next step was to recalculate the same converted amount using exchange rates captured during execution of aaExtractExchangeRates.feature file.
      4. Using assert logic to check if both are same.
      5. Wisely documented scenario outline for the same using amount, source currency, target currency columns. Used background to avoid repetency of test steps.
      6. Implemented the same in corresponding step definitions using hybrid framework.
      7. Used assertions at each step for validations.
2. Made sure that tests execute on Chrome, Firefox and IE using the hybrid framework.