Software Testing Approaches

EPAM RD, 2020

Anastasiia Rudik
Software Test Automation Engineer
rudikanastasiya@gmail.com

Content

TDD

TDD (Test Driven Development)

1. TDD is a Development practice
2. Integrated with CI

Implement new feature:

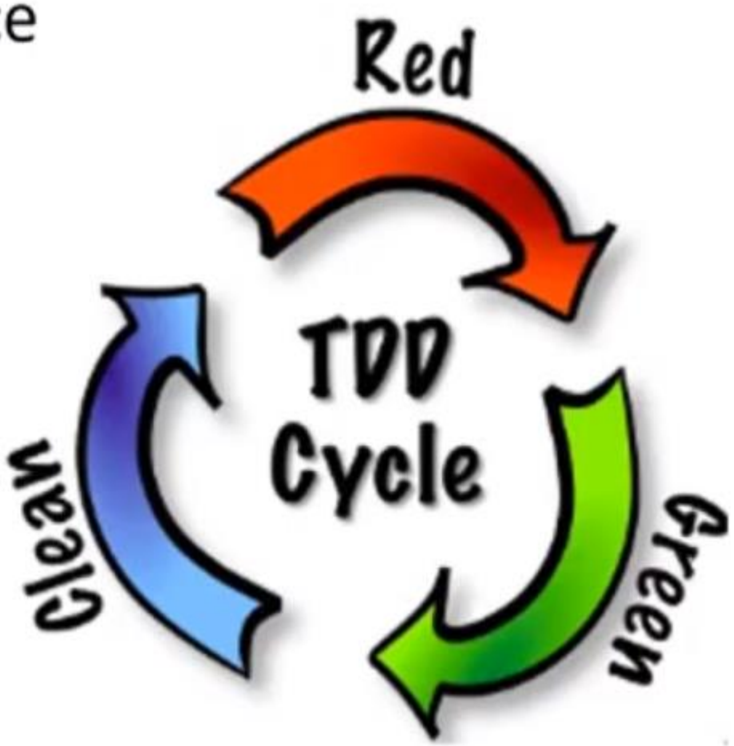
- ❑ Write test

CI > failed

- ❑ Implement feature

CI > passed

- ❑ Refactoring



TDD - 5 Rules

1. Fast: a test must be fast to be executed often.
2. Independent: tests must not depend on each other.
3. Repeatable: a test must be reproducible in any environment.
4. Self-Validating: a test must have a binary result (Failure or Success) for a quick and easy conclusion.
5. Timely: a test must be written at the appropriate time, i.e. just before the production code it will validate.

TDD - Pros

1. Forces good architecture
2. Documents your code better than documentation
3. Makes code easier to maintain and refactor
4. Makes collaboration easier and more efficient
5. Helps prevents defects
6. “Stupid” mistakes are caught almost immediately
7. Clean code

TDD - Cons

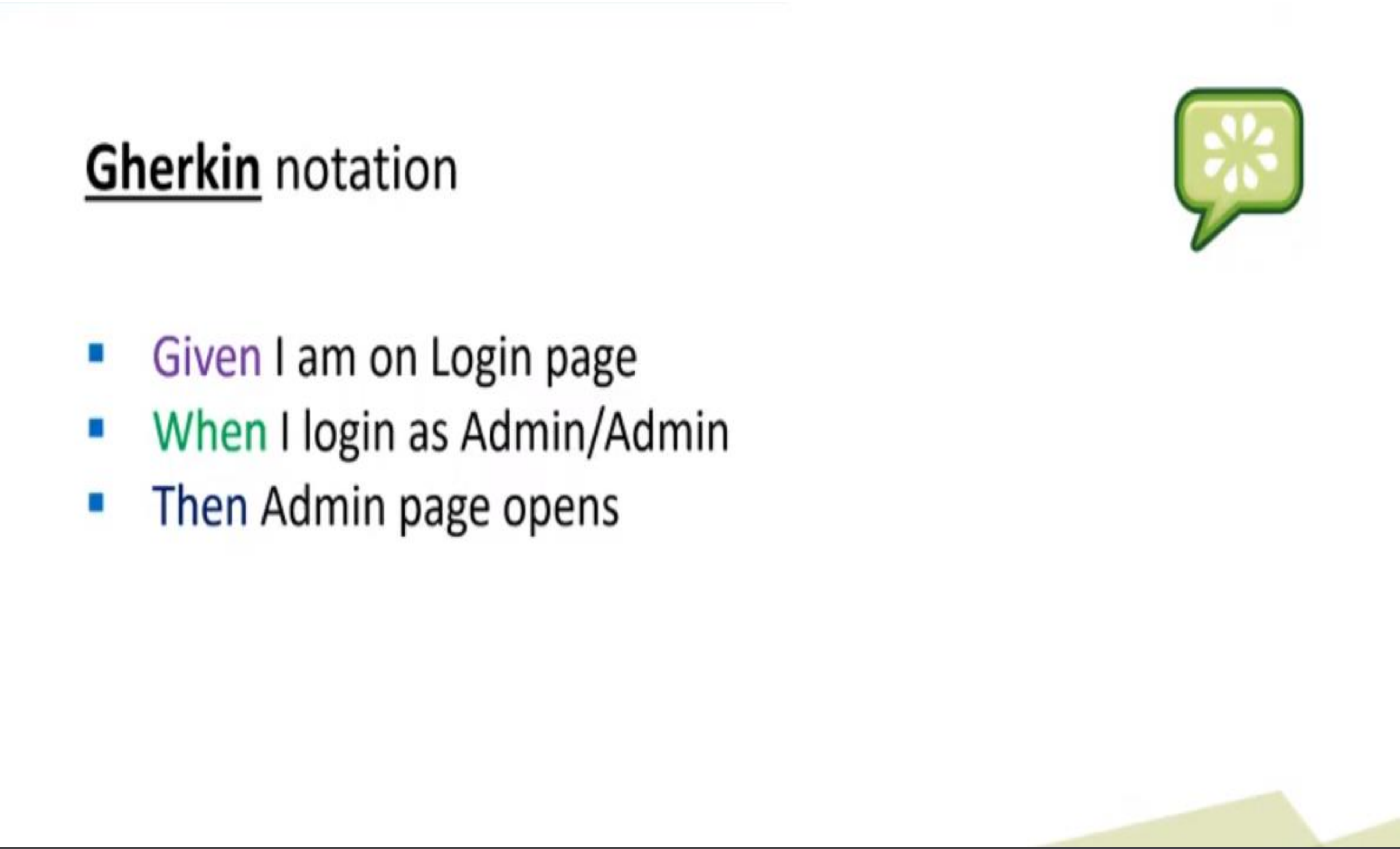
1. Hard to apply to existing legacy code.
2. Hard to start working this way without experience
3. Creating tests for failures can be tedious
4. Unless everyone on the team correctly maintains their tests, the whole system can quickly degrade.

BDD

What is BDD

BDD (Behavior Driven Development) is a software development process resulting from development through testing (TDD). This approach combines the general methods and principles of TDD with the ideas of domain design and object-oriented analysis. BDD is greatly facilitated by using a simple domain-specific language using natural language constructs (such as English sentences) that can express behavior and expected results. Test scripts have long become popular applications for subject-oriented languages with varying degrees of complexity.

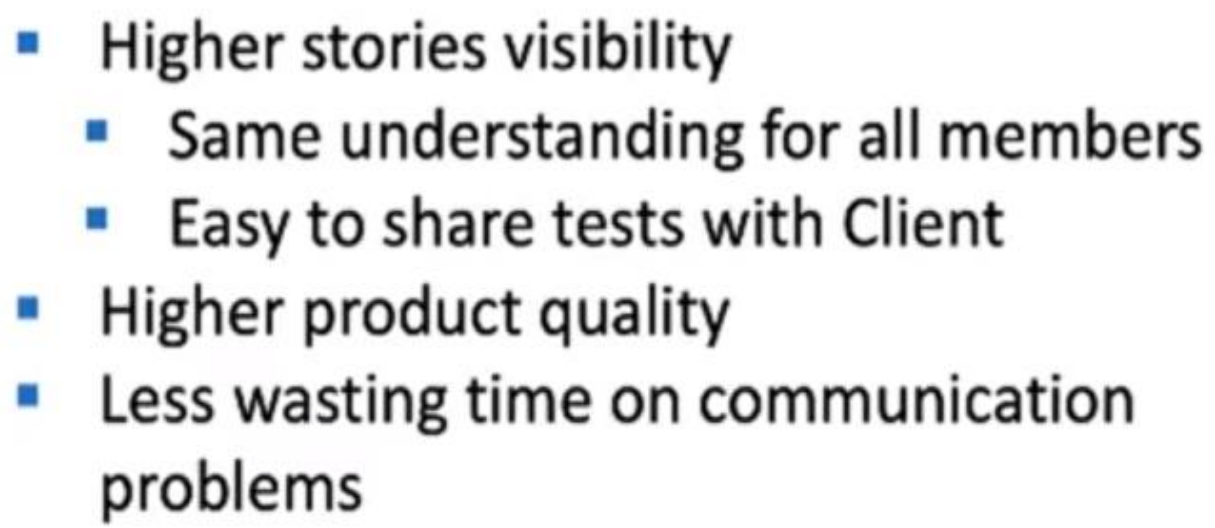
BDD language



Gherkin notation

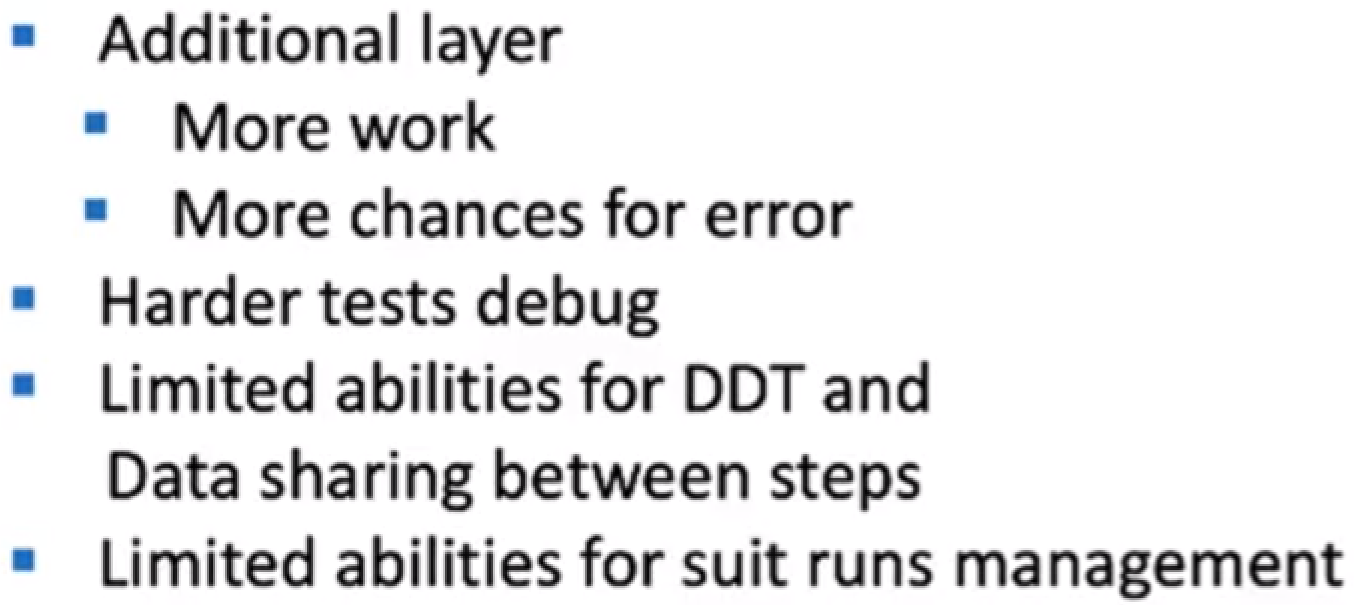
- Given I am on Login page
- When I login as Admin/Admin
- Then Admin page opens

BDD Pros



- Higher stories visibility
  - Same understanding for all members
  - Easy to share tests with Client
- Higher product quality
- Less wasting time on communication problems

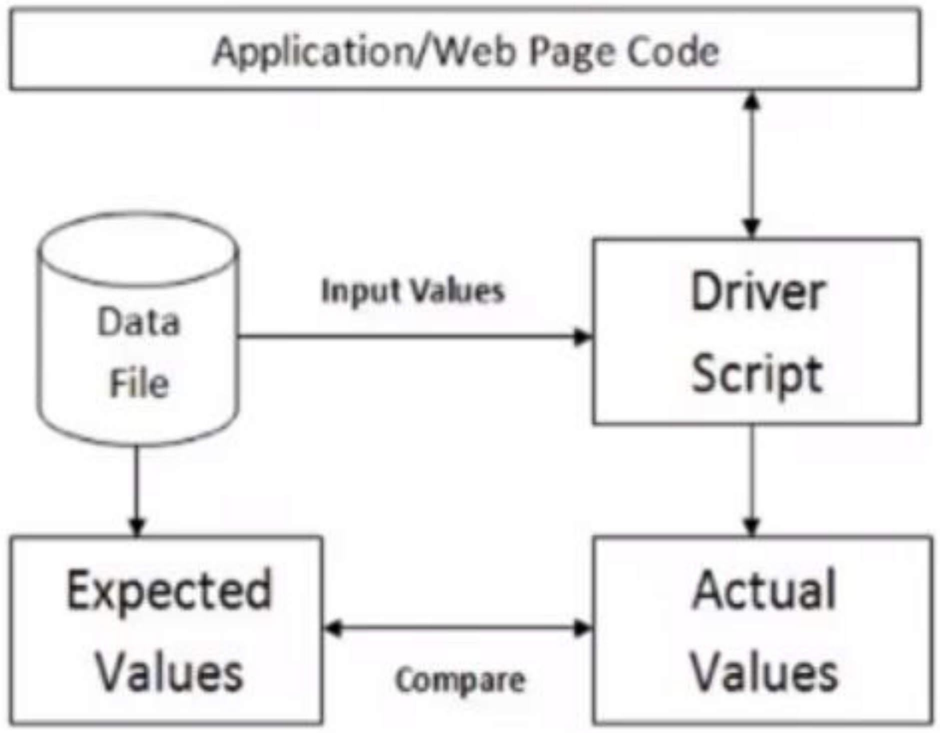
BDD Cons



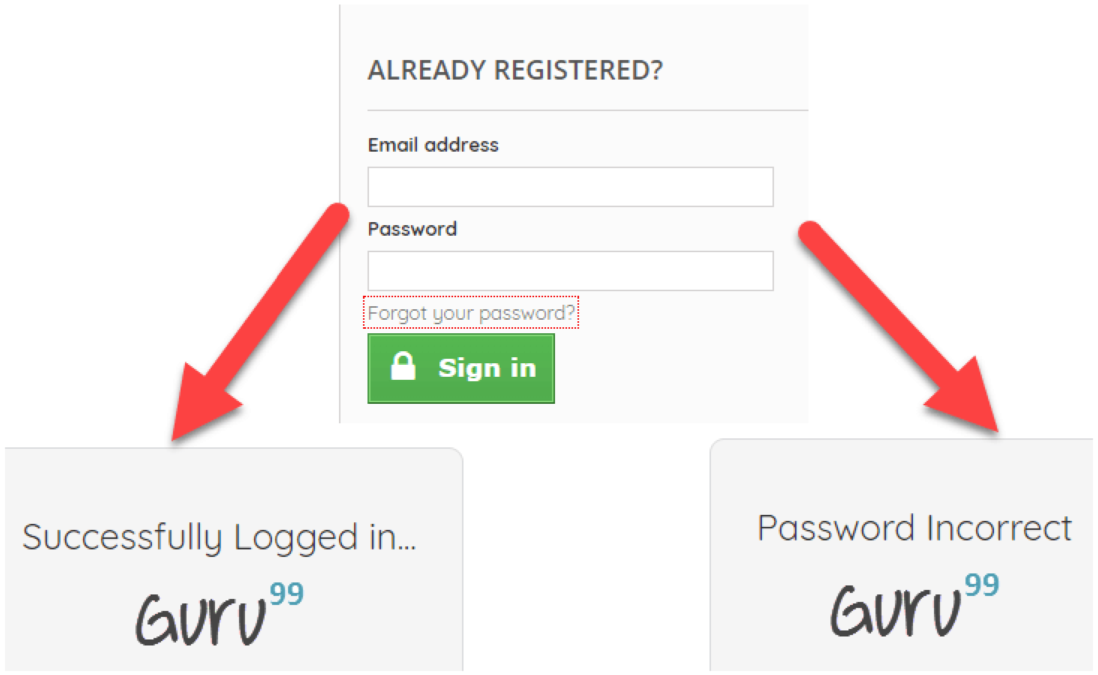
- Additional layer
  - More work
  - More chances for error
- Harder tests debug
- Limited abilities for DDT and
  Data sharing between steps
- Limited abilities for suit runs management

DDT

DDT (Data Driven Testing)



DDT - Example

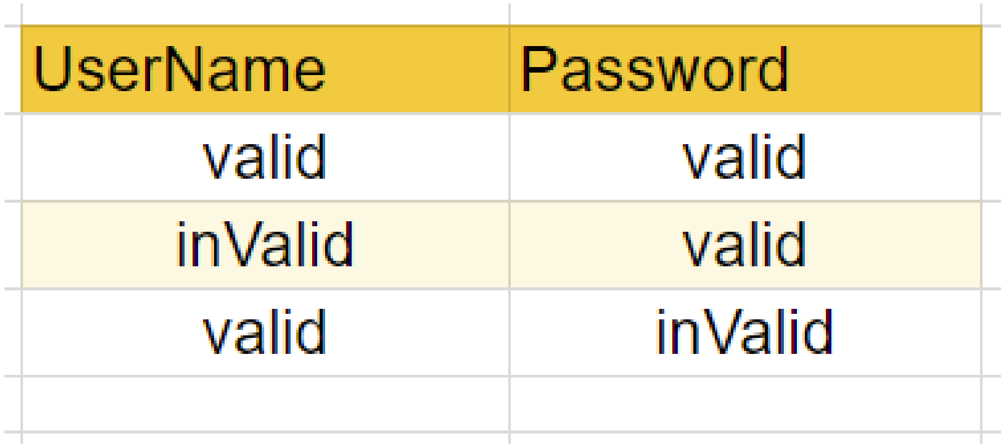


- Input Correct username and password - Login Success
- Input incorrect username and correct password - Login Failure
- Input correct username and incorrect password - Login Failure

DDT - Example

Test Case#	Description	Test Steps	Test Data	Expected Results
1	Check Login for valid credentials	1.Launch the application 2.Enter Username password 3.Click Okay 4.Check Results	Username: valid password: valid	Login Success
2	Check Login for invalid credentials	1.Launch the application 2.Enter Username password 3.Click Okay 4.Check Results	Username: invalid password: valid	Login Fail
3	Check Login for invalid credentials	1.Launch the application 2.Enter Username password 3.Click Okay 4.Check Results	Username: valid password: invalid	Login Fail

DDT - Example



UserName	Password
valid	valid
inValid	valid
valid	inValid

```
// Test Step 1: Launch Application
driver.get("URL of the Appliation");

// Test Step 2: Enter Password
txtbox_username.sendKeys("valid");

// Test Step 3: Enter Password
txtbox_password.sendKeys("invalid");

// Test Step 4: Check Results
If (Next Screen) print success else Fail
```

DDT - Example

```
// Loop 3 Times
for (i = 0; i & lt; = 3; i++) {
    // Read data from Excel and store into variables
    int input_1 = ReadExcel(i, 0);
    int input_2 = ReadExcel(i, 1);

    // Test Step 1: Launch Application
    driver.get("URL of the Application");

    // Test Step 2: Enter Password
    txtbox_username.sendKeys(input_1);
    // Test Step 3: Enter Password

    txtbox_password.sendKeys(input_2);
    // Test Step 4: Check Results
    If(Next Screen) Success
    else Fail
}
```

DDT - Where to store test data

- Comma-separated values (CSV) files
- Excel sheets
- Database table
- Parameterized class
- Script arrays
- Table variables

DDT - Pros

- Allows to test application with multiple sets of data values during Regression testing
- Test data and verification data can be organized in just one file, and it is separate from the test case logic.
- Base on the tool, it is possible to have the test scripts in a single repository. This makes the tests easy to understand, maintain and manage.
- Some tools generate test data automatically. This is useful when large volumes of random test data are necessary, which helps to save the time.
- The same test cases can be executed several times which helps to reduce test case and scripts.
- Any changes in the test script do not effect the test data

DDT - Cons

- Data validation is a time-consuming task when testing large amount of data.
- Maintenance is a big issue as large amount of coding needed for Data-Driven testing.
- High-level technical skills are required. A tester may have to learn an entirely new scripting language.
- There will be more documentation. Mostly related to scripts management tests infrastructure and testing results.

KDT

What is KDT

KDT is a software testing methodology that separates test case documentation from prescribing how to run test cases. The KDT architecture allows testers to create different versions of automated test cases using a fixed set of keywords without requiring programming skills.

KDT

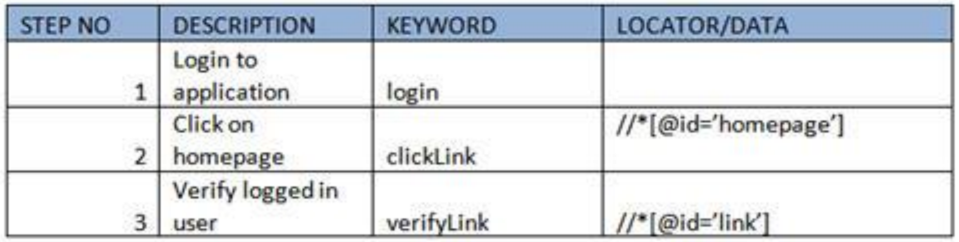
KDT framework in most cases based on three major factors:

- Vocabulary (keywords and phrases) based on the needs and terminology
- A parser (compiler), which is responsible for the keywords aggregation, syntax checking and data driven.
- Automation core framework – a set of generic service functions and element based functions, which is responsible for the test execution and reporting.

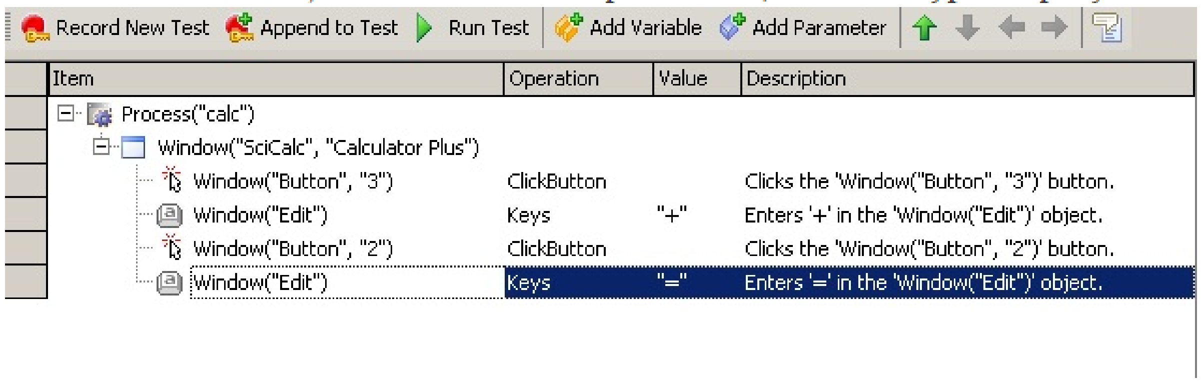
A typical keyword driven test will look like this:

Keyword	Object	Value
Open Browser	Chrome	www.google.com
Set Text	Search TextBox	"What is KDT?"
Press Button	Search Button	

Keyword Driven Testing framework



STEP NO	DESCRIPTION	KEYWORD	LOCATOR/DATA
1	Login to application	login	
2	Click on homepage	clickLink	//*[@id='homepage']
3	Verify logged in user	verifyLink	//*[@id='link']



Item	Operation	Value	Description
Process("calc")			
Window("SciCalc", "Calculator Plus")			
Window("Button", "3")	ClickButton		Clicks the 'Window("Button", "3")' button.
Window("Edit")	Keys	"+"	Enters '+' in the 'Window("Edit")' object.
Window("Button", "2")	ClickButton		Clicks the 'Window("Button", "2")' button.
Window("Edit")	Keys	"="	Enters '=' in the 'Window("Edit")' object.

KDT pros and cons

Pros

- Test cases are easy to manage, edit, modify.
- New test cases can reuse existing ones.
- Independence in programming language or automated tool implementation
- Test cases are readable for stake holders
- Test cases may be written by a manual QA engineer, BA.

Cons

- Comparing to manual testing or “Record & Play” technique KDT needs longer time to market
- The required effort to establish an effective KDT environment is **huge**
- When functionality structure is changed nothing left to reuse
- Learning proprietary vocabulary (keywords and phrases) will most likely be the same as learning a new scripting language
- Hard to maintain large scope of test scenarios

Final Task

Create Test Automation Framework for one of web sites:
https://www.ebay.com/
https://www.asos.com/
https://www.amazon.com/

Use should use: SeleniumWD, PageFactory (minimum 5 pages), TestNG
Implement 10 automation test scenarios that cover **general** features of the site. (try to write not only positive test cases but and negative too)
Separate you tests for different classes by functionality (if it’s necessary)
Create testng.xml for your tests (if you have positive and negative tests, create two testng.xml – one for negative tests, one for positive)

Deadline: 23.08.2020 23:00
Send to: rudikanastasiya@gmail.com
Subject: TA_Final_Task

Software Testing Approaches

RD, 2020

Anastasiia Rudik
Software Test Automation Engineer
rudikanastasiya@gmail.com