Pattern Driven Testing - CDT

A new automation testing framework

By Solomon A. and Anna D. (Sep. 2019)

2nd edition

This framework is recommended to be used in a system which has plenty of business processes that requires to accomplish numbers of series actions, with possible options, and set of attributes.

This framework can generate thousands of scenarios which are considered as testcase.

The CDT framework helps to:

1. Generate thousands of scenarios from possible options/combinations
2. It has scenario filter technique to pick part of it for your enough testing
3. Takes each scenario as a complete test case
4. Do an appropriate actions against each each option from the scenario
5. Set action flags so that they will be used for further actions or assertions
6. Assert actual and expected results of each selected scenario

The Framework Success Story:

The framework has been implemented on two companies of different projects:

1. **CollegeBoard => Project Advance Program (AP)**
    a. This is the project where we brought the framework idea and implemented it.
    b. The project has two complex processes (payment and billing)
        i. Payment process is factored by Course type, exam type, application time, processing time, committing time, and reordering time, ...

ii. Billing process is factored by Type of student, Subsidy by course level, exam level, subsidizing companies, by course date, exam date, …

c. Achievement :
   i. When we joined the company to support this project, they already completed the payment process but wanted to make big business requirements changes on it. The test script they had has ~2000 lines of code to cover ~40 payment scenarios out of more than 20, 000 possible payment scenarios. In addition to that most of them have been failing and were very unmanageable to fix existing one and add new scenarios on it.
   ii. Then we start defining our new approach, planned and ended to this Pattern Driven Testing framework.
   iii. The framework implementation tools ~300 lines of code (75% effective), able to cover all ~20, 000 scenarios, though we have limited to run only ~500 scenarios from enough testing rules.
   iv. Framework has also applied on billing which tooks ~200 lines of code with ~500 lines of code which can covered more than 50, 000 scenarios but limited to run ~1000 scenarios
   v. The framework gave them full confidence (100%) to change payment and billing modules any time
   vi. The framework also has detailed pass/fail results with better logs to debug.

2. Swift=> Finance (FIN) Project

a. The project is handling extremely complex money transactions between banks, companies, businesses, individuales, which is factored by amount, currency, time, sanctions, type of banks, dependencies, order settings, categories, ...
b. Big issue here was it is extremely difficult to generate test data that could cover all possible combinations
c. Big issue when requirement changes come on at least each year.
d. When we join the company test cases have been written in manual. They have hundreds of files per category and each file has hundreds of (even non enough) test cases, each test case has from 30, to 300 lines of test data. It was hell if that word can mention it.
e. We have followed the same Pattern Driven Test case generator.
f. Framework generates all test scenarios with in a second and put all of them on the appropriate file, each file, test case has been leveled and tagged
g. They had an issue to find what the objective of a test case, and we clearly show a mark line that the objective of the test case
h. Framework created a standard for each test. Science they had been writing them annually there was no standard to write the test case, so that

it was extremely difficult to identify the objective of test cases and at which line.

i. Saves 100% time, a tester might write/update max 20 test cases, with the help of the framework tester can generate thousands of test cases

Steps for implementation of Pattern Driven Testing framework with examples:

1)
2) Identify all possible actions as variables to complete a business process

```
// define all possible actions to be done to complete a business process
// properties for actionA
public static final String actionA1 ="actionA1";
public static final String actionA2 ="actionA2";
public static final String actionA3 ="actionA3";
public static final String actionA4 ="actionA4";

// properties for rebate and student reduced fee
public static final String actionB1 ="actionB1";
public static final String actionB2 ="actionB2";

// properties for exam date
public static final String actionC1 ="actionC1";
public static final String actionC2 ="actionC2";
public static final String actionC3 ="actionC3";

// properties for order date
public static final String actionD1 ="actionD1";
public static final String actionD2 ="actionD2";
public static final String actionD3 ="actionD3";
public static final String actionD4 ="actionD4";

// properties for chane enrollments
public static final String actionE1 ="actionE1";
public static final String actionE2 ="actionE2";
public static final String actionE3 ="actionE3";
public static final String actionE4 ="actionE4";

// properties for cancel enrollments
public static final String actionF1 ="actionF1";
public static final String actionF2 ="actionF2";
public static final String actionF3 ="actionF3";
public static final String actionF4 ="actionF4";
public static final String actionF5 ="actionF5";
public static final String actionF6 ="actionF6";
```

3) Prepare the Data Pull by grouping above actions into lists

```
// put all actions in a data pull
public static final List<String> actionA =Arrays.asList(actionA1, actionA2, actionA3, actionA4);
public static final List<String> actionB =Arrays.asList(actionB1, actionB2);
public static final List<String> actionC =Arrays.asList(actionC1, actionC2, actionC3);
public static final List<String> actionD =Arrays.asList(actionD1, actionD2, actionD3, actionD4);
public static final List<String> actionE =Arrays.asList(actionE1, actionE2, actionE3, actionE4);
public static final List<String> actionF=Arrays.asList(actionF1, actionF2, actionF3, actionF4,
actionF5, actionF6);
```

4) Generate Scenarios : Create a method to generate all possible scenarios from the Data Pull

- This method should take list of actions from the data pull as input.
- For example, if we need to take all actions from the data pull we can give inputs as shown below:

```
List<String> testCases = generateScenarios(actionA, actionB, actionC, actionD, actionE, actionF);
```

- The method should be designed to do cartesian product between actions.
- Generate all possible scenarios, and
- Return generated scenarios as a list.

A sample of generated scenarios is shown below:

```
****************************************** Number of test cases going to be run:
[--->actionA1--->actionB1--->actionC1--->actionD2--->actionE3--->actionF6,
--->actionA1--->actionB1--->actionC3--->actionD1--->actionE3--->actionF5,
.
.
.
--->actionA1--->actionB2--->actionC3--->actionD1--->actionE3--->actionF6,
.
.
.
]
```

5) Generate Scenario Sample Code

```
// this method is to generate all possible test scenarios for enrollment, order, change, and cancel actionD
public static  List<String> generateScenarios(List<String>... propertyPull) {
  List<String> testScenarios = new ArrayList<>();

  for(List<String> properties: propertyPull) {
     if(testScenarios.size()==0) {

        for(String attribute: properties) {
           testScenarios.add("--->" + attribute);
        }
     }else{

        List<String> updatedScenarios= new ArrayList<>();
        for(String testScenario: testScenarios){
           for(String attribute: properties)
              updatedScenarios.add(testScenario + "--->"+ attribute);
        }
        testScenarios=updatedScenarios;
     }
  }

  return testScenarios;
}
```

6) Include scenarios: create a method which parse through all above scenarios, filter out only the required scenarios that we would like to test

- **The method should take all scenarios as input t**
- **The method should take list of action keywords that we would like to include**
- **The method should iterate through all scenarios and filter out scenarios which contains the given keywords**
- **Are we going to include scenarios which contains all keywords or anyone from the given keywords?**
    - **To make such a flexible decision we should pass a boolean (true/false) flag.**
    - **Let us say if you pass "true" then your method should include scenarios if they contains all keywords.**
    - **If you pass "false" the method will include scenarios which contains any one from the keywords list.**

For example:

- **if you want to include scenarios which contains actionA2 or ActionA1 capstone or domestic non capstone courses do as follows**
    - **:testCases = *includeScenarios*(testCases, false, *actionA2*, *actionA1*);**

- **if you want to include scenarios which contains actionB2 and actionD2 use it as follows::**
    - **testCases = *includeScenarios*(testCases, true, *actionB2*, *actionD2*);**

7) Include Scenarios sample code

```
// this method is to make filter and add scenarios based on the given attribute and
logic(AND/OR)
public static List<String> includeScenarios(List<String> scenarios, boolean isAndLogic,
String... attributes) {
  List<String> filteredScenarios = new ArrayList<>();

  if(isAndLogic){
     filteredScenarios.addAll(scenarios);
     for(String scenario: scenarios) {

        for (String attribute : attributes) {
           if (!scenario.contains(attribute)){
              filteredScenarios.remove(scenario);
              break;
           }

        }
     }

  }else {
     for(String scenario: scenarios) {

        for (String attribute : attributes) {
           if (scenario.contains(attribute)){
              filteredScenarios.add(scenario);
              break;
           }

        }
     }
  }

  return filteredScenarios;
}
```

8) Exclude scenarios: create a method which parse through all scenarios, which are generated from the above step, remove scenarios which we consider as exhaustive. So finally we will get a modified list of scenarios, which we would like to test

- **The method should take all scenarios generated from the above step**
- **The method should take list of action keywords that we would like to exclude**
- **The method should parse through all scenarios and exclude all scenarios which contains those keywords**
- **Are we going to exclude scenarios which contains all or anyone from the given keywords?**
    - **To make such a flexible decision we should pass a boolean (true/false) flag.**
    - **Let us say if you pass "true" then your method should exclude scenarios if they contains all keywords.**
    - **If you pass "false" the method will exclude scenarios which contains any one from the keywords list.**

Sample exclude scenarios
testCases = *excludeScenarios*(testCases, false, *actionE1*, *actionF1*, *actionD3*, *actionF4*, *actionD4*);
testCases = *excludeScenarios*(testCases, true, *actionD1*, *actionF2*);

You can call this methods again and again until you get scenarios needed for your test

9) Exclude Scenarios sample code

```
// this method is to make filter and remove  scenarios based on the given attribute and logic(AND/OR)
public static List<String> excludeScenarios(List<String> scenarios, boolean isAndLogic, String...
attributes) {
  List<String> filteredScenarios = new ArrayList<>();

  if(isAndLogic){
    for(String scenario: scenarios) {

      for (String attribute : attributes) {
        if (!scenario.contains(attribute)){
          filteredScenarios.add(scenario);
          break;
        }

      }
    }

  }else {
    filteredScenarios.addAll(scenarios);
    for(String scenario: scenarios) {

      for (String attribute : attributes) {
        if (scenario.contains(attribute)){
          filteredScenarios.remove(scenario);
          break;
        }

      }
    }
  }

  return filteredScenarios;
}
```

10) Random scenario generator: if you still want to pick only few scenarios to test, you need to create a method which randomly picks limited number of test scenarios

- **For example, the below method will pick only 50 scenarios from the entire list**

Sample random scenario generator method looks like as follows
testCases = *randomScenarioGenerator*(testCases, 20);

- **There scenarios are now ready for test**

11) Pick a scenario and make a test process

- **Here one scenario is considered as one test case which presents one business process**
- **I recommend you to use TestNG as a data provider to iterate through each scenario as a test case.**
- **Pick one test case and take all real actions**
- **Different actions should be taken based on the keyworks that the scenario has**
- **For example, if the scenario contains “orderBeforeDeadline” then set the system deadline accordingly.**
- **You should have multiple methods to make a different operations and update flags according to the action taken**
    - **The method may take the scenario as string and flags as array list.**
    - **Method should take actions based on keywords it has**
    - **After taking action method should update necessary flags**
    - **Method should return the updated flags**

12) Define flags as a list

- List out all the flags which you will use them to store some info during test process
- Flags status should be accordingly updated while taking any system actions
- Flags will be used during final test assertion time for expected data assertion

```
// define default flags that we need to update/use during enrollment and assertion process
public static Map<String, Object> setFlags(String orgId, String studentId, String studentName ) {
  Map<String, Object> flags = new HashMap<>();
  flags.put("flag1", "defaultValue");
  flags.put("flag2", "");
  flags.put("flag3", "");
  flags.put("flag4", false);
  flags.put("flag5", true);
  flags.put("flag6", 0);
  flags.put("flag7", 0);
  flags.put("flag8", "");
  flags.put("flag9", 0);
  flags.put("flag10", -1); // -1 cancellation not yest happen
  flags.put("flag11", 0);
  flags.put("flag2", “defaultValue”);
  flags.put("flag13", “defaultValue”);
  flags.put("flag14", 0);
  flags.put("flag15", false);

  return flags;
}
```

13) Here is sample method usage

```
// set all flags variables and set order deadline
flags = setActionA(scenario, sections, flags);
flags = setActionB(scenario, flags);
flags = setActionC(scenario, flags);
flags = setActionD(scenario, flags);
System.out.println("============>Before New Order " + scenario + " \n flags\n" + flags);
```

14) Here is sample which shows how an action to set order deadlines:

```
public Map<String, Object> setActionA(String scenario, Map<String, String[]> sections, Map<String,
Object> flags) {
  if (scenario.contains(actionA1)) {
     flags.put("flag1", "value");
     flags.put("flag5", "value");
     flags = setSection(scenario, sections, flags);

  } else if (scenario.contains(actionA2)) {
     flags.put("flag1", "another value");
     flags.put("flag2", "another value");
     flags = setSection(scenario, sections, flags);

  } else if (scenario.contains(actionA3)) {
     flags.put("flag1", "other value");
     flags.put("flag5", "other value");
     flags = setSection(scenario, sections, flags);

  } else if (scenario.contains(actionA4)) {
     flags.put("flag6", "value");
     flags.put("flag2", "value");
     flags = setSection(scenario, sections, flags);
  }
  return flags;
}
```

15) Create assertion method: this method should be designed to prepare expected result, get actual result and validate.

- **You can make test assertion at any time**
- **This method should take flags as a list**
- **Generate expected result based on the flags status**
- **Get the actual result from system**
- **Validate the actual to expected on and show pass or fail**
- **You can see the example given below**
- **Sample assertion method to call:**
  - **assertOrderFee(scenario, flags);**

16) Sample assertion method .

```
public void assertMethod(String scenario, Map<String, Object> flags) {
   /*

   keep preparing expected  value
   keep getting actual value from end point
   validate expected and actual

    */

   String flag1 = flags.get("flag1").toString();
   int expectedValue1 = (Integer) flags.get("flag3") - (Integer) flags.get("flag4") - (Integer)
flags.get("flag9") + (Integer) flags.get("flag7") ;
   int actualValue1 = getActualValueFromEndPoint(flag1);
   validateValues(expectedValue1, actualValue1);

}
```

17) Final test report looks like as follows .

- **TestClass.testCase[--->actionA1--->actionB1--->actionC2--->actionD2--->actionE2--->actionF5]**passed
- **TestClass.testCase[--->actionA1--->actionB1--->actionC2--->actionD2--->actionE4--->actionF6]**passed
- ….