SPECIFICATION BASED TESTING TECHNIQUES

PBA SOFTWAREUDVIKLING/
BSC SOFTWARE DEVELOPMENT

Christian Nielsen cnls@cphbusiness.dk
Tine Marbjerg tm@cphbusiness.dk

SPRING 2019

TODAY'S TOPICS

- **Overview**
  - Learning objectives
  - Decision tables
  - State transition models
  - Data-driven testing
  - Repeated tests
  - Parameterized tests
  - Dynamic tests
  - Hamcrest
  - Examples
  - Exercises
  - Assignment

LEARNING OBJECTIVES?

- Describe the benefits and construction of decision tables and state transition models in the context of test case design
- Create and use decision tables and state transition models in tests
- Explain the concept and need for data-driven testing and getting test data from files
- Perform repeated, parameterized and dynamic tests with uniform input values from different sources, such as arrays, lists and files
- Explain the advantages and the disadvantages of different assertion libraries and using alternative matchers

EXERCISES

- DECISION TABLE:
  - LEAP YEARS
  - CREDIT CARD - DISCOUNT
- STATE TRANSITION MODEL:
  - ACCOUNT - DEPOSIT WITHDRAW INTEREST
- REPEATED TEST:
  - ACCOUNT - INTEREST
  - CREDIT CARD - DISCOUNT
- PARAMETERIZED TEST:
  - ACCOUNT - INTEREST
    - VALUESOURCE
    - METHODSOURCE
  - CREDIT CARD - DISCOUNT
    - VALUESOURCE
    - CSVSOURCE
    - CSVFILESOURCE
    - METHODSOURCE
- DYNAMIC TEST:
  - CREDIT CARD - DISCOUNT
- HAMCREST:
  - REWRITE TESTS

DECISION TABLES

A decision table is made up of conditions, condition alternatives, actions and action entries.

A decision table captures all combinations of variables and possible outcomes.

Overview of business rules

Systematic way to analyze combinations of inputs and the outcomes

A decision table consists of conditions, condition alternatives, actions and action entries

Start with all combinations of inputs, determine actions and finally optimize decision table

DECISION TABLES

EXAMPLE: ATM – Cash withdrawal

Customer requests a cash withdrawal.

ATM grants withdrawal if the customer has sufficient funds or if the customer has credit allowed.

1. Fill boolean combinations

Conditions				
Sufficient funds	T	F	T	F
Credit	T	T	F	F

2. Check for invalid combinations

Conditions				
Sufficient funds	T	F	T	F
Credit	-	T	-	F

3. Remove identical columns

Conditions			
Sufficient funds	T	F	F
Credit allowed	-	T	F

4. Determine actions

Conditions			
Sufficient funds	T	F	F
Credit allowed	-	T	F
Actions			
Withdrawal granted	X	X	
Withdrawal rejected			X

STATE TRANSITION MODELS

Used when some aspect of a system can be described in a finite state machine

For systems where output changes with same input, depending on what has happened before

A state transition model has four basic parts

1. The states that the software may occupy
2. The transitions between states
3. The events that cause a transition
4. The actions that result from a transition

Test cases are designed to execute valid and invalid state transitions

Tests can be derived from state transition diagram to test sequences of states and transitions

As a minimum, every state and transition should be tested and sequence length tested determines switch coverage

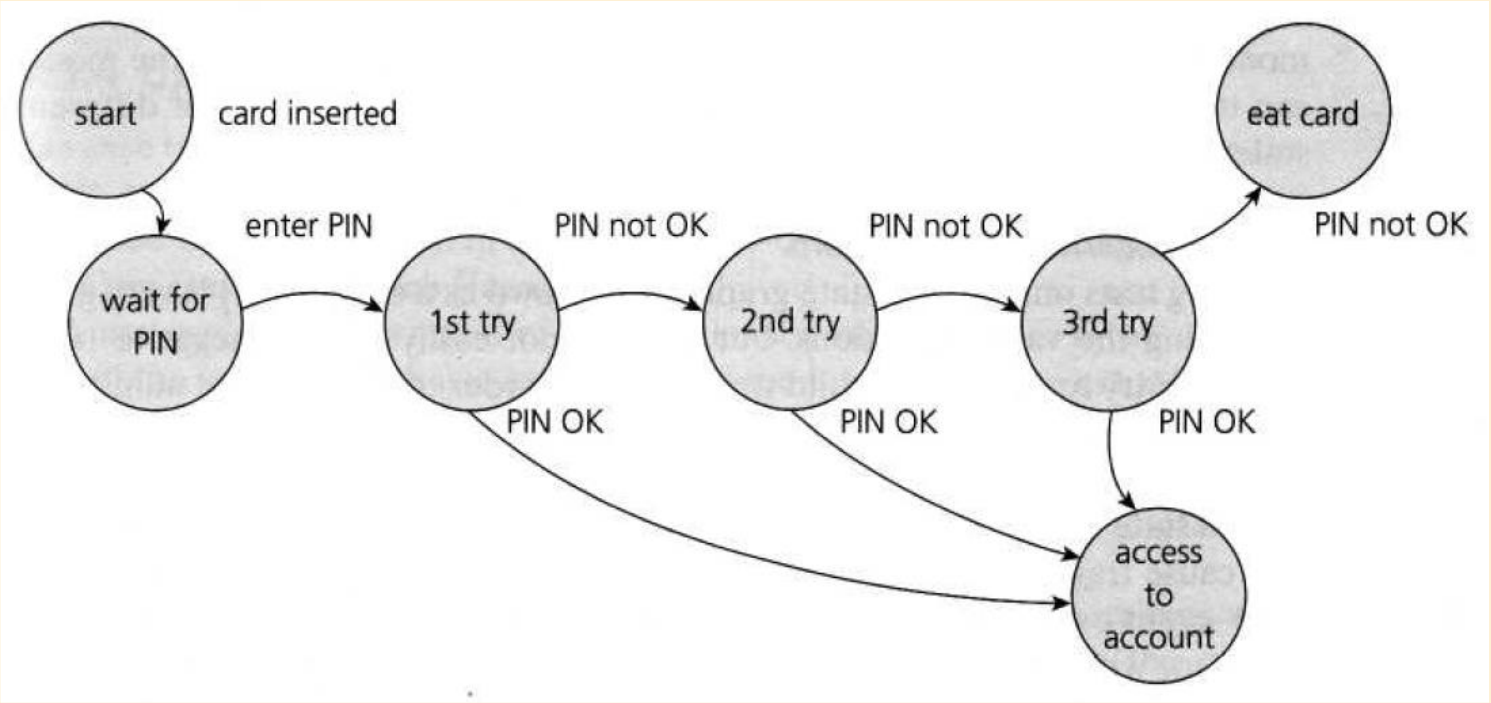
0 switch coverage refers to testing the individual transitions

1 switch coverage means that pairs of transitions are tested

STATE TRANSITION MODELS

EXAMPLE: ATM – Credit card insertion

Customer inserts credit card and enters pin code to access account.



DATA-DRIVEN TESTING

Software testing methodology

Data-driven testing is the creation of test scripts to run together with their related data sets in a framework

The data comprises variables used for both input values and output verification values

Testing done using a table with test inputs and verifiable outputs

Supplied inputs from a row in the table expects the output which occur in the same row

DATA-DRIVEN TESTING

Repeated test

Repeat a test a specified number of times with annotation @RepeatedTest and value for the total number of repetitions for the test

Parameterized test

Parameterized tests make it possible to run a test multiple times with different arguments with @ParameterizedTest and annotation for parameters source, such as @ValueSource, @CsvSource, @CsvFileSource, @MethodSource

Dynamic test

Test generated during runtime by a factory method annotated with the @*TestFactory* annotation

DATA-DRIVEN TESTING

Repeated test

```
@RepeatedTest(5)
void repeatedTestWithRepetitionInfo(RepetitionInfo repetitionInfo)
{
        assertEquals(5,repetitionInfo.getTotalRepetitions());
}
```

DATA-DRIVEN TESTING

Parameterized test

```
@ParameterizedTest
@ValueSource(strings={"racecar", "radar", "able was I ere I saw elbar"})
void testStrings(String string)
{
        assertTrue(string.endsWith("r"));
}
```

DATA-DRIVEN TESTING

Dynamic test

```
@TestFactory
Collection<DynamicTest> dynamicTestsFromCollection()
{
        return Arrays.asList(
                dynamicTest("1st dynamic test", () -> assertTrue(2==2)),
                dynamicTest("2nd dynamic test", () -> assertEquals("ABC", "ABC"))
        );
}
```

HAMCREST

Assertion library

Hamcrest is a framework for software tests

Hamcrest comes bundled with lots of useful matchers and it is possible to create custom matchers

Assertion library with matchers to be used in tests instead of asserting with JUnit asserts

Hamcrest is a framework that assists writing software tests and writing matcher objects allowing ‘match’ rules to be defined declaratively instead of imperatively

Using Hamcrest should improve tests and provide higher readability and better error messages over JUnit asserts

HAMCREST

AssertThat & Matchers

assertThat(actual, matcher)

- The subject of the assertion is the first method parameter

- The second method parameter is a matcher

Category	Matchers
Core matchers	is - decorator to improve readability
Logical matchers	allOf - matches if all matchers match, short circuits (like Java &&) anyOf - matches if any matchers match, short circuits (like Java ||) not - matches if the wrapped matcher doesn't match and vice versa
Object matchers	equalTo - test object equality using Object.equals notNullValue, nullValue - test for null
Collection matchers	hasEntry, hasKey, hasValue - test a map contains an entry, key or value hasItem, hasItems - test a collection contains elements
Number matchers	closeTo - test floating point values are close to a given value greaterThan, greaterThanOrEqualTo, lessThan, lessThanOrEqualTo - test ordering
Text matchers	equalToIgnoringCase - test string equality ignoring case equalToIgnoringWhiteSpace - test string equality ignoring differences in runs of whitespace containsString, endsWith, startsWith - test string matching

ASSIGNMENT

IMPLEMENT AND TEST BANK SYSTEM

1. IMPLEMENT ACCOUNT / CUSTOMER / CREDIT CARD / ATM CLASSES
2. TEST ACCOUNT, CREDIT CARD AND ATM CLASSES SUFFICIENTLY
3. INCORPORATE BOTH A REPEATED TEST, A PARAMETERIZED TEST AND A DYNAMIC TEST SOMEWHERE IN THE TESTS
4. USE HAMCREST MATCHERS THROUGHOUT THE TESTS INSTEAD OF JUNIT ASSERTS
5. DOCUMENT HOW EQUIVALENCE PARTITIONS, BOUNDARY VALUES, DECISION TABLES AND STATE TRANSITION MODELS HAVE BEEN USED AND APPLIED IN THE CREATION OF THE TESTS

RESOURCES…

JUnit

https://junit.org/junit5/docs/5.3.0/user-guide/index.pdf

Decision tables

https://www.guru99.com/decision-table-testing.html

State transition models

https://www.guru99.com/state-transition-testing.html

Data driven testing

https://en.wikipedia.org/wiki/Data-driven_testing

http://xunitpatterns.com/Data-Driven%20Test.html

Repeated tests

https://www.baeldung.com/junit-5-repeated-test

Parameterized tests

https://www.baeldung.com/parameterized-tests-junit-5

https://blog.codefx.org/libraries/junit-5-parameterized-tests/

Dynamic tests

https://www.baeldung.com/junit5-dynamic-tests

https://blog.codefx.org/libraries/junit-5-dynamic-tests/

Hamcrest

http://hamcrest.org/

https://code.google.com/archive/p/hamcrest/wikis/Tutorial.wiki

https://www.javacodegeeks.com/2015/11/hamcrest-matchers-tutorial.html

https://www.vogella.com/tutorials/Hamcrest/article.html#overview-of-hamcrest-mather