**Goals:**

- Develop an understanding of functionality suitable for modeling  
- Practice modeling using EFSMs  
- Apply model-based testing techniques  
- Understand the connection between model, adapter, and a system under test

**Introduction:**

This is an **individual assignment**. In this assignment, you will create a model of a system under test (SUT) and later use it for model-based testing. The SUT should include an Application programming interface (API) of your choice. Tip: Using sophisticated APIs like, for example, Google APIs for web development provides a challenging task.

A good SUT:

* has existing documentation,
* runs on your machine, and
* is complex enough to fulfil the requirements in Part 1, Task 1. (See below for details)

**Overview of your tasks:**

**Part 1, Modelling:**

**Task 1: Design a model**

* First, we ask you to create a model of your SUT or a part of its functionality. Your model should be an *extended finite state machine* allowing you to generate interesting, arbitrary long test sequences against your software under test.
* Your EFSM graph must have about 5 inputs and be able to generate infinite test sequences.
* Your model doesn’t need to cover the whole part of the SUT. Start by modeling a basic part and add more details progressively until you reach the desired number of inputs.
* Even if you end up with only a very basic model, it must contain some kind of loop. For example, this can be a connection for the last node in the model to the first node in the model.
* The model can be a combination of small models that share some state(s).
* We recommend using the tool "yEd" for graphical modeling. (https://www.yworks.com/downloads#yEd)

**Task 2: Realize your model in java code**

* Use the knowledge from the lectures and the tutorial to create the above model as executable Java code.
* We recommend using the tool "Graphwalker" for your implementation (http://graphwalker.github.io/Model\_design/)

**Create a first version of the final report (the report will be completed with the second part of the assignment), including the information:**

1. A description of the behavior of the modeled part of your SUT
2. The list of system inputs in your model with their description
3. The transition table
4. A graphical representation of the model
5. A test sequence which achieves full edge-coverage
6. A java file containing the code representation of the model

**Additional Resources:**

* http://graphwalker.github.io/
* http://www.cse.chalmers.se/edu/year/2017/course/DAT261/\_static/modeljunit2\_docs/nz/ac/waikato/modeljunit/package-summary.html

**Part 2: Model-based Testing:**

After finishing the model of your SUT, you are now asked to connect the model to it using a so-called adapter and run tests.

* Start with a reduced version of your model and connect the edges/transitions with the SUT by running actual SUT code. To have a clean separation, we recommend collecting all of this "glue-code" in a class called Adapter. When running the model using Maven and Graphwalker, the code of the SUT has to be invoked.
* Think of five tests to be added to your model. This can be general tests for functionality or robustness or single assertions similar to unit tests. Implement as much of your model as needed to construct meaningful tests.
* Investigate Model-Coverage Criteria. On your final model used for MBT, compare how runtime, state coverage, and transition coverage changes when using the random and the weighted\_random generators with the stop criteria edge\_coverage and vertex\_coverage.

Graphwalker does not provide a transition-pair coverage criterion. How can you implement it yourself, using your knowledge of graph theory? (This question is not mandatory. But it is good if you can think and provide some ideas about it.)

**Collect the following deliverables for the final report:**

1. The final model as a Java-file containing the tests
2. The adapter controlling the SUT
3. Your findings from the comparison of stop-criteria and generators
4. A description of five tests you are conducting with your model (What do they test? How to fail them?

**Submit the following artifacts and make them in a zip :**

* **A java-file** containing your model
* **A java-file** containing your adapter
* **A report** in a PDF format containing the following:
  + Part 1:
    - A description of the behavior of the modeled part of your SUT (provide a link to the origin of the SUT)
    - The list of system inputs in your model with their description
    - The transition table
    - A graphical representation of the model
  + Part 2:
    - Your findings from the comparison of stop-criteria and generators
    - A description of five tests you are conducting with your model (What do they test? How to fail them?)
    - An explanation of what to change to achieve transition-pair coverage, including possible figures and code samples. (Not mandatory)
  + Part 3:
    - Lessons learned from testing this particular SUT

\*any language can be used

\* 5 inputs - not very strict ,may be 4 or 3, [google calendar- day 0] [function call to library, api / action]

\*infinite test sequence

try to concentrate only on Part1- task 1 [20/04-24/4]

http://q2a.inf.mit.bme.hu/4049/lab4-generate-test-source-from-json

USE JAVA 1.8/ report - any file

https://groups.google.com/forum/#!topic/graphwalker/69T5h8vsHow