IN2013, Week 10 – Software Testing: Integration and System Testing

Dr Peter Popov

29th November 2018

This tutorial is about integration and system software testing.

Part 1: Operational profile in System Testing

Operational profile is an important concept in statistical software testing. Recall that the operational profile defines a ***probability distribution*** over the set of possible use cases. The probability P(A) associated with use case A will define the chances that a randomly chosen use case will be of type A.

Consider the case study shown in Appendix 1 (a simplified version of BAPERS used in the previous Tutorials).

Question 1:

Consider the following numbers that characterize the operational profile for the *simplified version* of BAPERS.

1. The Receptionist takes on average 90 new jobs a day. During the day the receptionist would log in on average 5 times (and would log out 5 times, of course).
2. All jobs taken in are processed and completed by a Technician by the end of the day. Processing a job would require from a Technician to do the following:
    a. Login
    b. Record the time spent on a job (i.e. ProcessJob use case) and
    c. Logout.
3. By the end of the day all jobs are collected by their respective customers (photographers who brought them in). The Receptionist would record the payment made by the respective customers. Each job is paid separately. No additional logins/logouts are needed for collecting payments. We assume that the Reception would stay logged-in most of the time, the 5 logins and logouts listed in point 1 above account for all logins and logouts by Receptionist during the day.

Build a sketch of the rationale that one might use in defining the ***operational profile*** for this simplified version of BAPPERS. Compute the probabilities of each use-case run by any of the two actors, the Receptionist and the Technician. These probabilities together define the operational profile for the simplified version of BAPPERS. Make sure that the sum of these probabilities is equal to 1.

Hint: Start your work by counting how many times on average each of the use cases shown in Appendix 1 will be 'run' by each of the actors – Receptionist and Technician during the day. Then you establish the total number of use cases executed during the day. You can establish the frequency with which each use case will be run by an actor by dividing the frequency of the use case by the total number of use cases run during the day.

Question 2: Compute the profile that can be associated with each of the actors (Receptionist or Technician), e.g. count only the use cases that each of them can undertake during the day, the total number of use cases that each of them runs and compute the (conditional) probabilities of the respective use cases given that a randomly chosen use cases is run by the particular actor.

Question 3 (complete at home especially if you have not installed jMeter on your laptop before ahead of tutorial): Conduct performance evaluation of example.org web-site using the tool jMeter. You can vary the number of "users" in the range [1 … 100]. Analyze the response times and the rate of error (i.e. the rate of responses which are returned with an error code that differ from 200, e.g. error code 502, 404, etc.). You may repeat the study with some other sites, e.g. Google.com or city.ac.uk.

Part 2: Integration testing with jUnit

In this part of the tutorial we will use jUnit to conduct integration testing of a small subsystem derived from BAPPERS, which is shown in Appendix 2. It consists of an interface I_JobTask, implemented by the class JT_Implementation, which in turn is associated with some of the BAPPER classes that we have worked with in the previous tutorials. The diagram can be found under the "Implementation Model" of the vpp file released on Moodle.

Question 4:

Generate Java code from Visual Paradigm design class diagram (Implementation Model -> JobTask_CD). Create a Netbeans project for these files. Using the Netbeans support for jUnit, create a test class to test the methods listed in the I_JobTask interface. Write a setup Java code (@BeforeAll) in which you need to instantiate the implementation class and create several instances of TaksDescription: these instances must exist before a new Job can be created.

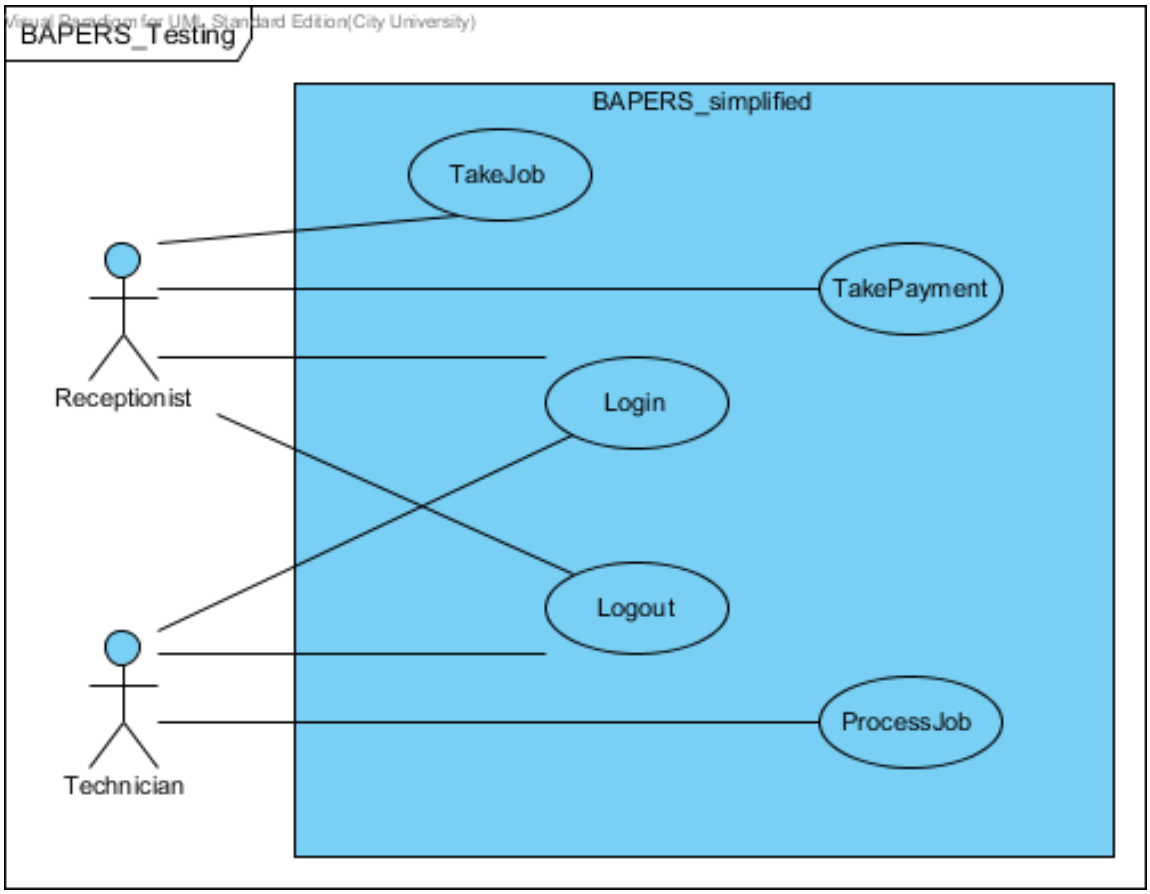
Question 5:

Write tests for getJobPrice(jobID:int). This test should traverse an existing :Job object and take the sums of the prices of all Tasks, which belong to the particular :Job. Consider testing the following important cases:

- jobID refers to an existing :Job object

- jobID refers to a non-existent :Job. Discuss with a colleague how you can detect this situation and the options to report to the jUnit the situation.

APPENDIX 1



APPENDIX 2