CITS5501 Software Testing and Quality Assurance Semester 1, 2020

Week 9 workshop – Exercise solutions

Assessed exercise

You are part of the software testing team for TimeOff, a business which books adventure and recreation tours. You have been tasked with the testing of their new online booking system, which is currently being developed.

For each item below, provide:

- A description of what sort of test you would write to test it, and what techniques from lectures you would apply.
- Based on any reasonable assumptions about the system requirements, one example test case.

Justify your answer.

Items:

a. Test that the `addNewTourMember` method for the `Tour` class works properly in combination with the system database.
b. Test that the system adequately protects customers' credit card details.
c. Test that the system remains responsive under high load.

This workshop exercise is worth 5% of your final grade. All work is to be done individually.

Sample solutions:

One possible answer:

a. We are asked to test that the `addNewTourMember` method for the `Tour` class works properly "in combination with" the system database. Therefore, this is an integration test – we are checking that one component (the `Tour` class) works correctly with another (the database). Relevant techniques therefore are the various integration testing approaches discussed in the lectures such as bottom-up and top-down integration.

We aren't told exactly what the `addNewTourMember` method does, but it seems reasonable to assume that if we call it with some sort of `TourMember` object, then that object will get added to the database.

We would not use a mock object for the *database*, since we want to test that it actually works. We *would* use fixtures (that is: state that needs to be set up for the test) – namely, some sort of test database containing bogus data, and a connection to that database.

A sample test case might be something like the one given below. We assume a `TourMember` class exists and can be straightforwardly initialized, and that the `Tour` class has access to a database connection (e.g. as an instance variable). We assume `addNewTourMember` takes no other parameters beyond a `TourMember` object.

Test `addNewTourMember` method	
Description	Calls `addNewTourMember` with a member not already in the database, and confirms the member is added and that no other change is made to the database.
Setup[1]	1. Create database instance with known contents. Call the set of pre-existing members `pm`. 2. Initialize `Tour` object `t`, including any instance variables[2] and database connection. 3. Initialize `TourMember` object `m`, including any instance variables.[2]
Test	Call `t.addNewTourMember` method with argument `m`.
Obtain results[3]	Query the database for all members – call the query result `qr`.
Expected results	$\mathtt{qr} = \mathtt{pm} \cup \{\mathtt{m}\}$
Cleanup	*(Elided for brevity)*

[1] i.e., prefix values.

[2] Exactly which instance variables are mocks and which are not will depend on what other classes have already been integrated.

[3] i.e., verification postfix values.

b. We would need to perform *security testing*. The main security testing technique covered in lectures is *penetration testing*:[*] the tester adopts the role of an attacker, and attempts to obtain the secured data.

Recall that "testing" usually refers to techniques involving dynamic execution of the software – i.e., software is actually executed, and its behaviour observed.

Penetration testing normally consists of a whole series of tests, but one *very* basic test would be to check whether the website (and all content – images, stylesheets etc) are being served over HTTPS (secure HTTP) – if they are not, it might be possible to intercept web traffic and obtain confidential data.

A test case might look like this:

	Test web assets being served over HTTPS
Description	Checks to see whether the website (including any assets served such as images and stylesheets) are being served over HTTPS.
Setup	1. Ensure the live system[1] is available via the web. 2. Start browser and open Developer Tools 3. Open 'Network' tab.
Test	Visit entry page for site.
Obtain results	Record the network requests made when the site is loaded.
Expected results	All network requests should be made over HTTPS.

[1] Or possibly a test system, if doing penetration before the system goes live.

As far as other software quality assurance techniques go: we might also make use lightweight formal methods – for instance, use of the type system as a way of limiting to where in the system confidential data is passed. And we would want to ensure our process included careful *review* of design and code, to ensure security best practices were implemented; and we should probably make use of *static analysis* techniques to check for defects in our code.

[*] Other security testing techniques include vulnerability scanning, to test for known security vulnerabilities, and network sniffing, to determine whether confidential data may be inadvertently being leaked. These can be run automatically as part of a test suite; though note that they can also be run manually (and penetration testers are likely to do so).

c. We would need to perform *stress* or *load testing*. Typically this is done by generating high levels of random, but realistic, input. For instance, for this system, we might generate random web access to the system, which are as similar as possible to ones that might be performed by real users.

Specialised software exists to generate such traffic: see for some examples https://www.dnsstuff.com/network-traffic-generator-software and https://www.icir.org/models/trafficgenerators.html. Additionally, techniques from the lectures on syntax-based testing can be used to generate custom traffic.

For the following test case, we assume that when the load reaches a level determined to be "too high" and a customer attempts to perform some operation (e.g. joining a tour group), the server should degrade gracefully by showing an error message. A complete test case would include a definition of what exactly "overloaded" means (e.g. more than 1000 web requests per second, say). We also assume traffic generation software is available to put the system under high load.

Verify error message is shown when the system is overloaded	
Description	Verifies that when the system is overloaded, and a customer attempts to join a tour group, a proper error message is shown saying that the system is under heavy load and to try again later.
Setup	1. Ensure the live system[1] is available via the web. 2. Log in and visit the member page. 3. Generate random traffic which puts the system over its load limit.
Test	Attempt to join a new tour group.
Obtain results	Record the page displayed.
Expected results	An error page should be displayed meeting the criteria specified.

[1] Or possibly a test system, if doing load testing before the system goes live.