Southern Cross UNIVERSITY

ISYS3001

# Managing Software Development

**Workshop 2**

**Blackbox Testing**

# Contents

# Blackbox Testing

## Objectives

On completion of this topic you should be able to:

1. Design tests for acceptance or system testing
2. Design detailed black-box tests

## Overview

Black-box testing, as you have learned in the study guide, is testing software without access to the code. In this workshop, we will look at different test aims and strategies for both low level testing at the individual field level, and higher level testing.

There are several presentation methods used in this workshop to organise testing and test data. These are not a standard way to present test information but you are welcome to use these methods in your assignments. The aim here is to give enough information to understand the various processes and strategies. I real world testing there will be additional information, such as who did the testing, where faults should be sent and other administrative information.

This workshop only looks at acceptance tests and detailed black-box testing. The study guide also discussed many other types of testing that we will not cover here, e.g. usability testing, performance testing, etc.

## Acceptance tests (also system tests)

Acceptance testing and systems testing have similar goals. Both are designed to provide complete systems testing and usually involve having real users perform the tests. The difference is that system tests are usually designed by the software developers, whereas acceptance tests are designed by organisation that are paying for the software. Because of this, system testing usually expects to find errors that the developers can fix. Acceptance testing may find errors, but since contractual issues are at stake, found errors may need to be negotiated. The difference in the tests themselves is not significant so we will take the part of acceptance testers in this workshop and any assessment in this unit.

Acceptance tests are high level tests, usually unconcerned with the way the system was developed. They are concerned with the software requirements, but they must take into account how the system will be used. The main goal of acceptance testing is to see whether the system meets the specification that was agreed between the developers and the users. You have learned that this is not always easy as both groups have difficulties understanding teach other's languages. **In this workshop you will need to play the role of the user, and not the developer!** However, your understanding of IT will mean that you can critically analyse the testing plans with an understanding of what could go wrong. To aid you as a user of the software we will use software examples whose requirements you will be familiar with.

### Example 1: MySCU forums

A facility you are familiar with is the MySCU forums. As a user you may have specified them as follows.

*MySCU forums are an asynchronous communication method for students and staff in a unit that allow:*

1. *Messages to be posted so students can ask questions, staff can make announcements and both students and staff can respond, respond to responses, etc.*
2. *Messages should be organised into subjects with responses easily accessed*
3. *There can be multiple forums to organise messages for different purposes*
4. *Message should be editable and allow font changes, font style, etc. like a word processor. A person can delete their own message and the unit assessor can delete any post.*

5. *All messages should show the poster, date and time of post.*

Now, you may think of additional requirements or more detailed requirements. For example, it would be nice to be notified by e-mail when something is posted. We will leave it at the above for this exercise. You should also note that software supplied may address more requirements than you have requested. This is the usual case with packaged software but can also be the case with software developed for you.

So how do we go about designing an acceptance test? The study guide has told us that a good approach is to look at user scenarios. So we can design our acceptance test around a series of user scenarios. To be thorough, we should attempt to design our tests to include *all* user scenarios. However, at this point we have to be pragmatic. To test all user scenarios in detail is not practical. For example, we cannot test every possible message that can be entered in the system, but we can test each high level use case to some degree. This means that as an acceptance tester we want to perform enough tests to test as many scenarios as possible *within the resources available*.

Another aspect to notice is that our specification above is divided into a list of requirements rather than a list of scenarios. This is not uncommon. Actual user scenarios may involve more than one specification detail. This means we usually need to design tests separately from the specification but we must ensure all of the specification is covered with the tests.

Let's start with a list of use case scenarios that address the first specification point above:

| **Scenario** | **Expected outcome** |
|---|---|
| S1.1: a user starts a new subject with a posting | A new "conversation" appears in a list of conversations.<br>The message is clearly readable by all people who can access the system with poster's name, data and time displayed.<br>A "subject" is clearly visible<br>Both students and staff can post messages |
| S1.2: another user responds to the posting | Same visibility as original post<br>The original post should be visible<br>A different computer can be used<br>Both students and staff can post messages |
| S1.3: a third user responds to the first posting. | The original post should be visible<br>It should be clear which post the response is for (see S1) |
| S1.4: a user responds to the first response | It is easy to follow the chain of posts, with the chain of posts preferably visible.<br>Both students and staff can post messages. |
| S1.5: add five more responses | Check that all posts remain accessible (though probably not visible at the same time) |

This is definitely not complete because we have only listed scenarios that apply to the first specification point. Also, notice how we have not gone into detail about what the messages and responses contain and we have not mention the organisation of subjects and other attributes that are described in subsequent specification points.

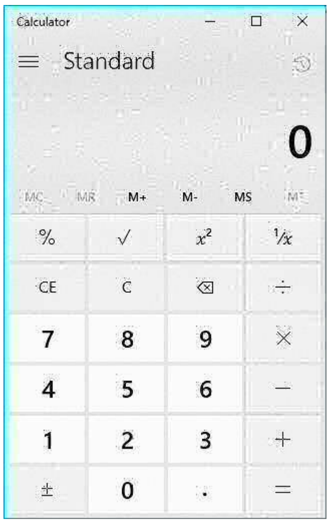
As mentioned above, the actual MySCU software has quite a few more options than the specification has stated. This is not uncommon for acceptance testing, since we really only need to test what we have specified because that is the agreement with the software suppliers and is our stated requirement.

### *Activity 2-1: Example1: Acceptance tests*

1. Add scenarios for acceptance testing to address the second specification point (2).
2. Add scenarios for acceptance testing to address the second specification point (3).
3. (Optional) complete the list of scenarios for the entire specification. This is probably more that you require to understand the process but will give you an indication of how extensive this may become for a complex system.

## Example 2: a calculator app

A simple app is the Microsoft calculator that can be used for basic calculations on a computer's screen. The Windows 10 version is pictured below.



An organisation's requirements for an onscreen calculator will be described below. Note that by stating requirements in this case, we may find that the Microsoft calculator is not acceptable so we will need to look elsewhere. Remember that we are to perform acceptance tests based on these requirements.

*The organisational requirements are:*

*1. Perform basic operation on integers and floating point numbers (plus, minus, times, divide), including more than two values*

*2. Optionally display as percentages*

*3. Allow calculations with brackets, e.g. 7 * (34 + 99)*

*4. Perform 1/x calculations and allow number signs to be changed*

*5. Allow money to be numbers representing money to be calculated, up to 100 Billion dollars.*

You should not that this specification is from a user's point of view rather than a mathematicians or programmers point of view. This means the description may not match directly with the product. For example, the calculations using parentheses can be achieved using the memory function on the Microsoft calculator.

We can start describing the acceptance tests for the first specification point.

| **Scenario** | **Expected outcome** |
|---|---|
| S1.1: +, -, x integer numbers and check accuracy of answer. | Answers are accurate.<br>Integers are produced as answer |
| S1.2: division produces accurate answer | If result is integer, then integer is displayed<br>If result contains fraction, then at least 6 decimal places displayed. |
| S1.3 floating point +, -, x and ÷ produce accurate answers to sufficient decimal points | All operation produce more than 6 decimal point accuracy |
| S1.4: multiple calculations can be performed sequentially (including previous result) | We can add a list of ten numbers<br>We can randomly insert other arithmetic operation (+, -, x and ÷) |
| S1.5: We can start new calculation without restarting calculator | Someway of clearing the current result and starting a new calculation |

One of the interesting attributes of testing the calculator is that we need to compare answers with an alternate calculation method. This may be by calculating answers by hand, or it may be by comparing answers with a trusted alternate calculator. This is particularly important with the detailed black-box testing that we look at below.

You may notice that working with numbers is quite different that our MySCU example which was mainly text processing. The above specification is quite general. A mathematician or someone expert in numerical analysis may have been much more demanding on the accuracy of the calculator. There are issues with computer based calculations that we have not mentioned above.

### *Activity 2-2: Example1: Acceptance tests*

1. Add scenarios for acceptance testing to address the specification points 2 and 3. If you apply the test you will see the Microsoft calculator has some operational issues with some with these tests.

2. (Optional) complete the list of scenarios for the entire specification. This is probably more that you require to understand the process but will give you an indication of how extensive this may become for a complex system.

Next, we look at detailed black box testing.

## Detailed Black-box tests

Detailed black-box testing is applied by software developers with a greater emphasis on finding bugs than the acceptance testing above. Black-box testing require different strategies to the acceptance tests in the previous section, which were designed to see if the product matches the specification. Detailed black-box testing has an emphasis on finding program errors and programmer's mistakes in programming. However, the two groups are not entirely separate, even though they as usually performed by different people. Nobody want to have bugs in their software, so acceptance testing will have some recognition of bugs that are found. Similarly, detailed black-box testing will recognise and report specification mismatches/errors.

The study guide recognised several strategies for detailed testing, and we mentioned some of these in the unit tests workshop. The textbook (section 8.1.2) mentioned strategies for lists and strings:

1. Test middle and boundary data (programmers make mistakes at boundaries)
2. Use different sequences and different sizes in data (programmer may have only tested trivial cases).
3. Use single value sequences (programmers may have tested with multiple value sequences)

The textbook also suggests some general strategies for detailed testing that may be applicable to black-box testing:

1. Test error messages (force error messages you would expect or perhaps referred to in documentation)
2. Try to force buffer overflow.
3. Repeat same series of inputs.
4. Try to force invalid outputs
5. Force computation that produces too large or too small values (causes overflow or underflow)

These are not the only strategies. Unique applications will have unique errors so experience and careful thought will find other tests that are applicable.

In the examples that follow we will take the general approach to apply the above strategies (and others) as follows:

1. Test individual fields, buttons and other widgets.

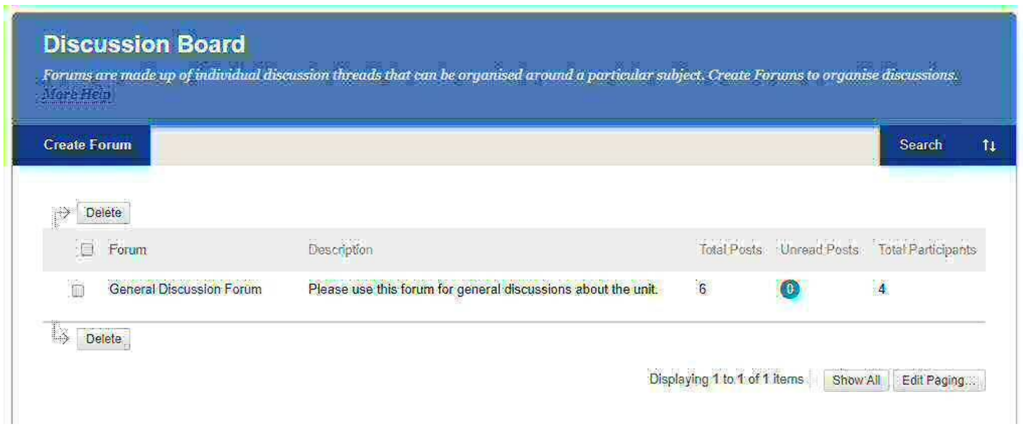
2. Think about how the individual fields interact with each other. This requires an understanding of the data model of the application.

Remember that the following examples are a testing strategy. You may think of better tests and different ways to test the same thing. No two strategies will be the same!

## Example 1: MySCU forums

We will now look at test strategies for the same software product as we looked at for acceptance testing. Remember, we are now looking at the product from a developer point of view, that is we are trying to find bugs as well as any other errors. Note also that we do not expect to find bugs because this is a commercial product.

Now the MySCU forums have several screens that require testing. The forums start with a list of forums. For example, the forum for this unit initially looks similar to the following:



Discussion Board

*Forums are made up of individual discussion threads that can be organised around a particular subject. Create Forums to organise discussions. More Help*

| Forum | Description | Total Posts | Unread Posts | Total Participants |
|---|---|---|---|---|
| General Discussion Forum | Please use this forum for general discussions about the unit. | 6 | 0 | 4 |

This is the instructor's version, a student cannot delete a forum. This page presents a collection of controls for us to test. We can systematically describe tests for each control by scanning them from top to bottom and left to right. The following table presents this list of controls and testing to do:

***Screen: MySCU forum main page***

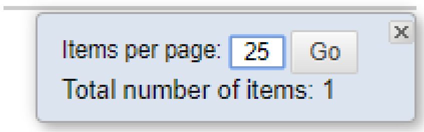
| Widget | Tests | Expected result |
|---|---|---|
| "More Help" link | Check help is displayed | Help screen |
| Create Forum button | Check "Create forum" screen is displayed | Create Forum screen |
| Search button | Check "search" dialogue is displayed | Search dialog |
| Delete Button | 1. Delete first forum<br>2. Delete last forum<br>3. Delete all forums and try to delete again using checkbox<br>4. Add 100 forums and delete first, last and middle forums | 1. Empty forum list<br>2. Removed from list<br>3. No effect<br>4. List adjusted appropriately |
| Checkbox next to "Forum" | 1. Check box<br>2. Click again<br>3. Delete with checked box (all selected)<br>4. Add 100 forums and click checkbox<br>5. Delete with all 100 using checkbox | 1. all forums selected<br>2. all forum's checkbox cleared<br>3. all forums deleted<br>4. all forums selected<br>5. All deleted |

| | | |
|---|---|---|
| Checkbox next to individual forum | See Delete button tests | |
| Total posts field | 1. Check forum with zero posts<br>2. Check Forum with 100 posts (manual count) | 1. Zero reported<br>2. Match manual count |
| Unread posts | 1. Check forum with zero unread posts<br>2. Check forum with about half unread<br>3. Check forum with all read posts | 1. Zero reported<br>2. Manual count matches<br>3. Manual count matches |
| Total participants | 1. Check forum with 1 participant and 1 post<br>2. Check forum with 1 participant and 20 posts<br>3. Check forum with 5 participants and 5 posts<br>4. Check forum with 5 participants and 20 posts<br>Note that for 3 and 4 make sure 60% are replies and 20% are replies to replies | 1. Manual count matches<br>2. Manual count matches<br>3. Manual count matches<br>4. Manual count matches |
| "Show All" button | 1. Click button with zero forums<br>2. Click button with 1 forum<br>3. Add 100 forums and click button with default paging | 1. No effect<br>2. No effect<br>3. Page should display all (default is smaller) |
| Edit Paging button | Click button | Display paging dialog |

There are several important things to note about this table:

1. The table refers to other screens and dialogs, e.g. the create forum screen and the search dialog. These will have a separate table for testing. This is just the way we have chosen to structure the tests in this example. If you wanted, you could do all of the tests on one very large table.
2. The testing certainly has not tested every possible input. We have chosen random data while keeping the strategy of testing limits of the data in mind.
3. The table indicates widgets for individual testing (left hand column) but the tests include interaction between individual widgets. For example, the checkbox next to the "forum" label selects the other checkboxes next to each forum. This is assumed in several of the tests. The implication is that an error detected for the test under this checkbox may appear under another widget's test (e.g. the "Delete" button). This interaction testing is part of the tester's knowledge of the data model.

One of the dialogs referred to is the "paging dialog". When you click on the "paging" button it appears as follows.


Items per page: 25
Total number of items: 1

This has the default value display in the editable text field. Text field processing is the source of many errors so it will require several tests. There are only four widgets to test:

| ***Screen: Paging dialog*** | | |
|---|---|---|
| **Widget** | **Tests** | **Expected result** |
| "x" button | 1. Click "x" without changing value<br>2. Change paging to 5, then click "x" | 1. Paging remains the same<br>2. Paging unchanged |
| Editable number field | 1. Type default value again, GO<br>2. Change to "5", GO<br>3. Change to "0", GO<br>4. Change to "1000000000", GO<br>5. Change to "abc", GO<br>6. Change to "" (empty string), GO | 1. Paging unchanged<br>2. 5 forums per page<br>3. Error<br>4. Error<br>5. Error<br>6. Error |
| "GO" button | This is tested with the editable field | |
| Total items label | 1. Delete all forums<br>2. Add 5 forums<br>3. Add 100 forums | 1. Shows "0"<br>2. Shows "5"<br>3. Shows "100" |

Notice how the text field requires a lot of tests in this case. Since it is a numeric field, this requires consideration of the format of the data as well, i.e. the text entered should be a number. Most text fields do have restrictions on the format of data entered so it is not just numeric fields that require extra tests.

Also in this example, the "go" button widget does not have a test because of its close association with the text field. This linking will occur in many situations, e.g. radio buttons.

It is also worth noting that the MySCU system re-uses dialogs in different parts of the system, which is a common approach to system design. This may mean that the way we split testing into screen and dialog tests will mean that we may not have to test these dialogs for each page. On the other hand, you should be careful that the underlying data model is equivalent and in many cases, it is wise to repeat the tests when a dialog appears in a different part of the system. For example, the search facility in MySCU is operating of different data in the posting list that the forum list because posts have replies and post data is quite different. On the other hand, the paging dialog data is simply a list so may not need additional testing.

You should perform the above tests for the paging dialog because you will find that the expected results (right hand column) for the editable field do not occur as predicted. Obviously, the developers of MySCU did not think this was a bug even though some of us would.

### *Activity 2-3: Blackbox tests*

1. Draw up a black-box detailed test table in the above format for the "search" dialog (click on search) for the MySCU forums. This has different widgets but you can use the same strategies as above.
2. Draw up a detailed black-box test for the MySCU individual forum page. (Click on any forum in the forum list). This is more complicated than the above, so just do some of the widgets if you are pressed for time.

## Example 2: a calculator app

We can approach the calculator app used in the previous section in the same way as MySCU. You should note that, because we are working with numbers, there are some calculation issues we will ignore. Remember that for black-box detailed testing, we are approach this as a developer of the calculator, not an actual user.

We will start with the standard calculator format presented as a default and systematically test each of the first few widgets. We will have additional tests because of the numeric operations, and remember that the expected results must be compared to a number calculated manually or by a trusted alternate calculator.

***Screen: Default calculator screen***

| **Widget** | **Tests** | **Expected result** |
|---|---|---|
| "Ξ" button | Click "Ξ" | Shows selection of other calculator formats |
| 🕘 button | 1. Shows history screen after several calculations completed<br>2. Second click brings back calculator screen | 1. Accurately show previous calculations<br>2. Calculator returned without changes to any field or memory |
| Calculator description field | 1. Change to each of the calculator formats with the "Ξ" button | 1. Ensure this field has correct label |
| History field | 1. Do two number operation for +, -, x and / (do not press '=')<br>2. Do three number operation, e.g. 1+2–3 (do not press '=')<br>3. Do 20 number operation (do not press '=') | 1. History filed shows previous data<br>2. Shows two numbers and two operations<br>3. Shows all history within space provided but all accessible |
| Current result field | This is tested with the operation buttons | |
| MC button | 1. number to memory with M+ button<br>2. Click MC button next<br>3. 1&2 with M- and MS buttons<br>4. Save 5 numbers with MS, then click MC once | 1. MC button become clickable<br>2. MC button not clickable<br>3. Repeats 1&2<br>4. All five numbers removed from memory (check with Mv button) |
| MR button | 1. Number to memory with M+ button<br>2. Click MC button<br>3. Number to memory with M+, then type new number, then click MR<br>4. Add 5 numbers then click MR<br>5. Store 5 numbers with MS, then click MR three times | 1. MR becomes clickable<br>2. MR button not clickable<br>3. Typed number replaced with original number<br>4. Original number recalled<br>5. Only last saved number recalled each time |

*More tests to come ...*

Some of these tests may not be obvious unless you use the calculator to see how it works. This assumes it is working as intended in this exercise. We have also skipped the Windows buttons that appear on the top frame of the window. Note that if you have a previous version of Microsoft calculator utility there are differences between the way they operate.

These tests have also tested the interaction between various widgets via the underlying data model. This is particularly obvious with the memory tests, where the memory is a stack of values in the calculator but the buttons may not operate as you expect.

## *Activity 2-4: Black-box tests: Microsoft Calculator*

1. Add the rest of the buttons on the MC and MR line of the calculator. Make sure you know how the calculator works and assume they work as intended so your tests will be accurate.
2. Write the tests for the %, Square root, square and inversion buttons.
3. Write the tests for the +, -, x and ÷. Just use normal arithmetic expectation here.
4. (optional) complete the table. You do not have to worry about the 1, 2, ..., 0 or '.' Buttons as there are tested enough in the other operations (and are very basic).

That is enough unit testing for this week. Remember that the testing is really an art form so everyone's testing ideas will be different. The idea is to do as much testing with the time and resources you have available because it is impossible to test all possible inputs for most applications.