Software Testing

Foundation University Rawalpindi Campus
BS (Software Engineering)
Spring 2014

Instructor: Sohaib Altaf
sohaib@hybriditservices.com
http://www.hybriditservices.com/course/FU-BSSE8-ST

Lecture 10

Types of Black Box Testing

Functional Testing: Black Box Techniques

Example

Consider a basic test case to ensure that players can move on a Monopoly board. Example of poorly specified test case is shown below Table, WHY POOR?

Test ID	Description	Expected Results	Actual Results
1	Player 1 rolls dice and moves.	Player 1 moves on board.	
2	Player 2 rolls dice and moves.	Player 2 moves on board.	

Functional Testing: Black Box Techniques

Example

Consider a basic test case to ensure that players can move on a Monopoly board. Example of poorly specified test case is shown below Table, WHY POOR?

Test ID	Description	Expected Results	Actual Results
1	Player 1 rolls dice and moves. How Much to Move? 3 or 4 or ?	Player 1 moves on board. Where to move?	
2	Player 2 rolls dice and moves.	Player 2 moves on board.	

Functional Testing: Black Box Techniques

Example: How to Prepare the Test Execution ? Take Example of Driver, how Can we test if he understand the signal or not during his journey when he is Passing from G.T. Road? Test if he knows the Signal of controlling the speed and Resume speed if he in the town or out of town……?

Test ID	Description	Expected Results	Actual Results
1	Player 1 rolls dice: 3 and moves.	Player 1 moves on board and go to Jail	Now prepare your test execution & Observe the actual
2	Player 2 rolls dice: 4 and moves.	Player 2 Passes from Go and landed at Green Park (Collected $ 20 after passing from GO)	

Now your test is prepared with (2 Tests), Test your Game Software now…..

Report: Use test case (test id 1 & 2) and record if Player 1 & 2 is now moving as per description or not, and fill the information as required in the format

Functional Testing: Black Box Techniques

Example

Requirement: Players can move on a Monopoly board when dice rolls (**If requirement is poor specified? Contact Manager or Lead..**)

Ok Test to ensure that if players can move on a board by all aspects

Test ID	Description	Expected Results	Actual Results
1	Player 1 rolls dice: 3 and moves.	Player 1 moves on board and go to Jail	Player 1 moved to Jail.
2	Player 2 rolls dice: 4 and moves.	Player 2 Passes from Go and landed at Green Park (Collected $ 20)	Player 1 moved to Jail.

Report: If your 1st Test Passed? What was required? What you test? And what are results? If test completed? What are steps of your test, if objective fulfilled…. Or not? If your 2nd Test Passed? NO? if it is minor or severe defect, if is low priority, who is developer of this code, to whom we assign for fixing…….. Report ALL

Functional Testing: Black Box Techniques

Example

…….Ensure that players can move on a Monopoly board.

Now Developer debugged & fixed the defect, re-check & inform to Test. Now tester re-test it & report the results. If to Close or Re-open?

Test ID	Description	Expected Results	Actual Results
1	Player 1 rolls dice: 3 and moves.	Player 1 moves on board and go to Jail	Player 1 moved to Jail.
2	Player 2 rolls dice: 4 and moves.	Player 2 Passes from Go and landed at Green Park (Collected $ 20 after passing from GO)	Player 2 Passes from Go and landed at Parking (not Collected $ 20)

Report: Use test case (test id 2) and report that Player 2 is now moving as per description, but partial requirement is not correct (not collected $20)

Functional Testing: Black Box Techniques

1. The problem is that the description does not give exact values of how many spaces the players moved.
2. Program can crash for some reason when Player 1 and Player 2 land on the same spot.
3. If you don't remember what was actually rolled (you let the rolls be determined randomly and don't record them), you might never be able to cause the problem to happen again because you don't remember the circumstances leading up to the problem.
4. Recreating the problem is essentially important in testing so that problems that are identified can be repeated and corrected. Instead write specific descriptions, such as shown in below table.
5. Sometime partial requirement is ok and partial is incorrect

Functional Testing: Black Box Testing

Test ID	Description	Expected Results	Actual Result
3	Precondition: Game is in test mode, Simple Game Board is loaded, and game begins. (Specification of Execution) Number of players: 2 Money for player 1: $1200 Money for player 2: $1200 Player 1 dice roll: 3 ----------	---------- Player 1 is located at Blue 3.	
4	Precondition: Test case 3 has successfully completed ---------- Player 2 dice roll: 3 ----------	Player 1 is located on Blue 3. ---------- ---------- Player 2 is located on Blue 3.	

Preferred Specification of a Test Case. Specification of Execution

Functional Testing: Black Box Techniques

Note the test case

1. **Notice the Precondition in the Description field when one test case depends on another test case. (Cont..)**

   a. The precondition defines what has to happen before the test case can run properly. There may be an order of execution whereby a test case may depend upon another test case running successfully and leaving the system in a state such that the second test case can successfully be executed.

      For example, may be one test case B tests whether a new user can create an ID in a system. Other test case C may depend upon this new user logging in. In this case test case C will depend on new login. If Login Fails test case C will fail also. In this case test case B will re-run before re-run test case C

   b. If test case B pass, even then test case C fails.

   c. We can run test case C by re-initializing database or the system before a test case can run. (Stub)

Functional Testing: Black Box Techniques

Note the test case & prepare for test execution

2. Notice How can the test case ensure the player rolled a 3 when the value the dice rolls needs to be random in the real game? Roll Range? Other tests

In this case we may have to add a bit of extra functionality to put a program in "test mode" so we can run our test cases in a repeatable manner and so we can easily force a condition happen. For example, we may want to test what happens when a player lands on "Go" or on "Go to Jail" and want to force this situation to occur.

The Monopoly programmers needed to create a test mode in which
(1) the dice rolls could be input manually and
(2) the amount of money each player starts with, is input manually.
(3) run some non-repeatable test cases in the regular game mode to test whether random dice input does not appear to change expected behavior.

Functional Testing: Black Box Techniques

What is Ideal:

How much to test? What are other scenarios where game ends.
To test every possible thing that can be done with our program.
But…… writing and executing every test cases is expensive.

Prepare Selective Test Cases

Our objective is to find as many defects as possible in as few test cases as possible. To accomplish this objective, we use some techniques

1. We write test cases for the kinds of things that the customer will do most often or even fairly often. (Profile)
2. We want to avoid writing redundant test cases that won't tell us anything new (because they have similar conditions to other test cases we already wrote).
3. Each test case should probe a different mode of failure.
4. Design the simplest test cases
5. Test cases can be error-prone, re-check

Types of Functional Testing: Black Box Techniques

1. Tests of Customer Requirements

a. Black box test cases are based on customer requirements.

b. Look at each customer requirement.

c. Every single customer requirement has been tested at least once.

d. As a result, we can trace every requirement to its test case(s) and every test case back to its stated customer requirement.

e. For Each Requirement, write most-used *success path* for that requirement. E.g. execute *some desirable functionality (something the customer wants to work) without any error conditions*.

f. Success path: a test case that execute some desirable functionality (something the customer wants to work) without any error conditions

g. Failure path: a test case that intentionally forces an error condition to occur

Types of Functional Testing: Black Box Techniques

1. Tests of Customer Requirements

h. Some test cases that execute *failure paths*. failure paths *intentionally have some kind of errors in them, such as errors that users can accidentally input*. We must make sure that the program behaves predictably and gracefully in the face of these errors.

i. Plan the execution of our tests out so that the most troublesome, risky requirements are tested first. This would allow more time for fixing problems before delivering the product to the customer. It would be shocking to find a critical flaw right before the product is due to be delivered.

j. It is impossible to test every single possible combination of input. We'll outline an incomplete sampling of test cases and reason about

Types of Functional Testing: Black Box Techniques

Requirement: *When a user lands on the "Go to Jail" cell, the player goes directly to jail, does not pass go, does not collect $200. On the next turn, the player must pay $50 to get out of jail and does not roll the dice or advance. If the player does not have enough money, he or she is out of the game.*

There are many things to test in this short requirement above, including:

1. Does the player get sent to jail after landing on "Go to Jail"?
2. Does the player receive $200 if "Go" is between the current space and jail?
3. Is $50 correctly decremented if the player has more than $50?
4. Is the player out of the game if he or she has less than $50?

How to Start: (See your plan which you prepared, or update it)

1. First: it is good to start out by testing some input that you know should definitely pass or definitely fail. Success and fail path
2. If these kinds of tests don't work properly, just quit testing and put the code back into development.

Types of Functional Testing: Black Box Techniques

Test ID	Description	Expected Results	Actual Result
5	1. Precondition: Game is in test mode. Number of players: 1 Money for player 1: $1200 Player 1 dice roll: 3 Player 1 clicks “End Turn” button. ---------- ---------- 3. Player 1 clicks “Get Out of Jail” button.	---------- 2. Player 1 is sent to jail Only “Get Out of Jail” button is enabled for Player 1. ---------- ---------- 4. Money for Player 1: $1150	
6	1. Precondition: Game is in test mode. Number of players: 2 Money for player 1: $1200 Money for player 2: $1200 Player 1 dice roll: 3 Player 1 clicks “End Turn” button. 3. Player 2 dice roll: 2 Player 2 clicks “End Turn” button. 5. Player 1 clicks “Get out of Jail” button.	2. Player 1 is sent to jail 4. Only “Get Out of Jail” button is enabled for Player 1 6. Money for Player 1: $1150	

Test Plan #1 for the Jail Requirement

Types of Functional Testing: Black Box Techniques

Test Case 5 & 6

1. IMP NOTE: T*est the simplest possible means to force the condition we are trying to achieve*. For example, in Test Case 5, we only have one player so we temporarily didn't have to spend our time with Player 2.

2. We add Player 2 in Test Case 6 so we can observe that the loss of $50 and dice roll occurs on the next turn (after Player 2 goes).

3. Test many more aspects of the above requirement.

Types of Functional Testing: Black Box Techniques

2. Equivalence Partitioning

a. Keep down our testing costs

b. Don't write several test cases that test the same aspect of our program.

c. Equivalence partitioning is a type of black box testing that can be used to reduce the number of test cases that need to be developed.

d. Equivalence partitioning divides the input domain of a program into classes. For each of these equivalence classes, the set of data should be treated the same by the module under test and should produce the same answer.

e. Test cases should be designed so the inputs lie within these equivalence classes.

For example, for tests of "Go to Jail" the most important thing is whether the player has enough money to pay the $50 fine. Therefore, the two equivalence classes can be partitioned, as shown below

Less than $50	$50 or more

Types of Functional Testing: Black Box Techniques

2. Equivalence Partitioning (Cont..)

f. Once partitions are identified, choose test cases from each partition. See Test Case 7 & 8 for valid & invalid classes for test cases

g. Test Cases 6 (Player 1 has $1200) and 7 (Player 1 has $100) are both in the same equivalence class. Therefore, Test Case 7 is unlikely to discover any defect not found in Test Case 6.

h. Test Cases 7 (Player 1 has $100) and 8 (Player 1 has $25) are both in the different equivalence class. Therefore, Test Case 7 will likely find different classes of defects from Test Case 8.

Types of Functional Testing: Black Box Techniques

Test ID	Description	Expected Results	Actual Results
7	1. Precondition: Game is in test mode. Number of players: 2 Money for player 1: $100 Money for player 2: $100 Player 1 dice roll: 3 Player 1 clicks "End Turn" 3. Player 2 dice roll: 2 Player 2 clicks "End Turn" 5. Player 1 clicks "Get Out of Jail"	2. Player 1 is sent to jail 4. Only "Get Out of Jail" is enabled for Player 1. 6. Money for Player 1: $50	
8	1. Precondition: Game is in test mode. Number of players: 2 Money for player 1: $25 Money for player 2: $25 Player 1 dice roll: 3 Player 1 clicks "End Turn" 3. Player 2 dice roll: 2 Player 2 clicks "End Turn" 5. Player 1 clicks "Get out of Jail"	2. Player 1 is sent to jail 4. Only "Get Out of Jail" is enabled for Player 1. 6. Player 1 is out of game	

Test Plan #2 for the Jail Requirement

Types of Functional Testing: Black Box Techniques

2. Equivalence Partitioning (Cont..)

Define test case using following guideline for each equivalent class

1. If input conditions specify a *range of values*, create one valid and one or two invalid equivalence classes.
   In the above example, this is (a) less than 50/invalid;

   (b) 50 or more/valid.

2. If input conditions require a certain value (for example A and D for the for sorting a list in Ascending & Descending, create an equivalence class of the valid values (A and D) and one of invalid values (all other letters other than A and D).

   In this case, you need to test all valid values individually and several invalid values.

3. If an input condition is a Boolean, define one valid and one invalid class.

Types of Functional Testing: Black Box Techniques

3. Boundary Value Analysis

1. Boris Beizer, well-known author of testing book advises, "Bugs creep around in corners and congregate at boundaries."

2. Programmers often make mistakes on the boundaries of the equivalence classes/input domain.

3. We need to focus testing at these boundaries. This type of testing is called Boundary Value Analysis (BVA)

4. This techniques guides you to create test cases at the "edge" of the equivalence classes.

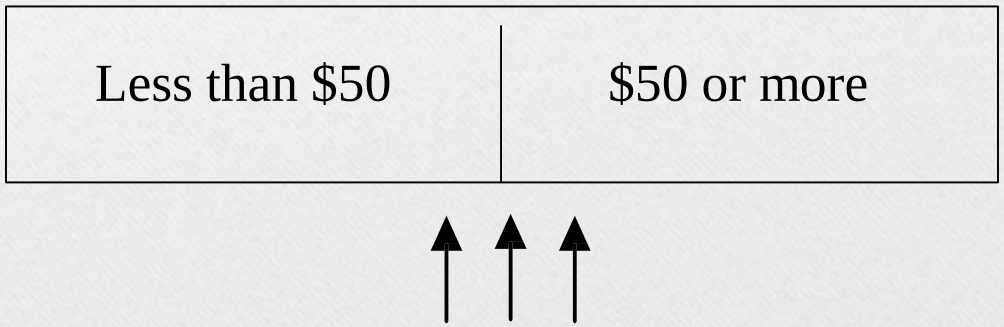
5. *Boundary value* is defined as a *data value that corresponds to a minimum or maximum input, internal, or output value specified for a system or component.*

Types of Functional Testing: Black Box Techniques

3. Boundary Value Analysis

5. For example, the boundary of the class is at 50,
We should create test cases for the Player 1 having $49, $50, and $51.
These test cases will help to find common off-by-one errors, caused by errors like using >= when you mean to use >.

= $ 0, < $ 50 ,OR, = $ 50 , > $ 50



Boundary Value Analysis. Test cases should be created for the Boundaries (arrows) between equivalence classes.

Types of Functional Testing: Black Box Techniques

Important when creating BVA test cases:

1.If input conditions have a range from **a** to **b** (such as a=100 to b=300), create test cases:

- immediately below **a** (99)
- at **a** (100)
- immediately above **a** (101)
- immediately below **b** (299)
- at **b** (300)
- immediately above **b** (301)

2. If input conditions specify a *number* of values that are allowed, test these limits. For example, input conditions specify that only one train is allowed to start in each direction on each station. In testing, try to add a second train to the same station/same direction. If (somehow) three trains could start on one station/direction, try to add two trains (pass), three trains (pass), and four trains (fail).

If 2 is allowed then test 2 & 3,

If 2, 3, 4 is allowed then test 2, 3, 4, & 5

For Integers, test an equivalence class from integers, and test any single value exceeding the highest limit of integer , > 33000 or < - 33000

Types of Functional Testing: Black Box Techniques

4. Decision Table Testing

1. Decision tables are used to record complex business rules that must be implemented in the program, and therefore tested.

2. A sample decision table is found below. In the table, the conditions represent possible input conditions.

3. The actions are the events that should trigger, depending upon the makeup of the input conditions.

4. Each column in the table is a unique combination of input conditions (and is called a rule) that result in triggering the action(s) associated with the rule.

5. Each rule (or column) should become a test case.

If a Player (A) lands on property owned by another player (B), A must pay rent to B. If A does not have enough money to pay B, A is out of the game.

	Rule 1	Rule 2	Rule 3
Conditions			
A lands on B's property	Yes	Yes	No
A has enough money to pay rent	Yes	No	--
Actions			
A stays in game	Yes	No	Yes

Decision table

Types of Functional Testing: Black Box Techniques

5. Failure ("Dirty") Test Cases

1. Think of every possible thing a user could possibly do with your system to demolish the software.

2. You need to make sure your program is robust – in that it can properly respond in the face of erroneous user input. This type of testing is called *robustness testing*,

3. Whereby test cases are chosen outside the domain to test robustness to unexpected, erroneous input and is included in *defensive testing* which includes *tests under both normal and abnormal conditions* [5].

4. Look at every input. Does the program respond "gracefully" to these error conditions?

Types of Functional Testing: Black Box Techniques

5. Failure ("Dirty") Test Cases; Checklist

1.Can any form of input to the program cause division by zero? Get creative!

2. What if the input type is wrong? (You're expecting an integer, they input a float. You're expecting a character, you get an integer.)

3. What if the customer takes an illogical path through your functionality?

4. What if mandatory fields are not entered?

5. What if the program is aborted abruptly or input or output devices are unplugged?

6. Defensive testing

Testing which includes tests under both normal and abnormal conditions

Types of Functional Testing: Black Box Techniques

Scaffolding code
computer programs and data files built to support software development and testing but not intended to be included in the final product

Stubs
computer program statement substituting for the body of a software module that is or will be defined elsewhere

Test driver
software module used to involve a module under test and often, provide test inputs, controls, and monitor execution and report test results