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 11

Types of White Box Techniques

White-Box Testing

- White-box testing is a verification technique software engineers can use to examine if their code works as expected.

- White-box testing is testing that takes into account the internal mechanism of a system or component (IEEE, 1990).

- White-box testing is also known as structural testing, clear box testing, and glass box testing (Beizer, 1995).

- "Clear box" and "glass box" appropriately indicate:
that you have full visibility of the internal workings of the software product, specifically, the logic and the structure of the code.

- Using the white-box testing techniques design test cases that..

(1) exercise independent paths within a module or unit;

(2) exercise logical decisions on both their true and false side;

(3) execute loops at their boundaries

(4) exercise internal data structures to ensure their validity (Pressman, 2001).

White-Box Testing

- There are six basic types of life cycle phase testing: unit, integration, function/system, acceptance, regression, and beta.

- White-box testing is used for three of these six types:

- Unit Testing

  - *Unit testing*, which is *testing of individual hardware or software units or groups of related units* (IEEE, 1990).

  - A *unit* is *a software component that cannot be subdivided into other components* (IEEE, 1990).

  - Software engineers write white-box test cases to examine whether the unit is coded correctly.

  - Unit testing is important for ensuring the code is solid before it is integrated with other code.

  - Once the code is integrated into the code base, the cause of an observed failure is more difficult to find.

  - Software engineer writes and runs unit tests him or herself, companies often do not track the unit test failures that are observed– making these types of defects the most “private” to the software engineer.

  - Software engineers own mistakes - have the opportunity to fix them without others knowing.

  - Approximately 65% of all bugs can be caught in unit testing (Beizer, 1990).

  .

White-Box Testing

- Integration Testing

- *Integration testing*, which is *testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them* (IEEE, 1990).

-Black Box test cases are written which explicitly examine the interfaces between the various units

- Tester understands that a test case requires multiple program units to interact.

- Alternatively, white-box test cases are written which explicitly exercise the interfaces that are known to the tester.

- Regression Testing

- *Regression testing*, which is *selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements* (IEEE, 1990).

- As with integration testing, regression testing can be done via black-box test cases, white-box test cases, or a combination of the two.

- White-box unit and integration test cases can be saved and re- run as part of regression testing.

White-Box Testing

1 White-Box Testing by Stubs and Drivers

- With white-box testing, run the code with predetermined input and check to make sure that the code produces predetermined outputs.

- Often programmers write stubs and drivers for white-box testing.

- A *driver* is *a software module used to invoke a module under test and, often, provide test inputs, control and monitor execution, and report test results* (IEEE, 1990)

- or most simplistically a *line of code that calls a method and passes that method a value*.

- For example, if you wanted to move a Player instance,Player1, two spaces on the board, the driver code would be

```
movePlayer(Player1, 2);
```

This driver code would likely be called from the main method. A white-box test case would execute this driver line of code and check

`Player.getPosition()` to make sure the player is now on the expected cell on the board.

White-Box Testing

1 White-Box Testing by Stubs and Drivers

- A *stub* is a *dummy component or object used to simulate the behavior of a real component* (Beizer, 1990) until that component has been developed.

- For example, if the movePlayer method has not been written yet, a stub as given below might be used temporarily that moves any player to position 1.

```
public void movePlayer(Player player int diceValue)
{
   player.setPosition(1);}
```

- Developing the stub allows the programmer to call a method in the code being developed, even if the method does not yet have the desired behavior.

- Dummy method would be completed with the proper program logic.

- Stubs and drivers are often viewed as throwaway code (Kaner, Falk et al., 1999).

- However, they do not have to be thrown away: Stubs can be “filled in” to form the actual method. Drivers can become automated test cases.

White-Box Testing

2 Deriving Test Cases

- Various methods for devising a thorough set of white-box test cases.

- Act of careful, complete, systematic test **design** will catch as many bugs as the act of testing.

Example of Monopoly

2.1 Basis Path Testing

- *Basis path testing* (McCabe, 1976) is a means for ensuring that all independent paths through a code module have been tested.

- To introduce the basis path method, we will draw a flowgraph of a code segment.

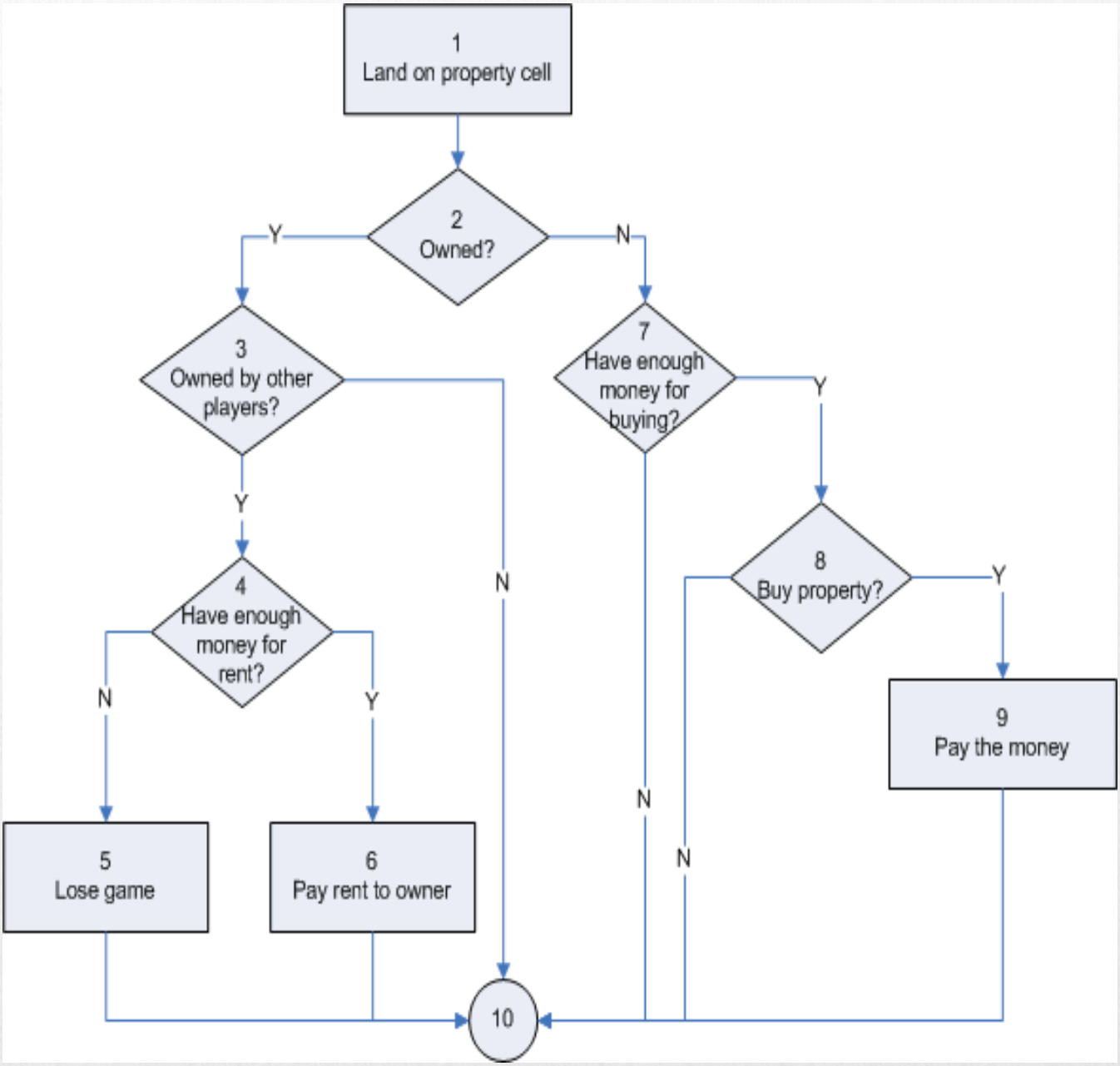
- A flowgraph of purchasing property

If a player lands on a property owned by other players, he or she needs to pay the rent. If the player does not have enough money, he or she is out of the game. If the property is not owned by any players, and the player has enough money buying the property, he or she may buy the property with the price associated with the property.

White-Box Testing

Example of Monopoly

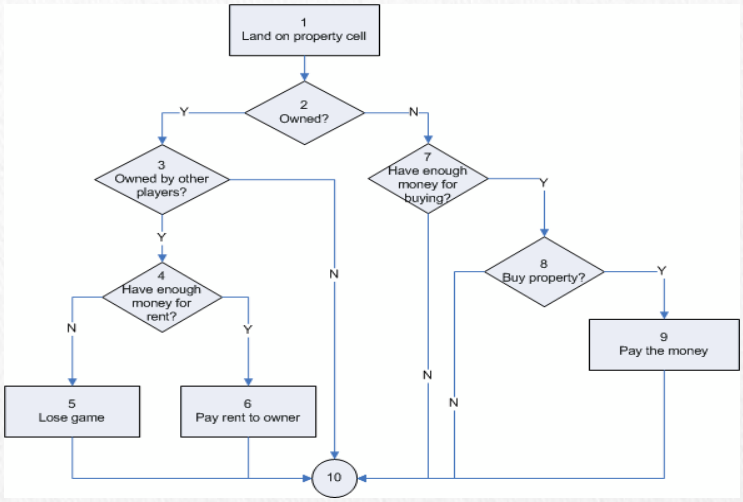
2.1 Basis Path Testing



White-Box Testing

Example of Monopoly

2.1 Basis Path Testing



- Compute the number of independent paths through the code.

- Using a metric called the *cyclomatic number* (McCabe, 1976), which is based on graph theory.

- The easiest way to compute the cyclomatic number is to count the number of conditionals/predicates (diamonds) and add 1.

- In above example, there are five conditionals.

- Cyclomatic number is 6, --> six independent paths through the code i.e.

1. 1-2-3-4-5-10 (property owned by others, no money for rent)
2. 1-2-3-4-6-10 (property owned by others, pay rent)
3. 1-2-3-10 (property owned by the player)
4. 1-2-7-10 (property available, don't have enough money)
5. 1-2-7-8-10 (property available, have money, don't want to buy it)
6. 1-2-7-8-9-10 (property available, have money, and buy it)

- Write a test case to ensure that each of these paths is tested at least once.

White-Box Testing

Example of Monopoly

2.2 Equivalence Partitioning/Boundary Value Analysis

- In case of any number, write Test Case using Equivalence partitioning (EP) & Boundary Value Analysis (BVA)

- For example, a person might want to buy a house, but may or may not have enough money. Considering the following 2 EP Test Cases; one for enough money and one for don't have enough money

1. property costs $100, have $200 (equivalence class "have enough money")
2. property costs $100, have $50 (equivalence class, "don't have enough money")

- In case of boundary value analysis (BVA) write 3 white-box test cases.

- With programming loops (such as while loops), For BVA, you will want to ensure that you execute loops right below, right at, and right above their boundary conditions.

3. property costs $100, have $100 (boundary value)
4. property costs $100, have $99 (boundary value)
5. property costs $100, have $101 (boundary value)

White-Box Testing

Example of Monopoly

3 Control-flow/Coverage Testing

- Devise a good set of white-box test cases is to consider the control flow of the program.

- The adequacy of the test cases is often measured with a metric called coverage.

- *Coverage* is a measure of the completeness of the set of test cases.

- See example below that will demonstrate the various kinds of coverage

```
1     int foo (int a, int b, int c, int d, float e)
2     {float e;
3           if (a == 0)  {
4           return 0;
5          }
6           int x = 0;
7           if ((a==b) OR ((c == d)AND bug (a)) {
8        x=1;
9          }
10       e = 1/x;
11     return e;
12      }
```

White-Box Testing

Example of Monopoly

3.1 Method Coverage

- *Method coverage* is *a measure of the percentage of methods that have been executed by test cases*.

- Tests call 100% of your methods.

- It seems irresponsible to deliver methods in your product when your testing never used these methods.

- Ensure you have 100% method coverage.

- In above program, attain 100% method coverage by calling the foo method.

- Consider Test Case 1: the method call **foo(0, 0, 0, 0, 0.)**, expected return value of 0. If you look at the code, you see that if a has a value of 0, it doesn't matter what the values of the other parameters are –make them all 0. Through this one call you attain 100% method coverage.

White-Box Testing

Example of Monopoly

3.2 Statement Coverage

- Statement coverage is a measure of the percentage of statements that have been executed by test cases.

- Objective should be to achieve 100% statement coverage through your testing.

- Identifying your cyclomatic number and executing this minimum set of test cases will make this statement coverage achievable.

- Test Case 1, executed the program statements on lines 1-5 out of 12 lines of code and had 42% (5/12) statement coverage from Test Case 1.

- You can attain 100% statement coverage by one additional test case, Test Case 2: the method call **foo(1, 1, 1, 1, 1.)**, expected return value of 1.

- With this method call, we have achieved 100% statement coverage because we have now executed the program statements on lines 6-12.

White-Box Testing

Example of Monopoly

3.3 Branch Coverage

- Branch coverage is a measure of the percentage of the decision points (Boolean expressions) of the program have been evaluated as both true and false in test cases.

The small program given above has two decision points: – one on line 3 and the other on line 7.

```
3    if (a == 0)  {
7            if ((a==b) OR ((c == d) AND)) {
```

- For decision/branch coverage, we evaluate an entire Boolean expression as one true-or-false predicate even if it contains multiple logical-and or logical-or operators – as in line 7.
- Ensure that each of these predicates (compound or single) is tested as both true and false. Table 1 shows our progress so far:

White-Box Testing

Example of Monopoly

3.3 Branch Coverage

Line #	Predicate	True	False
3	(a == 0)	Test Case 1 **foo(0, 0, 0, 0, 0)** **return 0**	Test Case 2 **foo(1, 1, 1, 1, 1)** **return 1**
7	((a==b) OR ((c == d) AND bug(a)))	Test Case 2 **foo(1, 1, 1, 1, 1)** **return 1**	

- Executed three of the four necessary conditions & achieved 75% branch coverage thus far.

- Add Test Case 3 for 100% branch coverage: **foo(1, 2, 1, 2,)**

- Now look at the code to calculate an expected return value, this test case uncovers a previously undetected division-by-zero problem on line 10!

- Fix the code and protect from such an error.

- Value of test planning: Through the test case plan, 100% branch coverage achieve

- the objective is to achieve 100% branch coverage in your testing, though in large systems only 75%-85% is practical. Only 50% branch coverage is practical in very large systems of 10 million source lines of code or more Beizer, 1990).

White-Box Testing

Example of Monopoly

3.4 Condition Coverage

- Going one step deeper and examine condition coverage.

- Condition coverage is a measure of percentage of Boolean sub-expressions of the program that have been evaluated as both true or false outcome [applies to compound predicate] in test cases.

- Notice that in line 7 there are three sub-Boolean expressions to the larger statement (a==b), (c==d), and bug(a). Condition coverage measures the outcome of each of these sub-expressions independently of each other.

- With condition coverage, you ensure that each of these sub-expressions has independently been tested as both true and false. See below

Predicate	True	False
(a==b)	Test Case 2 **foo(1, 1, x, x, 1)** **return value 0**	Test Case 3 **foo(1, 2, 1, 2, 1)** **division by zero!**
(c==d)		Test Case 3 **foo(1, 2, 1, 2, 1)** **division by zero!**
bug(a)		

- In above table, our condition coverage is only 50%. The true condition (c==d) has never been tested.

- Additionally, if c==d is false we cannot execute bug(int), therefore we plan another test case having c and d == 1, and we will determine that our bug method should return a value of true when passed a value of a=1.

White-Box Testing

Example of Monopoly

3.4 Condition Coverage

- We write Test Case 4 to address test (c==d) as true: **foo(1, 2, 1, 1, 1)**, expected return value 1.

- Run the test case, if bug (1) is returning false value or if bug (3) is giving false, and in case of division by zero, fix the code. So we create Test Case 5, **foo(3, 2, 1, 1, 1)**, expected return value "division by error".

- Develop additional test cases
- This test is too much time consuming

- There are commercial tools available, called *coverage monitors*, that can report the coverage metrics for your test case execution.

- Often these tools only report method and statement coverage.

- Some tools report decision/branch and/or condition coverage.

- These tools often also will color code the lines of code that have not been executed during your test efforts.

- It is recommended that coverage analysis is automated using such a tool because manual coverage analysis is unreliable and uneconomical (IEEE, 1987).

White-Box Testing

Example of Monopoly

4 Data Flow Testing

- In data flow-based testing, about how the program variables are defined and used.
- Different criteria exercise with varying degrees of precision how a value assigned to a variable is used along different control flow paths.

Glossary of Chapter Terms

Word	Definition	Source
branch coverage	a measure of the percentage of the decision points (Boolean expressions) of the program have been evaluated as both true and false in test cases	
condition coverage	a measure of the percentage of Boolean sub-expressions of the program that have been evaluated as both true or false outcome [applies to compound predicate] in test cases	
driver	software module used to invoke a module under test and, often, provide test inputs, control and monitor execution, and report test results	(IEEE, 1990)
integration testing	testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them	(IEEE, 1990)
method coverage	a measure of the percentage of methods that have been executed by test cases.	
regression testing	selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements	(IEEE, 1990)
statement coverage	a measure of the percentage of statements that have been executed by test cases	
stub	computer program statement substituting for the body of a software module that is or will be defined elsewhere	(IEEE, 1990)
unit	a separable, testable element specified n the design of a computer software component; a software component that cannot be subdivided into other components	(IEEE, 1990)
unit testing	testing of individual hardware or software units or groups of related units	(IEEE, 1990)
white-box testing	testing that takes into account the internal mechanism of a system or component	(IEEE, 1990)