SE 342 Course Notes - Lecture 2

General Overview of Testing

Dr. Uzay Çetin

Maltepe University
Software Engineering Department

April 19, 2017

Outline

Testing Essentials
- Debugging
- Testing Methods

Levels of Testing
- Functional Testing
- Non functional Testing

Testing Documentation
- Quiz

What is testing?

What is testing?

Evaluation of a system whether it satisfies the specified requirements or not. This process detects the differences (bugs/error/defects) between the desired system and the actual.

Who does testing?

- Developpers (unit tests)
- Software Testers
- Managers
- End users

When to begin testing?

When to begin testing?

As soon as possible. Bugs found later costs more to fix.

Testing in different SDLCs

Software Development Life Cycle (SDLC)

- Water fall model $\rightarrow$ at Testing phase
- Incrementel model $\rightarrow$ at the end of each increment
- V-shape model $\rightarrow$ relevant test phase at every level

When to stop testing?

> *Testing can only show the presence of bugs, but it can not show the absence of bugs. -Dijkstra*

No clear answer, It depends

- Manager decisions, deadlines etc.
- Completion of test cases for a sufficient code coverage
- Low and acceptable Bug rates is reached

Complete Testing is NOT Possible. You accept the risk and stop testing at some point.

Testing hints

- Love computer a little less.
- Learn to suspect and mistrust!
- Be selective! Test the most riskiest parts first!
- Think alternatives scenarios, try to crash the Software

Early testing saves both time and money!!

Verification vs Validation

Verification

Are we building the product right?

- related to number of bugs.
- done by developpers
- has an objective nature

Validation

Are we building the right product?

- related to specificaiton!
- done by testers.
- has a subjective nature

Testing vs Debugging

Testing

- Identification of bug/error/defect without correcting it.
- performed in the testing phase.

Debugging

- Identification of bug/error/defect in order to fix them
- done upon encountering an error
- part of White box or Unit Testing
- performed in the development phase

Testers vs Developpers role in testing

Both try to detect bugs.

- Developpers are responsible for their specific code component
- Testers are responsible for the overall workings of the software, what the dependencies are and what the impacts of one module on another module are.

Testing types

Manual Testing

- The role of end user, (no script and tool)

Automated Testing

- The use of other software tools to test the software at hand.

GUI items, connections, field validations in forms can be effectively tested via automation.

Testing Methods I

Blackbox Testing

- Testing without having access the source code.
- Interact with the system interface, provide input and compare the output.

\+ is good when there exists large amount of code, provides testing with user's perspective.

\- inefficient testing, blind coverage

Testing Methods II

Whitebox Testing

- Detailed investigation of internal structure and logic of the source code.

\+ open, complete and detailed knowledge of code.

\- Difficult to perform, costly

Greybox Testing

- Testing with limited knowledge

Levels of Testing

Functional Testing

A type of block-box testing performed on a completed software

1. Determine intended functionality of the software
2. Create test data according to the specification
3. Determine the corresponding output
4. Determine test scenarios and execute test cases
5. Compare actual and expected outputs.

Functional Testing

Unit Testing

Performed by developers before the execution of test cases. Do individual parts correctly function wrt the requirements?

Integration Testing

Test the interaction between modules. Does the combination of parts correctly function wrt the requirements?

- Bottom-up integration followed by Top-down integration.

Functional Testing

Regression Testing

conducted to understand the effect of a change. Ex: A bug fix can cause an unintended error in an another code segment.

System Testing

Test the system as a whole in different environments.

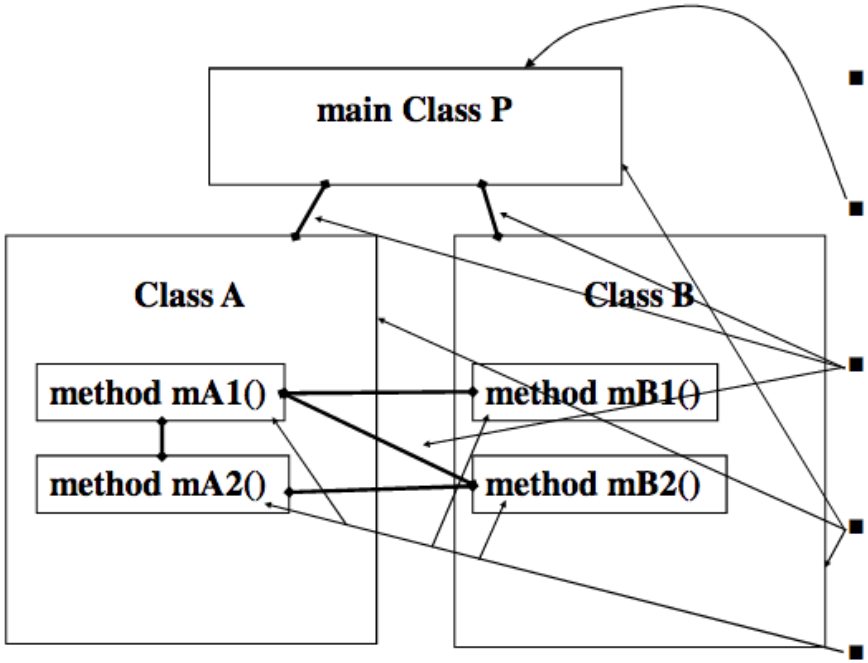
Functional Testing

Acceptance Testing

Validation of user requirements

- Alpha Test: conducted within the company by another team excluding devolopers and testers Ex: test loading times for poorest machines)
- Beta Test: pre-release test conducted by a sample of interested audience outside the company

Functional Testing



- **Acceptance testing: Is the software acceptable to the user?**
- **System testing: Test the overall functionality of the system**
- **Integration testing: Test how modules interact with each other**
- **Module testing: Test each class, file, module or component**
- **Unit testing: Test each unit (method) individually**

Ref: Jeff Offutt's course

Levels of Testing

Non functional Testing

- Performance, security, user interface, portability testing

Testing Documentation

Documentation

- Test Plan
  - involves strategy, resources, assumptions related to testing environment, list of test cases, and a schedule
- Test Scenario
  - several test cases in a specific order within a context
- Test Case
  - detailed steps and conditions for which a software passes or fails for the given inputs and expected outcomes.
- Traceability Matrix
  - Test Case Id is linked to the requirements and bug

..

- ..

Test case

Test Case ID:	<TC ID>	**Test Engineer:**	<Test Engineer>
Product Module:	Home Page	**Testing Date:**	29-08-2011
Product Version:		**Testing Cycle:**	1
Revision History:		**Status:**	
Purpose:	<Purpose>		
Assumptions	<Assumptions>		
Pre-Conditions:	<Pre-Condition>		
Steps to Reproduce:	<Steps to Reproduce>		
Expected Results:	<Expected Outcome>		
Actual Outcome:	<Actual Outcome>		
Post Conditions	<Purpose>		

Ref: www.tutorialspoint.com

Test case

Search routine specification

procedure Search (Key : INTEGER ; T: array 1..N of INTEGER;
Found : BOOLEAN; L: 1..N) ;

Pre-condition
-- the array has at least one element
1 <= N
Post-condition
-- the element is found and is referenced by L
(Found and T (L) = Key)
or
-- the element is not in the array
(**not** Found **and**
not (**exists** i, 1 >= i >= N, T (i) = Key))

Ref: Spiros Mancoridis' course

Test case

Check for

- Valid inputs (conforming the pre-conditions)
- Non valid inputs (not conforming the pre-conditions)
- Empty input

Table: Test array with single element

Array	Key	Found	L
[2]	2	True	0
[3]	2	False	?

Test case

Check first, middle, last elements (boundary conditions) of an array with multiple elements. Check non-existing element.

Table: Test array with multiple elements

Array	Key	Found	L
[2,3,4,5]	2	True	0
[2,3,4,5]	3	True	1
[2,3,4,5]	5	True	3
[2,3,4,5]	6	False	?

Control Flowgraph

Typical nodes are

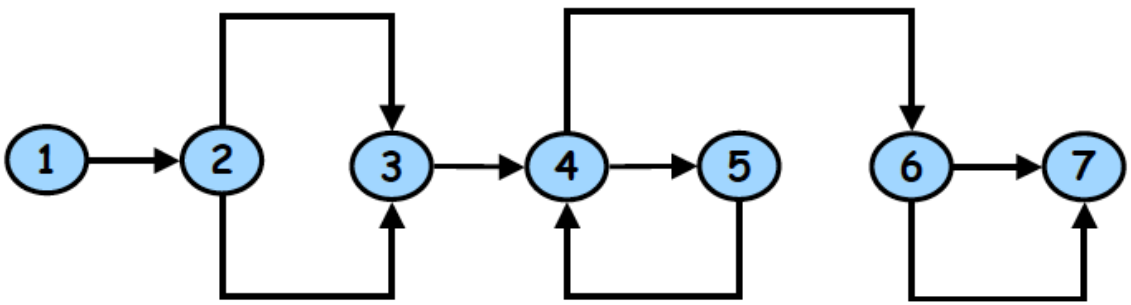
1. Initial nodes
2. Decisions: where branches (due to if-else/switch) occur and program diverge
3. Junctions: where branches merge
4. Process Block: sequence of statements
5. Final nodes

A path starts from an initial nodes, passes from decisions, junctions and process blocks and ends in a final node. Tests are intended to cover the graph.

Control Flowgraph

Exponentiation Algorithm

```
1   scanf("%d %d",&x, &y);
2   if (y < 0)
        pow = -y;
    else
        pow = y;
3   z = 1.0;
4   while (pow != 0) {
        z = z * x;
        pow = pow - 1;
5       }
6   if (y < 0)
        z = 1.0 / z;
7   printf ("%f",z);
```

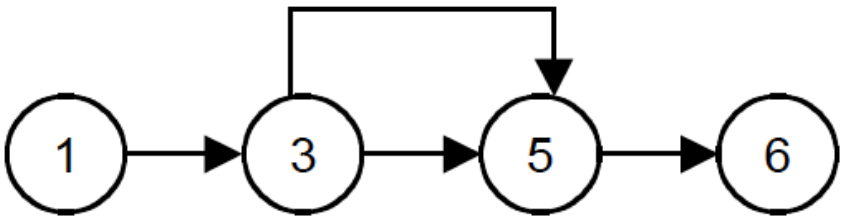


Ref: Spiros Mancoridis' course

Control Flowgraph

The Flowgraph for ABS

```
/* ABS
   This program function returns the absolute value of the integer
   passed to the function as a parameter.
    INPUT: An integer.
    OUTPUT: The absolute value if the input integer.
*/
1          int ABS(int x)
2          {
3          if (x < 0)
4              x = -x;
5          return x;
6          }
```

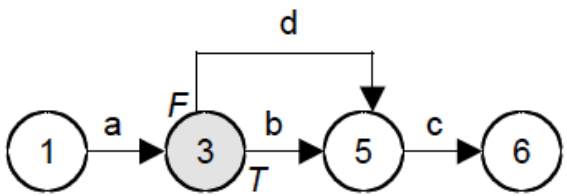


Ref: Spiros Mancoridis' course

Control Flowgraph

Test Cases to Satisfy **Statement Testing** Coverage for ABS

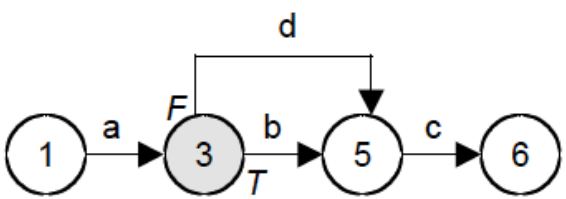
PATHS	PROCESS LINKS				TEST CASES	
	a	b	c	d	INPUT	OUTPUT
abc	√	√	√		A Negative Integer, x	-x
adc	√		√	√	A Positive Integer, x	x



Control Flowgraph

Test Cases to Satisfy **Branch Testing** Coverage for ABS

PATHS	DECISIONS	TEST CASES	
		INPUT	OUTPUT
abc	T	A Negative Integer, x	-x
adc	F	A Positive Integer, x	x



Control Flowgraph

Look for paths within a control flowgraph of a program

Complete Testing

- Exercise every path from entry to exit.
- Exercise every statement at least once.
- Exercise every branch at least once.

Ornek

```
1   public static void computeStats (int [ ] numbers){
2      int length = numbers.length;
3      double med, var, sd, mean, sum, varsum;
4      sum = 0;
5      for (int i = 0; i < length; i++){
6         sum += numbers [ i ];
7      }
8      med   = numbers [length / 2];
9      mean = sum / (double) length;
10     varsum = 0;
11
12     for (int i = 0; i < length; i++){
13        varsum = varsum  + ((numbers [ I ] - mean) * (numbers [ I ] - mean));
14     }
15     var = varsum / ( length - 1.0 );
16     sd  = Math.sqrt ( var );
17
18     System.out.println ("mean:                    " + mean);
19     System.out.println ("median:                  " + med);
20     System.out.println ("variance:                " + var);
21     System.out.println ("standard deviation: " + sd);
22  }
```

Ornek

```
1   public static void computeStats (int [ ] numbers){
2      int length = numbers.length;
3      double med, var, sd, mean, sum, varsum;
4      sum = 0;
5      for (int i = 0; i < length; i++){
6         sum += numbers [ i ];
7      }
8      med   = numbers [length / 2];
9      mean = sum / (double) length;
10     varsum = 0;
11
12     for (int i = 0; i < length; i++){
13        varsum = varsum  + ((numbers [ I ] - mean) * (numbers [ I ] - mean));
14     }
15     var = varsum / ( length - 1.0 );
16     sd  = Math.sqrt ( var );
17
18     System.out.println ("mean:                    " + mean);
19     System.out.println ("median:                  " + med);
20     System.out.println ("variance:                " + var);
21     System.out.println ("standard deviation: " + sd);
22  }
```

$$1 \rightarrow 5 \nearrow\!\hookleftarrow 7 \rightarrow 12 \nearrow\!\hookleftarrow 14 \rightarrow 22$$

Quiz

Quesitons

- Give an example of a defect who is to be fixed immediately(high pirority), but has a little impact on functionality (low severity)?
- What is a good test?
- What is the simplest test process?
- What is the effect of incomplete specifications?
- How testing can guarantee the absence of a bug?

Quiz

Answers

- Misspelled Logo
- A good test has a high probability of finding error.
- (1) run the test, (2) observe the actual outcome, (3) compare it to the expected outcome
- You can do what you shouldn't do, you can ignore what you should have done
- Impossible to guarantee the absence of a bug.

JUnit

- Create a Java Project (Project Name : Testing101)
- Right click on the project → New → Source Folder → test
- Right click in the test → New → JUnit Test Case
  - Name it MathTest
  - select all stubs, setup() etc.

Testing101: MathTest.java

```
1  import static org.junit.Assert.*;
2  import org.junit.After;
3  import org.junit.AfterClass;
4  import org.junit.Before;
5  import org.junit.BeforeClass;
6  import org.junit.Test;
7  public class MathTest {
8      @BeforeClass
9      public static void setUpBeforeClass() throws Exception {
10     }
11
12     @AfterClass
13     public static void tearDownAfterClass() throws Exception {
14     }
15
16     @Before
17     public void setUp() throws Exception {
18     }
19
20     @After
21     public void tearDown() throws Exception {
22     }
23
24     @Test
25     public void test() {
26         fail("Not yet implemented");
27     }
28 }
```

Annotations

- @Test annotation specifies that method is the test method.
- @Test(timeout=1000) annotation specifies that method will be failed if it takes longer than 1000 milliseconds (1 second).
- @BeforeClass annotation specifies that method will be invoked only once, before starting all the tests.
- @Before annotation specifies that method will be invoked before each test.
- @After annotation specifies that method will be invoked after each test.
- @AfterClass annotation specifies that method will be invoked only once, after finishing all the tests.

Ref: javatpoint - junit-tutorial

Annotations

- void assertEquals(boolean expected,boolean actual):
  - checks that two primitives/objects are equal. It is overloaded.
- void assertTrue(boolean condition):
  - checks that a condition is true.
- void assertFalse(boolean condition):
  - checks that a condition is false.
- void assertNull(Object obj):
  - checks that object is null.
- void assertNotNull(Object obj):
  - checks that object is not null.

Ref: javatpoint - junit-tutorial