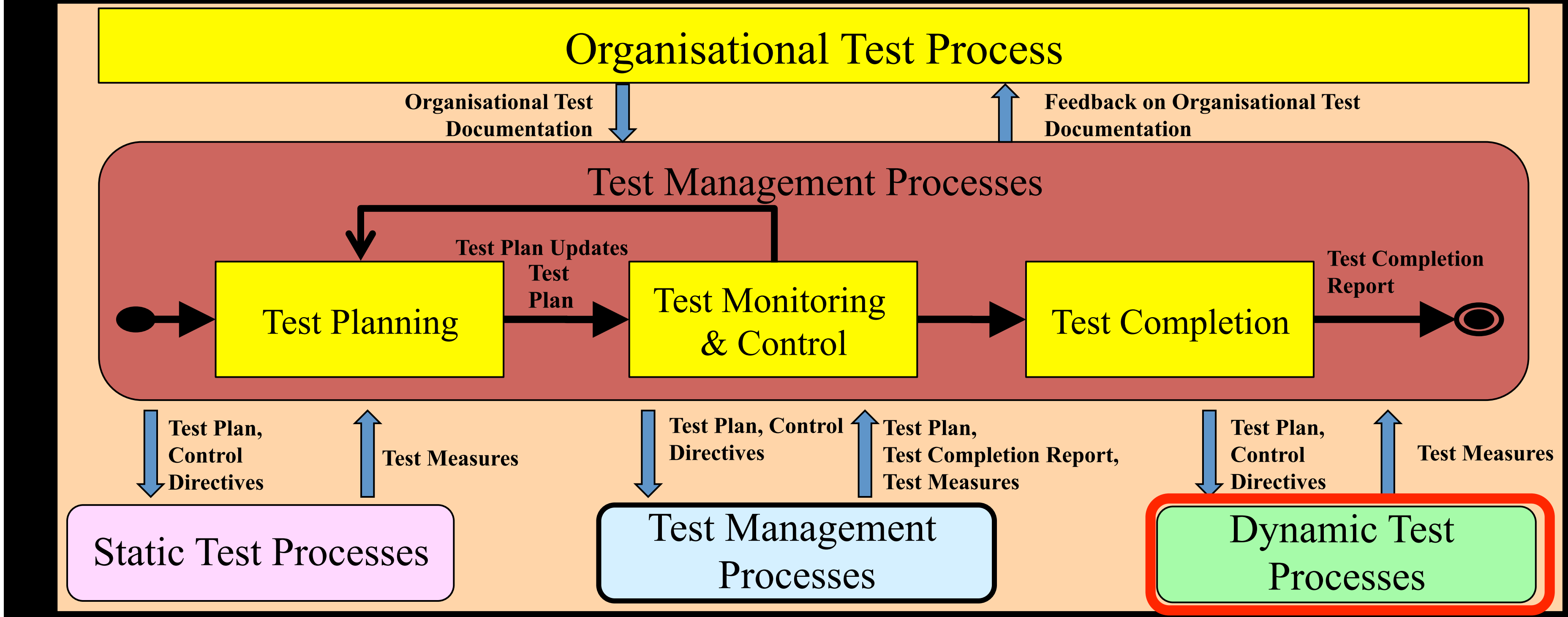
Testing Techniques

Jesper Andersson

Linnæus University

Test Management Processes – Functionality and Type



Organisational Test Process

Organisational Test Documentation

Feedback on Organisational Test Documentation

Test Management Processes

Test Plan Updates

Test Plan

Test Planning

Test Monitoring & Control

Test Completion

Test Completion Report

Test Plan, Control Directives

Test Measures

Test Plan, Control Directives

Test Plan, Test Completion Report, Test Measures

Test Plan, Control Directives

Test Measures

Static Test Processes

Test Management Processes

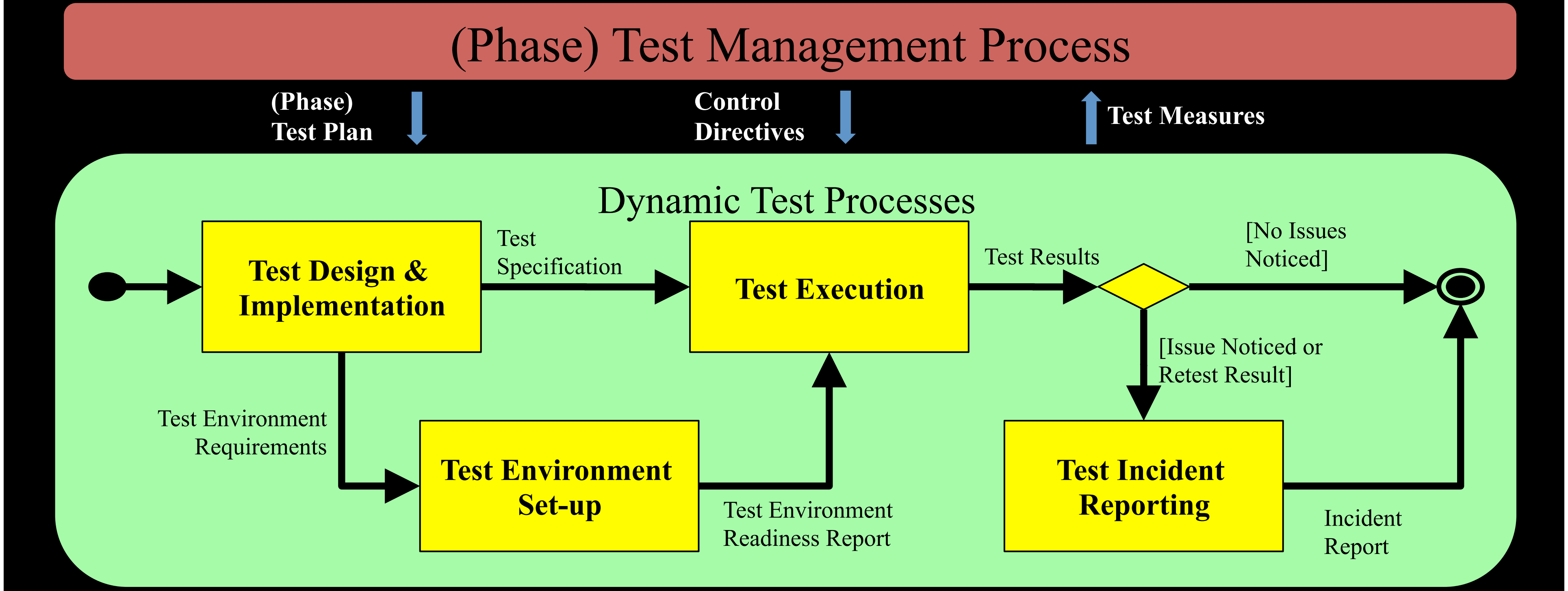
Dynamic Test Processes

Linnæus University
Sweden

Planning at different levels – Test Plan

- ✓ Plan *iteratively* and *incrementally*.
- ✓ Comparable to project planning
- ✓ Test plan → long-term
  - Why do we test? Test objectives
  - What should we test? Test objects
  - How should we test? Test type and Test technique
  - Test environment

Dynamic Test Processes



(Phase) Test Management Process

(Phase) Test Plan

Control Directives

Test Measures

Dynamic Test Processes

Test Design & Implementation

Test Specification

Test Execution

Test Results

[No Issues Noticed]

[Issue Noticed or Retest Result]

Test Environment Requirements

Test Environment Set-up

Test Environment Readiness Report

Test Incident Reporting

Incident Report

Planning at different levels – Iteration

- ✓ In each iteration ➔ short-term
  - – Plan based on iteration goals. What can you do in this iteration?
    - Early stages, for instance elaboration, some things
    - During construction, much more
  - – Test design
  - – Test refinement, including test plan!!!

Test planning – Create a Work Breakdown Structure

- ✓ A hierarchical representation of test activities and tasks
- ✓ Start with the most important testing milestones.
- ✓ Refines milestones into activities and tasks with their own mile-stones.
- ✓ Managing the WBS is a continuous process, its an ”active document” that will change during the course of the testing project.

Testing requires Development Resources!

- ✓ Include this in your planning!
- ✓ Setup test environment
  - – Test management tools
    - • Manage test cases, specification and documentation
    - • Manage test reports
  - – Drivers - Mockups
  - – Emulators/simulators/hardware in the loop
- ✓ Develop test-scripts
  - – Test-data and expected output

Testing in an Iteration

- ✓ In each iteration you must plan for
  - Preparation
  - Execution
  - Analysis
- ✓ Caution! These activities require time!

How?

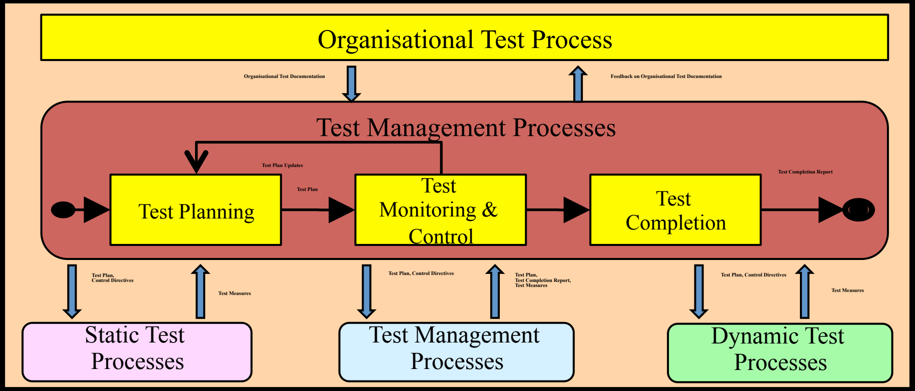
What?

Why?

Linnæus University
Sweden

Test Object, Objectives & Techniques

1. Identify - Test Object ➔ What!
2. Define - Test Objective ➔Why!
3. Select - Test Technique ➔How!



Organisational Test Process
Organisational Test Documentation
Feedback on Organisational Test Documentation
Test Management Processes
Test Plan Updates
Test Plan
Test Completion Report
Test Planning
Test Monitoring & Control
Test Completion
Test Plan, Control Directives
Test Measures
Test Plan, Control Directives
Test Plan, Test Completion Report, Test Measures
Test Plan, Control Directives
Test Measures
Static Test Processes
Test Management Processes
Dynamic Test Processes

Test Suite!

Linnæus University
Sweden

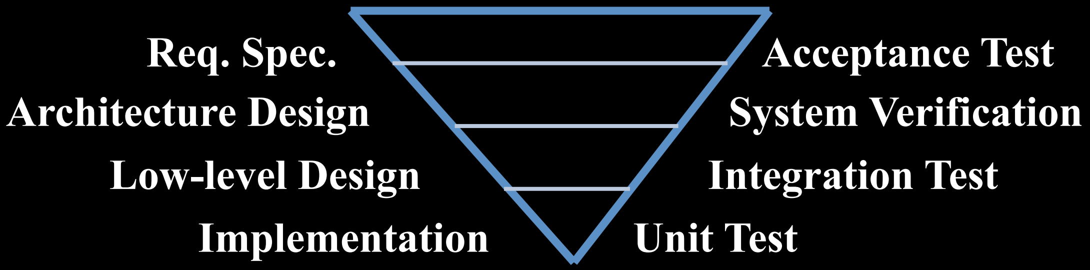
Test Suite

- ✓ A collection of *test cases* that tests a software system
- ✓ A test suite can contain groups of test cases for different object, objectives, and techniques, and information on the system configuration to be used during testing.
- ✓ A group of test cases also contain information on how to setup the test environment.

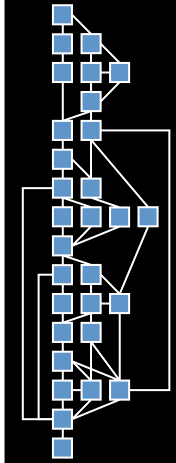
Test Levels/Phases

✓ Initial development/evolution

- Basic/Unit Test
- Function/Integration Test
- System Verification
- Acceptance Test



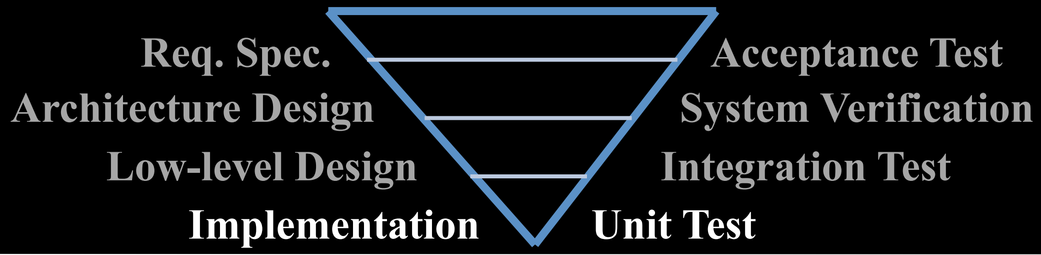
Unit Test (UT)



- ✓ UT, verify the smallest testable pieces in the system.
- ✓ The ***Test Object*** is the unit (class or method)
- ✓ The ***Test Objective*** is to detect *defects* in the code.
- ✓ Dynamic Testing
  - UT often use ”white box testing”, i.e. mostly performed by unit designer, coder, with access to the source code.
  - The *code coverage* is measured.
- ✓ Static testing
  - Code standards
  - Language

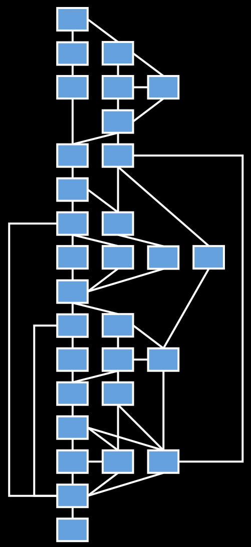
Unit Test

- ✓ Use tools for the validation.
  - Test Case generators and Execution support
  - Code coverage tools
  - Emulators
  - Simulators
- ✓ Test specifications are based on unit and design specifications.



Unit Testing – Test Techniques

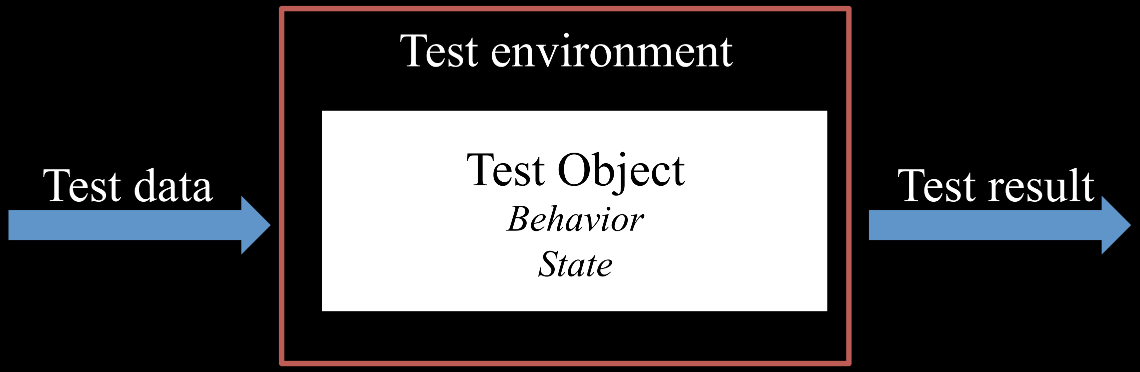
- ✓ Objective - Defect testing
- ✓ Challenge
  - – Cover as much code as possible
  - – Using a minimal set of test cases
- ✓ Techniques/Strategies
  - – Black-box, module viewed as a function
  - – White-box, structural coverage techniques



Controllability and Observability

- ✓ **Controllability**
  - Controlled execution to a point we would like to test
- ✓ **Observability**
  - Observe the application’s actual behavior
- ✓ Goals for a test case
  - Reach a fault
  - Produce an error
  - Make the error visible as a failure

Controllability and Observability



Test environment

Test data

Test Object

Behavior

State

Test result

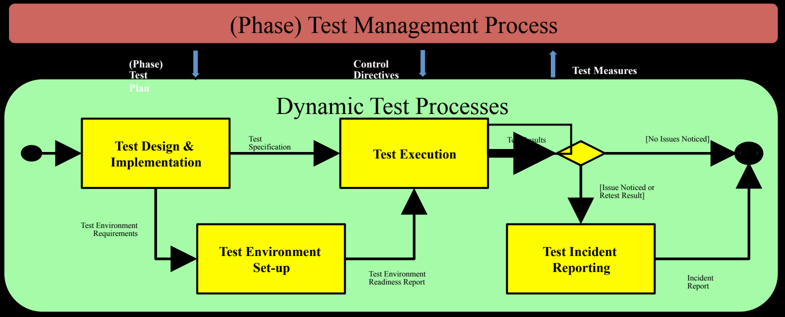
What can we control and observe?

Technique - Structural Testing

- ✓ Code level – Analyze the unit
  - signature,
  - specification,
  - and implementation
- ✓ Based on structure of implementation derive test-cases (call-graph)
- ✓ Generate test-data to “cover” code

Structural Testing – Process

1. Select a *coverage criteria*
2. Generate Control flow graphs
3. Instrument the code
4. Derive test cases
5. Execute tests
6. Analyze coverage
    1. Repeat 4
    2. Finish

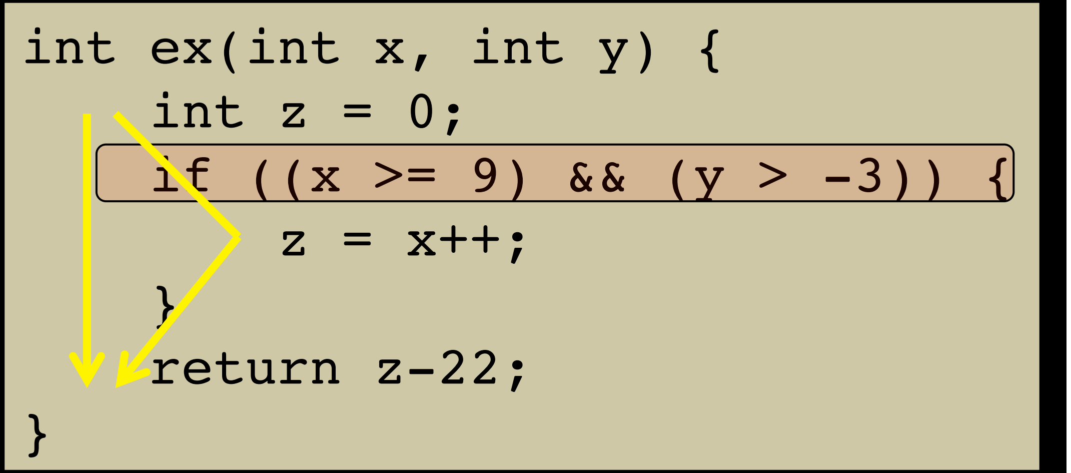


Linnæus University
Sweden

White-box testing – Coverage Criteria

- ✓ All-Paths (Infeasible)
- ✓ Statement coverage
- ✓ Branch-coverage
- ✓ Decision Coverage
- ✓ Entry-Exit coverage

x >= 9	y > -3
T	T
T	F
F	T
F	F



```
int ex(int x, int y) {
    int z = 0;
    if ((x >= 9) && (y > -3)) {
        z = x++;
    }
    return z-22;
}
```

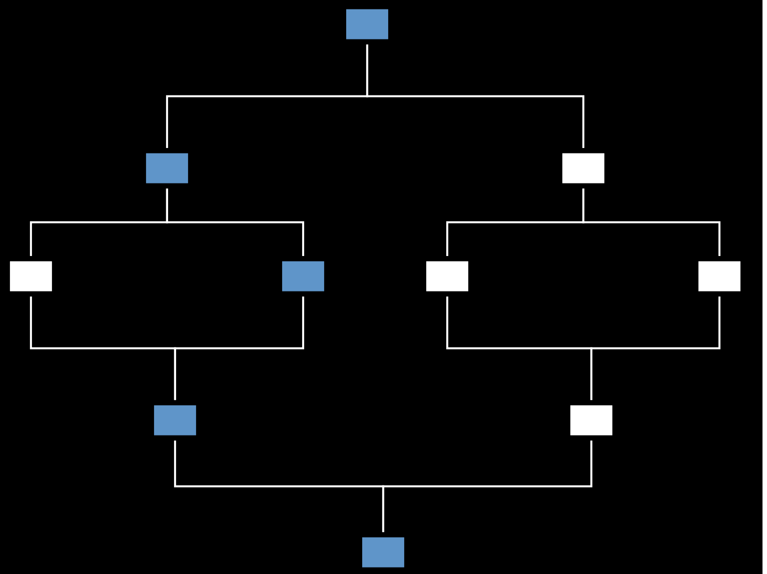
Coverage – Example

```
if …
   if …
   else …
else …
   if …
   else …
endif
```

Instruction 5/10, 50%

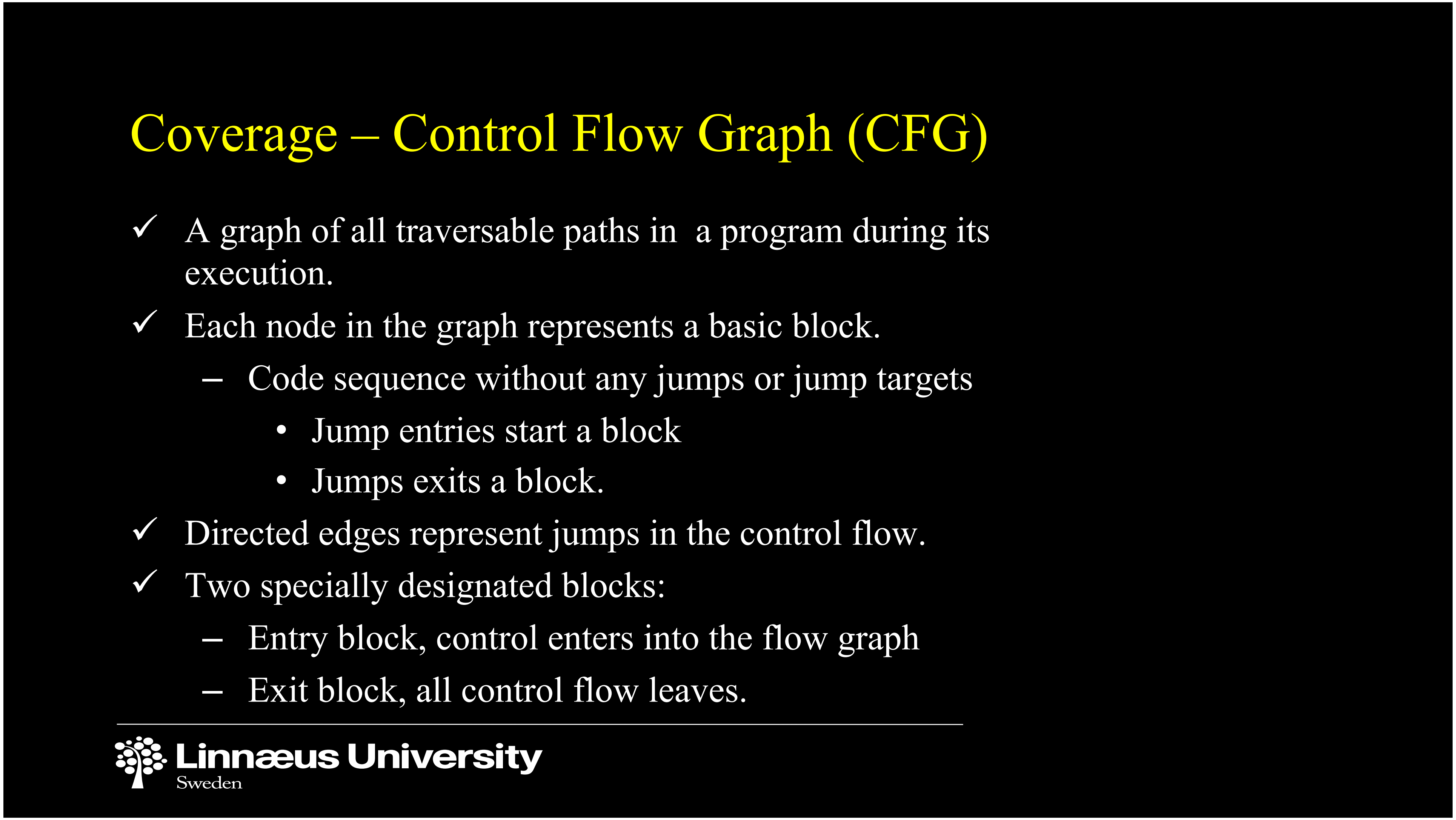
Branch 2/6, 33%

Path 1/4, 25%



Measure Coverage – Instrumentation

- ✓ Instrumentation is a technique which annotates a software object with extra functionality
  - – Instrumentation of source code
  - – Instrumentation of object code
- ✓ Instrumented code “removed” for other test types, when not needed.
- ✓ **Why?**
  - – Negative effects on for example performance
  - – Makes debugging more difficult

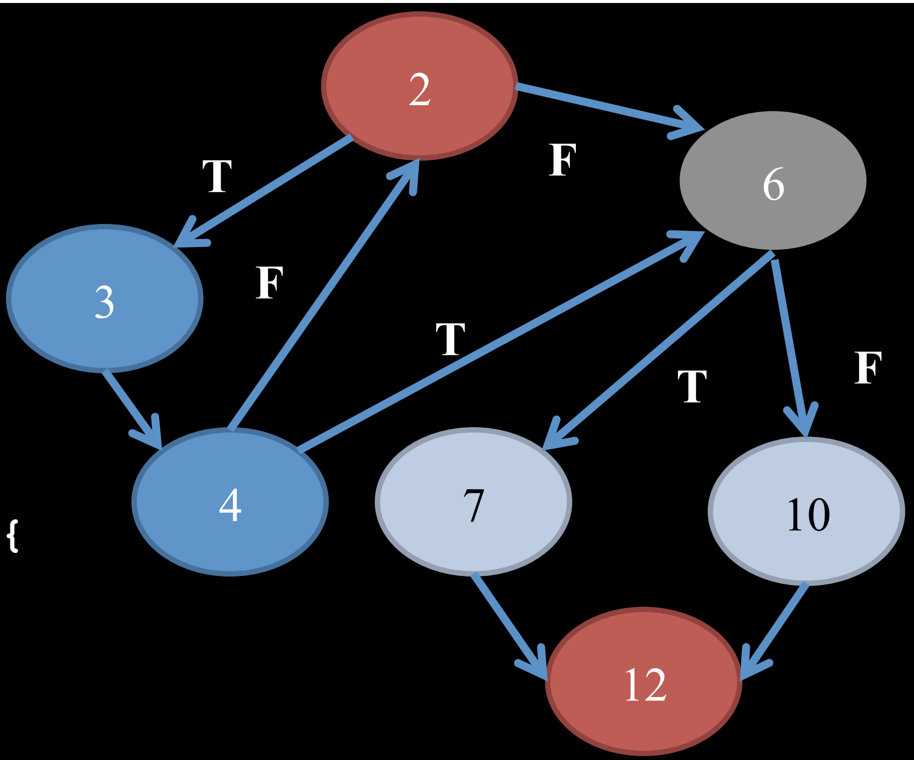


Coverage – Control Flow Graph (CFG)

- ✓ A graph of all traversable paths in a program during its execution.
- ✓ Each node in the graph represents a basic block.
  - – Code sequence without any jumps or jump targets
    - • Jump entries start a block
    - • Jumps exits a block.
- ✓ Directed edges represent jumps in the control flow.
- ✓ Two specially designated blocks:
  - – Entry block, control enters into the flow graph
  - – Exit block, all control flow leaves.

Call-graph

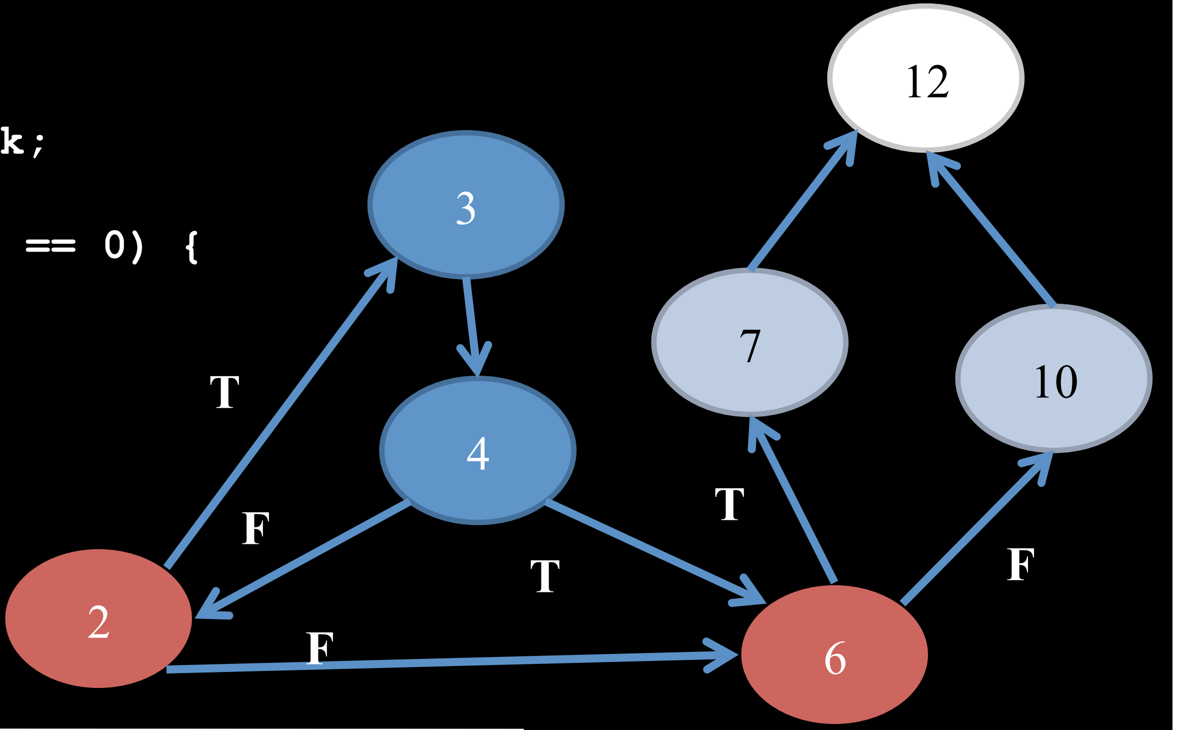
```
1  int p(int x, int y) {
2    while (x > 10 ) {
3        x = x - 10;
4        if (x == 10) break;
5    }
6    if ( y < 20 && x % 5 == 0) {
7        y = y+ 20;
8    }
9    else {
10       y = y - 20;
11   }
12   return 4*(x+y);
13}
```



Linnæus University
Sweden

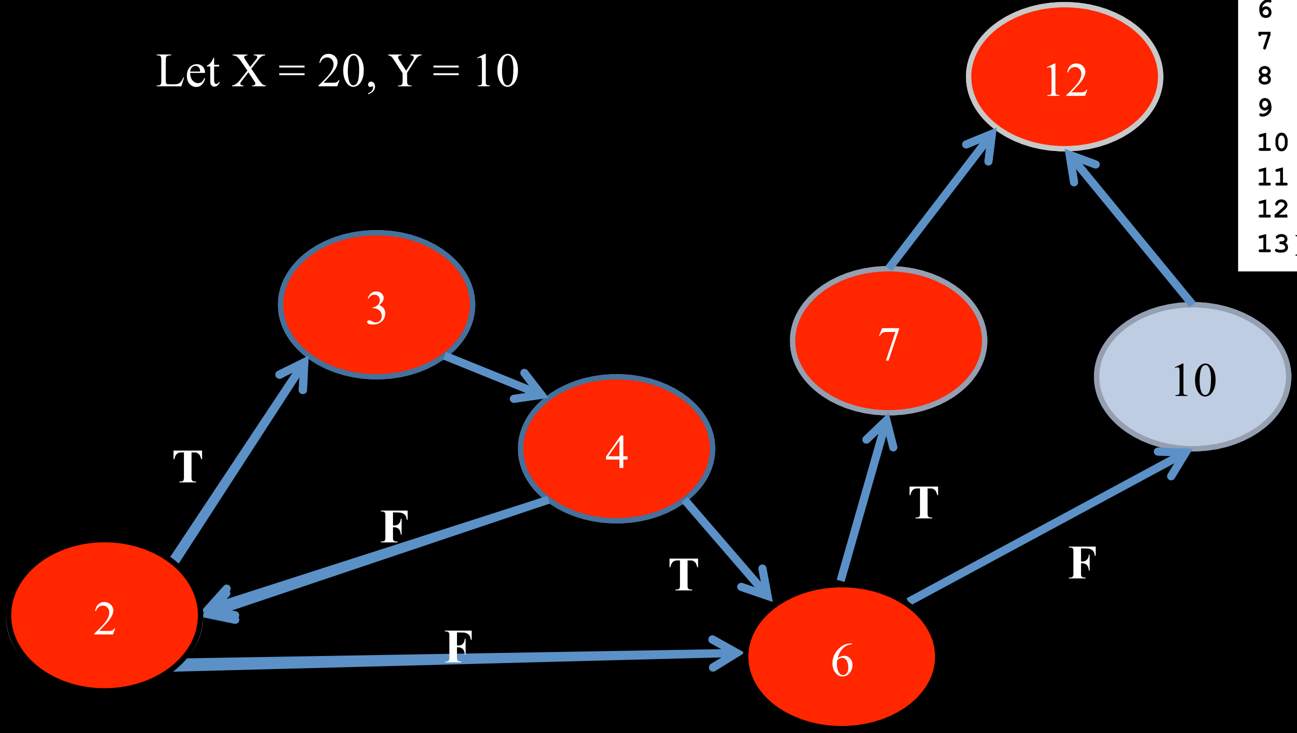
Example – CFG, Statement Coverage

```
1  int p(int x, int y) {
2    while (x > 10 ) {
3       x = x - 10;
4       if (x == 10) break;
5    }
6    if ( y < 20 && x % 5 == 0) {
7       y = y+ 20;
8    }
9    else {
10      y = y - 20;
11   }
12   return 4*(x+y);
13}
```



Test case 1: Statement Coverage

Let X = 20, Y = 10

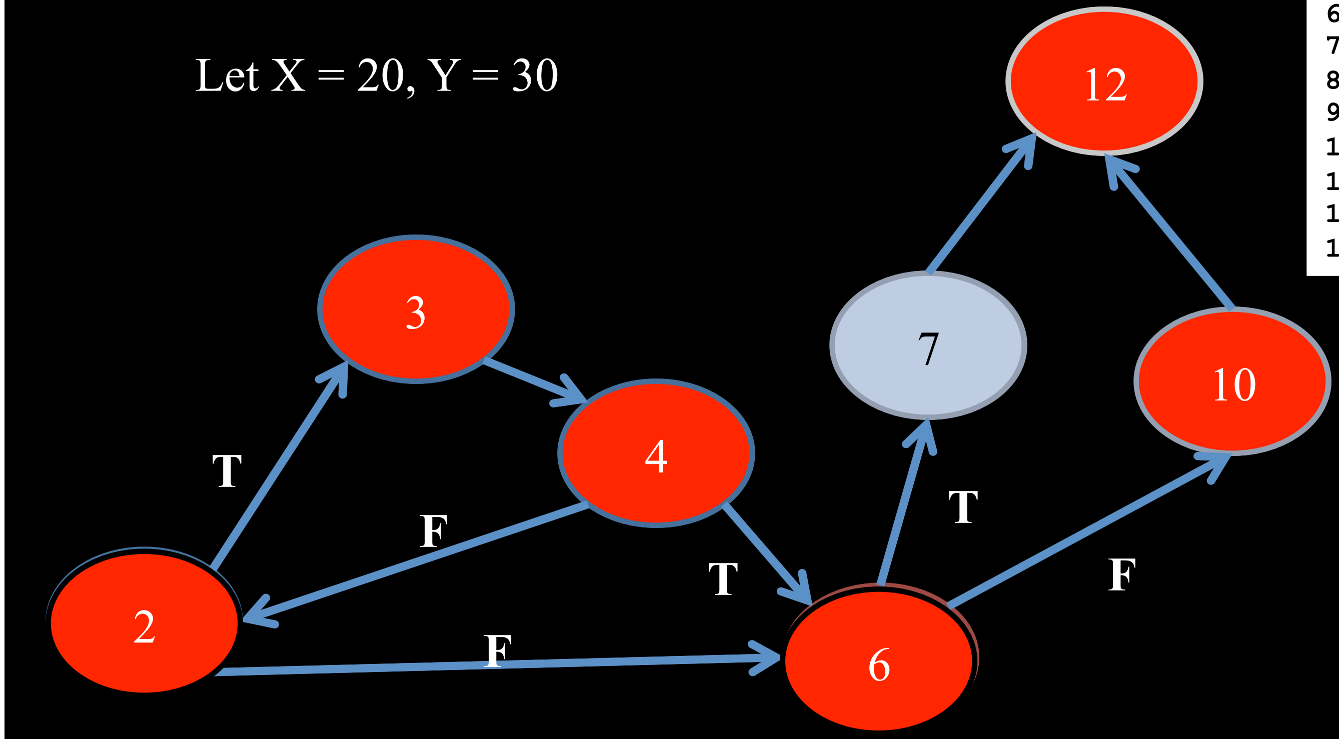


```
1  int p(int x, int y) {
2    while (x > 10 ) {
3       x = x - 10;
4       if (x == 10) break;
5    }
6    if ( y < 20 && x % 5 == 0) {
7       y = y+ 20;
8    }
9    else {
10      y = y - 20;
11   }
12   return 4*(x+y);
13}
```

Test case 2: Statement Coverage

Let X = 20, Y = 30

```
1 int p(int x, int y) {
2   while (x > 10 ) {
3       x = x - 10;
4       if (x == 10) break;
5   }
6   if ( y < 20 && x % 5 == 0) {
7       y = y+ 20;
8   }
9   else {
10      y = y - 20;
11  }
12  return 4*(x+y);
13}
```

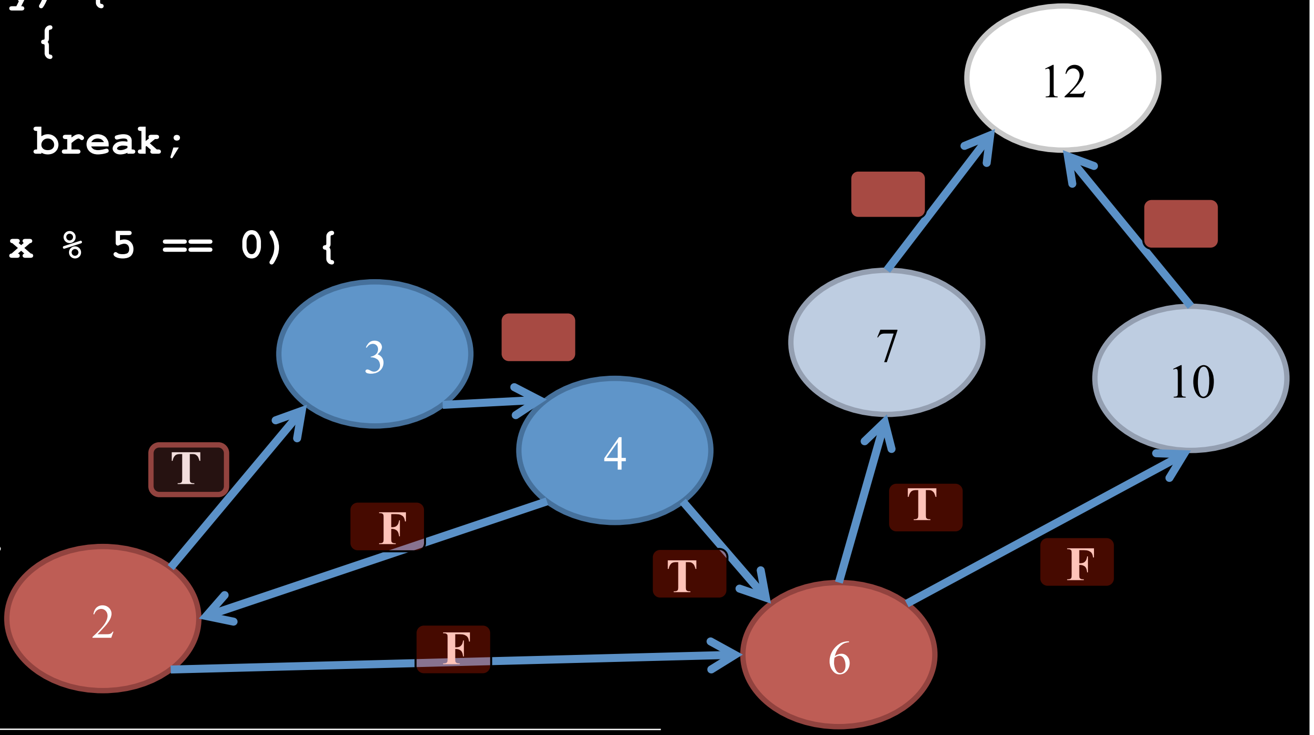


Test case 1, Let X = 20, Y = 10

Test case 2, Let X = 20, Y = 30

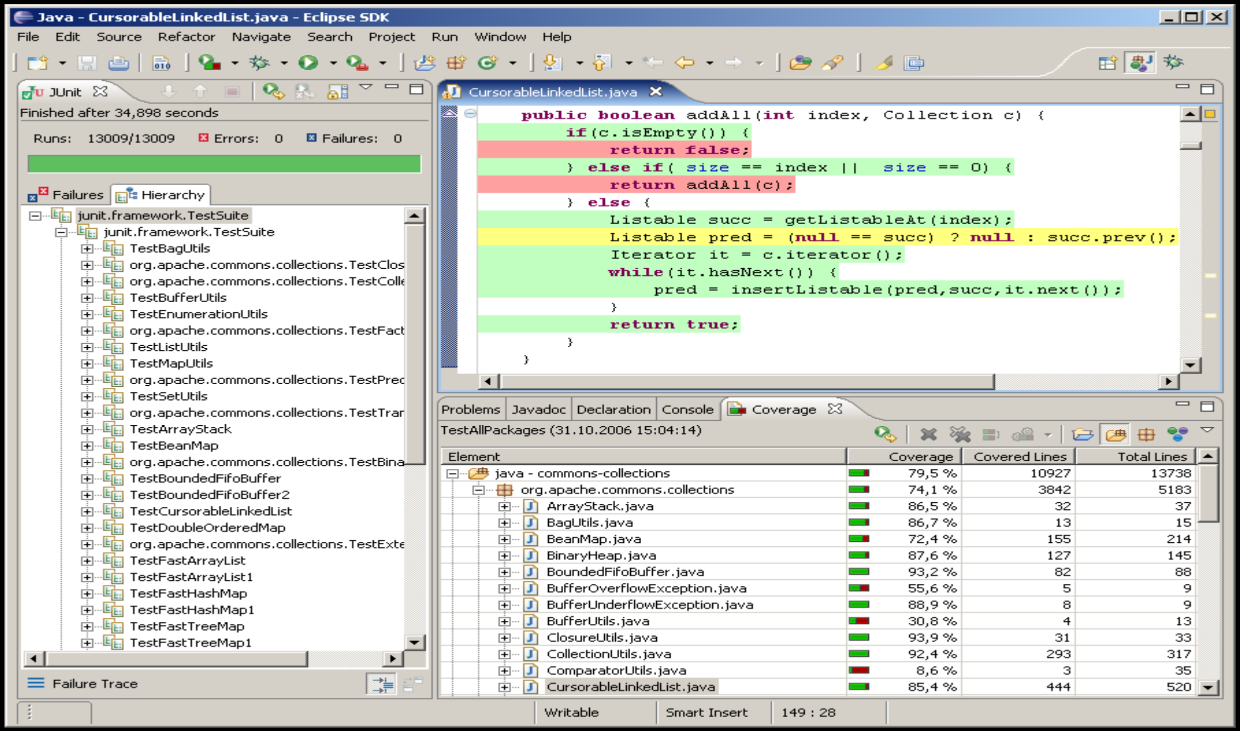
Example – CFG, Branch Coverage

```
1 int p(int x, int y) {
2   while (x > 10 ) {
3       x = x - 10;
4       if (x == 10) break;
5   }
6   if ( y < 20 && x % 5 == 0) {
7       y = y+ 20;
8   }
9   else {
10      y = y - 20;
11  }
12  return 4*(x+y);
13}
```



Linnæus University
Sweden

Coverage Tools – EclEmma



Finished after 34,898 seconds

Runs: 13009/13009 ☒ Errors: 0 ☒ Failures: 0

```
⊟ junit.framework.TestSuite
  ⊟ junit.framework.TestSuite
    ⊞ TestBagUtils
    ⊞ org.apache.commons.collections.TestClos
    ⊞ org.apache.commons.collections.TestColle
    ⊞ TestBufferUtils
    ⊞ TestEnumerationUtils
    ⊞ org.apache.commons.collections.TestFact
    ⊞ TestListUtils
    ⊞ TestMapUtils
    ⊞ org.apache.commons.collections.TestPred
    ⊞ TestSetUtils
    ⊞ org.apache.commons.collections.TestTrar
    ⊞ TestArrayStack
    ⊞ TestBeanMap
    ⊞ org.apache.commons.collections.TestBina
    ⊞ TestBoundedFifoBuffer
    ⊞ TestBoundedFifoBuffer2
    ⊞ TestCursorableLinkedList
    ⊞ TestDoubleOrderedMap
    ⊞ org.apache.commons.collections.TestExte
    ⊞ TestFastArrayList
    ⊞ TestFastArrayList1
    ⊞ TestFastHashMap
    ⊞ TestFastHashMap1
    ⊞ TestFastTreeMap
    ⊞ TestFastTreeMap1
```

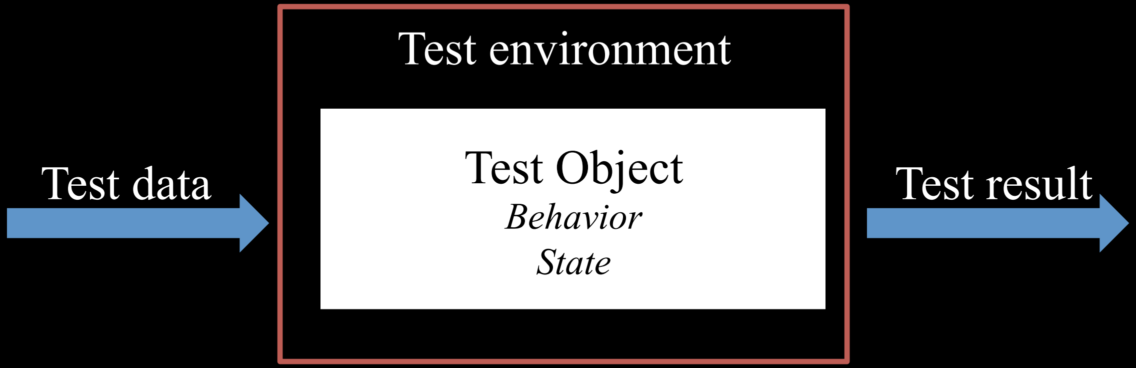
```java
    public boolean addAll(int index, Collection c) {
        if(c.isEmpty()) {
            return false;
        } else if( size == index ||  size == 0) {
            return addAll(c);
        } else {
            Listable succ = getListableAt(index);
            Listable pred = (null == succ) ? null : succ.prev();
            Iterator it = c.iterator();
            while(it.hasNext()) {
                pred = insertListable(pred,succ,it.next());
            }
            return true;
        }
    }
```

TestAllPackages (31.10.2006 15:04:14)

Element	Coverage	Covered Lines	Total Lines
⊟ java - commons-collections	79,5 %	10927	13738
⊟ org.apache.commons.collections	74,1 %	3842	5183
⊞ ArrayStack.java	86,5 %	32	37
⊞ BagUtils.java	86,7 %	13	15
⊞ BeanMap.java	72,4 %	155	214
⊞ BinaryHeap.java	87,6 %	127	145
⊞ BoundedFifoBuffer.java	93,2 %	82	88
⊞ BufferOverflowException.java	55,6 %	5	9
⊞ BufferUnderflowException.java	88,9 %	8	9
⊞ BufferUtils.java	30,8 %	4	13
⊞ ClosureUtils.java	93,9 %	31	33
⊞ CollectionUtils.java	92,4 %	293	317
⊞ ComparatorUtils.java	8,6 %	3	35
⊞ CursorableLinkedList.java	85,4 %	444	520

Writable | Smart Insert | 149 : 28

Controllability and Observability



Test environment

Test data

Test Object

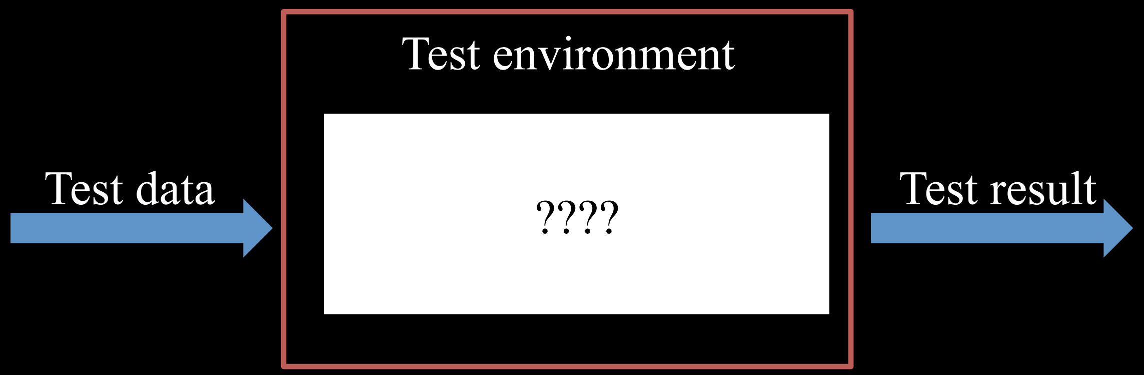
Behavior

State

Test result

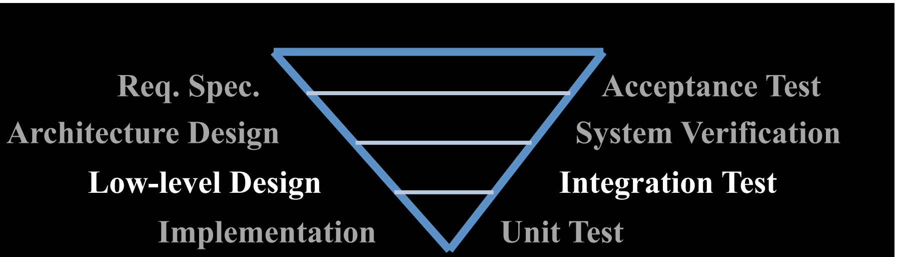
Now we can we control and observe?

New Problem – no access to source code!



Controllability back on square 1

Example – Integration testing



Req. Spec.

Architecture Design

Low-level Design

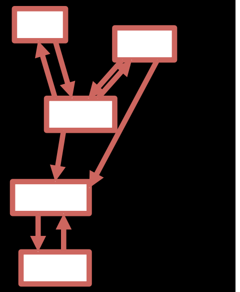
Implementation

Acceptance Test

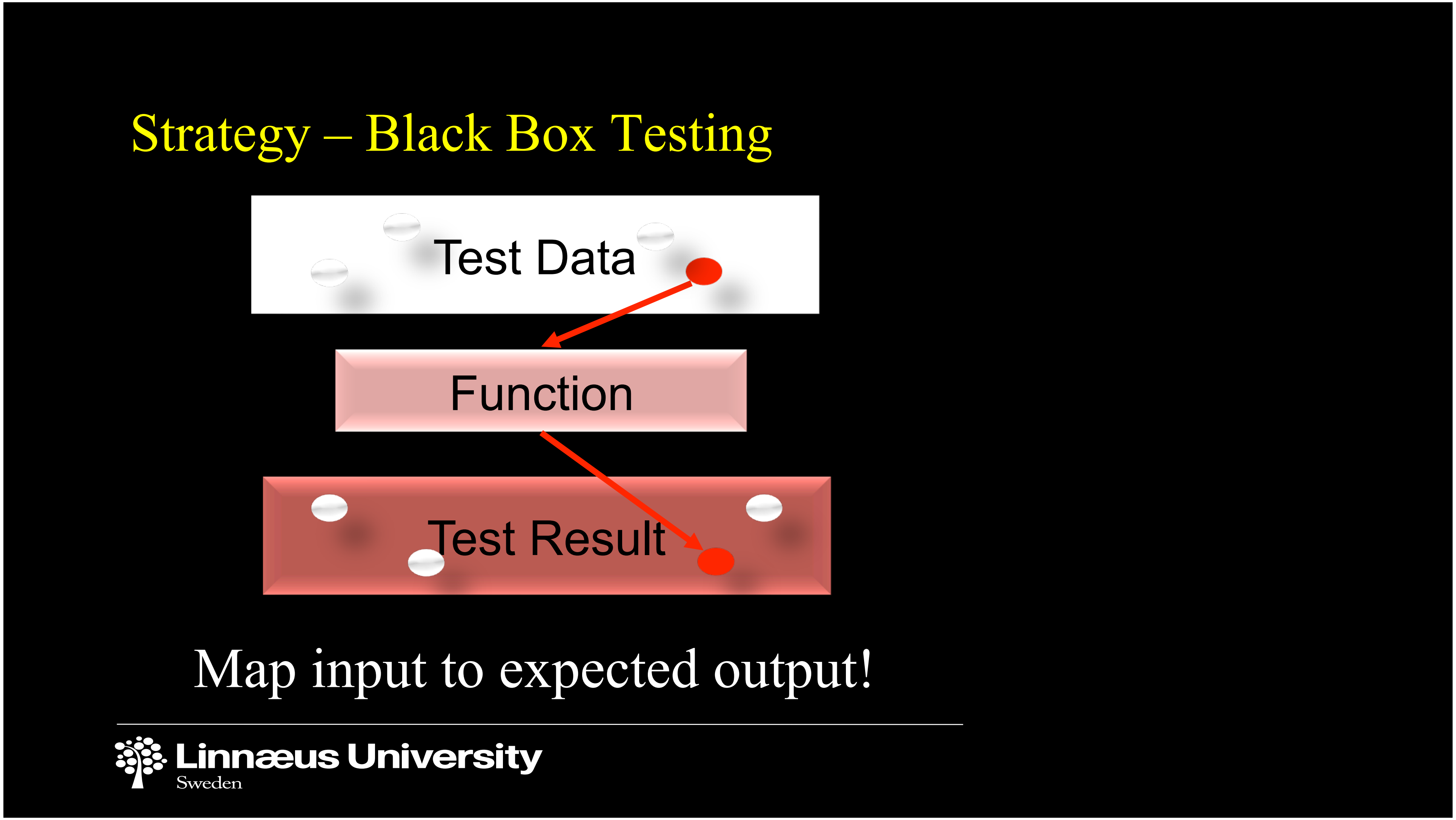
System Verification

Integration Test

Unit Test



- ✓ The purpose of Integration test is to integrate the software into large pieces.
  - – Test objectives, *functionality* or *quality*
  - – Integration test are mostly executed in *target* environment.
- ✓ Strategies include ”white box” and ”black box” testing, i.e. sometimes the integration tester have access to the source code and sometimes not
- ✓ Test cases could be based on **use cases**(functional specifications).



Strategy – Black Box Testing

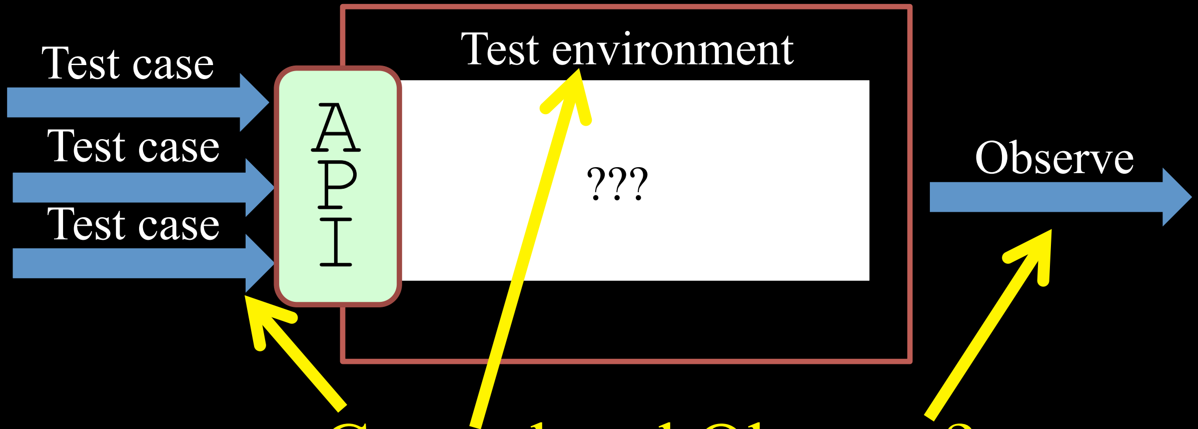
Test Data

Function

Test Result

Map input to expected output!

Example – API testing



Control and Observe?

Exercise the API

Black Box Testing – Category Partitioning

- ✓ API - Testing technique
- ✓ Analyze the function under test
  - Signature
  - Specification
- ✓ Find partitions of *equivalent inputs* and *outputs*. Removes redundant tests and minimizes test suite size

- ✓ Look at boundary values!
- ✓ Tester does “not know” about internals!

Category Partitioning Steps

1. Decompose specifications into units
2. Identify parameters and environment state
3. Find categories in state and parameters
4. Partition each category into choices.
5. Write test specification
6. Define test data
7. Implement test case

Example: **Sorting**
Specification:
    Input: Variable length array of arbitrary type
    Output: Permutation of input, sorted.
        Minimum value
        Maximum value
Parameters: Array

Sorting Example

- ✓ Categories:
  - – Array’s size
  - – Type of elements
  - – Maximum value
  - – Minimum value
  - – Position of Max and Min values
- ✓ Choices:
  - – Size: { 0, 1, 2 .. 100, 100 .. INF }
  - – Type: { Integer, Character, Array, Record, Ref}
  - – Max: . . .

Example: **Sorting**
Specification:
    Input: Variable length array of arbitrary type
    Output: Permutation of input, sorted.
        Minimum value
        Maximum value
Parameters: Array

Test Data

- ✓ [Size, PosMax, PosMin, MAX=MIN]
- ✓ [A[0], U,U,F]
- ✓ [A[1], HEAD,HEAD,T]
- ✓ [A[2],HEAD,TAIL,F]
- ✓ [A[2],TAIL,HEAD,F]
- ✓ [A[2],HEAD,TAIL,T]
- ✓ [A[2],TAIL,HEAD,T]
- ✓ …
- ✓ …

```
for i ← 1 to length(A) - 1
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    end while
end for
```

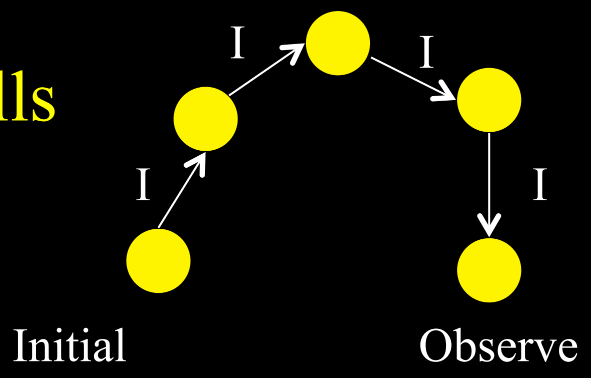
This is what we exercise!

Test Case

- ✓ [A[2],HEAD,TAIL,F]

- ✓ Test Fixture (Test sequencing)
  - Array that matches test data pattern
  - Invoke test object with (possibly with data)
  - Compare result (Observe A’s state)

Another approach – Sequences of calls



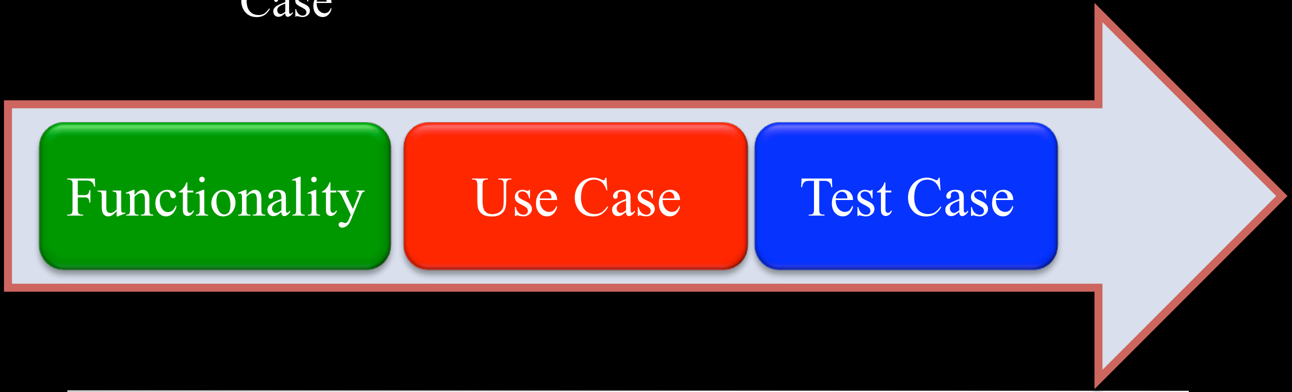
- ✓ For APIs (or any integration type)
  - Test that sequences of invocations work
  - Test for functionality

1. Tested, working, units
2. Now we test if the work together!

Linnæus University
Sweden

Integration Testing (API) for Functionality

- ✓ At the System or Subsystem level
- ✓ Tests
  - Functionality → Use Case → Test Case
  - Quality → Quality Attribute Scenario → Test Case



Deriving Test Cases from Use Cases

Acknowledgement: Chris Collins, *www.christophertcollins.com/presentations/Understanding**UseCases**.ppt*

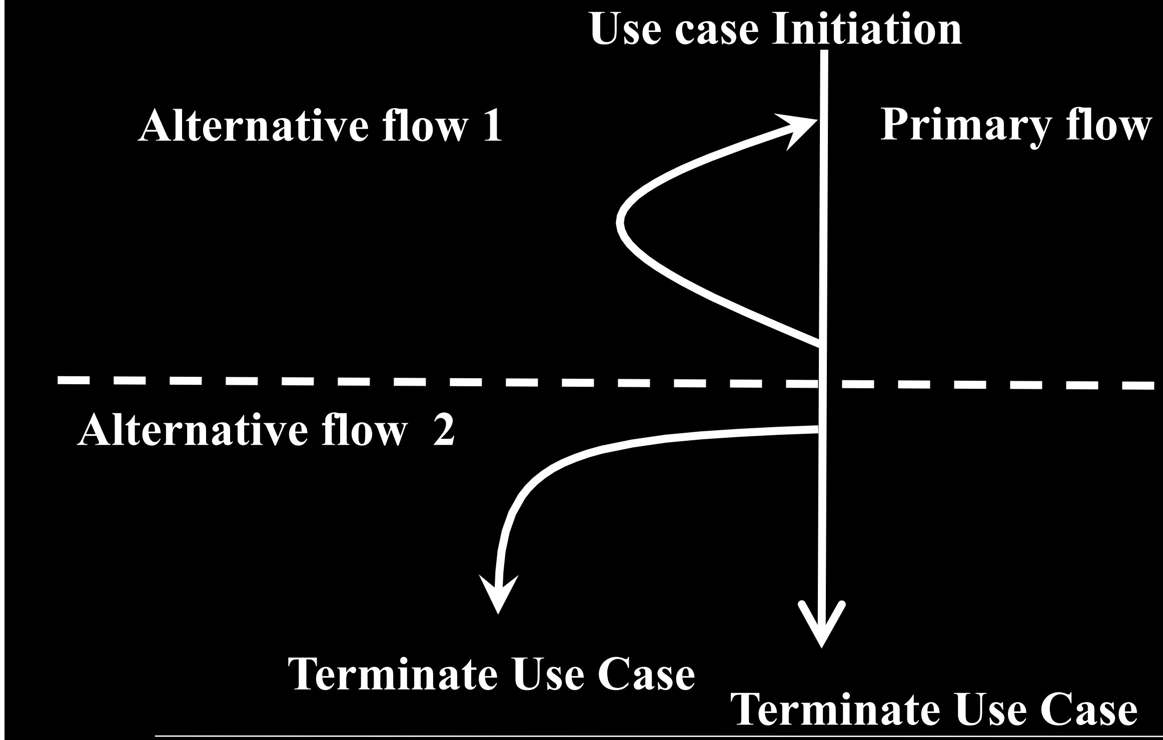
- ✓ Similar to CFG based generation aim for coverage
- ✓ We don't now the structure of the code but the structure of the flow!
- ✓ Four Step Process
  - **Step 1**: Identify the **paths** through the Use Case – {Scenario}
  - **Step 2**: Identify the Test Cases – One or more per scenario
  - **Step 3**: Identify the Test Choices
  - **Step 4**: Add Data Values to Complete the Test Case

Example – Unlock Screen Use Case

Unlock Screen

- ✓ Basic Flow
  1. The user selects unlock command
  2. System brings up logon screen
  3. The user enters valid user Id and password
  4. The user selects to logon
  5. The system unlocks the screen
- ✓ Alternate Flow 1 – Invalid Password
  - 3a. The user enters valid user Id and invalid password
  - 4a. The user selects to logon to the system
  - 5a. The system indicates error logging on and returns to logon screen
- ✓ Alternate Flow 2 – Cancel
  - 3b. The user select to cancel
  - 4b. The system does not log user on and returns locked screen

Step 1: Scenarios for a Use Case



Step 1: Scenario Matrix – Identifying Scenarios

Scenario Number	Originating Flow	Alternate Flow	Next Alternate	Next Alternate
1	Basic flow			
2	Basic flow	Alternate flow 1		
3	Basic flow	Alternate flow 1	Alternate flow 2	
4	Basic flow	Alternate flow 2		

How can we exercise these combinations of flows?
Controllability!

Example – Unlock Screen Use Case

Unlock Screen

- ✓ Basic Flow
  1. The user selects unlock command
  2. System brings up logon screen
  3. The user enters valid user Id and password
  4. The user selects to logon
  5. The system unlocks the screen
- ✓ Alternate Flow 1 – Invalid Password
  - 3a. The user enters valid user Id and invalid password
  - 4a. The user selects to logon to the system
  - 5a. The system indicates error logging on and returns to logon screen
- ✓ Alternate Flow 2 – Cancel
  - 3a. The user select to cancel
  - 4a. The system does not log user on and returns locked screen

Step 2: Test Cases – Choices

Id	Scenario	Unlock Screen Cmd	UserId	Pwd	Logon Cmd	Expected Result	Actual Result
1	Scenario 1						
2	Scenario 2						
3	Scenario 4						

Step 3: Test Conditions

Id	Scenario	Unlock Screen Cmd	UserId	Pwd	Logon Cmd	Expected Result	Actual Result
1	Scenario 1	Valid	Valid	Valid	Yes	Logon	
2	Scenario 2	Valid	Valid	Invalid	Yes	Failure Return to logon	
3	Scenario 4	Valid	N/A	N/A	No	Return to Screen Lock	

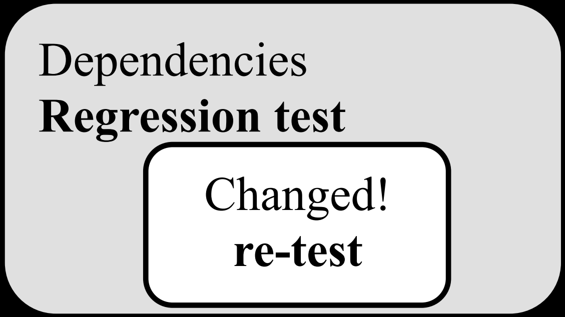
Step 4. Add Values Complete Test Case

Id	Scenario	Unlock Screen Cmd	UserId	Pwd	Logon Cmd	Expected Result	Actual Result
1	Scenario 1	Ctr-atl-del	ctc	abc	Ok Btn	Logon	Logged on
2	Scenario 2	Ctr-atl-del	ctc	abd	Ok Btn	Failure Return to logon	Returned to Logon
3	Scenario 4	Ctr-atl-del	N/A	N/A	Cancel Btn	Return to Screen Lock	Returned to Screen lock

Iterative development – Regression testing

- ✓ In each iteration you develop an “increment”
- ✓ Testing the increment is not sufficient, due to *interactions.*
- ✓ You must test all dependencies!

- ✓ Identify faults introduced during a modification of the system
- ✓ Verify that the modification have not effected old implementations



Dependencies

Regression test

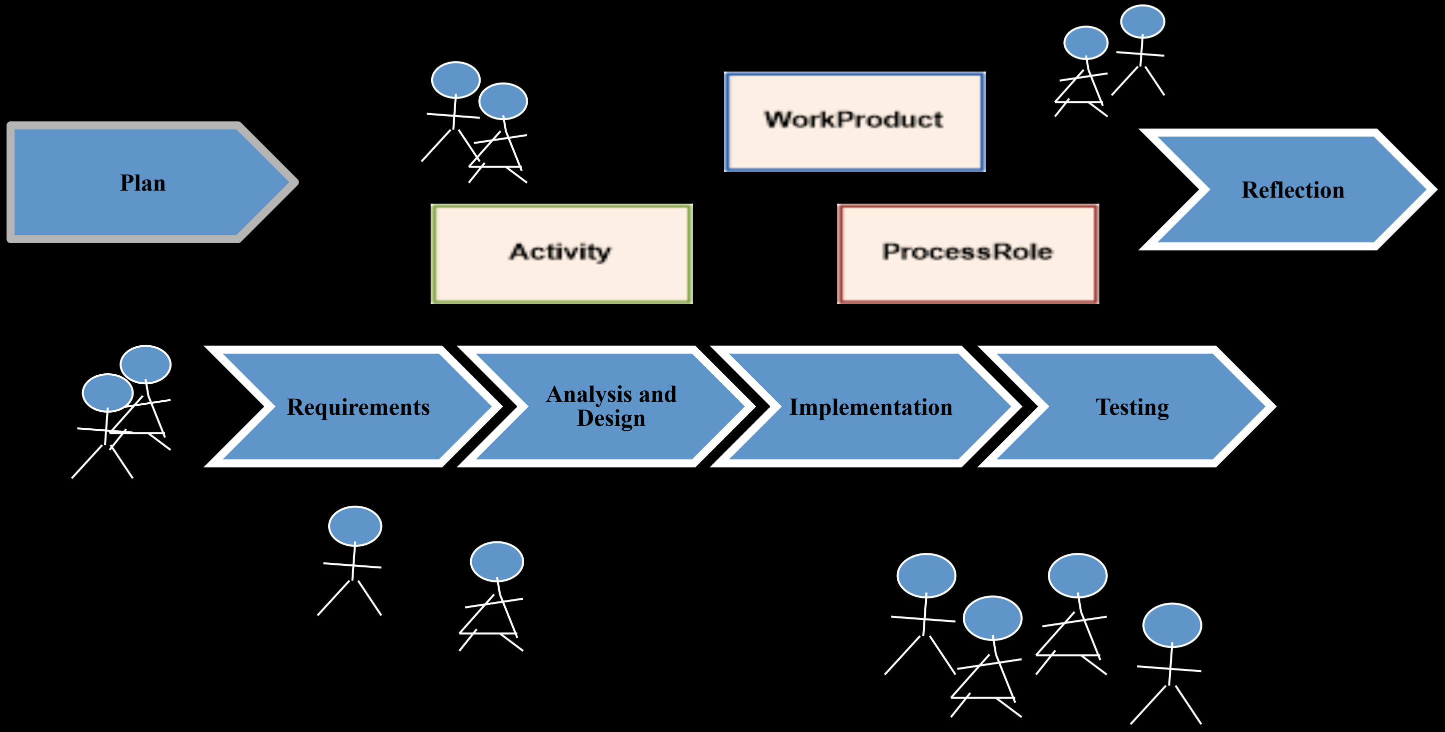
Changed!

re-test

Todays takeaways

- ✓ Observability and Controllability
- ✓ Code level – structural coverage testing
- ✓ API or Integration level – more challenging
  - Category partitioning
  - Use-case based generation

Course Takeaway – Understand a Software Process



Linnæus University
Sweden

Thank You!

Linnæus University
Sweden