POWERGATE FUNDAMENTALS OF SOFTWARE TESTING

THE TALENT POOL IN VIETNAM

MAY, 2012

PowerGate Software
The Talent Pool in Vietnam

CONTENTS

- Duration: 2 hours
- Agenda:

TYPICAL PROJECT …

Project Kick off

Bug Fixing Just started..

First bug fixed...

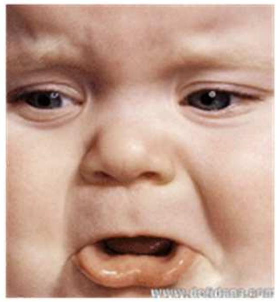
Regression Testing..

Fixed Regression too..

Requirement Changes..

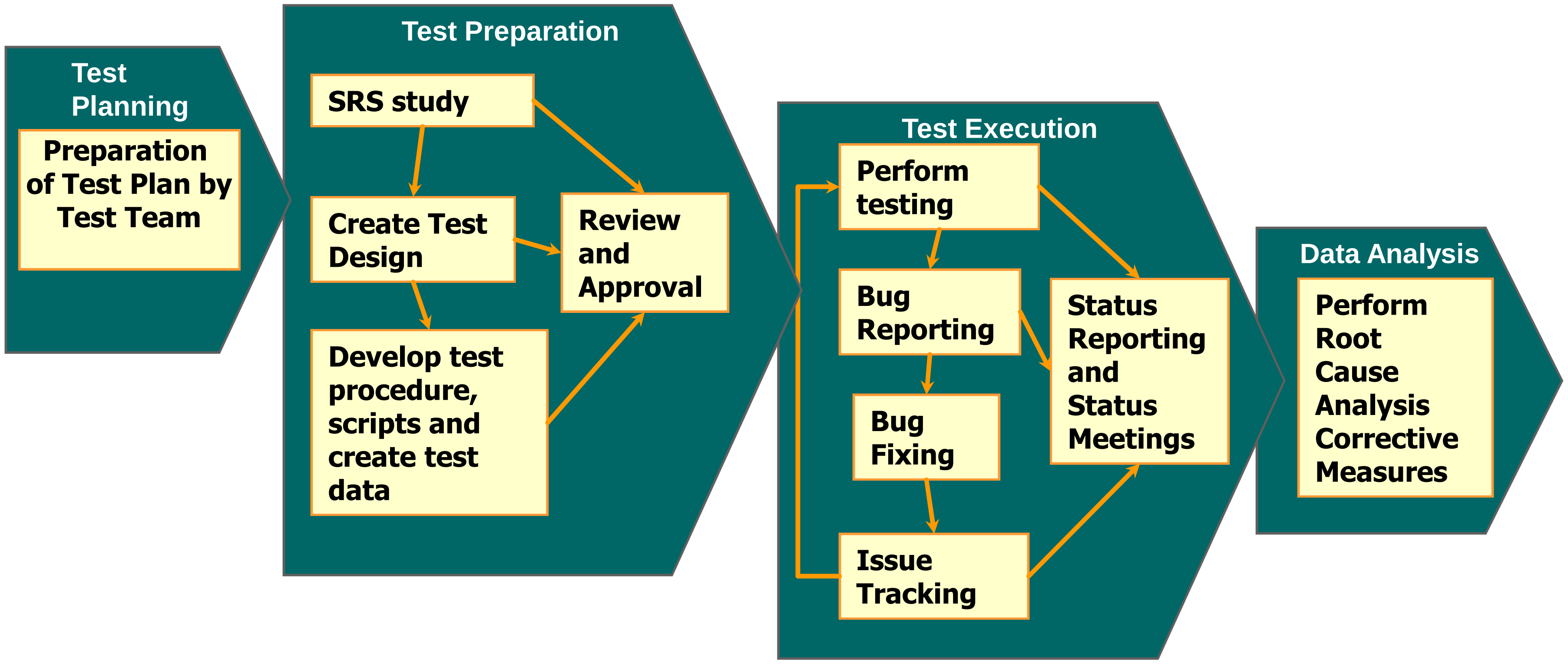
What to do??

When will testing stop?


Next Increment, please...?!

Quality.. God Let me sleep?

TEST PROCESS


Test Planning
Preparation of Test Plan by Test Team
Test Preparation
SRS study
Create Test Design
Review and Approval
Develop test procedure, scripts and create test data
Test Execution
Perform testing
Bug Reporting
Bug Fixing
Issue Tracking
Status Reporting and Status Meetings
Data Analysis
Perform Root Cause Analysis Corrective Measures

TEST PROCESS – INPUT/OUTPUT

- Input:
  - ❖Customer requirements and Acceptance criteria
  - ❖Change requests
  - ❖Software Requirement Specification (SRS)
  - ❖Design documents
  - ❖Programs (Modules)

- Output:
  - ❖Test documents: Test plan, Test cases and procedures, Test script, Test data
  - ❖ Defect list
  - ❖Test execution log
  - ❖Summary report
  - ❖Defect analysis report

SOFTWARE ERRORS

What is a software error?

One common definition of a software error is a mismatch between the program and its specification.

Definition #1:

“A mismatch between the program and its specification is an error in the program if and only if the specification exists and is correct.”

Definition #2:

“A software error is present for when the program does not do what its end user reasonability expects to do.” (Myers, 1976)

Definition #3:

“There can never be an absolute definition for bugs, nor an absolute determination of their existence. The extent to which a program has bugs is measured by the extent to which it fails to be useful. This is a fundamentally human measure.” (Besizer, 1984)

CATEGORIES OF SOFTWARE ERRORS

- User interface errors, such as output errors, incorrect user messages.
- Function errors
- Defect hardware
- Incorrect program version
- Testing errors
- Requirements errors
- Design errors
- Documentation errors
- Architecture errors
- Module interface errors
- Performance errors
- Error handling
- Boundary-related errors
- Logic errors, such as calculation errors
- State-based behavior errors
- Communication errors
- Program structure errors, such as control-flow errors

SOFTWARE TEST PLANNING

Like other activities in software engineering phases, it is impossible to have a cost-effective software test process without a very good planning,

The major objective of software test planning:
- generate a well-defined software test plan.

What content should be included in a software test plan?

- Testing activities and schedule
- Testing tasks and assignments
- Selected test strategy and test models
- Test methods and criteria
- Required test tools and environment
- Problem tracking and reporting
- Test cost estimation

Other needed items: quality control process and standards

SOFTWARE TEST REQUIREMENTS

Before starting test design, we must identify our test objectives, focuses, and test items.
The major purpose is to help us understand what are the targets of software testing.

This step can be done based on:

- Requirements specifications
- Inputs from developers
- Feedback from customers

Benefits are:

- Identify and rank the major focus of software testing
- Check the requirements to see if they are correct, completed, and testable
- Enhance and update system requirements to make sure they are testable
- Support the decision on selecting or defining test strategy

For example,
- for performance testing, we need clear requirements on system performance.

SOFTWARE TEST REQUIREMENTS

The essentials of testing requirements include:

- Specified testing methods
- Required test types am d test coverage criteria
- Selected or required test tools
- Testing focuses and test items for each type of software testing

An example of performance testing requirements:

“Check the system performance to make sure it meet 99% system reliability requirements”

A typical example for required test items is:

Test item #I: “Test the call waiting feature (REQ #j) during system testing based on the given requirements specifications.”

SOFTWARE TEST DESIGN

Software test design is an important task for software test engineers.

A good test engineer always know how to come out quality test cases and perform effective tests to uncover as many as bugs in a very tight schedule.

What do you need to come out an effective test set ?

- Choose a good test model and an effective testing method
- Apply a well-defined test criteria
- Generate a cost-effective test set based on the selected test criteria
- Write a good test case specification document

What is a good test case?

- It must have a high probability to discover a software error
- It is designed to aim at a specific test requirement
- It is generated by following an effective test method
- It must be well documented and easily tracked
- It is easy to be performed and simple to spot the expected results
- It avoids the redundancy of test cases

SOFTWARE TEST DESIGN

- What content should be included in a test case?.

Test Case ID:	Test Item:
Wrote By: (tester name)	Documented Date:
Test Type:	Test Suite#:
Product Name:	Release and Version No.:

Test case description:

Operation procedure:

Pre-conditions:	Post-conditions:
Inputs data and/or events:	Expected output data and/or events:

Required test scripts:

SOFTWARE TEST EXECUTION

- Test execution can be performed:
  - using manual approach
  - using a systematic approach
  - using a semi-automatic approach
- Basis activities in test execution are:
  - Select a test case
  - Set up the pre-conditions for a test case
  - Set up test data
  - Run a test case following its procedure
  - Track the test operations and results
  - Monitor the post-conditions of a test case & expect the test results
  - Verify the test results and report the problems if there is any
  - Record each test execution

SOFTWARE TEST EXECUTION

- What do you need to perform test execution?
  - a test plan
  - test design specification with test case specifications
  - a test suite with documented test cases (optional)
  - test supporting facility, such as test drivers, test stubs, simulators

- What are the outcome of an test execution:
  - Text execution record and report
  - Problem and bug reports
  - Error logs

- With automatic test execution tools (or test control tools), we can do:
  - automatic test runner
  - record and replay

PROBLEM ANALYSIS AND REPORT

- When do we issue a problem?
- Whenever a bug or problem is found, we need to write down a problem report immediately.
- What are the content of a problem report?

Problem ID	current software name, release no. and version no.		
Test type	Reported by	Reported date	Test case ID
Subsystem (or module name)	Feature Name (or Subject)		
Problem type (REQ, Design, Coding, …)	Problem severity (Fatal, Major, Minor, ..)		
Problem summary and detailed description			
Cause analysis	How to reproduce?	Attachments	

PROBLEM ANALYSIS AND REPORT

- <u>How to track, control, and manage issued problems?</u>
  - A systematic solution is needed to track and maintain the reported problems in a repository.
  - Define and implement a problem control and analysis process to control problem tracking, reporting, analysis, fixing and resolutions.

- <u>Characteristics of a problem report:</u>
  - Simple and understandable
  - Traceable and numbered
  - Reproducible
  - Non-judgmental

- <u>Problem analysis:</u>
  - Finding the most serious consequences
  - Finding the simplest and most general conditions
  - Finding alternative paths to the same problem
  - Finding related problems

SOFTWARE TEST REVIEW

What is a review?

- A review is a verification activity to assure a correct method has been used to create a software product.

- Participants in a review take full responsibility for results.

There are two types of reviews:

- Formal reviews:
    - use a well-defined review method (or technique)
    - generate formal review results

- Informal reviews
    - use a desk checking approach
    - conduct an informal review
    - generate information review results

SOFTWARE TEST REVIEW

Products should be reviewed during software testing:

Test Plan Test Design Specification
Test Procedure Specification
Test Report Problem Reports

What do reviews accomplish?

- Reviews provide the primary mechanism for reliably evaluating progress.

- Reviews train and educate the participants and have a significant positive effect on staff competence.

- Reviews give early feedback and prevent more serious problems from arising.

- Reviews bring individual capability to light.

TEST MANAGEMENT

- Test management encompass:
  - management of a test team
  - management of a test process
  - management of test projects

A test manager's role:

- Play as a leadership role in:
  - planning projects - setting up a direction -motivating people
  - build a team - manage engineers

- Play as a controller in:
  - product evaluation - performance evaluation
  - changing to a new direction

- Play as a supporter in:
  - assist and train engineers - train engineers
  - enforce and control test methods, standards, and criteria
  - select and develop test tools

TEST MANAGEMENT

- Test management functions:

- Management	- Manage test projects
	- Manage team members
	- Manage test processes
- Motivation	- Motivate quality work from team members
	- Simulate for new ideas and creative solutions
- Methodology	- Control of setting up test methodology, process, standards.
	- Control of establishing test criteria
- Mechanization	- Control the selection and development of test tools
	- Mechanism for the configuration management of test suites
	- Control of setting up an integrated test environment
- Measurement	- Measure test cost, complexity and efforts
	- Measure engineer performance
	- Measure test effectiveness
	- Measure product quality

TEST ENGINEERS' TASKS

- What does a test engineer do?

  - Ensure that testing is performed
  - Ensure that testing is documented
  - Ensure that testing methodology, techniques, standards are established and developed

  The basic skills for a qualified test engineer:

Controlled	- Organized individual, systematic planning - Good planning on testing
Competent	- Technical awareness of testing methods, tools, and criteria
Critical	- Inner determination to discover problems
Comprehensive	- Total understanding of the given system and specifications - Pay good attention to the details
Considerate	- Ability to related to others and resolve conflicts

BASIC TEST ENGINEERS’ TASKS

- The basic tasks of a test engineer include:

  - Prepare testing plans and organize testing activities
  - Design test cases and document them using well-defined test methods
  - Develop test procedures and test data
  - Write problem reports and text execution records
  - Use testing tools and aids
  - Review test designs, results and problems
  - Test maintenance changes
  - Oversee acceptance tests

TEST AUTOMATION AND TOOLS

What is software test automation?

Software test automation refers to a process and activities that reduce manual testing activities and efforts in a software development lifecycle.

Why software test automation?

- Reduce software testing time, cost, and efforts
- Increase the quality of software testing
- Apply well-defined test methods through tools
- Relieve the complicated and redundant work from test engineers

What do we need to automate software testing?

- Limited cost and schedule
- Select and introduce quality test tools
- Develop necessary effective and efficient test tools and facility
- Apply well-defined testing methods and coverage
- Form an integrated test environment supporting various software testing activities for products on a production line

TEST AUTOMATION AND TOOLS

Basic steps for test automation:

Level #1: Automatic test management
- test case & suite management, and documentation
- test script management

Level #2: Automatic test execution
- black-box test control and execution
- white-box test control and execution

Level #3: Automatic test generation
- black-box test generation
- white-box test generation

Level #4: Automatic test measurement
- test coverage and metrics measurement
- test cost and complexity measurement

TEST AUTOMATION AND TOOLS

Classification of software test tools:

- GUI record and replay tools
- Program specification-based test tools

- Test management and configuration management tools
- Test generation tools, such as random test tools

- Program-based test tools (or white-box test tools)
- Program test metrics tools

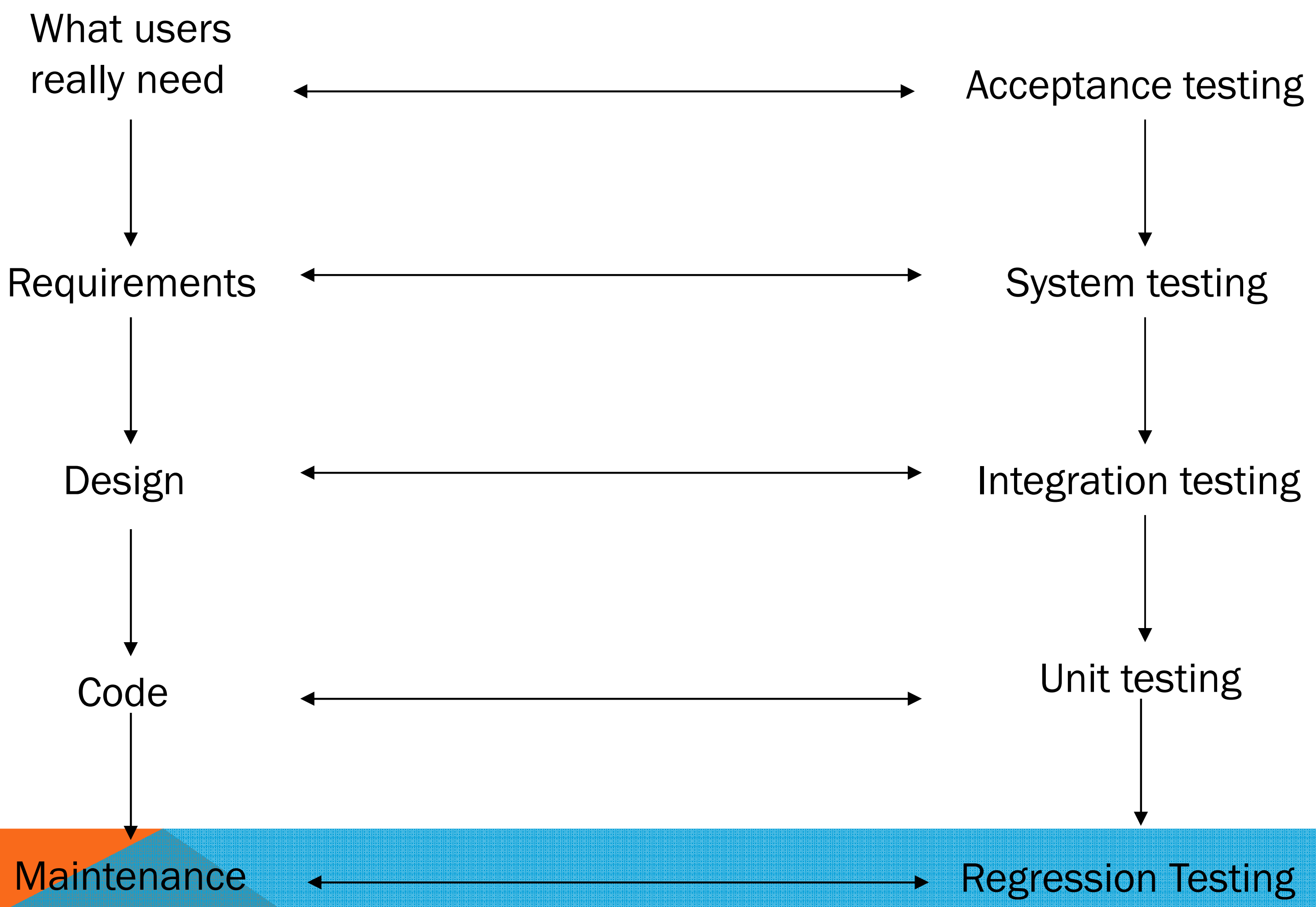
- Program performance test tools
    - performance monitoring tools
    - performance evaluation tools
    - Program load test tools

- Protocol-based confirmation tool tools
- Program regression test tools (such as program change analysis tool)

LEVELS OF TESTING

- Component/Unit testing
- Integration testing
- System testing
- Acceptance testing
- Regression testing

LEVELS OF TESTING



COMPONENT TESTING

- Testing of individual program components;
- Usually the responsibility of the component developer (except sometimes for critical systems);
- Tests are derived from the developer's experience.
- Require knowledge of code
  - High level of detail
  - Deliver thoroughly tested components to integration
- Stopping criteria
  - Code Coverage
  - Quality

COMPONENT TESTING

- Test case
  - Input, expected outcome, purpose
  - Selected according to a strategy, e.g., branch coverage
- Outcome
  - Pass/fail result
  - Log, i.e., chronological list of events from execution

INTEGRATION TESTING

- Test assembled components
  - These must be tested and accepted previously
- Focus on interfaces
  - Might be interface problem although components work when tested in isolation
  - Might be possible to perform new tests

INTEGRATION TESTING

- Strategies
  - Bottom-up, start from bottom and add one at a time
  - Top-down, start from top and add one at a time
  - Big-bang, everything at once
  - Functional, order based on execution

- Simulation of other components
  - Stubs receive output from test objects
  - Drivers generate input to test objects
  - Note that these are also SW, i.e., need testing etc.

INTEGRATION TESTING

There are two groups of software integration strategies:

- Non Incremental software integration
- Incremental software integration

Non Incremental software integration:

Big bang integration approach

Incremental software integration:

Top- down software integration
Bottom-up software integration
Sandwich integration

INTEGRATION TESTING

- Involves building a system from its components and testing it for problems that arise from component interactions.

- Top-down integration
  - Develop the skeleton of the system and populate it with components. Use stubs to replace real components.
  - Two strategies: depth first and breadth first.

- Bottom-up integration
  - Integrate infrastructure components then add functional components. Use drivers to test components

- To simplify error localisation, systems should be incrementally integrated.

SYSTEM TESTING

- Testing of groups of components integrated to create a system or sub-system;
- The responsibility of an independent testing team;
- Tests are based on a system specification.
- Functional testing
  - Test end to end functionality
  - Requirement focus
    - Test cases derived from specification
  - Use-case focus
    - Test selection based on user profile

SYSTEM TESTING

- Non-functional testing
- Quality attributes
  - Performance, can the system handle required throughput?
  - Reliability, obtain confidence that system is reliable
  - Timeliness, testing whether the individual tasks meet their specified deadlines
  - etc.

ACCEPTANCE TESTING

- User (or customer) involved
- Environment as close to field use as possible
- Focus on:
  - Building confidence
  - Compliance with defined acceptance criteria in the contract

RE-TEST AND REGRESSION TESTING

- Conducted after a change

- Re-test aims to verify whether a fault is removed
  - Re-run the test that revealed the fault

- Regression test aims to verify whether new faults are introduced
  - How can we test modified or newly inserted programs?
    - Ignore old test suites and make new ones from the scratch or
    - Reuse old test suites and reduce the number of new test suites as many as possible
  - Should preferably be automated

TEST STRATEGIES

- Code coverage strategies, e.g.
  - Decision coverage
  - Path coverage
  - Data-Flow analysis (Defines -> Uses)
- Specification-based testing, e.g.
  - Equivalence partitioning
  - Boundary-value analysis
  - Combination strategies
- State-based testing

TEST STRATEGIES

- Black-box or behavioral testing
  - knowing the specified function a product is to perform and demonstrating correct operation based solely on its specification without regard for its internal logic
- White-box or glass-box testing
  - knowing the internal workings of a product, tests are performed to check the workings of all possible logic paths

TEST STRATEGY - CODE COVERAGE

- Statement coverage
  - Each statement should be executed by at least one test case
  - Minimum requirement

- Branch/Decision coverage
  - All paths should be executed by at least one test case
  - All decisions with true and false value

TEST STRATEGY - MUTATION TESTING

- Create a number of mutants, i.e., faulty versions of program
  - Each mutant contains one fault
  - Fault created by using mutant operators

- Run test on the mutants (random or selected)
  - When a test case reveals a fault, save test case and remove mutant from the set, i.e., it is killed
  - Continue until all mutants are killed

- Results in a set of test cases with high quality
- Need for automation

TEST STRATEGY - SPECIFICATION-BASED TESTING

- Test cases derived from specification
- Equivalence partitioning
  - Identify sets of input from specification
    - Assumption: if one input from set s leads to a failure, then all inputs from set s will lead to the same failure
  - Chose a representative value from each set
  - Form test cases with the chosen values

TEST STRATEGY - BLACK BOX TESTING
EQUIVALENCE PARTITIONING

- Input data and output results often fall into different classes where all members of a class are related.
- Each of these classes is an equivalence partition or domain where the program behaves in an equivalent way for each class member.
- Test cases should be chosen from each partition.

TEST STRATEGY – BLACK BOX TESTING
EQUIVALENCE PARTITIONING

- Black-box technique divides the input domain into classes of data from which test cases can be derived.
- An ideal test case uncovers a class of errors that might require many arbitrary test cases to be executed before a general error is observed.

TEST STRATEGY - SPECIFICATION-BASED TESTING

- Boundary value analysis
  - Identify boundaries in input and output
  - For each boundary:
    - Select one value from each side of boundary (as close as possible)
  - Form test cases with the chosen values

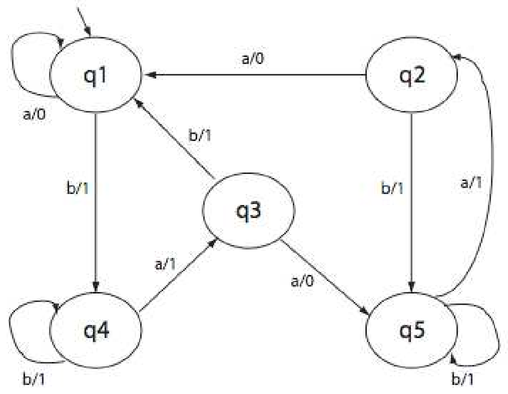
TEST STRATEGY - BLACK BOX TESTING
BOUNDARY VALUE ANALYSIS

- Black-box technique
  - focuses on classes and also on the boundaries of the input domain.

- Guidelines:
  - If input condition specifies a range bounded by values a and b, test cases should include a and b, values just above and just below a and b
  - If an input condition specifies a number of values, test cases should exercise the minimum and maximum numbers, as well as values just above and just below the minimum and maximum values

TEST STRATEGY - STATE-BASED TESTING

- Model functional behavior in a state machine (communication – protocol ...)
- Select test cases in order to cover the graph
  - Each node
  - Each transition
  - Each pair of transitions
  - Each chain of transitions of length n



TEST STRATEGY - WHITE BOX TESTING
STRUCTURAL TESTING

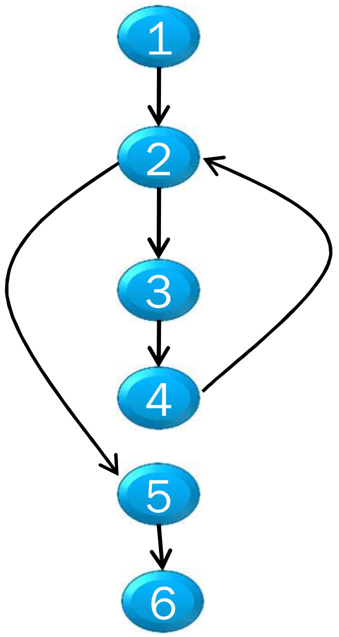
- The objective of path testing is to ensure that the set of test cases is such that each path through the program is executed at least once.

- The starting point for path testing is a program flow graph that shows nodes representing program decisions and arcs representing the flow of control.

- Statements with conditions are therefore nodes in the flow graph.

PATH TESTING – CONTROL FLOW GRAPH

- White-box technique is based on the program flow graph (CFG)



Many paths between 1 (begin) and 6 (end)

1, 2, 5, 6

1, 2, 3, 4, 2, 6

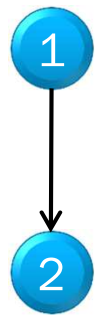
1, 2, 3, 4, 2, 3, 4, 2, 5, 6

…

- Prepare test cases that will force the execution of each path in the basis set.
- Test case : ((inputs …) , (expected outputs …))

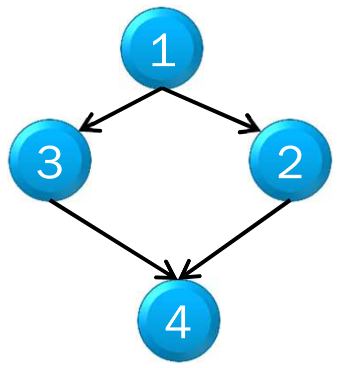
PROGRAM FLOW GRAPH

BASIC CONTROL FLOW GRAPHS



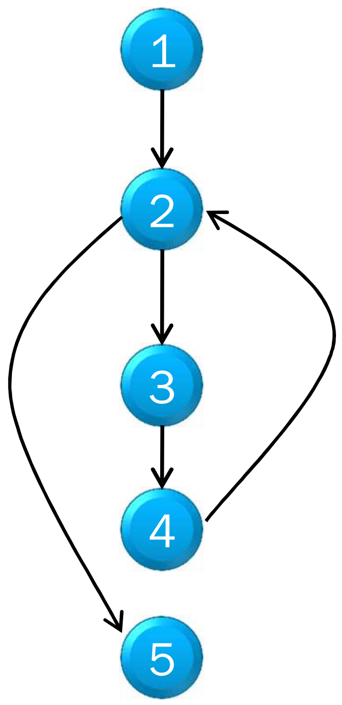
A sequence:

X = 1;
Y = X * 10;



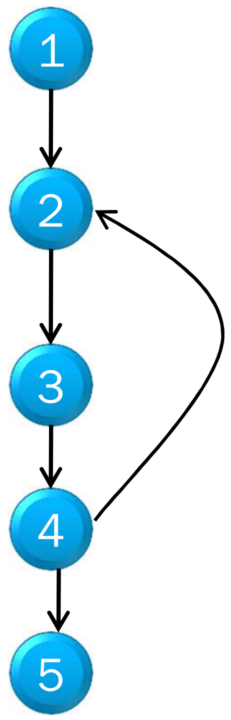
If condition:

```
If … Then
    …
Else
    …
End if
```



While loop:

```
While … do
    …
statements
    …
End while
```



Do While loop
(Repeat until):

```
do
    …
statements
    …
While …
```