CSC7056
Software Testing & Verification

(Study Notes, Topics 1 - 4)

IMPORTANT NOTE

The purpose of this document is to provide a summary of the lecture notes for the first four topics in this module, not replace them. Accordingly, this resource should be used in combination with the lecture slides, videos and other available resources.

It SHOULD NOT be assumed that class test material will be limited to the content of this document.

Due the relatively small size of topics 5 & 6, their material has not been summarised in this document.

Class Test Summary

A two hour, closed-book, lab-based assessment conducted under exam conditions.

Two Parts

Part 1: 30 multiple choice questions spanning all topic areas.

Part 2: Two open response (essay style) questions. May include multiple parts (e.g. Q1a, Q1b etc.).

Topic 1: Fundamentals of Testing (I)

Why is Testing Necessary (Lecture 1 & 2)

Context
Software is pervasive, it can be found in all aspects of life.

If a software system fails, the consequences range from minor inconvenience to loss of life.

Causes of Defects
Human error – any human being can make an error (mistake).

Errors result in a defect (bug) being introduced to the system.

If a defect is executed, this can result in a system failure.

Why We Test
1. Find Defects.
2. Check if Fit for Purpose.
3. Measure Quality.
4. Mitigate Risk
5. Build Confidence

Role of Testing
To conduct an investigation and present results to stakeholders.

Testing & Quality
Testing measures the quality of software system under test.

How Much Testing
Depends on how much risk.

What is Testing (Lecture 1 & 2)

Testing is a Process
- Planning and Control
- Choosing test conditions
- Designing and executing test cases
- Checking results
- Evaluating exit criteria
- Reporting test progress
- Reviewing documents
- Conducting static analysis

Towards These Objectives
Verification: Correct to Spec

Validation: Correct Software

Detecting Defects:
Improve quality & reduce risk.

Testing Principals (Lecture 3)

Shows Presence of Defects
Can show defects are present but cannot prove absence of defects.

Testing reduces the number of unknown defects in the software. No defects is not proof of correctness.

Exhaustive Testing is Impossible
Testing all combinations of inputs is not feasible. Use risk analysis & prioritisation to focus testing efforts.

Early Testing
Start testing as early as possible in the SDLC. When detected early, defects are often cheaper to resolve.

Defect Clustering
A small number of modules usually contains most of the defects.

Pesticide Paradox
If the same kinds of tests are repeated again and again, eventually the same set of test cases will no longer be able to find any new bugs.

Testing is Context Dependant
Testing is context dependent, for example, safety-critical software is tested differently from a free app.

Absence of Errors Fallacy
If the system built is unusable and does not fulfil the user's needs and expectations then finding and fixing defects does not help.

Psychology of Software Testing (Lecture 3)

Objectives
People and projects are driven by different objectives, so It is important to define the objectives of testing.

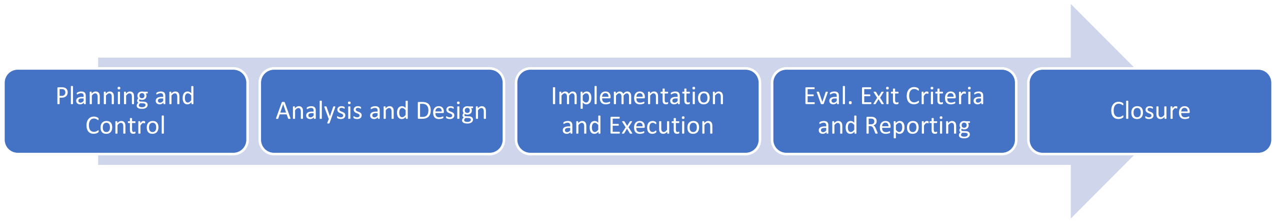
Mindsets
When considering software quality:
Dev: Focuses on positive aspects.
Tester: Focuses on negative aspects.

Levels of Independence (Low to High)
Person(s) who wrote the software
Another person(s) from the Dev team.
Person(s) from a different group.
Person(s) from a different org.

Communication
- Is critical skill for testers. Must be able to explain what a defect is and why it must be fixed.
- Must be able to communicate the state of play to stakeholders.

Topic 1: Fundamentals of Testing (II)

Five Main Test Activities (Lecture 3)


Planning and Control
Analysis and Design
Implementation and Execution
Eval. Exit Criteria and Reporting
Closure

Test Planning and Control (Lecture 3)

- Definition of the testing objectives and corresponding testing activities.
- Comparing Planned Progress against Actual Progress.

Test Analysis and Design (Lecture 3)

- Reviewing the test basis.
- Evaluating the testability of the test basis.
- Identifying and prioritising test conditions.
- Designing and prioritising high level test cases.
- Identifying necessary test data.
- Designing the test environment set-up.
- Identifying any required infrastructure and tools.
- Creating bi-directional traceability:
  - Test cases to test basis.
  - Test basis to test cases.

Test Implementation and Execution (Lecture 3)

- Finalising and prioritising test cases.
- Finalising the test data.
- Developing and prioritising test procedures.
- Creating the test data.
- Preparing test harnesses.
- Writing automated test scripts.
- Creating test suites from the test procedures for efficiency.
- Verifying test environment has been set up correctly.
- Verifying traceability between test basis and test cases.
- Executing test procedures in planned sequence.
  - Manually or using test execution tools.
- Logging the outcome (results) of each test execution.
- Comparing actual results with expected results.
- Recording discrepancies as incidents.
- Analysing incidents for causes.
- Repeating test activities (with necessary regression).

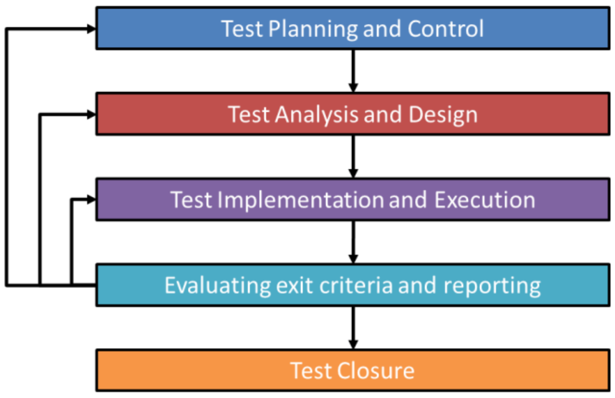
Evaluating Exit Criteria & Reporting (Lecture 3)

- Checking logs against exit criteria specified in Test Planning.
- Assessing if more tests are required.
- Assessing if exit criteria need changing.
- Writing a test summary for stakeholders.

Test Closure (Lecture 3)

- Checking planned deliverables have been delivered.
- Closing incident reports.
- Raising change records for remaining open incidents.
- Documenting the acceptance of the system.
- Finalising and archiving test-ware and test environment
- Analysing lessons learned.
- Using information gathered to improve test maturity (retrospective).

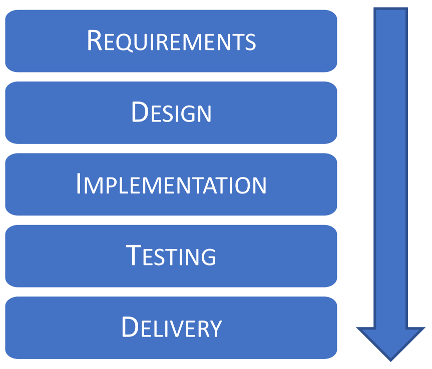
Testing Process Possible Paths (Lecture 3)


Test Planning and Control
Test Analysis and Design
Test Implementation and Execution
Evaluating exit criteria and reporting
Test Closure

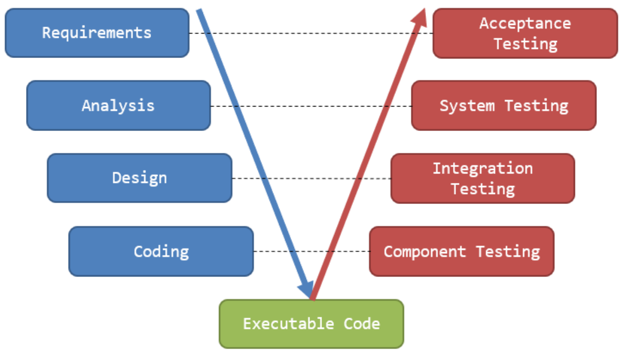
Topic 2: Testing Through the Software Life Cycle (I)

Development Models (Lecture 4 & 5)

Waterfall Model



V-Model



Iterative-Incremental Models
Iterative-incremental development is the process of establishing requirements, designing, building and testing a system, done as a series of shorter development cycles.

Verification & Validation (Lecture 5)

Verification
Confirmation by examination and through the provision of objective evidence that specified requirements have been fulfilled. [ISO 9000]

- *Are we building the product right?*

Validation
Confirmation by examination and through the provision of objective evidence that the requirements for a specific intended use or application have been fulfilled. [ISO 9000]

- *Are we building the right product?*

Characteristics of Good Testing (Lecture 8)

- For each development activity there is a corresponding testing activity.
- Every test level has objectives specific to that level.
- Test Analysis and design of tests for a specific test level should begin during the corresponding activity.
- Testers should be involved in reviewing documents as soon as drafts are available in the SDLC.

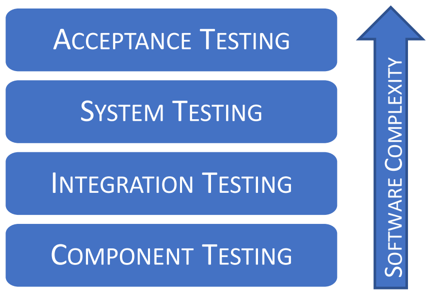
Maintenance Testing (Lecture 9)

- Testing of live, operational systems
- Includes planned & unplanned changes
- Migration & retirement
- Impact analysis
- Missing documentation?

Test Levels (Lecture 5)

Test Levels
A project may have several test levels. The number and content of levels in a project can and do vary.

Common Test Levels



Test Level Items/Consideration

- The generic objectives.
- The test basis documentation.
- The test object.
- Typical failures and defects expected.
- Test harness requirements.
- Tool support.
- Specific approaches.
- Responsibilities.

Topic 2: Testing Through the Software Life Cycle (II)

Levels of Software Testing (Lecture 5)

Component Testing
Searches for defects in and verifies the functioning of Software modules, programs, objects, classes etc. that are separately testable.
- Usually takes place during the coding phase.
- Includes functional and non-functional characteristics.
- Usually conducted by developers / author of the code.
- Defects are fixed as they are identified. Details not usually recorded.

Typical Test Objects:
- Components, Programs

Typical Test Basis:
- Component Requirements, Detailed Design, Code

Integration Testing
Testing performed to expose defects in the interfaces and interactions between integrated components or systems.
- Includes functional and non-functional characteristics.

Typical Test Objects:
- Sub-System database implémentation, Infrastructure, Interfaces.

Typical Test Basis:
- Software and System Design, Architecture, Workflows, Use Cases

System Testing
Concerned with the behaviour of the whole software system as defined by the scope of project.
- The test environment should correspond to the final target or production environment as much as possible.
- Includes functional and non-functional characteristics.
- Often carried out by an independent test team.

Typical Test Objects:
System, user & operation manuals, system configuration.

Typical Test Basis:
System & software requirement specification, use cases, functional specification, risk analysis reports.

Acceptance Testing
The aim is to establish confidence in the system. The detection of defects is not the main concern here.
- Usually a responsibility of the customers/users.
- Includes functional and non-functional characteristics

Typical Test Objects:
Business processes on fully integrated system, operational & maintenance processes, user procedures, forms, reports.

Typical Test Basis:
user requirements, system requirements, use cases, business processes, risk analysis reports.

Types of Software Testing (Lecture 6)

Functional Testing
- Testing 'what' the system does (it's external behaviour).
- Typically evaluated using black-box testing.
- Applicable to all levels of testing. The higher the level, the more complex the box.

Non-Functional Testing
- Testing 'how' the system behaves.
- Applicable to all levels of testing.
- Includes: Performance, Load, Stress, Usability, Maintainability, Reliability and Portability Testing.

Structural Testing
- Involves testing of the software structure.
- Applicable to all levels of testing
- Typically evaluated using white box testing.
- Main benchmark is the percentage of coverage (the amount of the structure than has been exercised).
- Examples include statement and decision coverage.

Change-Based Testing
- Testing related to changes in the software system.
- Key Principal: All tests must be repeatable!
  This ensures that the system can be checked after any software changes, e.g. bug fixes and software updates.
- Re-testing (Confirmation Testing): Tests conducted to verify that a defect has been fixed.
- Regression testing: retesting of unchanged areas of a software system after a software change.

Topic 3: Static Testing Techniques (I)

Overview (Lecture 7)

What is Static Testing?
Testing that does not involve the execution of code.

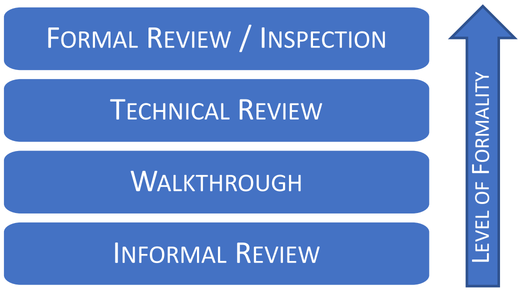
How is it done?
By examining a document or (unexecuted) code.
Manually: By visual review.
Automation: Analysis of code without execution by a tool.

What are the benefits?
Early detection of defects.

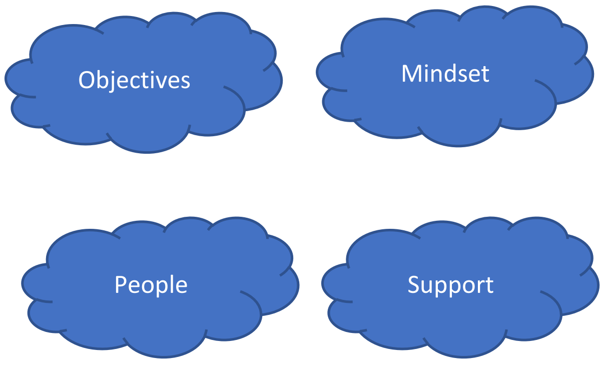
What can be tested using a Static approach?
Any deliverable related to the project that can be printed.

Types of Review (Lecture 7)

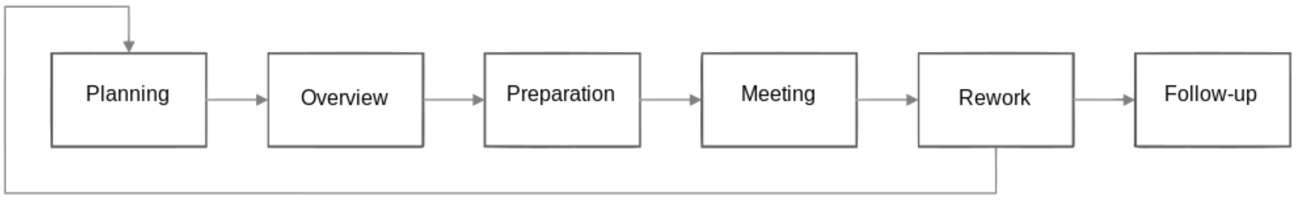
Four different types of review. Each is differentiated by their objectives and level of formality.



Review Success Factors (Lecture 7)



Activities of a Formal Review (Lecture 7)



(1) Planning
- Allocating Roles
- Defining Entry/Exit Criteria
- Scope of Document to Review

(2) Kick-Off (Optional)
- Distribute Documents
- Define Review Objectives
- Explain Review Process
- Check Entry Criteria

(3) Individual Preparation
- Individually Review Document
- Note potential defects, questions, comments, etc.

(4) Review Meeting
- Conduct
  - Log, Discuss, Decide
- Report

(5) Rework
- Fix Defects
- Log Defects

(6) Follow-Up
- Check Fixes
- Gather Metrics
- Check Against Exit Criteria

Roles and Responsibilities in a Formal Review (Lecture 7)

Manager
- Decides whether a review will be authorised – they are expensive.
- Determines if the review objectives have been met.

Moderator
- Leads the formal review.
- Plans, chairs and conducts follow-up activities.
- Mediates any disputes.

Author
- The person (or lead person) who for the document being reviewed.

Reviewers
- Individuals with a specific technical or business background - Sometimes called inspectors.
- Investigate/Review documents and describe findings - Often centres on defects.
- Chosen to represent different perspectives in the review group.

Scribe
Documents the discussion points raised during the meeting
- Issues, problems and open points.

Topic 3: Static Testing Techniques (II)

Review Types (Lecture 7)

Informal Review
Any review not based on a formal procedure. Can take the form of pairs programming or a technical lead reviewing designs and code.

Level of Formality:
No documented process to follow. Results may be documented, e.g. minutes of meeting.

Aim:
Inexpensive way which yields various levels of usefulness.

Walkthrough
An author led step-by-step presentation of a project object, e.g. storyboards, prototype system etc.

Level of Formality:
Can vary from informal to very formal

Aim:
Largely educational – build knowledge and understanding in the team / stakeholders. Useful for finding defects.

Technical Review
Focuses on reviewing technical details/procedures to build an consensus. Participants can include peers, technical experts and optional management participation.

These reviews are led by a trained moderator (not the author). One of the outcomes is a review report which includes findings and recommendations.

Level of Formality:
Generally formal, but in practice this can vary.

Aim:
Depends on the objectives set – general discussion, making decisions, finding defects, addressing technical problems, checking conformance to standards/policies/legislation.

Inspection / Formal Review
Centres around a visual examination of documentation to detect defects, e.g. a requirements spec document.

These reviews are led by a trained moderator (not the author). Outcomes includes a review report consisting of findings, recommendations and statistics.

Level of Formality:
Very formal and follows a strict procedure. Set roles and responsibilities and set entrance/exit criteria. Includes a follow-up procedure

Aim:
To find defects.

Static Analysis by Tools (Lecture 9)

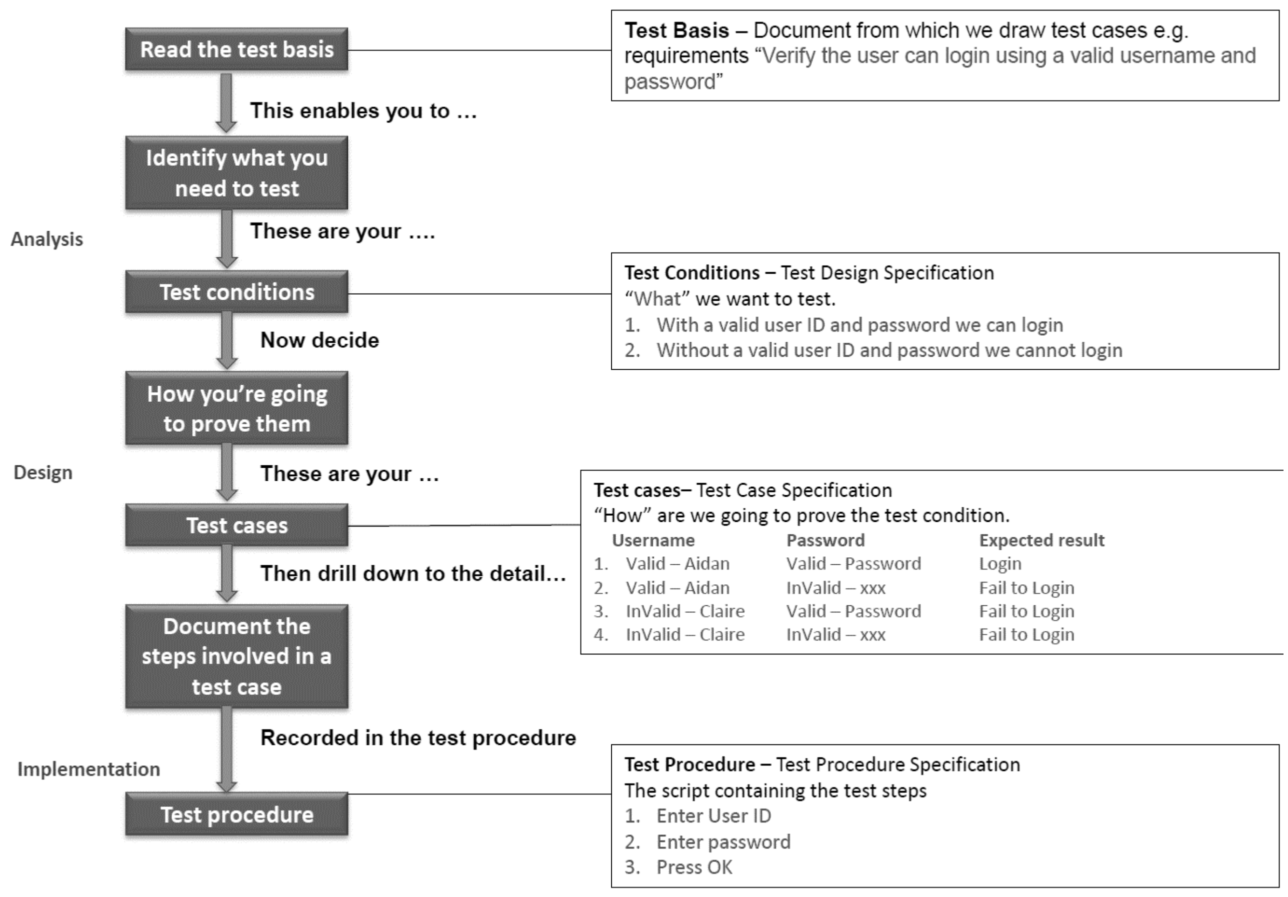
Definition
The use of tools to find defects, most often in unexecuted code, but in documentation as well.

How is it Done
- By finding defects (not failures) in documentation and code.
- By enforcing standards, usually on code.
- By generating metrics on code:
    - Cyclomatic Complexity, Depth of Nesting, Number of Lines of Code

Typical defects discovered by static analysis:
- Referencing a variable with an undefined value.
- Inconsistent interface between modules and components.
- Variables that are unused, or improperly declared.
- Unreachable code.
- Missing and erroneous logic (e.g. infinite loops).
- Programming standards violations.
- Security vulnerabilities
- General syntax errors.

Topic 4: Test Design Techniques (I)

Test Specification Process (Lecture 11)


Analysis
Design
Implementation
Read the test basis
This enables you to …
Identify what you need to test
These are your ….
Test conditions
Now decide
How you're going to prove them
These are your …
Test cases
Then drill down to the detail…
Document the steps involved in a test case
Recorded in the test procedure
Test procedure

Test Basis – Document from which we draw test cases e.g. requirements "Verify the user can login using a valid username and password"

Test Conditions – Test Design Specification
"What" we want to test.
1. With a valid user ID and password we can login
2. Without a valid user ID and password we cannot login

Test cases– Test Case Specification
"How" are we going to prove the test condition.

	Username	Password	Expected result
1.	Valid – Aidan	Valid – Password	Login
2.	Valid – Aidan	InValid – xxx	Fail to Login
3.	InValid – Claire	Valid – Password	Fail to Login
4.	InValid – Claire	InValid – xxx	Fail to Login

Test Procedure – Test Procedure Specification
The script containing the test steps
1. Enter User ID
2. Enter password
3. Press OK

Test Design Techniques (Lectures 12- 15)

Structure Based
Techniques that derive tests from an items structure.

Specification Based
Techniques that derive tests from an items specification.

Experience Based
Techniques that derive tests from a persons experience or intuition. Includes error guessing, exploratory testing and ad-hoc testing.

Choosing Test Techniques (Lectures 12- 15)

Choices are based on:
- Type of system
- Regulatory standards
- Customer/contractual requirements
- Level of risk
- Type of risk
- Test objectives
- Documents available
- Testers knowledge
- Time and budget
- SDLC
- Use case models
- Previous experience with defects found.

Experience-Based Techniques (Lecture 15)

Common Factors
Knowledge of testers, developers, users and other stakeholders about the software, its usage and its environment.

Knowledge about likely defects and their distribution in the software.

Error Guessing
A test design technique where the experience of the tester is used to anticipate what defects might be present in the component or system under test, as a result of errors made! and to design tests specifically to expose them.

Exploratory Testing
Exploratory testing is concurrent test design, execution, logging and learning, based on a test charter containing test objectives, and carried out within time-boxes.

Most useful for inadequate specs And/or severe time pressure. Or to augment other, more formal testing.

Topic 4: Test Design Techniques (II)

Equivalence Partitioning (Lectures 12)

A type of specification based testing used to reduce the number of tests required.

Done by identifying groups of values (inputs or outputs) that the system will treat the same using a single value as a representation of that entire range.

E.g. “To pass an exam, students must achieve >= 50%”

Process:
1. Identify the partitions.
2. Assign a classification – valid or invalid

<0	0 - 49	50 – 100	> 100
Invalid	Valid	Valid	Invalid

Boundary Value Analysis (Lectures 12)

Specification-based test, useful for testing boundaries – e.g. in an if statement.

“Only customer aged over 18 may purchase alcohol”

Process:
1. Identify the boundary.
2. Test the boundary, plus one value below and above.

17	18	19
Boundary - 1	Boundary	Boundary + 1

Decision Table Testing (Lectures 13)

A type of specification-based testing useful for evaluating against complex business rules.

Login Form Example:

	Rule 1	Rule 2	Rule 3	Rule 4
Conditions				
Valid username Entered	Y	Y	N	N
Valid password entered	Y	N	Y	N
Actions				
Process login attempt	Y			
Username or password error message		Y	Y	Y

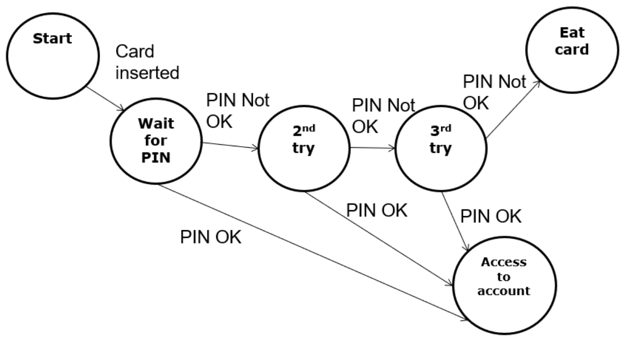
Process:
1. Determine the outcome for a given scenario, e.g.
   Valid Username & Password = Process Login

State Transition Testing (Lectures 13)

A type of specification based testing useful for evaluating embedded software.

To ensure 100% 0-switch coverage, all transition lines must be tested.

Bank ATM Example:



Structure-Based Techniques (Lecture 14)

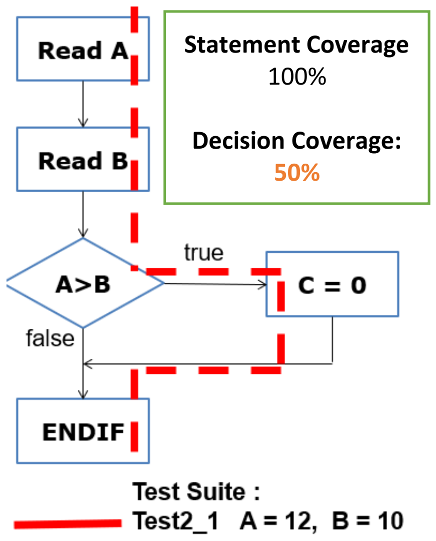
Aim: To achieve 100% code coverage

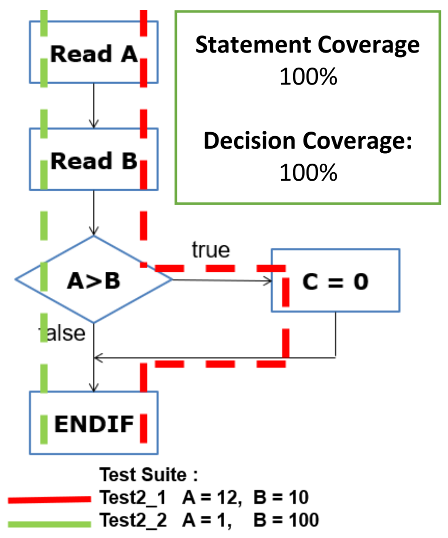
Statement Coverage
% of executable statements covered.

Decision Coverage
% of decision statements covered.

Example Pseudo Code:

```
READ A
READ B
IF A > B THEN C = 0
ENDIF
```


Read A
Read B
A>B
true
false
C = 0
ENDIF
Statement Coverage
100%
Decision Coverage:
50%
Test Suite :
Test2_1 A = 12, B = 10


Read A
Read B
A>B
true
false
C = 0
ENDIF
Statement Coverage
100%
Decision Coverage:
100%
Test Suite :
Test2_1 A = 12, B = 10
Test2_2 A = 1, B = 100