Debre Markos University
Institute of Technology
School of Computing, Software Eng A/program

Software Testing and Quality Assurance(SEng4051)

Chapter One

SQA Concepts

Contents

- Introduction
- Quality
- Detection Vs Prevention
- Verification and Validation
- Testing

Introduction

Quality?

- Quality is an intangible concept.
- The terms **good quality**, **poor quality** are used in our everyday life to tell how **good** or **bad** a product functions.
- **Quality** is a complex concept it means different things to different people, and *it is highly context dependent.*
- To be able to capture the quality concept, it is important to study quality from a broader perspective. This is because the concept of quality predates software development.

Views of Quality

Transcendental View

- In the **transcendental view**, quality is something that can be recognized through experience but is not defined in some tractable form.
- Quality is viewed to be something ideal, which is too complex to lend itself to be precisely defined. However, a good-quality object stands out, and it is easily recognized.

User's View

- It perceives quality as fitness for purpose. According to this view, while evaluating the quality of a product, one must ask the key question: "**Does the product satisfy user needs and expectations?**"

Cont...

Manufacturing View :

- Here quality is understood as conformance to the specifications.
- The quality level of a product is determined by the extent to which the product meets its specifications

Product View :

- Quality is viewed as tied to the inherent characteristics of the product.
- A product's internal qualities determine its external qualities.
- Quality is that high degree of modularity, which is an internal property, makes a software testable and maintainable.

Value - Based View :

- Quality, in this perspective, depends on the amount the customer is willing to pay for it.

What is Software?

Based on IEEE definition

- **Software** is: computer programs, procedures, and possibly associated documentation and data pertaining to the operation of a computer system.
- ISO definition lists out the following components of software:
  - Computer programs (the "code")
  - Procedures
  - Documentation
  - Data necessary for operating the software system

Software Quality

Pressman's definition:

- Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

IEEE definitions:

- The degree to which a system, component, or process meets specified requirements.
- The degree to which a system, component, or process meets customer or user needs or expectations.

Detection vs Prevention

- **Quality Assurance (QA):** Is a set of activities for ensuring quality in the process by which products are developed.
- **QA** focuses on improving software development process.
- **Quality Control (QC)** - Is a set of activities for ensuring quality in products.
- **QC** does not deal with the processes used to create a product; rather it examines the quality of the "end products" and the final outcome

Cont…

Quality Assurance	Quality Control
Aim is to prevent the defect	Aim is to identify and improve the defects
Proactive approach	Reactive approach
Does not involve executing the program	Involves executing the program
About engineering the process	Examine the product
Is the technique of managing quality	Is a method to verify quality
Is responsible for full software development life cycle	Responsible for software testing life cycle
All team members are responsible for QA	Testing team is responsible for QC

Software Verification Vs Software Validation

- **Verification** helps in ensuring if the software is being developed in the right way.
- The focus of verification is on the quality of software that is being developed, whether it follows all the standards or not, is it well engineered?
- **Validation** on the other hand helps in building the right software.
- While carrying out validation we look at whether the software is in line with customer's requirement or not.

Cont…

Verification	Validation
Verification includes checking documents, design, code and program	It is a dynamic mechanism of testing and validating the actual product
It does not involve executing the code	It always involves executing the code
Verification uses methods like reviews, walkthroughs, inspections and desk-checking etc.	It uses methods like Black Box Testing, White Box Testing and non-functional testing
It checks whether the software conforms to standards or not	It checks whether software meets the requirements and expectations of customers or not
It finds bugs early in the development cycle	It can find bugs that the verification process can not catch
The target is a software architecture, specification, complete design, high level and data base design etc.	The target is an actual product
QA team does verification by using the SRS, design, etc. documents.	With the involvement of testing team validation is executed on software code.
It comes before validation	It comes after verification

Software Testing

- An activity of checking whether the actual results match the expected results and to ensure that the software is defect free.
- The process of verifying and validating a software application to check whether it is working as expected or not.
- The intent is to find defects and improve the product quality.
- Software testing also helps to identify errors, gaps or missing requirements in contrary to the actual requirements.
- It can be done either manually or using automated tools.

Objective of Software testing

- To find any defects or bugs that may have been created when the software was being developed
- To increase confidence in the quality of the software
- To prevent defects in the final product
- To ensure the end product meets customer requirements as well as the company specifications
- To provide customers with a quality product and increase their confidence in the company

Defect, Error, Bug, Failure

A mistake in coding is called error;

an error found by a tester is called defect;

a defect accepted by a development team is called bug;

if a build* does not meet the requirements, it is a failure.

Cont…

- **An Error** is a mistake made in the code due to which compilation or execution fails.
- **A Defect** is a deviation between the actual and expected output.
- **A bug** refers to defects which means that the software product or the application is not working as per the adhered requirements set.
- **Failure** is the accumulation of several defects that ultimately lead to Software failure and results in the loss of information in critical modules thereby making the system unresponsive.

Causes of Software Errors

The nine causes of software errors are:

- Faulty requirements definition
- Client-developer communication failures
- Deliberate deviations from software requirements
- Logical design errors
- Coding errors
- Non-compliance with documentation and coding instructions
- Shortcomings of the testing process
- User interface and procedure errors
- Documentation errors

Cont…

Faulty Requirements Definition

- Usually considered the root cause of software errors
- Incorrect requirement definitions
  - Simply stated, 'wrong' definitions (formulas, etc.)
- Incomplete definitions
  - Unclear or implied requirements
- Missing requirements
  - Just flat-out 'missing.' (e.g. Program Element Code)
- Inclusion of unneeded requirements
  - (many projects have gone amuck for including far too many requirements that will never be used.
  - Impacts budgets, complexity, development time, …

Cont…

Client-developer communication failures

- Misunderstanding of instructions in requirements documentation (written / graphical instructions)
- Misunderstanding of written changes during development.
- Misunderstanding of oral changes during development.
- Lack of attention
- ✓ to client messages by developers dealing with requirement changes and
- ✓ to client responses by clients to developer questions
- Very often, these very talented individuals come from different planets, it seems.
- ✓ Clients represent the users; developers represent a different mind set entirely some times!

Cont…

Deliberate deviations from software requirements

- Developer reuses previous / similar work to save time.
- Often reused code needs modification which it may contain contain unneeded / unusable extraneous code.
- Book suggests developer(s) may overtly omit functionality due to time / budget pressures.
  - ✓ Another BAD choice; System testing will uncover these problems to everyone's dismay!
  - ✓ I have never seen this done intentionally – but understand it!
- Developer inserting unapproved 'enhancements' (perfective coding; a slick new sort / search); may also ignore some seemingly minor features, which sometimes are quite major.
  - ✓ Have seen this and it too causes problems and embarrassment during reviews.

Logical design errors

- Definitions that represent software requirements by means of erroneous algorithms.
  - Wrong formulas
  - Wrong Decision Logic Tables
  - Incorrect descriptions in text
- Process definitions: procedures specified by systems analyst not accurately reflecting the real business process.
  - Note: all errors are not necessarily software errors.
  - This seems like a procedural error, and likely not a part of the software system. But they are errors nonetheless!
- Erroneous Definition of Boundary Condition – a common source of errors.
  - The "absolutes" like 'no more than' "fewer than" "n times or more" "the first time" etc

Cont…

Coding errors

- Too many to try to list.
  - Syntax errors (grammatical errors)
  - Logic errors (program runs; results wrong)
  - Run-time errors (crash during execution)

Cont…

Non-compliance w/documentation & coding instructions

- Non-compliance with published templates (structure)
- Non-compliance with coding standards (attribute names)
- Size of program;
  - Other programs must be able to run in environment!
  - Data Elements and Code.
  - Required documentation manuals and operating instructions
- SQA Team: testing not only execution software but coding standards; manuals, messages displayed; resources needed; resources named (file names, program names).

Cont…

Shortcomings of the Testing Process

- Likely the part of the development process cut short most frequently!
- Incomplete test plans
  - Parts of application not tested or tested thoroughly!
- Failure to document, report detected errors and faults
  - So many levels of testing….we will cover.
- Failure to quickly correct detected faults due to unclear indications that there 'was' a fault
- Failure to fix the errors due to time constraints
  - Many philosophies here depending on severity of the error.

Cont…

User interface and procedure errors

- Missing or wrong functionality
- Spelling, actual and context error

Documentation errors

- Errors in the design documents
  - Trouble for subsequent redesign and reuse
- Errors in the documentation within the software for the User Manuals
- Errors in on-line help, if available.
- Listing of non-existing software functions
  - Planned early but dropped; remain in documentation!

Software Testing Principles

- A principle of software testing refers to the brief mentioned and proven concepts which guide testing professionals during software testing process.

Principle 1.Testing shows the presence of bugs

- Testing shows the defects but cannot prove that there are no defects. Meaning that the testing team cannot say that the product is 100% defect-free. ◦
- Testing reduces the number of undiscovered defects in the application. ◦
- Therefore, it is important to design test cases which find as many defects as possible.

Principle 2. Exhaustive testing is impossible

- It is impossible to test all possible combinations of input cases and data.
  - For instance, consider there are 15 fields in one screen which contain 5 possible values. To test all combinations, you would need 515 = (30,517,578,125 tests. But, project timescales would never allow testing a large number of combinations.
- For this reason, severity and priority are used to concentrate on the most important aspects to test.

Principle 3. Early testing

- The sooner we start the testing activities, the better we can utilize the available time.
- As soon as the initial products, such as the requirement or design documents are available, we can start testing.
- It is common for the testing phase to get squeezed at the end of the development lifecycle, i.e. when development has finished, so by starting testing early, we can prepare testing for each level of the development lifecycle.
- When defects are found earlier in the lifecycle, they are much easier and cheaper to fix.
- The main advantage of early testing is testers can easily detect errors and help in each level of development with fewer costs and efforts.

Principle 4. Defect clustering

- During testing, it can be observed that most of the reported defects are related to small number of modules within a system.
- This is the application of the Pareto Principle to software testing: approximately 80% of the problems are found in 20% of the modules.
- The Pareto Principle defines "the vital few, the trivial many" Bugs are uneven in frequency – a vital few contribute the majority of the program failures. Fix these first.

Principle 5. The pesticide paradox

- If you keep running the same set of tests over and over again, no more new defects will be discovered by those test cases.
- Because as the system evolves, many of the previously reported defects will have been fixed and the old test cases do not apply anymore.
- Anytime a fault is fixed or a new functionality added, we need to do regression testing to make sure the new changed software has not broken any other part of the software.
- However, those regression test cases also need to change to reflect the changes made in the software to be applicable and hopefully find new defects.

Principle 6. Testing is context dependent

- Different methods, techniques and types of testing are related to the type and nature of the application.
- For example, a software application in a medical device needs more testing than a game software. More importantly a medical device software requires risk based testing, be compliant with medical industry regulators and possibly specific test design techniques.
- Similarly, a very popular website, needs to go through rigorous performance testing as well as functionality testing to make sure the performance is not affected by the load on the servers.

Principle 7. Absence of errors fallacy

- Just because testing didn't find any defects in the software, it doesn't mean that the software is ready to be shipped.
- Were the executed tests really designed to catch the most defects? or
- Were they designed to see if the software matched the user's requirements?
- There are many other factors to be considered before making a decision to ship the software.

What is SQA?

- The degree to which a system, component, or process meets specified requirements.
- The degree to which a system, component or process meets customer or user needs or expectations.
- SQA encompasses the entire software development process such as
  - software requirements
  - software design
  - Coding
  - source code control
  - code reviews
  - change management
  - configuration management
  - release management

THE END
THANK YOU!!!

Debre Markos University
Institute of Technology
School of Computing, Software Eng A/program

Software Quality Assurance and Testing (SEng4112)

Chapter Two

Software Testing

Contents

- Introduction
- What is software testing
- Basics of Software Testing
- Software Testing Process

Introduction

- On average, software developers spend 50% of their programming time on finding and fixing bugs.
- Professional programmers have 1-3 bugs per 100 lines of code after it is "done".
- The global cost of software debugging rises to $312 billion annually.
- Software bugs cost the U.S economy an estimation of $59.5 billion per year.
- About $22.2 billion could be eliminated by improved testing that enables earlier and more effective identification and removal of defects.

Cont…

- Anyone who has worked with computers or software at tests to the wonders that technology can accomplish.
- But for software companies, creating the actual software is only half the battle; devising the method of testing it is equally challenging.
- And in order to create excellent software, an effective and efficient software testing is needed.

What is software testing?

- Software testing is
  - an activity of checking whether the actual results match the expected results and to ensure that the software is defect free.
  - the process of verifying and validating a software application to check whether it is working as expected or not.
- The intent is to find defects and improve the product quality.
- Software testing also helps to identify errors, gaps or missing requirements in contrary to the actual requirements.
- It can be done either manually or using automated tools.

Why software testing is required?

- No matter how well the software design looks on paper, once the development starts and you start testing the product, you will definitely find lots of defects in the design.
- It is not possible to create software with zero defects
- The purpose of testing is to find problems – find as many problems as possible
- The purpose of finding problems is to fix them – then fix the most important problems, as there isn't enough time to fix all of them.

Why…

- **Generally**, software testing is required
  - ✓ To ensure that the product is inline with the requirement of the client
  - ✓ To ensure that the system is free from any bug that can cause any kind of failure
  - ✓ To ensure the quality of the product
  - ✓ To check the reliability of the software
  - ✓ To make sure that the final product is user friendly
  - ✓ To stay in the business

Who does the software testing

- There is often a debate on who should actually test the software. People often question that why developers are not allowed to test.
- A developer generally checks his code several times before he submits it for testing and still in most cases it is never error-free because a developer is generally blind to his own mistakes.
- A tester on the other hand looks at software from clients' point of view. He is unbiased and his focus is only on the specifications and the requirements. So, a tester is able to look into areas that a developer may ignore. So, the testing should always be carried out by independent testers.

Who…

- Many times, developers share their work amongst each other and test each other's work. This is known as buddy/pair testing.
- When the development and testing teams are different, there is often a communication gap and sometimes, developers become careless towards coding and do not revise their code because they think that it is all a tester's job thereby increasing the burden on the tester.

Who…

- Every development team should have dedicated testers and every project generally has at least one dedicated testing team.
- Some companies believe in having separate teams for different types of testing, this means different teams for usability, performance, security and other forms of testing.
- Some companies believe in outsourcing software testing work which means they hire a firm or independent testers or consultants to have a look at and test their project.

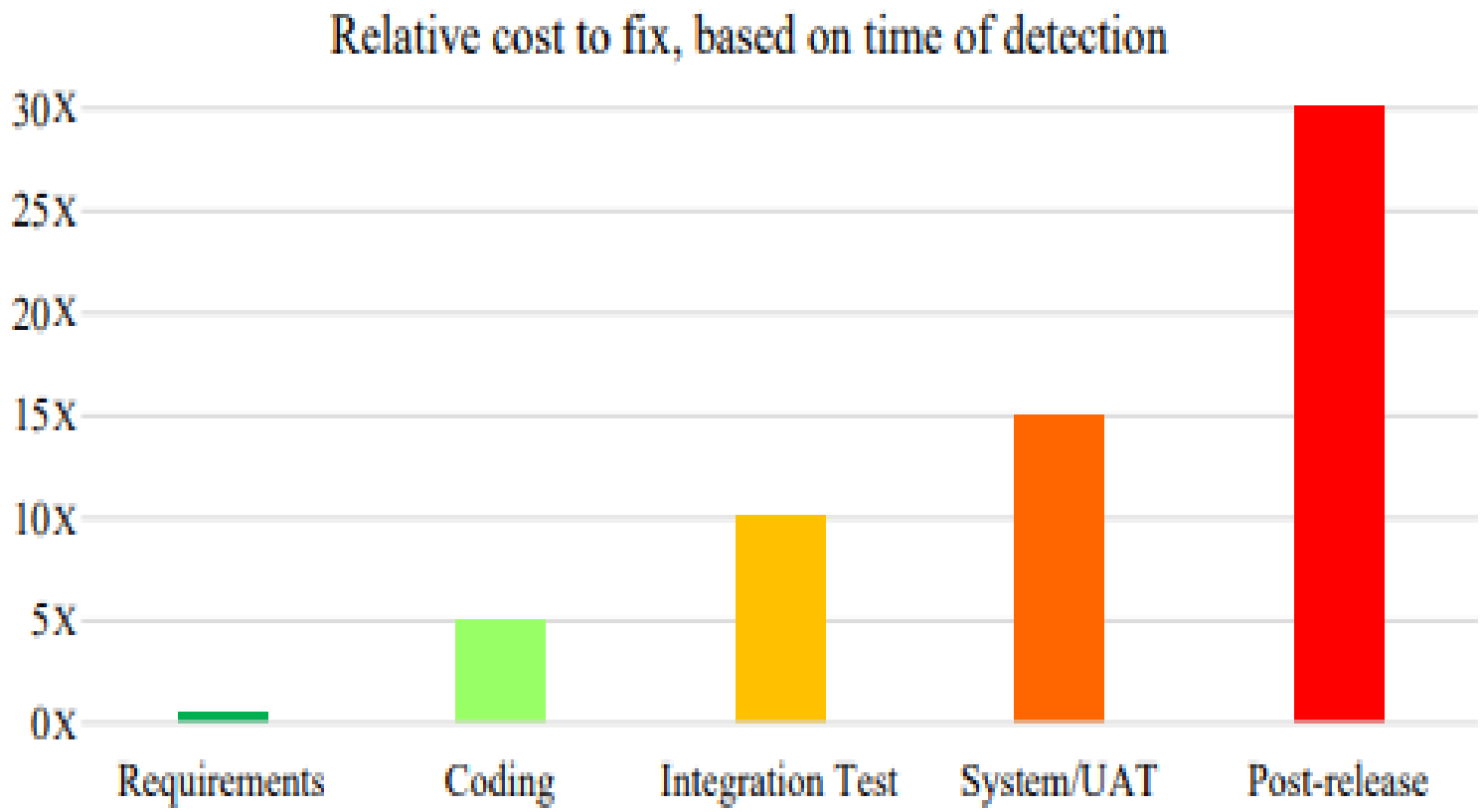
What has to be really tested

- The tester should have a good understanding about the project requirements.

When is the software testing done?

- The earlier the testing team starts testing the software, the easier it would be for the developers to complete the project on time and this would also save a lot of time, money and effort.
- Starting testing in the later stages of development can turn out to be an expensive matter as it is very difficult to rectify defects once the software has reached the final stages of development.
- Dividing software development into stages and then testing work done in every stage before moving on to the next stage helps in finishing the software development in time with good results.

When…



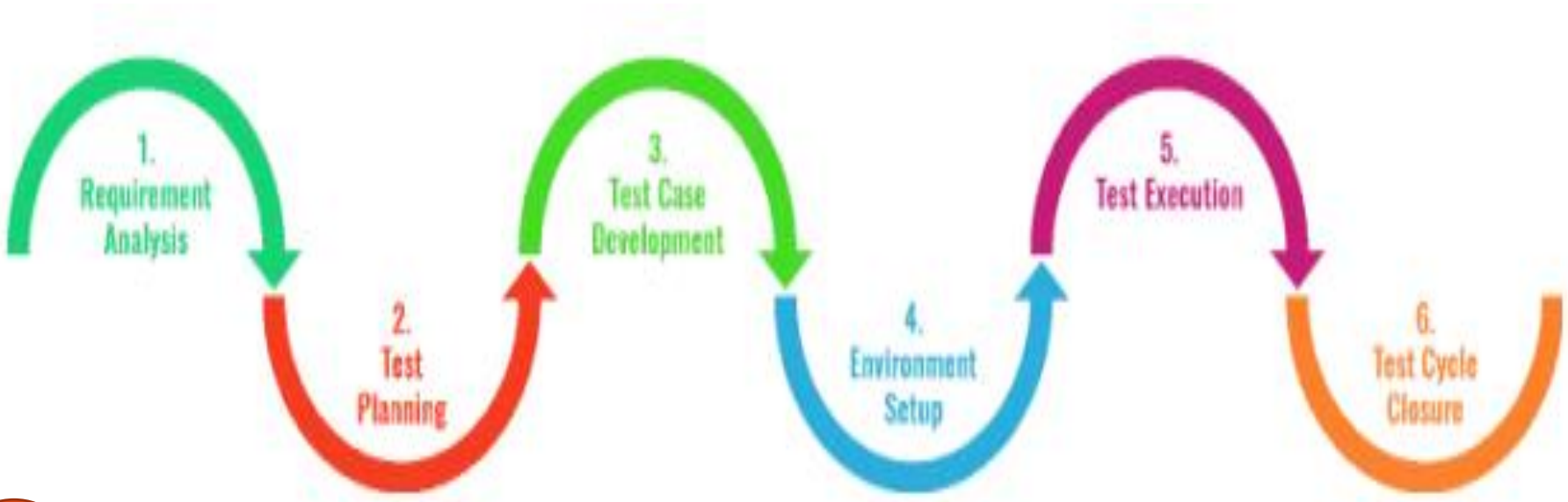
Source: National Institute of Standards and Technology

Software testing process

- A process is a series of activities performed to fulfill a purpose and produce a tangible output based on a given input.
- Processes can be described and hence monitored and improved. A process description must always include:
  - A definition of the input
  - A list of activities—the procedure
  - A description of the output

Software testing life cycle (STLC)

- Software testing life cycle defines the stages in testing of software.
- It is executed in a systematic and planned manner.
- In STLC process, various activities are carried out to improve the quality of the product.
- STLC, in general, comprises of the following phases:



1. Requirement Analysis

- Before we go for designing any software, we need to analyze the requirements. Like
  - ✓ What is the purpose of the application?
  - ✓ How many users are going to use this?
  - ✓ What is the traditional process of the client and how our software is going to help our client?
- Here testing team gathers as much information as possible about the software which they are going to test.
- It's obvious that if the testing team has good knowledge about the software, then it can test it very well.

2. Software test planning

- Software test planning is the practice of documenting software testing requirements in an organized manner.
- The test plan serves as a blueprint to conduct software testing activities as a defined process which is minutely monitored and controlled by the test manager.

Cont…

- A test plan is a detailed document that outlines the
  - ✓ test tasks, (what)
  - ✓ test strategy, (how)
  - ✓ test approach, (how)
  - ✓ testing objectives, (why)
  - ✓ resources (manpower, software, hardware) required for testing,
  - ✓ test schedule, (when)
  - ✓ test estimation and (how much)
  - ✓ test deliverables. (outputs)

Importance of Software Test Plan

- Test plan guides the testing phases to be followed.
- Test plan helps us determine the effort needed to validate the quality of the application under test
- Helps people outside the test team such as developers, business managers, customers understand the details of testing.
- It can be reviewed by the management team and re-used for other projects.

How to write a Software Test Plan

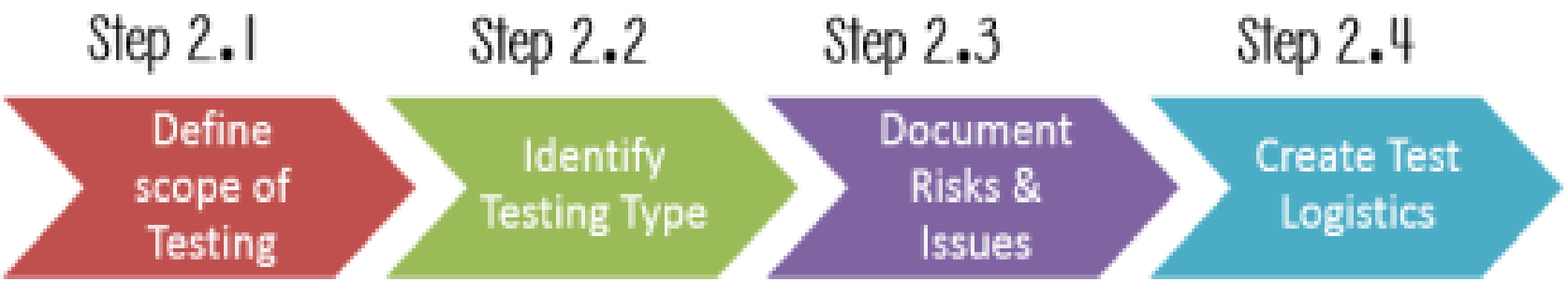
- Follow the eight steps below to create a test plan as per IEEE 829 Standard

I. Analyze the product

- You must learn a product thoroughly before testing it. How can you test a product without any information about it? The answer is **impossible**.
- You should research clients or the end users to know their needs and expectations from the application.
  - Who will use the software?
  - What is it used for?
  - How will it work?
  - What software/ hardware the product uses?

II. Develop Test Strategy

- Test strategy is a critical step in making a test plan.
- A test strategy is a high-level document which is usually developed by test manager.
- This document defines:
  - The project's testing objectives and the means to achieve them
  - Determines testing effort and costs
- You should follow the steps below



2.1. Define Scope of Testing

- Before the start of any testing activity, scope of the testing should be known. You must think hard about it.
- The components of the system to be tested (hardware, software, middleware, etc.) are defined as "in scope"
- The components of the system that will not be tested also need to be clearly defined as being "out of scope."
- Defining the scope of your testing project is very important for all stakeholders. A precise scope helps you
  - Give everyone confidence & accurate information of the testing you are doing
  - All project members will have clear understanding about what is tested and what is not

How you determine scope of

- To determine scope, you must have
  - Precise customer requirement
  - Project budget
  - Product specification
  - Skills and talent of your test team
- For instance, if a customer wants you to conduct stress and load testing, but the project budget does not permit to do so. In such a case what will you do?

Cont…

- Well, in such case you need to convince the customer that stress and load testing is extra work and will consume significant resources. Give him data supporting your facts. Tell him if we conduct these tests, the budget will increase by XYZ amount. If the customer disagrees, you have to modify your "in scope" and "out of scope" testing
- As the software requirement specs, only focus on testing the **functions** and external interface (**in scope** testing)
- Non-functional testing such as stress and load testing currently will not be tested. (**out of scope**)

2.2. Identify Testing Type

- A testing type is a standard test procedure that gives an expected test outcome.
- Each testing type is formulated to identify a specific type of product bugs. But, all testing types are aimed at achieving one common goal "early detection of all the defects before releasing the product to the customer"
- Your team cannot have enough efforts to handle all kind of testing.
- As test manager, you must set **priority** of the testing types
  - Which testing types should be **focused**?
  - Which testing types should be **ignored** for saving cost?

2.3. Document Risk & Issues

- Risk is future's uncertain event with probability of occurrence and a potential for loss.

Risk	Mitigation
Team members lack the required skill for website testing.	Plan training courses to skill up your members
The project schedule is too tight; it's hard to complete this project on time	Set test priority for each of the test activity.
Test manager has poor management skill	Plan leadership training for the manager
Lack of cooperation negatively affects your employees' productivity	Encourage each team member in his task, and inspire them to greater efforts.
Wrong budget estimate and cost overruns	Establish the scope before beginning work, pay a lot of attention to project planning and constantly track and measure the progress

2.4. Create Test Logistics

- In test logistics, the test manager should answer the following questions:
  - Who will test?
  - When will the test occur?
- You may not know exact names of the tester who will test, but the **type of tester** can be defined.
- To select the right member for specified task, you have to consider if his skill is qualified for the task or not, also estimate the project budget. Selecting wrong member for the task may cause the project to **fail** or **delay**.
- A person having the following skills is most ideal for performing software testing:
  - Ability to understand customers point of view
  - Strong desire for quality
  - Attention to detail
  - Good cooperation
- In your project, the member who will take in charge for the test execution is the **tester**. Based on the project budget, you can choose in-source or outsource member as the tester.

III. Define Test Objective

- Test objective is the overall goal and achievement of the test execution. The objective of the testing is
  - finding as many software defects as possible;
  - ensure that the software under test is **bug free** before release.
- To define the test objectives, you should do the following two steps:

  1. List all the software features (functionality and non-functional) which will be tested.

  2. Define the **target** or the **goal** of the test based on the above features

IV. Define Test Criteria

- Test criteria is a standard or rule on which a test procedure or test judgment can be based. There're two types of test criteria:

(a) Suspension criteria

- Specify the critical suspension criteria for a test. If the suspension criteria are met during testing, the active test cycle will be **suspended** until the criteria are **resolved**.

Example: if your team members report that **40%** of test cases failed, you should **suspend** testing until the development team fixes all the failed cases.

(b) Exit Criteria

- It specifies the criteria that denote a **successful** completion of a test phase.
- The exit criteria are the targeted results of the test and are necessary before proceeding the next phase of development.

Example: 95% of all critical test cases must pass.

Cont...

- Some methods of defining exit criteria are by specifying a targeted run rate and pass rate.
  - Run rate - is the ratio between **number of test cases executed/total test cases** of test specification.

For example, the test specification has a total of 120 TCs, but the tester only executed 100 TCs. So, the run rate is 100/120 = 0.83 (83%)

  - Pass rate - is the ratio between **number of test cases passed / test cases executed**.

For example, in the above example, 100 TCs executed and 80 TCs passed. So, the pass rate is 80/100 = 0.8 (80%)

- This data can be retrieved in test metric documents.
- **Run rate** is mandatory to be 100% unless a clear reason is given.
- **Pass rate** is dependent on project scope, but achieving high pass rate shall be a goal.

V. Resource Planning

- Resource plan is a detailed summary of all types of resources required to complete project task. Resource could be human, physical (equipment and materials) and financial needed to complete the test.
- The test manager can make the correct schedule & estimation for the project based on the resources he has.
- This section represents the recommended resources for your project.

VI. Plan Test Environment

- A testing environment is a setup of software and hardware on which the testing team is going to execute test cases.
- The test environment consists of real business and user environment, as well as physical environments, such as server, front end running environment.
- To finish this task, you need a strong cooperation between test team and development team
- You should ask the developer some questions to understand the web application under test clearly.
- Here're some recommended questions. Of course, you can ask the other questions if you need.

  1. What is the maximum user connection which this website can handle at the same time?

  2. What are hardware/software requirements to install this website?

  3. Does the user's computer need any particular setting to browse the website?

VII. Schedule & Estimation

- In the test estimation phase, suppose you break out the whole project into **small tasks** and add the estimation for each task as shown below.

Task	Members	Estimate effort
Create the test specification	Test Designer	170 man-hour
Perform Test Execution	Tester, Test Administrator	80 man-hour
Test Report	Tester	10 man-hour
Test Delivery		20 man-hour
Total		280 man-hour

VIII. Determine Test Deliverables

- Test deliverables are list of all the documents, tools and other components that have to be developed and maintained in support of the testing effort.
- There are different test deliverables at every phase of the software testing process.
- They can be grouped into **three**.

1) Test deliverables provided before the testing phase.
2) Test deliverables are provided during the testing
3) Test deliverables are provided after the testing cycles is over.

Cont...

1) **Test deliverables provided before the testing phase.**
   - Test plans document.
   - Test cases documents
   - Test design specifications.
2) **Test deliverables are provided during the testing**
   - Test scripts
   - Simulators.
   - Test data
   - Test traceability matrix
   - Error logs and execution logs.
3) **Test deliverables are provided after the testing cycles is over.**
   - Test results/reports
   - Defect report
   - Release notes

Outline of a test plan

- Test plan format and content may vary depending upon different standards. The following format details the points usually covered in a test plan.

Test plan identifier

Introduction

Test items

Features to be tested

Features not to be tested

Approach

Item pass/fail criteria

Suspension criteria

Test deliverables

Testing tasks

Environmental needs

Roles and Responsibilities

Staffing and training needs

Schedule

Risks and contingencies

Approvals

Brief description

- **Test Plan Identifier**: Provides a unique identifier for the document. Every deliverable has a unique identification number which could be numeric or alphanumeric based on the company configuration management.
- **Introduction:** Brief introduction about the project and objective of the current release. Project could be platform configuration tool and objective could new mobile App interface or new feature / enhancement in existing product or defect fixes.
- **Test item:** Introduction and overview of Software Under Test.
- **Features to test:** In scope features. This could be newly added or updated features. Indirect features that have technical or functional dependency on newly added or updated features.

Cont…

- **Features not to test:** Out of scope features. Excluded product features from current Test Plan. [Note: Provide reasoning for exclusion, like, non-impacted / less impacted / less priority features, as applicable.]
- **Approach**: Strategy to test the software. Includes types of tests and how to test. Functional, performance, security testing using combined [manual + automation], manual only, automation only approach.
- **Test deliverables:** All the deliverables from the testing e.g. approaches, test cases, reports etc.
- **Item pass/fail criteria:** Entry and Exit criteria for all items. **E.g**.
  - Test Case: All Steps passed
  - Feature: All test cases executed and no defects are detected.
- **Testing tasks:** All tasks / steps to execute for test planning and execution
  - **Environmental needs:** Infrastructure required for application and testing.

Cont…

- **Responsibilities:** Roles and responsibilities for various testing / supported activities.
- **Staffing and training needs:** Training / hiring needs to bridge the gap of available and expected skill.
- **Schedule:** Test estimation (Efforts) and high-level schedule. Schedule should be for key deliverables or important milestones. Ideally, all test deliverables included in the test plan should be scheduled. Detailed test schedule (at feature or defects or resource level) is prepared at appropriate time during test execution. Risks and Mitigation: Risk identification for applicable items, assumptions, and mitigation plan.
- **Approvals**: Approvals and sign of dates.

Test plan is a guideline based on which test execution should be tracked. For successful testing and good product test delivery, it is important to update and make required changes in the plan as per changes in the any of the parameter which was basis of the test plan.

3. Test Design

- In software engineering, test design is the process of creating and writing test artifacts for testing a software.
- Tasks include:

✓Identifying test basis

✓Developing test scenarios

✓Identifying and describing test cases

✓Developing test suite

✓Identifying and structuring test scripts

Test Basis

- Test basis is the source to create test scenarios and test cases.
- It can be
  - the application itself or
  - the requirement documents such as
    - SRS (Software Requirement Specification),
    - BRS (Business Requirement Specification), etc.

Test Scenario

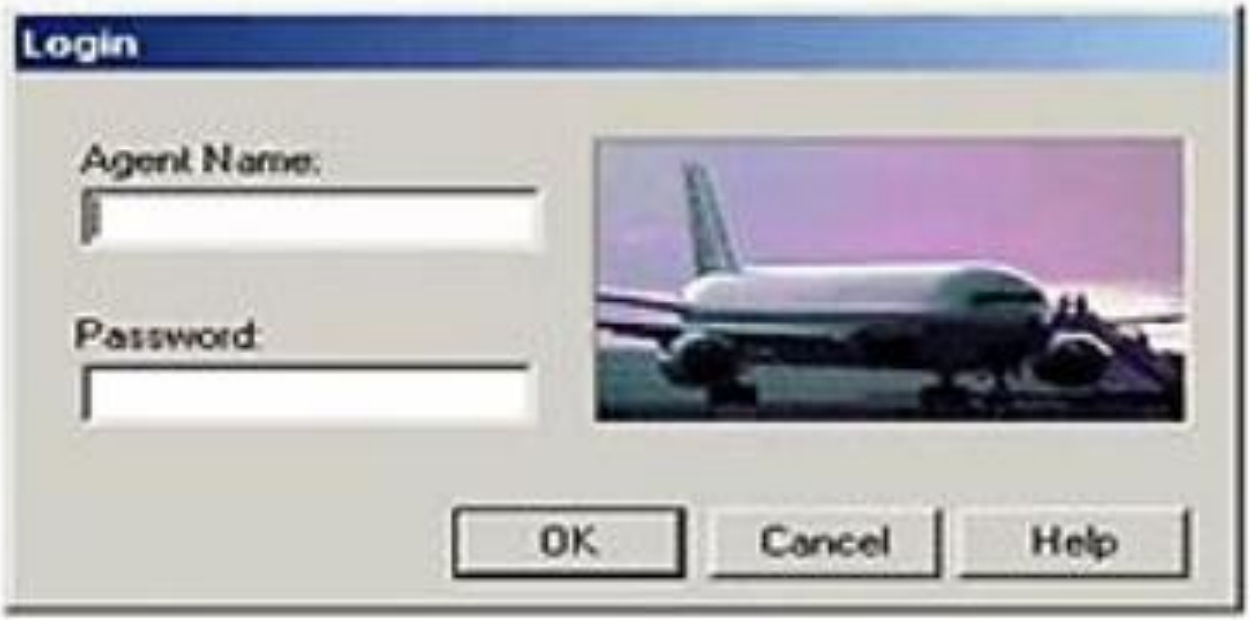
- A **test scenario** is any functionality that can be tested.
- It is also called test condition or test possibility.
- As a tester, you may put yourself in the end user's shoes and figure out the real-world scenarios and use cases of the Application Under Test (AUT).
- **Test scenario** is what to be tested and a **test case** is how to be tested.

...Test Scenario

Example 1:Test Scenario for Flight Reservation

- For the Flight Reservation Application, a few test scenarios would be

➢ **Test Scenario 1**: Check the Login Functionality



...Test Scenario

- **Test Scenario 2:** Check that a New Order can be created
- **Test Scenario 3:** Check that an existing Order can be opened
- **Test Scenario 4:** Check that a user, can FAX an order
- **Test Scenario 5**: Check that the information displayed in the HELP section is correct
- **Test Scenario 6:** Check that the information displayed in About section, like version, programmer name, copy right information is correct

...Test Scenario

- Apart from these six scenarios, here is the list of all other scenarios
  - Update Order
  - Delete Order
  - Check Reports
  - Check Graphs and so on.
- We have already learned exhaustive testing is not possible.
- Suppose you have time only to execute 4 out of those 6 scenarios which two low priority scenarios of these six will you eliminate.
- I am sure most of you would have guessed scenarios 5 & 6 since they are not the core functionality of the application. This is nothing but test prioritization.

Why to create test scenarios?

- Test scenarios are created for the following reasons:
  - To ensure complete test coverage
  - To get approval from various stakeholders like business analyst, developers, customers to ensure the application under test is thoroughly tested. It ensures that the software is working for the most common use cases.
  - To quickly determine the testing work effort and accordingly create a proposal for the client or organize the workforce.
  - To determine the most important end-to-end transactions or the real use of the software applications.

Test Case

- A test case is a set of actions executed to verify a particular feature or functionality of your software application.
- A test case is a document which consists of a set of conditions or actions which are performed on the software application in order to verify the expected functionality of the feature.
- Here we describe the end to end logical flow of a specific requirement with test data, prerequisites and expected results
- Now, consider the Test Scenario: **Check Login functionality there are many possible cases like**
- Test Case 1: Check results by entering valid User Id & Password
- Test Case 2: Check results by entering Invalid User ID & Password
- Test Case 3: Check response when User ID is Empty & Login Button is pressed, and many more.

Cont…

- Below is format of a standard login Test case

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
TU01	Check Customer Login with valid Data	1. Go to site http://demo.guru99.com 1. Enter UserId 2. Enter Password 3. Click Submit	Userid = guru99 Password = pass99	User should Login into application	As Expected	Pass
TU02	Check Customer Login with invalid Data	1. Go to site http://demo.guru99.com 1. Enter UserId 2. Enter Password 3. Click Submit	Userid = guru99 Password = glass99	User should not Login into application	As Expected	Pass

Characteristics of Good Test Case

1. Test Cases need to be simple and transparent

- Create test cases that are as simple as possible. They must be clear and concise as the author of test case may not execute them.
- Use assertive language like go to home page, enter data, click on this and so on. This makes the understanding of the test steps easy and test execution faster.

2. Create Test Case with End User in Mind

- Ultimate goal of any software project is to create test cases that meets customer requirements and is easy to use and operate. A tester must create test cases keeping in mind the end users' perspective.

3. Avoid test case repetition.

- Do not repeat test cases. If a test case is needed for executing some other test case, call the test case by its test case id in the pre-condition column

Cont…

4. Do not Assume

- Do not assume functionality and features of your software application while preparing test case. Stick to the specification documents.

5. Ensure 100% Coverage

- Make sure you write test cases to check all software requirements mentioned in the specification document. Use Traceability Matrix to ensure no functions/conditions is left untested.

6. Test Cases must be identifiable.

- Name the test case id such that they are identified easily while tracking defects or identifying a software requirement at a later stage.

7. Implement Testing Techniques

- It's not possible to check every possible condition in your software application. Testing techniques help you select a few test cases with the maximum possibility of finding a defect. **Example**: BVA, EP, etc.

8. Self cleaning

- The test case you create must return the Test Environment to the pre-test state and should not render the test environment unusable. This is especially true for configuration testing.

9. Repeatable and self-standing

- The test case should generate the same results every time no matter who tests it

10. Peer Review.

- After creating test cases, get them reviewed by your colleagues. Your peers can uncover defects in your test case design, which you may easily miss.

Test Suit

- In software engineering, a test suite is a collection of test cases that are intended to be used to test a software program, to show that it has some specified set of behaviors.
- Test suite is a container that has a set of tests which helps testers in executing and reporting the test execution status.
- A test suit is a collection of test cases that are grouped for test execution purposes.

Test Script

- A test script is a set of instructions (written using a scripting/programming language) that is performed on a system under test to verify that the system performs as expected.
- Test scripts are used in automated testing.

Test Data

- Test data is the documented data that is basically used to test the software program.
- Test data may be produced by the tester, or by a program or function that aids the tester. Test data may be recorded for re-use, or used once and then forgotten.
- Test data is divided into two categories.
  - **Positive test data** which is generally gives to system to generate the expected result and
  - **Negative test data** which is used to test the unhandled conditions, unexpected, exceptional or extreme input conditions.
- ***If the test data is inadequately designed, then such test inputs will not cover all possible test scenarios, which impact the quality of the software application under test***

4. Test Environment Setup

- Every software is developed by considering hardware and software requirement.
- Here we setup the testing environment (e. g. server/client/network, or install the setup if it is window's application etc.) with the goal of replicating the end-users' environment.
- This step decides on which Platform/OS the tester needs to perform testing of project.
  - ✓ Example: Use only Internet Explorer, Resolution must be like 136 x 768, or in many large applications they indicate the minimum hardware and software requirement like Photoshop, latest version of Matlab, Oracle, large games etc.
- In this phase, according to requirements, they configure required hardware and software.
- Tester may or may not involve in setting the configuration.

5. Test Execution

- Once the preparation of test case development and test environment setup is completed, then test execution phase can be kicked off.
- Test execution is the process of executing the code and comparing the expected and actual results.
- In this phase, testing team start executing test cases based on prepared test planning & prepared test cases in the prior step.
- Once the test case is passed then same can be marked as passed.
- If any test case is failed then corresponding defect can be reported to the developer team via bug tracking system & bug can be linked for corresponding test case for further analysis.
- Ideally, every failed test case should be associated with at least single bug.

Cont…

- Once the bug fixed by development team then same test case can be executed based on your test planning.
- If any of the test cases are blocked due to any defect then such test cases can be marked as Blocked, so we can get the report based on how many test cases passed, failed, blocked or not run etc.
- Once the defects are fixed, same Failed or Blocked test cases can be executed again to re-test the functionality.

6. Test Reporting and closure

- Call out the testing team member meeting & evaluate cycle completion criteria based on test coverage, quality, cost, time, and critical business objectives.
- Discuss what all went good, which area needs to be improved & taking the lessons from current STLC as input to upcoming test cycles, which will help to improve bottlenecks in the STLC process.
- Test cases & bug reports will analyze to find out the defect distribution by type and severity.
- Once complete the test cycle then test closure report & test metrics will be prepared.

THE END
THANK YOU!!!

Debre Markos University
Institute of Technology
School of Computing, Software Eng A/program

Software Quality Assurance and Testing (SEng4112)

Chapter Three

Levels of Software Testing

Objective

- **After successful completion of this chapter, students will be able to explain**
  - ✓ **Software testing levels**

Software Testing Levels

- In order to develop products of superior quality and functionality, testers and developers use unique and different levels of software testing.
- These levels of testing include several testing methods that are used while conducting software testing to gain intended results and to fulfil the demands of the stakeholders.
- Each of these levels of testing has a particular purpose that provides extra value to the software development life cycle.

Cont…

- Levels of software testing are process in which every component or small unit of software is tested.
- There are different levels of software testing; each has its features, advantages, and disadvantages.
- These different levels of software testing are designed to test the software performance and behaviors of the software at different stages.
- There are different stages in the software development lifecycle like requirements, design, coding, and execution. **Levels of software testing** are used to find those missing areas between these stages of the software development lifecycle.

Cont...

- There are **four** different levels of software testing, which are as follows:

    1. Unit testing
    2. Integration testing
    3. System testing
    4. Acceptance testing

1. Unit Testing

- **Unit testing** is also known as the first level of testing.
- A **unit** means a small component.
- **Unit testing** is the process of testing small components of software or product.
- This testing is performed using the white box testing method.
- One of the main advantages of unit testing is it allows each small component of the software to be tested separately.
- The **developers** perform this testing.
- This testing aims to test each component of the software to check that
  - it is functioning well or not,
  - does it works as it is intended,
  - does it fulfills the specified requirements.

Advantages of Unit Testing

- **Unit testing** improves the quality of the code.
- As **unit testing** tests only components of the software, it is easy to debug the software. If some error occurs, only the latest changes that are made need to be tested.
- In **unit testing**, the development process is faster because it requires fewer efforts to find and fix bugs than another type of testing.
- Bugs detected at earlier stage are easier to be fixed.
- Unit testing actually saves tester's time, effort and cost. The cost of solving defects is less as compared with others.
- Unit testing gives confidence to the developing team.
- Modules whose unit testing has been done successfully can be reused by the developer.

Limitations of Unit Testing

- Unit testing can't catch every error in a program. Unit testing by its nature focuses on a unit of code. Hence, it can't catch integration errors or broad system level errors.
- It is not possible to evaluate all execution paths even in the most trivial programs

Integration Testing

- This is the second level of software testing, where each component or unit is combined and tested as a whole.
- The goal behind this testing is to ensure that software or product is working properly when the unit of software is combined together with other components.
- As its name suggests, it is used to identify errors between interacted components.
- A typical software project consists of multiple software modules, coded by different programmers.
- Integration testing focuses on checking data communication among these modules.

Why do Integration Testing?

- Although each software module is unit tested, defects still exist for various reasons like
  - A module in general is designed by an individual software developer whose understanding and programming logic may differ from other programmers.
  - Integration testing becomes necessary to verify the software modules work in unity
  - At the time of module development, there are wide chances of change in requirements by the clients. These new requirements may not be unit tested and hence system integration testing becomes necessary.
  - Interfaces of the software modules with the database could be erroneous
  - External hardware interfaces, if any, could be erroneous
  - Inadequate exception handling could cause issues.

Approaches of Integration Testing

- The software industry uses variety of strategies to execute integration testing:

1. Big Bang Approach
2. Incremental Approach- which is further divided into
   - ✓ Top Down Approach,
   - ✓ Bottom Up Approach and
   - ✓ Sandwich Approach - Combination of top down and bottom up approaches

Big Bang Approach

- Here all components are integrated together at once, and then tested.
- In this testing, the individual modules are not integrated until all the modules are ready.
- In this type of testing, some disadvantages might occur like, defects can be found at the later stage.
- It would be difficult to find out whether the defect arouse in interface or in module.

Big Bang…

Advantages:

- Convenient for small systems.

Disadvantages:

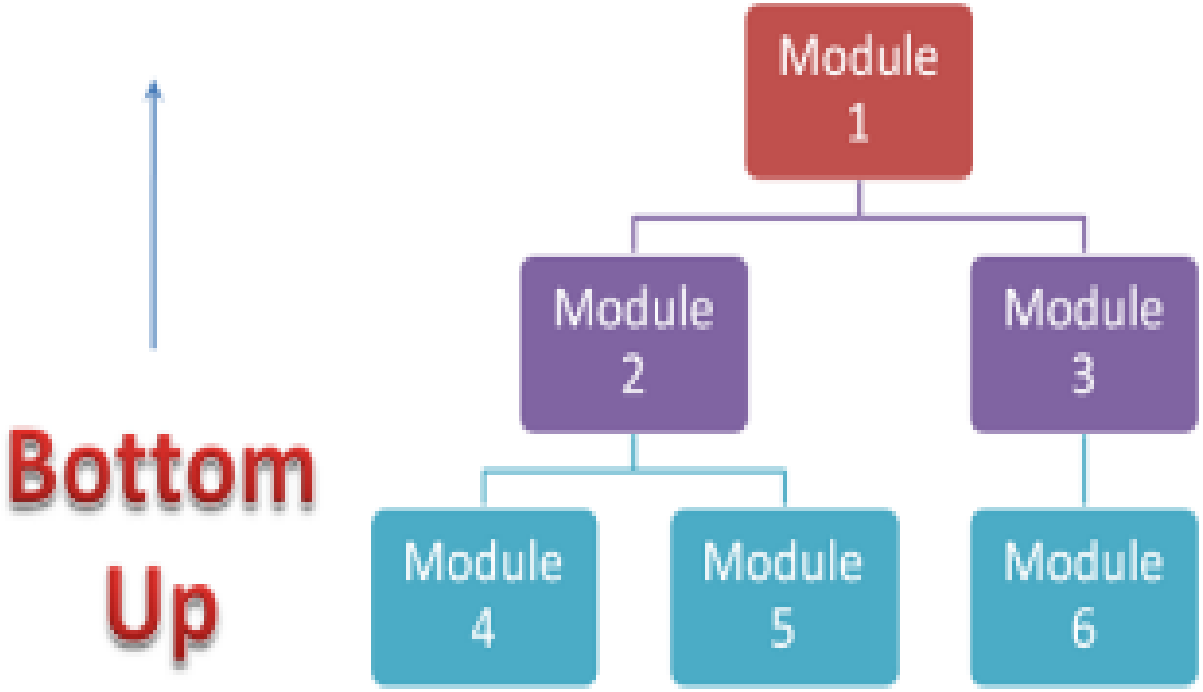
- Fault localization is difficult.
- Given the sheer number of interfaces that need to be tested in this approach, some interface links to be tested could be missed easily.
- Since the integration testing can commence only after "all" the modules are designed, testing team will have less time for execution in the testing phase.
- Since all modules are tested at once, high risk critical modules are not isolated and tested on priority. Peripheral modules which deal with user interfaces are also not isolated and tested on priority.

Incremental Approach

- In this approach, testing is done by joining two or more modules that are logically related. Then the other related modules are added and tested for the proper functioning.
- The process continues until all of the modules are joined and tested successfully.
- Incremental approach in turn is carried out by the following methods:
  - **Bottom up**
  - **Top down**
  - **Sandwich**

Bottom up Integration

- In the bottom up strategy, each module at lower levels is tested with higher modules until all modules are tested.



Bottom up…

Advantages:

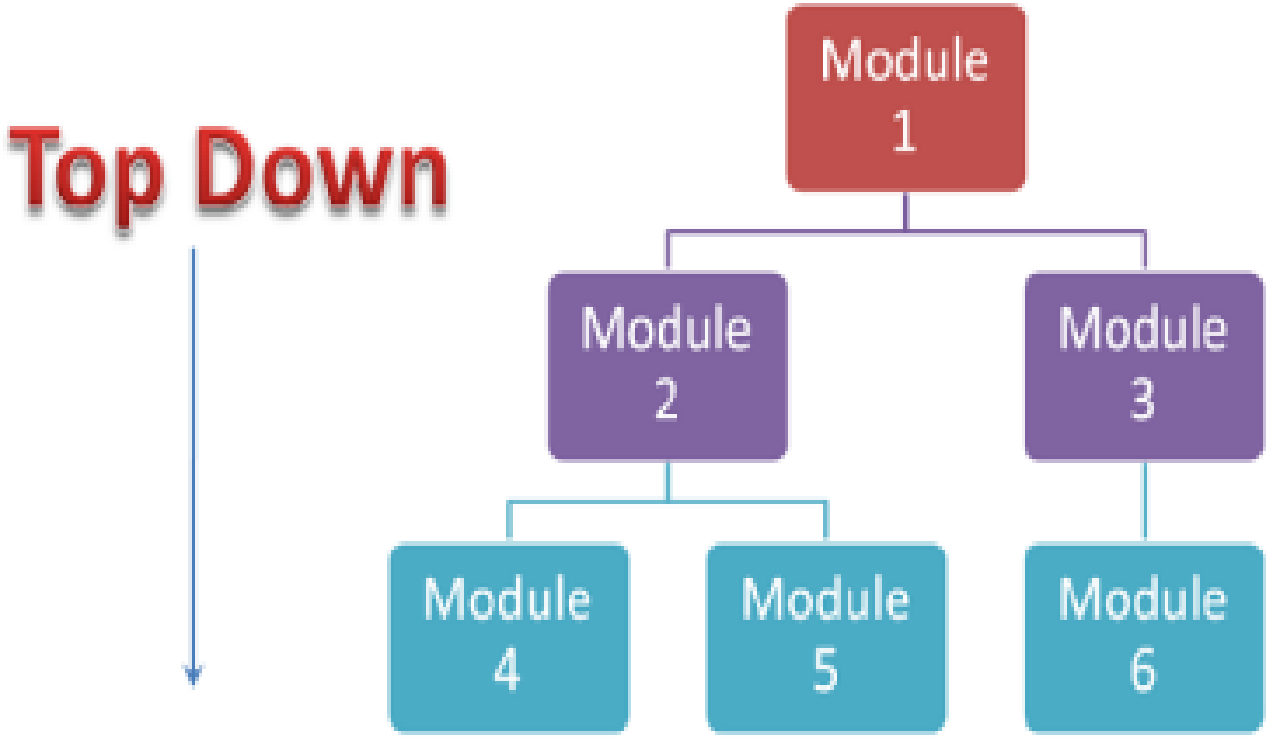
- Fault localization is easier.
- No time is wasted waiting for all modules to be developed unlike big-bang approach

Disadvantages:

- Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
- Early prototype is not possible

Top down Integration:

- In top-to-down approach, testing takes place from top to down following the control flow of the software system.



Top down…

Advantages:

- Fault localization is easier.
- Possibility to obtain an early prototype.
- Critical modules are tested on priority; major design flaws could be found and fixed first.
- Needs

Disadvantages:

- many stubs.
- Modules at lower level are tested inadequately.

Sandwich/Hybrid Integration Approach

- This approach is a combination of both top-down and bottom up integration testing.
- In this approach, the integration starts from middle layer and testing is carried out in both directions - towards higher level modules(upwards) and towards lower level module (downwards).
- This way includes benefits of both top-down and bottom up approach and helps in faster testing of the module's interfaces.

System Testing (End-to-End Testing)

- This is the first level of functional testing where the application is tested as a whole.
- The goal at this level is to evaluate systems compatibility with the outlined requirements and to see that it meets the anticipated quality standards.
- To verify the functional and technical specifications of the software, comprehensive testing is performed.
- The application is tested in an environment that is similar to the environment where the application will be deployed.
- Enables the testers to test, verify, and validate the business requirements as well as the application architecture.
- Performed by a knowledgeable and specialized testing team.

Users Acceptance Testing (UAT)

- Acceptance testing or user acceptance testing, is conducted by the Quality Assurance (QA) Team or customers to check whether the application meets the intended specifications and satisfies clients' requirements.
- Acceptance testing allows the tester to check for any undetected bugs, which can crash the system in future.
- Once acceptance testing is accomplished successfully, the application becomes suitable for release.
- The QA team must have a set of pre-written scenarios and test cases that will be used to test the application.

… UAT

- More ideas will be shared about the application and more tests can be performed on it to device its accuracy and the reasons why the project was initiated.
- Acceptance tests are not only intended to point out simple spelling mistakes, cosmetic errors, or interface gaps, but also to point out any bugs in the application that will result in system crashes or major errors in the application.
- By performing acceptance tests on an application, the testing team will reduce how the application will perform in production.
- There are also legal and contractual requirements for acceptance of the system.
- UAT includes:

  a) Alpha Testing

  b) Beta Testing

  c) Gamma Testing

A) Alpha Testing

- Alpha testing is a type of acceptance testing performed to identify all possible issues/bugs before releasing the product to everyday users or public.
- It is performed within the software company or lab environment by highly qualified specialists usually not by developers of the product. The testers are internal employees of the organization.
- It can be referred to as a form of internal acceptance testing. Also in this stage, new functionality can be implemented in the software.
- The focus of this testing is to simulate real users by using black-box and white-box techniques.
- The aim is to carry out the tasks that a typical user might perform.
- This kind of testing is called alpha only because it is done early on, near the end of the development of the software, and before Beta Testing.

Alpha ...

- During this phase, the following aspects will be tested:
  - Spelling mistakes
  - Broken links
  - Cloudy directions
  - The application will be tested on machines with the lowest specification to test loading times and any latency problems.

Alpha …

Advantages:

- Provides better view about the reliability of the software at an early stage
- Helps simulate real time user behavior and environment.
- Detect many showstopper or serious errors
- Ability to provide early detection of errors with respect to design and functionality

Disadvantages:

- In-depth functionality cannot be tested as software is still under development stage
- Sometimes developers and testers are dissatisfied with the results of alpha testing

Beta Testing

- Beta testing is done outside the software developing organization, not by its personnel.
- Prerelease builds of the program are distributed to the end users who validate their functionality, reliability, usability and compatibility. It is known as a form of external acceptance testing.
- Beta testing lowers the risk of the product's failure and provides improved quality of the application by means of customer feedback.
- Direct feedback is a significant benefit of beta testing. This phase follows alpha testing.
- Beta testers often find a lot of defects of programs.

Beta…

- In this phase of software production, the development team bears no responsibility for the damage caused to the products when they are tried out by the clients or outsiders.
- No testing or lab environment is needed for beta tests.
- Beta testing companies engage a limited quantity of end users in the testing process to check the product in real world environment and decide if it is ready to be shipped to the customers.
- It aids partly to give the program a "real-world" test and partly to provide a preview of the next release.

Beta…

- In this phase, the audience tests the following:
  - Users install, run the application and send their feedback to the project team.
  - Typographical errors, confusing application flow and even crashes.
  - Getting the feedback, the project team can fix the problems before releasing the software to the actual users.
- The more issues you fix that solve real user problems, the higher the quality of your application will be.
- Having a higher-quality application when you release it to the general public will increase customer satisfaction.

Beta…

Advantages

- Reduces product failure risk via customer validation.
- It allows a company to test post-launch infrastructure.
- Improves product quality via customer feedback
- Cost effective compared to similar data gathering methods
- Creates goodwill with customers and increases customer satisfaction

Disadvantages

- Test management is an issue. As compared to other testing types which are usually executed inside a company in a controlled environment, beta testing is executed out in the real world where you seldom have control.
- Finding the right beta users and maintaining their participation could be a challenge

Types of Beta Testing

- There are different types of beta tests:

1. **Traditional beta testing** - Product is distributed to the target market, and related data is gathered in all aspects. This data can be used for product improvement.
2. **Public beta testing** - Product is publicly released to the outside world via online channels and data can be gathered from anyone. Based on feedback, product improvements can be done. For example, Microsoft conducted the largest of all Beta Tests for its OS -- Windows 8 before officially releasing it.

Types of Beta ...

3. Technical beta testing - Product is released to the internal group of an organization and gather feedback/data from the employees of the organization.

4. Focused beta testing- Product is released to the market for gathering feedback on specific features of the program. For example, important functionality of the software.

5. Post release beta testing - Product is released to the market and data is gathered to make improvements for the future release of the product.

Gamma Testing

- **Gamma testing** is a final stage of software testing before an appropriate public release, it is conducted primarily for safety purposes.
- It is intended to identify and fix minor defects that were found during beta testing.
- As a rule, this process is carried out with the involvement of as many end users as possible.
- In this final phase of software development, there is no need for going through all the in-house testing activities.
- Gamma tests can be run only when the application is ready for deployment to the specified requirements.

THE END
THANK YOU!!!

Debre Markos University
Institute of Technology
School of Computing, Software Eng A/program

Software Quality Assurance and Testing (SEng4112)

Chapter Four

Types of Software Testing

Objective

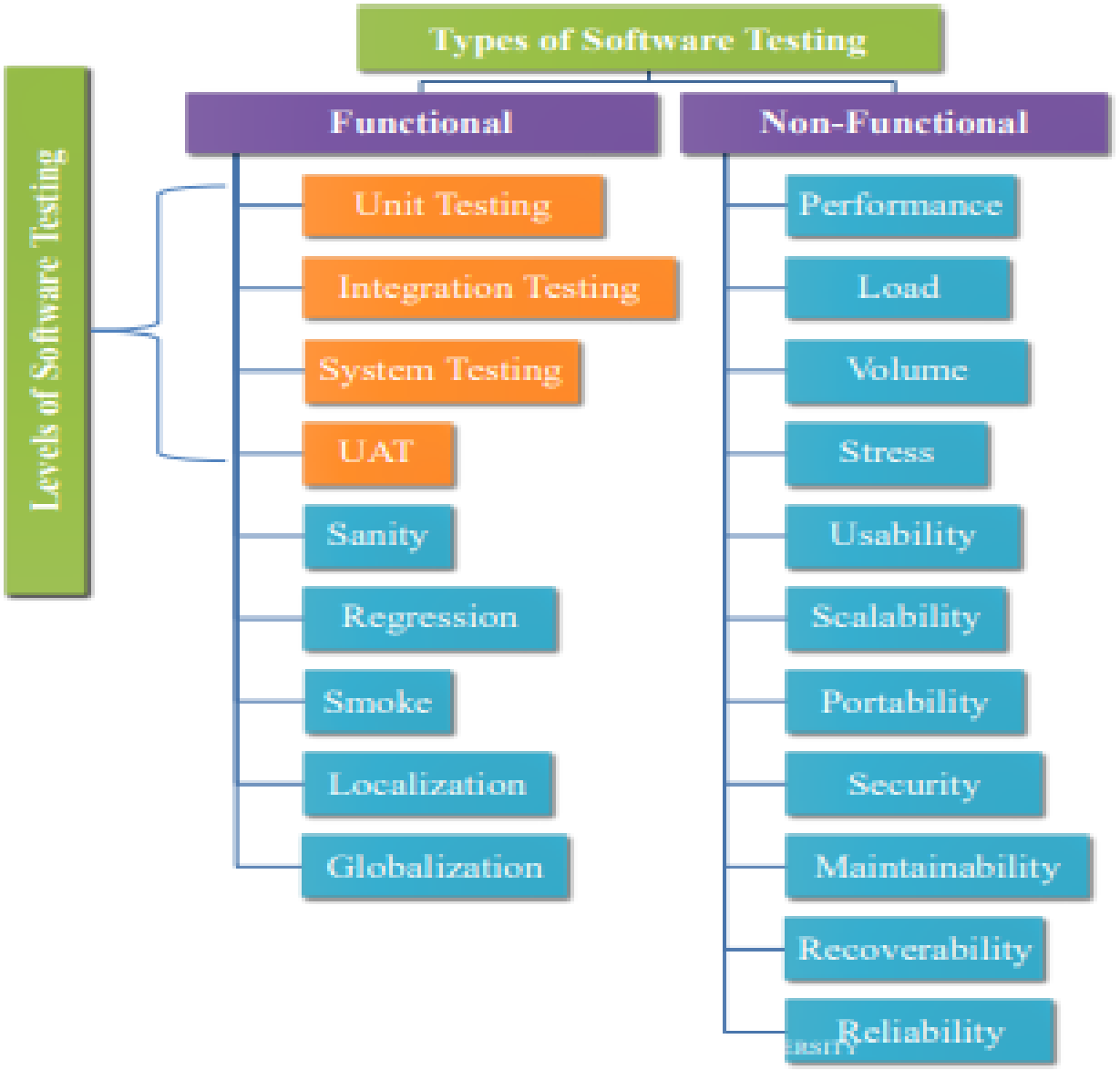
- **After successful completion of this chapter, students will be able to explain**
  - ✓ Software testing types
    - Functional software testing
    - Non-functional software testing

Software Testing Types

- The software testing is generally divided into two major categories, which further consist of numerous testing methods within them

    1. Functional testing
    2. Non-functional testing

Cont…


Types of Software Testing
Functional
Non-Functional
Levels of Software Testing
Unit Testing
Integration Testing
System Testing
UAT
Sanity
Regression
Smoke
Localization
Globalization
Performance
Load
Volume
Stress
Usability
Scalability
Portability
Security
Maintainability
Recoverability
Reliability

Functional Testing

- Functional testing is a software testing procedure used within software development wherein the software is tested to check that a software has all the desired functionality specific to its functional requirements.
- This type of testing is carried out based on the product's functional specification document.
- Functional testing is always done against the application's business requirement that means it is a black box testing type.
- Moreover, functional testing is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

Cont…

- Functional testing includes any user commands, data manipulation, searches, business processes, user screens, and integrations.
- This testing is done using the functional specifications provided by the client or by using the design specifications like use cases provided by the design team. So, this type of testing is strictly specific.
- For functional testing either manual testing or automation tools can be used but functionality testing would be easier using manual testing only.
- Functional tests tend to answer the question of
  - Can the user do this? or
  - Does this particular feature work?

Functional testing includes:

- Unit testing
- Integrity testing
- System testing
- User Acceptance Testing (UAT)
- Regression testing
- Sanity testing
- Smoke testing
- Localization testing
- Globalization testing

Regression Testing

- Whenever a change in a software application is made, it is quite possible that other areas within the application will be affected by the change.
- It is performed to verify that a fixed bug hasn't resulted in another functionality or business rule violation.
- The intent of regression testing is to ensure that a change, such as a bug fix should not result in another fault being uncovered in the application.

Cont…

- Regression testing is done due to the following reasons:
  - Minimizing the gaps in testing when an application with changes made has to be tested
  - Testing the new changes to verify that the changes made did not affect any other area of the application
  - Mitigates risks
  - Increasing test coverage without compromising timelines
  - Increasing speed to market the product

Mutation testing

- Mutation testing is a type of software testing where we mutate (alter/change) certain statements in the source code intentionally and check if the test cases are able to find the errors.
- It is a type of White Box Testing which is mainly used for unit testing.
- To determine if a set of test data or test cases are useful by deliberately introducing various bugs.
- Re-testing with the original test data/cases to determine if the bugs are detected.

Smoke Testing

- Smoke testing is a kind of software testing performed after software build to ascertain that the critical functionalities of the program are working fine.
- It is executed "before" any detailed functional or regression tests are executed on the software build.
- The purpose is to reject a badly broken application, so that the QA team does not waste time installing and testing the software application.
- In smoke testing, the test cases chosen cover the most important functionality or component of the system.
  - For example, a typical smoke test would be - check that the application launches successfully, Check that the GUI is responsive … etc.

Cont…

- Smoke testing makes sure that build received from the development team is testable or not.
- It is also named as the "Day 0" check and performed on the building level.
- It saves time as you don't have to test the whole application if the core functionalities of the program are not working.

Sanity Testing

- Sanity testing is a kind of software testing performed after receiving a software build, with minor changes in code, or functionality, to ascertain that the bugs have been fixed and no further issues are introduced due to these changes.
- The goal is to determine that the proposed functionality works roughly as expected.
- If sanity test fails, the build is rejected to save the time and costs involved in a more rigorous testing.
- Ensure that introduced changes doesn't affect existing functionalities.

Localization (L10N) Testing

- Localization refers to the adaptation of a product, application or document content to meet the language, cultural and other requirements of a specific target market (a locale).
- Localization is sometimes written as L10N, where 10 is the number of letters between L and N.
- Often thought of only as a synonym for translation of the user interface and documentation, localization is often a substantially more complex issue.

Cont…

- **It can entail customization related to:**
  - Numeric, date and time formats
  - Use of currency
  - Keyboard usage
  - Collation and sorting
  - Symbols, icons and colors
  - Text and graphics containing references to objects, actions or ideas which, in a given culture, may be subject to misinterpretation or viewed as insensitive.
  - Varying legal requirements

Globalization (G11N) Testing

- Globalization is the design and development of a product, application or document content that enables easy localization for target audiences that vary in culture, region, or language.
- Globalization is often written G11N, where 11 is the number of letters between G and N in the English word.
- Globalization testing is to ensure that application can function in any culture or locale (language, territory and code page)
- It is also called internationalization testing.

Cont...

- **Globalization testing typically entails:**
  - Designing and developing in a way that removes barriers to localization or international deployment.
  - Providing support for features that may not be used until localization occurs.
  - Enabling code to support local, regional, language, or culturally related preferences.
  - Separating localizable elements from source code or content, such that localized alternatives can be loaded or selected based on the user's international preferences as needed

2. Non-Functional

- This section is based upon testing an application from its non-functional attributes.
- Non-functional testing involves testing a software from the requirements which are non-functional in nature but important such as performance, security, user interface, etc.
- Some of the important and commonly used non-functional testing types are discussed below.

Performance Testing

- It is mostly used to identify any bottlenecks or performance issues rather than finding bugs in a software.
- There are different causes that contribute in lowering the performance of a software:
  - Network delay, Client-side processing, Database transaction processing, Load balancing between servers, Data rendering, etc.
- Performance testing is considered as one of the mandatory testing types in terms of the following aspects:
  - Speed (i.e. response time, data rendering and accessing)
  - Capacity – number of users or operations that your application can handle
  - Stability - ability to remain stable after modification process
  - Scalability - the capacity to be changed in size or scale
- Performance testing can be either qualitative or quantitative and can be divided into different sub-types such as Load testing, Stress testing & Volume.

Load Testing

- It is a process of testing the behavior of a software by applying maximum load in terms of software accessing and manipulating large input data.
- It can be done at both normal and peak load conditions.
- This type of testing identifies the maximum capacity of software and its behavior at peak time.
- Virtual users (VUsers) are defined in the automated testing tool and the script is executed to verify the load testing for the software.
- The number of users can be increased or decreased concurrently based upon the requirements.
- Most of the time, load testing is performed with the help of automated tools such as Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Studio Load Test, etc.

Volume Testing

- Volume testing is a non-functional performance testing where the software is subjected to a huge volume of data. It is also referred as flood testing.
- Volume testing is done to analyze the system performance by increasing the volume of data in the database.
- With the help of volume testing, the impact on response time and system behavior can be studied when exposed to a high volume of data.
  - For example, testing a music site behavior when there are millions of user to download the song.

What to test in volume testing?

- **In volume testing, the following things need to be tested**
  - Check if there is any data loss
  - Check the system's response time
  - Check if the data is stored correctly or not
  - Verify if the data is overwritten without any notification
  - Check for warning and error messages, whether it comes at all for volume problems
  - Check whether high volume data affects the speed of processing

Stress Testing

- Stress testing (sometimes called torture testing) is a form of deliberately intense or thorough testing used to determine the stability of a given system or entity.
- This testing is done beyond normal operational capacity; often to the breaking point, in order to observe the results.
- Stress testing includes testing the behavior of a software under abnormal conditions. **For example:**
  - Shutdown or restart of network ports randomly
  - Turning the database on or off
  - Taking away some resources
  - Applying a load beyond the actual load limit
  - Running different processes that consume resources such as CPU, memory, server, etc.

Recoverability Testing

- Recovery testing is the activity of testing how well an application is able to recover from crashes, hardware failures and other similar problems.
- Recovery testing is basically done in order to check how fast and better the application can recover from any type of crash or hardware failure or other catastrophic problems

Examples:

- This can be understood with a simple example of a browser. Say we have opened a browser with 6 sessions on it. Close it abruptly and relaunch it and check whether it is able to recover or not.

Usability Testing

- It is a black box testing technique, which is used to identify errors and defects as well as the improvements in the software by observing the users through their usage and operation.
- This type of testing assists the tester in finding any undetected defect before the release of an application.
- According to Nielsen, usability can be defined in terms of five factors, i.e.
  - Efficiency of use, learnability, memorability, safety and satisfaction.
- Check accessibility
- Check language and understandability
- Check the proper help text message
- Check the navigation

Security Testing

- **Security Testing** is a type of Software Testing that uncovers vulnerabilities, threats, risks in a software application and prevents malicious attacks from intruders.
- Through this type of testing the tester verifies system security, whether it can protect data and maintain functionality as intended.

Cont…

- **Some of the most important concepts covered by security testing are:**
  - **Confidentiality:** protecting against the disclosure of information to parties other than the intended recipient
  - **Integrity:** protecting information from being modified by unauthorized parties
  - **Authentication:** confirming the identity of a person, artifact or product
  - **Authorization:** determining that a requester is allowed to receive a service or perform an operation.
  - **Non-repudiation**: ensuring that a transferred message has been sent and received by the parties claiming to have sent and received the message. Guarantee no denial from both parties.
  - URL Manipulation
  - Session Expiry
  - Privilege Elevation
  - Cross Site Request Forgery

Maintainability Testing

- It basically defines that how easy it is to maintain the system. This means that how easy it is to analyze, change and test the application
- Maintainability testing shall use a model of the maintainability requirements of the software/system.
- It is the degree of measuring the software or system potential to undergo changes, to meet the requirements.
- These requirements may include
  - Correction of errors or faults
  - Additional functionality
  - Adapting the changing environment,
  - Prevention of unexpected failures,
  - Future maintenance, etc.

Scalability Testing

- Scalability testing, is the testing of a software application to measure its capability to scale up or scale down
- Scalability testing is the ability of a network, a system, a database or a process to continue to function well when changes are done in size or volume of the system to meet growing need.
- Scalability testing ensures that an application can handle the projected increase in user traffic, data volume, transaction counts frequency, etc.

Reliability Testing

- Reliability testing is a type of testing to verify that software is capable of performing a failure-free operation for a specified period of time in a specified environment.
- Reliability means "yielding the same," in other terms, the word "reliable" mean something is dependable and that it will give the same outcome every time.
- The same is true for reliability testing. Reliability testing in software assures that the product is fault free and is reliable for its intended purpose.
  - Example:- The probability that a PC in a store is up and running for eight hours without crashing is 99%; this is referred as reliability.

Portability Testing

- Portability testing is the process of determining the degree of ease or difficulty to which a software component or application can be effectively and efficiently transferred from one hardware, software or other operational or usage environment to another.
- The following are the strategies that can be used for portability testing:
  - Building executable (.exe) file to run the software on different platforms.
  - Transferring a developed software from one computer to another.
- This testing type includes overall testing of a software with respect to its usage over different environments.
- Computer hardware, operating systems, and browsers are the major focus of portability testing.

Examples

- Software designed to run on Macintosh OS and Microsoft Windows operating systems.
- Applications developed to be compatible with Google Android and Apple iOS phones.
- Video Games or other graphic intensive software intended to work with OpenGL and DirectX API's.
- Software that should be compatible with Google Chrome and Mozilla Firefox browsers.

Cont…

- The ISO 29119 (2013) standard describes Portability with the attributes of

    1. Install-ability
    2. Compatibility
    3. Interoperability and
    4. Adaptability

A. Install-ability testing

- Install-ability of a software is tested on its ability to effectively install the target software in the intended environment.
- Install-ability may include tests for:
  - Space demand,
  - Checking prerequisites,
  - Installation procedures,
  - Completeness,
  - Installation interruption,
  - Customization,
  - Uninstallation

B. Compatibility/ Co-existence- Testing

- It tests the compatibility of multiple, unrelated software systems to co-exist in the same environment, without effecting each other's behavior.
- This is a growing issue with advanced systems, increased functionality and interconnections between systems and subsystems that share components.
- Components that fail this requirement could have profound effects on a system.

  For example, if 2 sub-systems share memory or a stack, an error in one could propagate to the other and in some cases cause complete failure of the entire system.

C. Adaptability testing

- Adaptability testing is done to verify that the software can perform all of its intended behaviors in each of the target environments.
- Using communication standards, such as HTML can help with adaptability.
- Adaptability may include testing in the following areas:
  - Hardware dependency
  - Software dependency
  - Representation dependency
  - Standard language conformance
  - Dependency encapsulation and/or
  - Text convertibility.

D. Interoperability testing

- This is testing the capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units.

THE END
THANK YOU!!!

Debre Markos University
Institute of Technology
School of Computing, Software Eng A/program

Software Quality Assurance and Testing (SEng4112)

Chapter Five

Software Testing Techniques

Objective

- **After successful completion of this chapter, students will be able to understand**
  - ✓ The static and dynamic software testing techniques
  - ✓ White box and black box testing techniques
  - ✓ The benefits and limitations of those testing techniques

Software Testing Techniques

- Software testing technique includes the various strategies or approaches used to test an application to ensure how it functions and behaves as expected.
- Methods in which the internal structure, design or implementation of a software is tested
- These are software testing techniques which the organization must choose carefully, and implement on the software application.
- In order to get the most out of each type of testing, and choose the right tools for a given situation, it's crucial to understand the benefits and limitations of each type of testing technique.

Cont…

- **There are two software testing techniques**

1. **Static Testing Techniques**
2. **Dynamic Testing Techniques**

1. Static Testing Techniques

- Static software testing techniques are used for testing code, requirements and design documents and put review comments on the work document.
- In this type of testing, the code is not executed.
- It can be done manually or by a set of tools/software.
- With static testing, we try to find out the errors, code flaws and potentially malicious code in the software application.
- It starts earlier in development life cycle and hence, it is also called verification testing technique.

Cont…

- **In static testing technique, we use work documents such as:**
  - Requirement specifications
  - Design document
  - Source code
  - Test plans
  - Test cases
  - Test scripts
  - Help or user document

Static testing technique

- Static testing technique includes:
  - ✓ Informal Review
  - ✓ Walkthrough
  - ✓ Technical Review
  - ✓ Static Code Review
  - ✓ Management Review
  - ✓ Inspection
  - ✓ Audit

1.1. Informal Review

- Informal reviews are applied many times during the early stages of the life cycle of the document.
- A two persons team can conduct an informal review.
- In later stages, these reviews often involve more people and a meeting.
- The most important thing to keep in mind about the informal reviews is that they are not documented.

1.2. Walkthrough

- Walkthrough is not a formal process of review and is carried out by authors.
- The authors guide the participants through the document and try to understand their thought processes, in order to gain feedback and arrive at a common understanding.
- The author of the work product explains the product to his/her team.
- Participants can ask questions if any.
- Meeting is led by the author.

Cont…

- Scribe makes note of review comments
- Useful for the people if they are not from the software discipline, who are not used to or cannot easily understand software development process.
- Is especially useful for higher level documents like requirement specification, etc.

1.3. Technical review

- A technical review is carried out by a trained moderator or technical expert.
- It is less formal as compared to an inspection.
- It is usually carried out as a peer review without any participation from the management.
- A team consisting of your peers, review the technical specification of the software product and checks whether it is suitable for the project.
- They try to find any discrepancies in the specifications and standards followed.
- This review concentrates mainly on the technical documents related to the software such as test strategy, test plan and requirement specification documents.

1.4. Static code review

- This is a systematic review of the software source code, without executing the code.
- It checks the syntax of the code, coding standards, code optimization, etc.
- This review can be done at any point during development.
- It is basically applied in two ways:
  1. Rule-based Static Analysis
  2. Flow-based Static Analysis

1.5. Management Review

- The management review is a cross-functional review undertaken by an organization's top management.
- It includes analyses of
  - ✓ customer satisfaction,
  - ✓ determination of the cost of poor quality,
  - ✓ performance trends and
  - ✓ achievement of objectives defined in the business plan.

1.6. Inspection

- An inspection is the most formal type of review and is conducted by trained moderators.
- During an inspection, the documents and products are examined through a process of reviews and defects are identified following strict process to find the defects.
- The defects identified are documented in a logging list or issue log and a formal follow-up is carried out.
- The main purpose is to find defects, and the meeting is led by trained moderator.
- Reviewers have checklist to review the work products.
- They record the defect and inform the participants to rectify those errors.

1.7. Audit

- An audit is an independent examination of all the documents related to the software, carried out by an external agent.
- Audits provide increased assurance to the various stakeholders of the project that the document is up to standards and devoid of any defects.

2. Dynamic Test Techniques

- Dynamic testing involves execution of the test object (application) on a computer.
- This testing technique is also called execution technique or validation testing
- Dynamic testing is performed in runtime environment. When the code being executed is input with a value, the result or the output of the code is checked and compared with the expected output. With this we can observe the functional behavior of the software, monitor the system memory, CPU response time, performance of the system.

Cont…

- Dynamic testing is carried out during the validation process.
- Dynamic test design techniques can be further classified into:
  - **Specification-based (black-box, also known as behavioral techniques)**
  - **Structure-based (white-box or structural techniques)**
  - **Experience-based**

2.1. Specification-Based / Black Box Testing Technique

- Specification-based testing technique, also known as the black box testing technique uses the software's external descriptions such as the
  - technical specifications,
  - design,
  - customer requirements, etc.
- This implies that a tester who does not have any knowledge about the code or internal structure can also perform the test.

Cont…

- **Some of the well-known methods in this technique are:**
  - Equivalence Partitioning (EP)
  - Boundary Value Analysis (BVA)
  - Decision Table Testing
  - State Transition Testing

2.1.1. Equivalence Partitioning (EP)

- Equivalent partitioning (EP) or Equivalent Class Partitioning (ECP) is a black box testing technique (code is not visible to tester) which can be applied to all levels of testing like unit, integration, system and UAT.
- The idea behind this technique is to divide/partition a set of test conditions into groups or sets that can be considered the same or equivalent. This is used for reducing the total number of test cases to a finite set of testable test cases, still covering maximum requirements.
- One test value is picked from each class while testing. This is because we are assuming that all the conditions in one partition will be treated in the same way by the software.

Cont…

- If one condition in a partition works, we assume all of the conditions in that partition will work and if one condition fails, it is assumed that all others in the partition will fail and there is no point in testing others.
- In this technique, you divide the set of test conditions into a partition that can be considered the same.
  - It divides the input data of software into different equivalence data classes.
  - You can apply this technique, where there is a range in input field.

Cont...

- Let's consider the behavior of tickets in the flight reservation application, while booking a new flight.
- Ticket values 10 to 50 are considered valid & ticket is booked. While other values are considered invalid for reservation

Invalid	Valid	Invalid
<10	10-50	>50

Cont…

- The divided sets are called Equivalence Partitions or Equivalence Classes.
- Then we pick only one value from each partition for testing. ◦ The hypothesis behind this technique is that if one condition/value in a partition passes all others will also pass.
- Likewise, if one condition in a partition fails, all other conditions in that partition will fail.

2.1.2. Boundary Value Analysis (BVA)

- BVA is based on testing the boundary values of valid and invalid partitions.
- The behavior at the edge of each equivalence partition is more likely to be incorrect than the behavior within the partition, so boundaries are an area where testing is likely to yield defects.
- Every partition has its maximum and minimum values and these values are the boundary values of a partition.
- A boundary value for a valid partition is a valid boundary value. Similarly a boundary value for an invalid partition is an invalid boundary value.

Cont…

- Tests can be designed to cover both valid and invalid boundary values.
- When designing test cases, a test for each boundary value is chosen.
- This technique determines whether a certain range of values are acceptable by the system or not. It is very useful in reducing the number of test cases.
- Boundary value analysis is most common when checking a range of numbers.
- We should design test cases which exercise the program functionality at the boundaries, with values just inside and just outside the boundaries.

Cont…

- Here, the assumption is that if the program works correctly for these extreme cases, then it will work correctly for all values in between the valid partition.
- Testing has shown that defects that arise when checking a range of values the most defects are near or at the boundaries.

Cont...

- For each boundary, we test +/-1 in the least significant digit of either side of the boundary.
- Boundary value analysis can be applied at all test levels.

Example: Assume, we have to test a field which accepts Age between 18 – 56,

- Min-1
- Min
- Min+1
- Nominal
- Max-1
- Max
- Max+1

AGE Enter Age *Accepts value 18 to 56

BOUNDARY VALUE ANALYSIS		
Invalid (min -1)	Valid (min, +min, -max, max)	Invalid (max +1)
17	18, 19, 55, 56	57

2.1.3. Decision Table Testing

- A decision table is a tabular representation of inputs versus rules/cases/test conditions.
- Decision table testing is a testing technique used to test a system for different input combinations.
- This is a systematic approach where the different input combinations and their corresponding system outputs are captured in a tabular form.
- That is why it is also called as a Cause-Effect table where causes and effects are captured for better test coverage.

Cont…

- It also helps in better test coverage for complex business logic.
- The significance of this technique becomes immediately clear as the number of inputs increases.
- Number of possible combinations is given by 2^n, where n is the number of inputs.
- For n = 10, which is very common in the web based testing, having big input forms, the number of combinations will be 1024. Obviously, you cannot test all but you will choose a rich sub-set of the possible combinations using decision based testing technique

Cont...

- Let's create a decision table for a login screen.
- The condition is simple if the user provides correct username and password, the user will be redirected to the homepage. If any of the input is wrong, an error message will be displayed.

Legend:

- T - Correct username/password
- F - Wrong username/password
- E - Error message is displayed
- H - Home screen is displayed

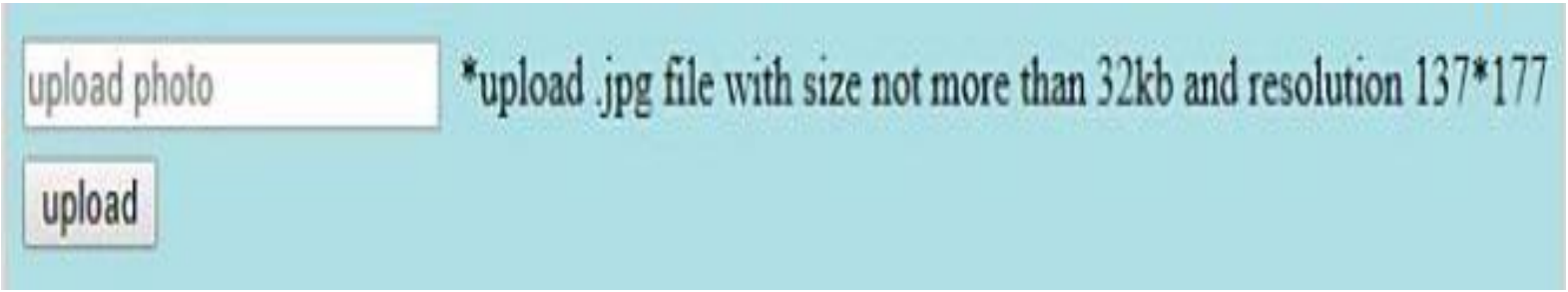
Conditions	Case 1	Case 2	Case 3	Case 4
Username (T/F)	F	T	F	T
Password (T/F)	F	F	T	T
Output (E/H)	E	E	E	H

Cont…

- Interpretation:
- Case 1 – Username and password both were wrong. The user is shown an error message.
- Case 2 – Username was correct, but the password was wrong. The user is shown an error message.
- Case 3 – Username was wrong, but the password was correct. The user is shown an error message.
- Case 4 – Username and password both were correct, and the user navigated to homepage

Cont…

- Decision Table for “Upload Screen”
- Now consider a dialogue box which asks the user to upload photo with certain conditions like
  1. You can upload only '.jpg' image format
  2. File size less than 32kb
  3. Resolution 137x177.
- If any of the conditions fails, the system will throw a corresponding error message stating the issue; and if all conditions are met, photo will be updated successfully



Cont…

- Let's create the decision table for this case.

Conditions	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7	Case 8
Format	.jpg	.jpg	.jpg	.jpg	Not .jpg	Not .jpg	Not .jpg	Not .jpg
Size	Less than 32kb	Less than 32kb	More than 32kb	More than 32kb	Less than 32kb	Less than 32kb	More than 32kb	More than 32kb
resolution	137*177	Not 137*177	137*177	Not 137*177	137*177	Not 137*177	137*177	Not 137*177
Output	Photo uploaded	Error message resolution mismatch	Error message size mismatch	Error message size and resolution mismatch	Error message for format mismatch	Error message format and resolution mismatch	Error message for format and size mismatch	Error message for format, size, and resolution mismatch

Cont…

- For this condition, we can create 8 different test cases and ensure complete coverage based on the above table.

1. Upload a photo with format '.jpg', size less than 32kb and resolution 137*177 and click on upload. Expected result is Photo should upload successfully

2. Upload a photo with format '.jpg', size less than 32kb and resolution not 137*177 and click on upload. Expected result is Error message resolution mismatch should be displayed

3. Upload a photo with format '.jpg', size more than 32kb and resolution 137*177 and click on upload. Expected result is Error message size mismatch should be displayed

4. Upload a photo with format '.jpg', size less than 32kb and resolution not 137*177 and click on upload. Expected result is Error message size and resolution mismatch should be displayed

Cont…

5. Upload a photo with format other than '.jpg', size less than 32kb and resolution 137*177 and click on upload. Expected result is Error message for format mismatch should be displayed

6. Upload a photo with format other than '.jpg', size less than 32kb and resolution not 137*177 and click on upload. Expected result is Error message format and resolution mismatch should be displayed

7. Upload a photo with format other than '.jpg', size more than 32kb and resolution 137*177 and click on upload. Expected result is Error message for format and size mismatch should be displayed

8. Upload a photo with format other than '.jpg', size more than 32kb and resolution not 137*177 and click on upload. Expected result is Error message for format, size and resolution mismatch should be displayed

2.1.4. State Transition Testing

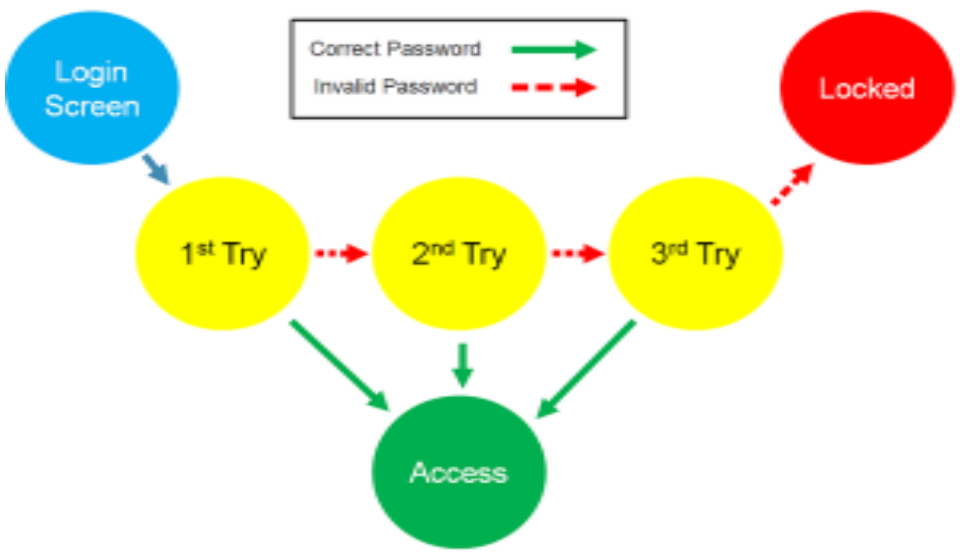
- In state transition testing technique, changes in input conditions cause state changes in the application under test.
- In this technique, the tester analyzes the behavior of an application under test for different input conditions in a sequence.
- The tester provides both valid and invalid input test values and record the system behavior.
- Any system where you get a different output for the same input, depending on what has happened before, is a finite state system.
- This technique is helpful where you need to test different system transitions.

Cont…

- When to use state transition?
  - ✓ When the tester is trying to test sequence of events that occur in the application under test i.e., this will allow the tester to test the application behavior for a sequence of input values.
  - ✓ When the system under test has a dependency on the events/values in the past.
  - ✓ This can be used when a tester is testing the application for a finite set of input values.
- When to not rely on state transition?
  - ✓ When the testing is not done for sequential input combinations.
  - ✓ If the testing is to be done for different functionalities like exploratory testing

Cont...

- Example:- Let's consider an ATM system function where if the user enters the invalid password three times, the account will be locked.
- In this system,
  - If the user enters a valid password in any of the first three attempts the user will be logged in successfully.
  - If the user enters the invalid password in the first or second, the user will be asked to re-enter the password.
  - And finally, if the user enters incorrect password 3rd time, the account will be blocked.



2.2. Structure-based / White Box Test Technique

- The exact opposite of the black box test design technique, white box test design technique necessitates knowledge of the internal structure of a program.
- Some of the methods implemented in this type of technique are:
  - Statement coverage
  - Decision coverage
  - Condition coverage
  - Multiple condition coverage
  - Control flow based criteria
  - Data Flow Testing

2.2.1. Statement coverage

- Statement coverage technique involves execution of all statements of the source code at least once.
- It is used to calculate the total number of executed statements in the source code out of the total statements present in the source code.
- Based on the input given to the program, some code statements are executed and some may not be executed.
- The goal of statement coverage technique is to cover all the possible executing statements and path lines in the code.
- $Statement\ coverage = \frac{Number\ of\ executed\ satements}{Total\ number\ of\ statements} * 100$

- **Example**

```
Test (int a, int b) {
int sum = a+b;
if (sum>0)
Print ("This is a positive result")
else
Print ("This is negative result")
}
```

Scenario 1: If a = 5 and b = 4

Total number of statements = 7
Number of executed statements = 5
Statement coverage = 5/7*100
= 500/7
= 71%

Scenario 2: If a = -2 and b = -4

Statement coverage = 6/7*100
= 6/7 *100
= 85%

But, we can see all the statements are covered in both scenarios and we can consider that the overall statement coverage is 100%.

2.2.2. Decision coverage technique

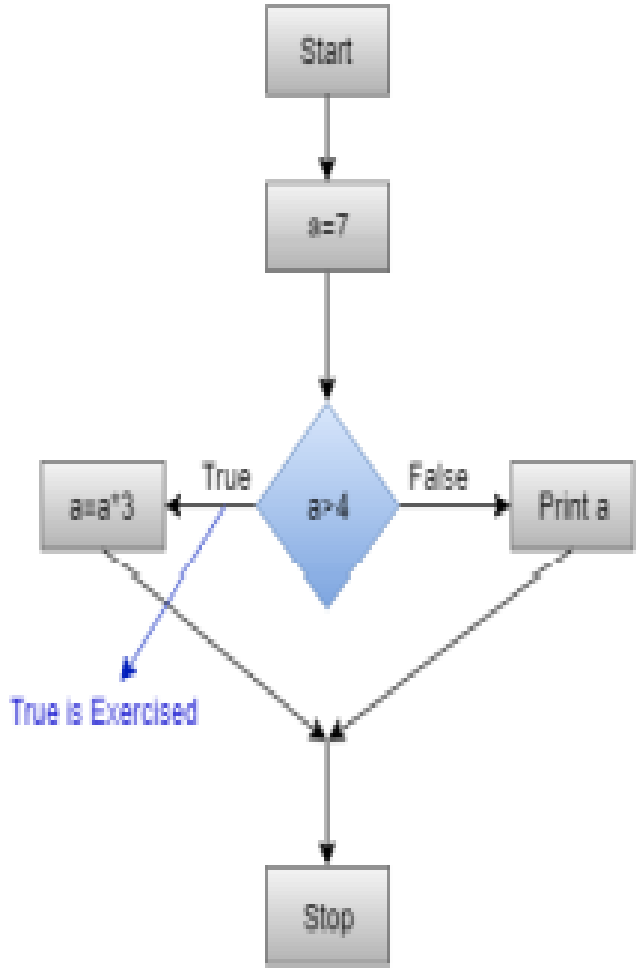
- This technique reports true and false outcomes of Boolean expressions.
- Whenever there is a possibility of two or more outcomes from the statements like **do while statement, if statement and case statement** (Control flow statements), it is considered as decision point because there are two outcomes either true or false.
- Decision coverage covers all possible outcomes of each and every Boolean condition of the code by using control flow graph or chart.
- $$\textit{Decision coverage} = \frac{\textit{Number of Decision Outcomes Exercised}}{\textit{Total Number of Decision Outcomes}} * 100$$

- Example 1

```
Test (int a)
{if (a>4)
a=a*3
Print (a)
}
```

Scenario 1: **If a=7**

The control flow graph when the value of a is 7



Only "True" is exercised

Decision Coverage = ½*100

Decision Coverage is 50%

2.2.3. Condition Coverage

- Condition coverage or predicate coverage ensures whether all the Boolean expressions have been evaluated to both TRUE and FALSE.
- In condition coverage the possible outcomes of ("true" or "false") for each condition are tested at least once. This means that each individual condition is one time true and false.
- Also the combinations of conditions are not relevant.
- Since there are only two possible outcomes of a condition (true or false), condition coverage results in 2 test situations per decision point.
- Example:

```
if (A OR B)
  {
    Print Z
  }
```

Cont...

Truth table

A	B	A OR B
1	1	1
1	0	1
0	1	1
0	0	0

Condition coverage
The possible outcomes ("true" or "false") of each condition are tested at least once.

IF number of books > 8 **OR** sum ≥ 100 THEN extra discount

	Number of books >8	Sum ≥ 100	Outcome
TS1	1	0	1 (extra discount)
TS2	0	1	1 (extra discount)

For condition coverage, only two test situations are needed per decision point

The condition "Number of books > 8" is one time True and one time False

For condition "Sum ≥ 100" is also one time True and one time False

Notice that the outcomes of the decision do not need to vary

2.2.4. Multiple Condition Coverage

- Every combination of 'true' or 'false' for the conditions related to a decision has to be tested in this technique.

Example

- Assume we want to test the following code extract:

```
if ( (A || B) && C )
    {
      /* instructions */
    }
  else
    {
      /* instructions */
    }
```

- where A, B and C represent atomic Boolean expressions (i.e. not divisible in other Boolean sub-expressions).

Cont…

- In order to ensure Multi-condition coverage criteria for this example, each combination of values for A, B and C should be tested at least one time.
- So, in our example, we need $2^3=8$ tests leading to the following evaluations of A, B and C to valid Multi-condition coverage:

A = true / B = true / C = true

A = true / B = true / C = false

A = true / B = false / C = true

A = true / B = false / C = false

A = false / B = true / C = true

A = false / B = true / C = false

A = false / B = false / C = true

A = false / B = false / C = false

Some languages evaluate the second operand of a logical AND only if the first operand is true (or evaluate the second operand of OR only if the first operand evaluates to false).

Cont…

- However, some programming languages such as C, C++, C#, Java Script etc, ensure multi-condition coverage using the results of tests in the following way:

  A = true / B = not evaluation / C = true

  A = true / B = not evaluation / C = false

  A = false / B = true / C = true

  A = false / B = true / C = false

  A = false / B = false / C = not evaluation

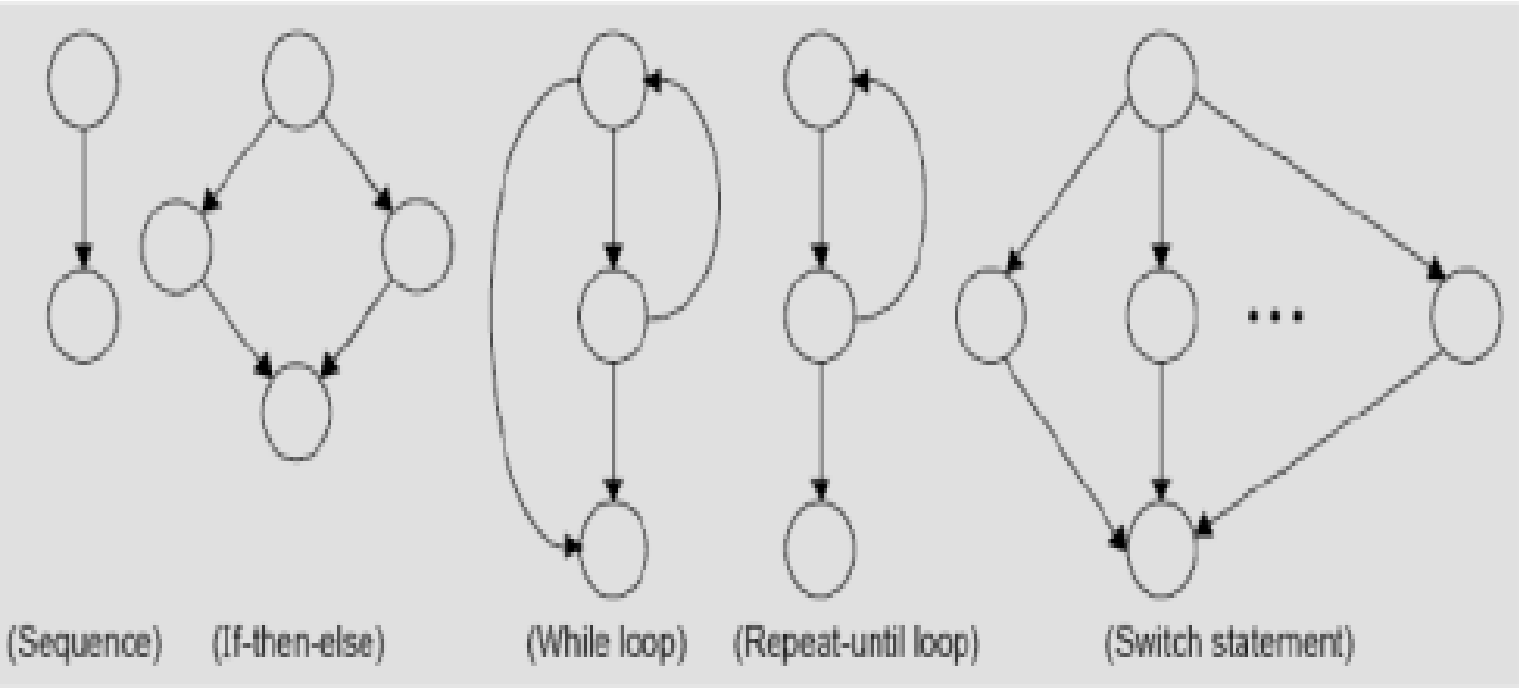
- E.g., for the 1st case, if A is "true", system will not calculate the value of B since in every way (A or B) will be evaluated to "true". In fact, it means that the 1st and 2nd cases of the 1st list will be evaluated in the same way by the system and that will follow this logic while presenting the results.

2.2.5 Control Flow Based Criteria

- Control flow testing is **a form of white-box testing where the implementation of the code is known to the tester**.
- The development team often performs control flow testing.
- This process determines and observes the execution paths of a program in a structured way.
- The technique used to develop the test cases.
- The **control flow** of a program can be analyzed using a graphical representation known as flow graph.
- The **flow graph** is a directed graph in which nodes are either entire statements or fragments of a statement, and edges represents flow of control.

Cont…

- **The basic construct of the flow graph**



Cont…

- **On a flow graph:**
  - Arrows called edges represent flow of control
  - Circles called nodes represent one or more actions.
  - Areas bounded by edges and nodes called regions.
  - A predicate node is a node containing a condition

Decision – to – Decision (DD) path Graph

- When we have a flow graph, we can easily draw another graph that is known as **decision-to-decision** or (DD) path graph, wherein we lay our main focus on the decision nodes only.
- The **nodes of the flow graph** are combined into a **single node** if they are in sequence.
- A path through a program is a node and edge sequence from the starting **node to a terminal** node of the control flow graph of a program.
- If a path has one new node compared to all other linearly independent paths, then the path is **linearly independent**.

Example 1

- Consider a program given for the identification of the greatest number. Draw the flow graph & DD Path graph. Also find the independent paths from the DD Path graph.

Cont…

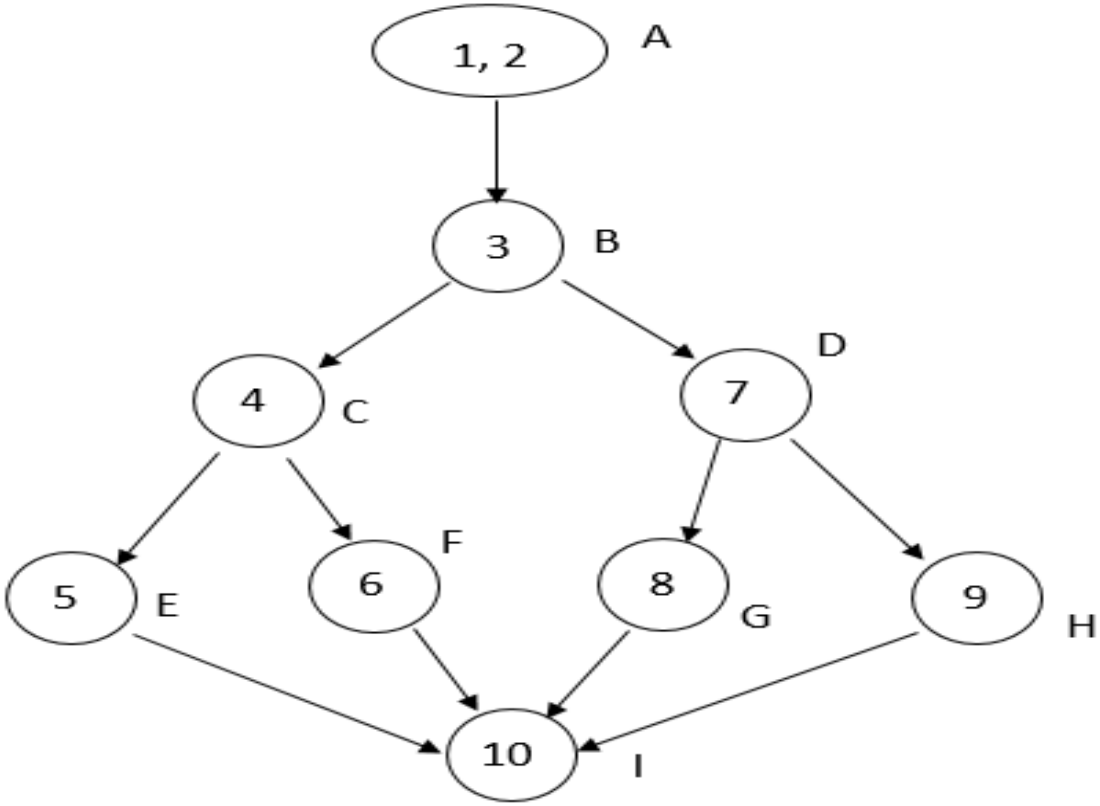
```
int main() {
1.    double n1, n2, n3;
2.   cin >> n1 >> n2 >> n3;
3.  if(n1 >= n2){
4.     if(n2>=n3){
5.      Max = n1;}
6.      else{ Max = n3;}}
7.    else if(n2 >=n3){
8            Max = n2;}
9.            else { Max = n3;}
10.        cout << "Largest number: " << max;}}
```

Cont...

- The flow Graph of the above code and the independent path are:

i) ABDHI ii) ABDGI

iii) ABCFI iv) ABCEI



Example 2

- Consider a program given for the classification of a triangle. Its input is a triple of positive integers (say a,b,c) from the interval [1,100]. The output may be [Scalene, Isosceles, Equilateral, Not a triangle]. Draw the flow graph & DD Path graph. Also find the independent paths from the DD Path graph.

Cont...

```
#include <stdio.h>
#include <conio.h>
1    int main()
2    {
3      int a,b,c,validInput=0;
4      printf("Enter the side 'a' value: ");
5      scanf("%d",&a);
6      printf("Enter the side 'b' value: ")
7      scanf("%d",&b);
8      printf("Enter the side 'c' value:");
9      scanf("%d",&c);
10     if ((a > 0) && (a <= 100) && (b > 0) && (b <= 100) && (c > 0)
           && (c <= 100)) {
11       if ( (a + b) > c) && ((c + a) > b) && ((b + c) > a)) {
12         validInput = 1;
13       }
14     }
15     else {
16       validInput = -1;
17     }
18     If (validInput==1) {
19       If ((a==b) && (b==c)) {
20         printf("The trinagle is equilateral");
21       }
22       else if ( (a == b) || (b == c) || (c == a) ) {
```

(Contd.)...

Cont...

```
23          printf("The triangle is isosceles");
24        }
25        else {
26          printf("The trinagle is scalene");
27        }
28      }
29      else if (validInput == 0) {
30        printf("The values do not constitute a Triangle");
31      }
32      else {
33        printf("The inputs belong to invalid range");
34      }
35      getche();
36      return 1;
37    }
```

Solution :

Flow graph of triangle problem is:

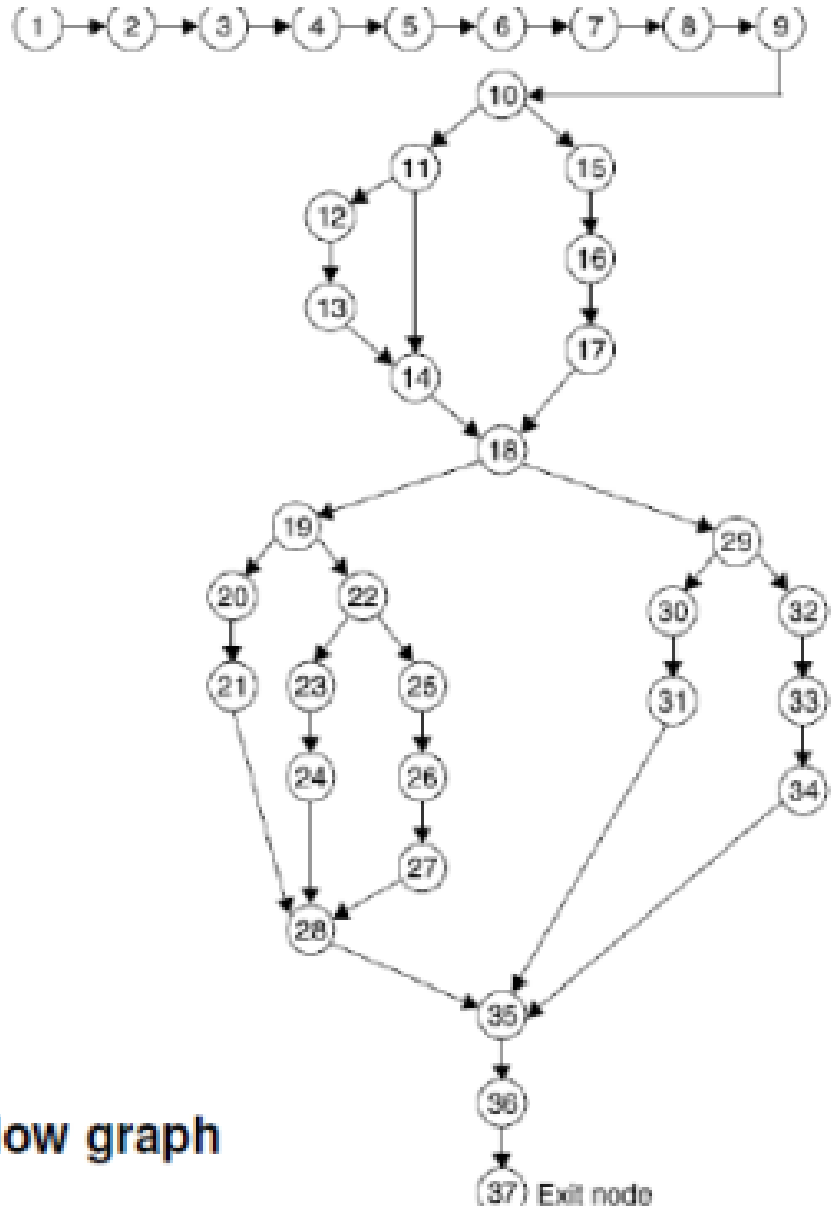


Fig.8. 20 (a): Program flow graph

Cont...

The mapping table for DD path graph is:

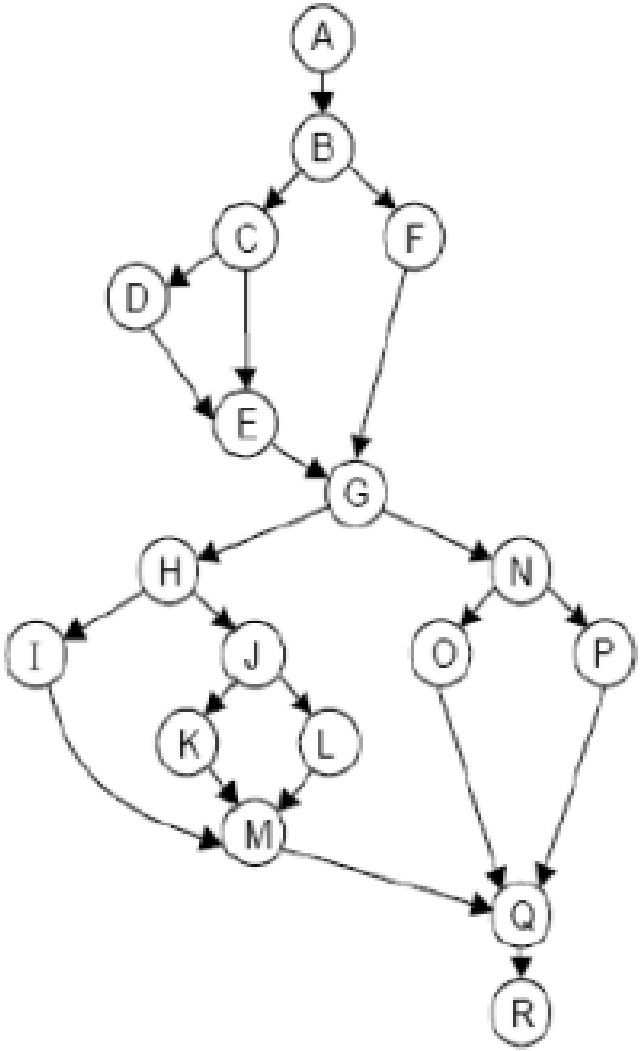
Flow graph nodes	DD Path graph corresponding node	Remarks
1 TO 9	A	Sequential nodes
10	B	Decision node
11	C	Decision node
12, 13	D	Sequential nodes
14	E	Two edges are joined here
15, 16, 17	F	Sequential nodes
18	G	Decision nodes plus joining of two edges
19	H	Decision node
20, 21	I	Sequential nodes
22	J	Decision node
23, 24	K	Sequential nodes
25, 26, 27	L	Sequential nodes

Cont...

Flow graph nodes	DD Path graph corresponding node	Remarks
28	M	Three edges are combined here
29	N	Decision node
30, 31	O	Sequential nodes
32, 33, 34	P	Sequential nodes
35	Q	Three edges are combined here
36, 37	R	Sequential nodes with exit node

Cont...

DD Path graph is given in Fig. 20 (b)



Independent paths are:

(i) ABFGNPQR
(ii) ABFGNOQR
(iii) ABCEGNPQR
(iv) ABCDEGNOQR
(v) ABFGHIMQR
(vi) ABFGHJKMQR
(vii)ABFGHJMQR

Fig. 20 (b): DD Path graph

Cyclomatic Complexity

- Cyclomatic complexity is a source code complexity measurement.
- It is calculated by developing a Control Flow Graph of the code that measures the number of linearly-independent paths through a program module.

Cyclomatic complexity (V(G)) = **E - N + 2*P**

where,

E = number of edges in the flow graph.

N = number of nodes in the flow graph.

P = number of nodes that have exit points

Note: for a single program (or subroutine or method), P is always equal to 1. so simple formula for a single subroutine is M = E-N+2 or

- Described (informally) as the number of decision points + 1

Cont...

- The cyclomatic complexity of the above example is

For Example 1

$$V(G) = E-N+2$$

$$= 11-9+2 = 4$$

For Example 2

$$V(G) = 23-18+2 = 7$$

Data Flow Testing

- **Data Flow Testing** is a type of structural testing.
- It is a method that is used to find the test paths of a program according to the locations of definitions and uses of **variables** in the program.
- It mainly **focuses** on the two **points:**
  - In which statement the variables are defined
  - In which statement the variables are used
- It uses the **control flow graph** to detect illogical things that can interrupt the flow of data.
- Anomalies in the flow of data are detected at the time of associations between values and variables due
  - If the variables are used without initialization.
  - If the initialized variables are not used at least once

Cont...

Example

```
1. read a, b, c, d;
2. if(a>b)
3.     X = a+1;
4.     print x;
   else
5.     x = b-1
6.     Y = b+1
7.     print Z;
8.     PrintY:
```

Variable	Define	Use
a	1	2,3
b	1	2,5,6
c	1	NA
d	1	NA
X	3,5	4
Y	6	8
Z	NA	7

Advantages of Data Flow Testing

- To identify a variable that is declared but never used with in the program.
- To identify a variable that is used but never declared.
- To identify a variable that is defined multiple times before it is used.
- Deallocating a variable before it is used.

2.3. Experience-Based Test Technique

- As the name suggests, the experience-based technique neither involves internal, nor the external structure, but is based on experience.
- This type of testing is most effective in the hands of experienced testers who will bring their knowledge to bear in the most productive ways possible. Why it is so?
  - Experienced testers know what needs to be documented to assess coverage and ensure some level of repeatability.
  - Experienced testers also know what to target to find the biggest bugs quickly and return a high yield during their testing time.
- These are not techniques for novices-but they are definitely appropriate and productive techniques for the advanced tester.

Cont…

- **Some of the methods followed are:**

1) Exploratory Testing Technique

2) Fault Attack Testing Technique

3) Error guessing Testing Technique

4) Checklist based Testing Technique

2.3.1. Exploratory testing technique

- Exploratory testing, is all about discovery, investigation and learning.
- It emphasizes on personal freedom and responsibility of the individual tester.
- Test cases are not created in advance but testers check system on the fly.
- They may take notes on ideas about what to test before test execution.
- The focus of exploratory testing is more on testing as a "thinking" activity.
- Normally, you design test cases first and later proceed with test execution. On the contrary, exploratory testing is a simultaneous process of test design and test execution.

2.3.2. Fault attack testing technique

- Software attacks, sometimes called fault attacks, are focused on trying to induce a specific type of failure.
- When performing attack testing, you should consider all areas of the software and its interaction with its environment as opportunities for failures.
- Attacks target the user interface, the operating system, interfacing systems, database interfaces and any file system interaction.
- Anytime data is being exchanged, it is potentially vulnerable to a failure and consequently is an excellent target for an attack.
- Coverage for attack testing is usually measured by determining if all the potentially vulnerable interfaces have been tested

2.3.3. Error guessing testing technique

- It doesn't sound very official, but it works.
- Error guessing as a testing technique is employed by the tester to determine the potential errors that might have been introduced during the software development and to devise methods to detect those errors as they manifest into defects and failures.
- Error guessing is commonly used in risk analysis to "guess" where errors are likely to occur in the error-prone areas.
- Error guessing coverage is usually determined based on the types of defects that are being sought.
- If there is a defect (or bug) taxonomy available, that can be used as the guideline. If taxonomy is not employed, the experience of the tester and the time available for testing usually determine the level of coverage.

2.3.4. Checklist-based testing technique

- Checklist-based testing is used by experienced testers who are using checklists to guide their testing.
- The checklist is basically a high-level list, or a reminder list, of areas to be tested.
- This may include items to be checked, lists of rules, or particular criteria or data conditions to be verified.
- Checklists are usually developed over time and draw on the experience of the tester as well as on standards, previous trouble-areas, and known usage scenarios.
- Coverage is determined by the completion of the checklist

THE END
THANK YOU!!!

Debre Markos University
Institute of Technology
School of Computing, Software Eng A/program

Software Quality Assurance and Testing (SEng4112)

Chapter Six

Software Testing Metrics and Tools

Objective

- **After successful completion of this chapter, students will be able to understand**
  - ✓ Software Testing Metrics
  - ✓ Software Testing Tools

Software Testing Metric

- In software testing, metric is a quantitative measure of the degree to which a system, system component, or process possesses a given attribute.
- In other words, metrics helps estimating the progress and quality of a software testing effort.
- Software testing metrics improve the effectiveness and efficiency of a software testing process.
- Software test metrics is used for monitoring and controlling processes and products.
- It helps to drive the project towards our planned goals without deviation.

Why test metrics?

- Test metrics helps us to
  - Take decision for next phase of activities
  - Take evidence of the claim or prediction
  - Understand the type of improvement required
  - Take decision or process or technology change

Software Test Metrics

- Software test metrics used in the process of **test preparation** and **test execution** phases of the STLC include:

1. Test Case Preparation Productivity
2. Test Design Coverage

Test Design Phase

3. Test Execution Productivity
4. Test Execution Coverage
5. Test Cases Passed
6. Test Cases Failed
7. Test Cases Blocked

Test Execution Phase

1. Test Case Preparation Productivity

- It is used to calculate the number of test cases prepared and the effort spent for the preparation of test cases.
- **Formula:**

$$\text{TCPP} = \frac{\boldsymbol{No.of\ TCs}}{\boldsymbol{Effort\ Spent\ for\ TC\ preparation}}$$

TCPP: Test Case Preparation Productivity

TCs: Test Cases

- E.g.:-
- No. of Test cases = 240
- Effort spent for Test case preparation (in hours) = 80 hours
- Test Case preparation productivity = 240/80 = 30 test cases/hour
- Asap, higher test case productivity is recommended

2. Test Design Coverage

- It helps to measure the percentage of test case coverage against the number of requirements.
- **Formula:**

$$\text{TDC} = \left(\frac{\textit{Total No.of requirements mapped to TCs}}{\textit{Total No.of requirements}}\right) * 100$$

TDC: Test Design Coverage

TCs: Test Cases

- E.g.:-
- Total number of requirements: 78
- Total number of requirements mapped to test cases: 69
- Test design coverage = (69/78)*100 = **88.5%**

3. Test Execution Productivity

- It determines the number of Test Cases that can be executed per hour
- **Formula:**

$$\text{TEP} = \frac{\textit{No.of TCs Executed}}{\textit{Effort Spend for Execution of TCs}}$$

TEP: Test Execution Productivity

TCs: Test Cases

- **E.g.:-**
- No. of Test cases executed = 180
- Effort spent for execution of test cases = 10 hours
- Test Execution Productivity = 180/10 = **18 test cases/hour**

4. Test Execution Coverage

- It is used to measure the number of test cases executed against the number of test cases planned to be executed.
- **Formula:**

$$\text{TEC} = \left(\frac{\textit{Total No.of TCs Executed}}{\textit{Total No. of TCs planned to be executed}}\right) * 100$$

TEC: Test Execution Coverage

TCs: Test Cases

- **E.g.:-**
- Total no. of test cases planned to be executed = 240
- Total no. of test cases executed = 180
- Test Execution Coverage = (180/240)*100 = **75%**

5. Test Cases Passed

- It measures the percentage of test cases passed during test execution. Also known as pass rate.
- **Formula:**

$$\text{TCP} = \left(\frac{\textit{Total No.of TCs Passed}}{\textit{Total No. of TCs Executed}}\right) * 100$$

TCP: Test Cases Passed

TCs: Test Cases

- **E.g.:-**
- Total number of test cases passed = 80
- Total number of test cases executed = 90
- Test Cases Passed = (80/90)*100 = **88.8 ≈ 89%**

6. Test Cases Failed

- It measures the percentage of test cases failed the test.

Formula:

$$\text{TCF} = \left(\frac{\textit{Total No.of TCs Failed}}{\textit{Total No. of TCs Executed}}\right) * 100$$

TCF: Test Cases Failed

TCs: Test Cases

E.g.:-

- Total no. of test cases failed = 10
- Total no. of test cases executed = 90
- Test Cases Failed= (10/90)*100 = **11.1 ≈ 11%**

7. Test Cases Blocked

- It measures the percentage of test cases blocked during the test.
- **Formula:**

$$\text{TCB} = \left(\frac{\textit{Total No.of TCs Blocked}}{\textit{Total No. of TCs Executed}}\right) * 100$$

TCB: Test Cases Blocked

TCs: Test Cases

- **E.g.:-**
- Total number of test cases blocked = 5
- Total number of test cases executed = 90
- Test Cases Blocked = (5/90)*100 = **5.5 ≈ 6%**

Defect Analysis Metrics

- Software test metrics used in the process of defect analysis phase of the STLC includes:

  1. Error Discovery Rate
  2. Defect Fix Rate
  3. Defect Density
  4. Defect Leakage
  5. Defect Removal Efficiency

1. Error Discovery Rate

- It determines the effectiveness of the test cases.
- **Formula:**

$$\text{EDR} = \left(\frac{\textit{Total No.of Defects Found}}{\textit{Total No. of TCs Executed}}\right) * 100$$

EDR: Error Discovery Rate

TCs: Test Cases

- **E.g.:-**
- Total number of defects found = 60
- Total number of test cases executed = 240
- Error Discovery Rate = (60/240)*100 = **25%**

2. Defect Fix Rate

- It helps to know the quality of a build in terms of defect fixing.
- **Formula:**

$$DFR = \left(\frac{\textit{Total No.of Defects reported as fixed} - \textit{Totla No. Dffects Reopened}}{\textit{Total No. of defects reported as fixed} + \textit{Total No.of new bug due to fix}}\right) * 100$$

DFR: Defect Fix Rate

- **E.g.:-**
- Total number of defects reported as fixed = 10
- Total number of defects reopened = 2
- Total number of new Bugs due to fix = 1
- Defect Fix Rate = ((10 – 2)/(10 + 1))*100 = (8/11)100 = **72.7 ≈ 73%**

3. Defect Density

- Defect density is the number of confirmed defects detected in the software or a component during a defined period of development or operation, divided by the size of the software.
- It is defined as the ratio of defects to requirements.
- Defect density determines the stability of the application.
- **Formula:**

$$\text{Defect Density} = \frac{\textit{Total No.of Defects identified}}{\textit{Actual Size of Software Product}}$$

Cont…

- E.g.: Suppose you have a software product which has been integrated from 4 modules and you found the following bugs in each of the modules.
  - Module 1 = 20 bugs
  - Module 2 = 30 bugs
  - Module 3 = 50 bugs
  - Module 4 = 60 bugs
- And the total line of code for each module is
  - Module 1 = 1200 LoC
  - Module 2 = 3023 LoC
  - Module 3 = 5034 LoC
  - Module 4 = 6032 LoC
- Then, we calculate defect density as
  - Total bugs = 20+30+50+60 = 160
  - Total Size= 15289 LOC
  - Defect Density = 160/15289

    = 0.01046504 defects/LoC

    = 10.5 defects/KLoC

4. Defect Leakage

- It is used to review the efficiency of the testing process before User Acceptance Testing (UAT).
- **Formula:**

$$\text{Defect Leakage} = \left(\frac{\textit{Total No.of Defects Found in UAT}}{\textit{Total No.of Defects Found Before UAT}}\right) * 100$$

- **E.g.:-**
- Number of defects found in UAT = 20
- Number of defects found before UAT = 120
- Defect Leakage = (20 /120) * 100 = **16.6 ≈ 17%**

5. Defect Removal Efficiency

- It allows us to compare the overall defect removal efficiency.
- It includes defects found pre and post-delivery.
- **Formula:**

$$\text{DRE} = \left(\frac{\textit{Total No.of Defects found pre_delivery}}{\textit{Total No. of defects found pre_delivery} + \textit{Total No. of defects found post_delivery}}\right) * 100$$

DRE: Defect Removal Efficiency

- **E.g.:-**
- Total no. of defects found pre-delivery = 80
- Total no. of defects found post-delivery = 10
- Defect Removal Efficiency = ((80) / ((80) + (10)))*100

  = (80/90)*100 = **88.8 ≈ 89%**

Manual Testing Vs Automated Testing

- Software testing is carried out using manual and automated software testing tools
- Tester should have the perspective of an end-user and to ensure all the features are working as mentioned in the requirement document.
- In this process, testers execute the test cases and generate the reports manually, without using any automation tools.

Manual Testing

Advantages:

- It can be done on all kinds of applications
- It is preferable for short life cycle products
- Newly designed test cases should be executed manually
- Application must be tested manually before it is automated
- It is preferred in the projects where the requirements change frequently and for the products where the GUI changes constantly
- It is cheaper in terms of initial investment compared to automation testing
- It requires less time and expense to begin productive manual testing
- There is no necessity to the tester to have knowledge on automation Tools

Cont…

Limitations:

- Manual testing is time-consuming mainly while doing regression testing.
- In the long run, expensive over automation testing

Automated Testing

- Automated testing is the process of testing a software using automated tools.
- Automated testing means running the software programs that carry out the execution of test cases automatically and produce the test results without any human intervention.
- In this process, executing the test scripts and generating the results are performed automatically by automated tools.

Cont…

Advantages:

- Automation testing is faster in execution
- It is cheaper compared to manual testing in a long run
- It is more reliable
- It is more powerful and versatile
- It is mostly used for regression testing
- It does not require human intervention.
- It helps to increase the test coverage

Limitations:

- It is recommended only for stable products
- Automation testing is expensive initially
- Most of the automation tools are expensive
- It has some limitations such as handling captcha, fonts, color
- Huge maintenance in case of repeated changes in the requirements
- Not all the tools support all kinds of testing.

Testing Tool Acquisition

- Now days we can get lots of software testing tools in the market.
- Testing tool acquisition define how and when the tool will be used.
- Selection of tools is totally based on the project requirements
  - Proprietary/commercial tools or
  - Free tools/open source tools
- Off course, free testing tools may have some limitation in the features list of the product, so it's totally based on what are you looking for & is that your requirement fulfill in free version or go for paid software testing tools.

How to select the right testing tools

- Understand your project requirements thoroughly.
  - Project type
  - Scope of project
  - Technical expertise
- Consider your existing test automation tool as benchmark
  - Understand pros and cons
  - Evaluate and determine the best tools
- Evaluate the selected tool on key criteria
  - Easy to use
  - OS compatibility
  - Multiple languages
  - Test reporting
  - Platform support
- Leverage a matrix technique for analysis

1. Selenium

- It is the most widely used automated testing tool among all web application testing tools.
- Selenium can be executed in multiple browsers and operating systems.
- It is compatible with several programming languages and automation testing frameworks.
- With selenium, you can come up with very powerful browser-centered automation test scripts which are scalable across different environments.
- You can also create scripts using Selenium that is of great help for prompt reproduction of bugs, regression testing, and exploratory testing.

2. Unified Functional Testing (UFT) QTP

- Unified Functional Testing (UFT) tool given by Hewlett-Packard Enterprise is one of the best automation testing software for functional testing.
- It was previously known as Quick Test Professional (QTP).
- It brings developers & testers coming together under one umbrella and provides high-quality automation testing solutions.
- It makes functional testing less complex and cost-friendly.

3. HP Quality Center

- It is basically an integrated IT quality management software.
- Automated testing is one of its key features which constantly allows you to test earlier and quicker.
- Asset sharing and reusability allows QC to deliver bug-free and reliable applications.
- HP-ALM helps us to manage project milestones, deliverables, and resources.

4. Testim.io

- It leverages machine learning for the authoring, execution, and maintenance of automated test cases.
- We use dynamic locators and learn with every execution.
- The outcome is super fast authoring and stable tests that learn, thus eliminating the need to continually maintain tests with every code change.

5.TestComplete

- It allows all level of users to quickly create powerful, reusable and time-saving GUI automation tests for the web, mobile, and desktop applications.
- It lets you combine the recorded scripts and tests into a single framework which reduces the training cost and testing time.
- The best part of the tool is TestComplete Visualizer, which is a screenshot based feature allows you to modify previously recorded tests.

6. Telerik Test Studio

- Telerik test studio is a comprehensive test automation solution.
- It is well suited for GUI, performance, and load.
- It allows you to test desktop, mobile and web applications.
- Its main features include Point-and-click test recorder, support for coding languages like C# and VB.NET, central object repository and continuous integration with source control.

7. Katalon Studio

- Katalon Studio is a powerful test automation solution for mobile, Web, and API testing. And it is completely FREE!
- It provides a comprehensive set of features for test automation, including recording actions, creating test cases, generating test scripts, executing tests, reporting results, and integrating with many other tools in the software development lifecycle.
- Katalon Studio runs on both Windows and MacOS, supporting automated testing of iOS and Android apps, web applications on all modern browsers, and API services.

Test Management Tools

Proprietary/Commercial Tools

- HP Quality Center/ALM
- QA Complete
- T-Plan Professional
- Automated Test Designer (ATD)
- Testuff
- SMARTS
- QAS.TCS (Test Case Studio)
- PractiTest
- Test Manager Adaptors
- SpiraTest
- TestLog
- ApTest Manager

Open Source Tools

- TET (Test Environment Toolkit)
- TETware
- Test Manager
- RTH

Functional Testing Tools

Open Source Tools

- ✓ Selenium
- ✓ Soapui
- ✓ Watir
- ✓ HTTP::Recorder
- ✓ WatiN
- ✓ Canoo WebTest
- ✓ Webcorder
- ✓ Solex
- ✓ Imprimatur
- ✓ SAMIE
- ✓ Swete
- ✓ ITP
- ✓ WET
- ✓ WebInject

Proprietary/Commercial Tools

- QuickTest Pro
- Rational Robot
- Sahi
- SoapTest
- Badboy
- Test Complete
- QA Wizard
- Netvantage Functional Tester
- PesterCat AppsWatch
- Squish
- actiWATE
- liSA
- vTest
- Internet Macros

Performance/Load Testing Tools

Proprietary/Commercial Tools

- WebLOAD Professional
- HP LoadRunner
- LoadStorm
- NeoLoad
- Loadtracer
- Forecast
- ANTS – Advanced .NET Testing System
- vPerformer
- Webserver Stress Tool
- preVue-ASCII

Open Source Tools

- ✓ Jmeter
- ✓ FunkLoad

THE END
THANK YOU!!!

Debre Markos University
Institute of Technology
School of Computing, Software Eng A/program

Software Testing and Quality Assurance(SEng4051)

Chapter Seven

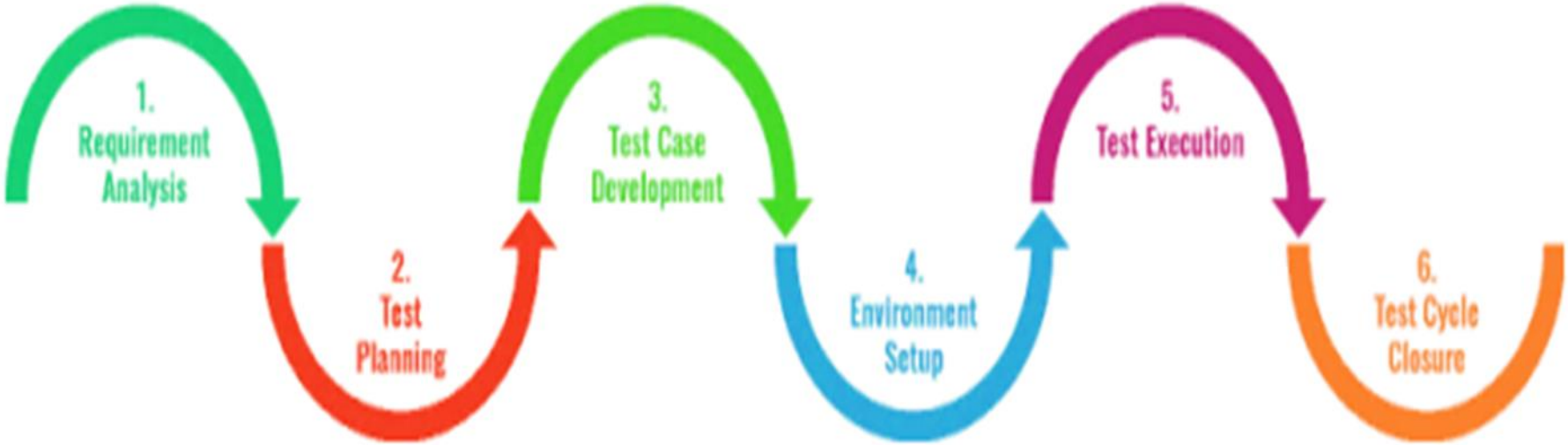
Test Process management and Improvement

Objective

- **After successful completion of this chapter, students will be able to understand**
  - ✓ Software Testing Process (revision)
  - ✓ Test Management
  - ✓ Test Estimation

Software Testing Process (revision)

- A process is a **series** of activities performed to fulfill a purpose and produce a tangible output based on a given input.
- **Software testing life cycle** defines the stages in testing of software.



Test Management

- **Test Management** is a process of managing the testing activities in order to ensure high quality and high-end testing of the software application.
- The method consists of organizing, controlling, ensuring **traceability** and **visibility** of the testing process in order to deliver the high quality software application.
- It ensures that the **software testing process** runs as expected.

Test Management Process

- **Test Management Process** is a procedure of managing the software testing activities from start to the end.
- The test management process provides planning, controlling, tracking and monitoring facilities throughout the whole project cycle.
- The process involves several activities like test planning, designing and test execution.

Cont…

- There are two main Parts of Test Management Process: –

- **Planning**
  - Risk Analysis
  - Test Estimation
  - Test Planning
  - Test Organization
- **Execution**
  - Test Monitoring and Control
  - Issue Management
  - Test Report and Evaluation

I, Planning: 1. Risk Analysis and Solution

- **Risk** is the probability of occurrence of an undesirable event.
- Risk should be identified and corresponding solutions should be determined before the start of the project.
- Risk Analysis is the first step which Test Manager should consider before starting any project. Because all projects may contain risks, early risk detection and identification of its solution will help Test Manager to **avoid** potential loss in the future & save on project cost.

Cont…

- **How to Perform Risk ANALYSIS?**
- It's a 3-Step process
  - Identify the Risks
  - Analyze Impact of each Identified Risk
  - Take counter measures for the identified & Analyzed risk

Step 1) Identify Risk

- Risk can be identified and classified into 2 types in software product

I)Project Risk

- Project risk can be defined as an **uncertain** event or activity that can impact the project's progress.
- There are primarily 3 categories of Project Risks
    - **Organizational Risk**
    - **Technical Risk**
    - **Business Risk**

Cont…

Organizational Risk

- It is a risk related to your **human resource** or your Testing team. **For example**:-
  - in your project, lack of technically skilled members is a risk.
  - Not having enough manpower to complete the project on time is another risk.
- To identify the Organizational Risk, you should make a list of few **questions** and answer them as self-exercise.
  - Is this a well-organized Team?
  - Does each team member has the skill to do his/her job?
  - Compare to project size and schedule, do we have enough human resource to finish this project at the deadline?

Cont…

Technical Risk

- Technical Risk is the probability of loss incurred during the **execution** of a technical process such as wrong testing procedure.

Example:-

- The Testing **environment** may not be setup properly like real business environment.

Business Risk

- The risk involves an **external** entity. It is the risk which may come from your company, your customer but **not** from your project.
- Example:-
  - Your project **budget** may cut by half because of business situation

II) Product Risk

- **Product risk** is the possibility that the system or software might fail to satisfy or fulfill the expectation of the customer, user, or stakeholder.

Example:-

- The software skips some **key** function that the customers specified in the users' requirement
- The software is **unreliable** and frequently **fails** to work.
- Software **fail** in ways that cause **financial** or other damage to a user or the company that uses the software.
- The software has problems related to a particular quality characteristic such as **security**, **reliability**, **usability**, **maintainability** or **performance**.

Step 2) Analyze the impact of the risk occurring

- Each risk should be classified on the basis of following two parameters
  - ✓The **probability** of occurrence
  - ✓The **impact** on the project

Priority = Probability * Impact

	Probability	**Impact**
High (3)	Has very high probability to occur, may impact to the whole project	Cannot continue with project activity if it is not solved **immediately**
Medium (2)	50% chance to occur	Cannot continue the project activity if it is not solved
Low (1)	Low probability of occurrence	Need to solve it but it is possible to take alternative solution for a while

Cont…

- Based on the above priority you can take the Risk Mitigation in Testing or counter measures.

Priority	**Risk Management**
High 6-9	*Take mitigation action immediately and monitor the risk every day until its status is closed.*
Middle 3-5	*Monitor the risk every week at internal progress meeting*
Low 1-2	*Accept the risk and monitor the risk on milestone basis*

Step 3) Take COUNTER MEASURES to mitigate the risk

- This activity is divided into 3 parts.

❑ Risk response

- The project manager needs to choose strategies that will reduce the risk to minima
  - Avoidance
  - Reduction
  - Sharing
  - Accept

❑ Register Risk

- All the risk must be recorded, documented and acknowledged by project managers, stakeholder and the project member.

Monitor and Control Risk

- Risks can be monitored on a continuous basis to check if any changes are made.
- New risk can be identified through the constant monitoring and assessing mechanisms.

2. Test Estimation

- Test Estimation is approximately determining
  - **how long** will this testing take?
  - **How much** will it cost?
- Accurate test estimates lead to better planning, execution and monitoring of tasks under a test manager's attention.
- Allow for more accurate scheduling and help realize results more confidently.

3. Test Planning

- A test plan gives **detailed** testing information regarding an upcoming testing effort, including:
  - Test Strategy
  - Test Objective
  - Exit /Suspension Criteria
  - Resource Planning
  - Test Deliverables

4. Test Organization

- **Test Organization in Software Testing** is a procedure of defining roles in the testing process.
- It defines who is responsible for which activities in testing process. Test functions, facilities and activities are also explained in the same process.
- The competencies and knowledge of the people involved are also defined however everyone is responsible for quality of testing process.

II, Execution: 1. Test Monitoring and Control

- Test Monitoring and Control is the process of overseeing all the metrics necessary to ensure that the project is running well, on schedule, and not out of budget.
- **Monitoring** is a process of collecting, recording, and **reporting** information about the project activity that the project manager and stakeholder needs to know
- To Monitor, Test Manager does following activities
  - **Define** the project goal, or project performance standard
  - **Observe** the project performance, and compare between the actual and the planned performance expectations
  - **Record** and **report** any detected problem which happens to the project

Cont...

- Project **Controlling** is a process of using data from monitoring activity to bring actual performance to planned performance.
- In **Controlling**, the Test Manager takes action to correct the deviations from the plan.
- In some cases, the plan has to be **adjusted** according to project situation.

2. Issue Management

- **Issue Management** is the process to make others aware of the problem and then resolve it as fast as possible

3. Test Report & Evaluation

- The project has already completed. It's now time for look back what you have done.
- "Test Evaluation Report" describes the results of the Testing in terms of Test coverage and exit criteria.

Test Estimation

- Estimation can be used to predict how much effort with respect to time and cost would be required to complete a defined task.
- Here's a list of Estimation Techniques for testing software:
  - Work Breakdown Structure (WBS)
  - 3-Point product Estimation Test
  - Function point and testing point breakdown
  - Wideband Delphi method
  - Use-case Methodologies etc.

Work Breakdown Structure (WBS)

- The essence of this technique is to divide a complex test project into small components.
- The project is broken down into sub-modules;
- Each sub-module, in turn, is divided into functionalities, which are split into sub-functionalities.

Three-Point Estimation

- This is a statistical method, but it also breaks down the task into subtasks (in this it is similar to WBS).
- Then, three possible scenarios should be estimated for each sub-task.
- **The best case:** assuming that:
  - you have a talented team,
  - all of the necessary resources,
  - no problem occurs and everything goes right
- you can complete the task, for example, in *100 man-hours (B)*. This is an optimistic scenario.
- **The most likely case:** assuming that:
  - you have a good team
  - enough resources
  - almost everything goes right, although some problems may occur,
- you can complete the task in *150 man-hours (M)*. This is a normal scenario.

Cont…

- **The worst case:** assuming that:
  - your team is not experienced
  - everything goes wrong
- you have to solve numerous problems, you can complete the task in *200 man-hours (W)*. This is a pessimistic scenario.
- Thus, you have three values:
  - B = 100
  - M = 150
  - W = 200

Cont…

- Now, you can calculate the average value for the test *estimation (E)* using the following formula:
- **E = (B + 4M + W) / 6**
- E = (100 + 4 x 150 + 200) / 6 = 150 man-hours
- As the average value may fluctuate a little bit, to be more accurate, you need to calculate *standard deviation (SD)* — the limits within which E may change. The formula is as follows:
- **SD = (W – B) / 6**
- SD = (200 – 100) / 6 = 16.7 man-hours
- You can present the final estimate as this: the team needs 150 +/- 16.7 (133.3 up to 166.7) person-hours to accomplish the sub-task.

Function Point Analysis (FPA)

- It estimates the value of the **total effort** that can be considered as **time or cost** of the task.
- FPA is based on specification documents, such as SRS document or Design.
- Again, as with WBS, the project is split into modules.
- Each module — depending on its complexity — is assigned a *functional point (FP)*.
- Simple tasks get lower points, difficult tasks get higher points.

Total Effort = Total FP x Estimate Defined per FP

Cont…

- Let's consider the total effort with respect to cost and take the estimate defined per FP as equal to **$100/points**.
- The whole project is divided into three groups of modules:
- Complex modules (FP is 5) — 3 pieces
- Medium modules (FP is 3) — 8 pieces
- Simple modules (FP is 1) — 4 pieces

Complexity	**Number of Modules**	**FP for each module**	**Total FP for each module**
Complex	3	5	15
Medium	8	3	24
Low(Simple)	4	1	4
			Total FP = 43

- Total Effort = 43 * $100 = $4,300.

Wideband Delphi method

- This is one of the most widely used testing estimation techniques based on surveys of the experts involved in the testing process.
- The essence is that a panel of experts discuss the given task under the guidance of a manager and make anonymous personal forecasts (how many man-hours this task will take), providing the reasons for their opinions.
- As a rule, after the first round, the range of answers is quite wide. Then, the experts are encouraged to revise their answers taking into account other members' judgments.
- Several rounds may take place until the range of answers decreases and the average value can be calculated.

Use-Case Point Method

- **Use-Case Points** (**UCP**) method is an estimation technique used for measuring software with different use cases.
- The Use-Case Points counting process has the following steps –
  - Calculate unadjusted UCPs
  - Adjust for technical complexity
  - Adjust for environmental complexity
  - Calculate adjusted UCPs

THE END
THANK YOU!!!