

Module 3

Syllabus

- ▶ UML Diagram introduction
- ▶ Design pattern
- ▶ Review Techniques(seminar)
- ▶ **Software testing strategy**
- ▶ Overview of DevOps and code management

Design Patterns


- ▶ The pattern is a description of the problem and the essence of its solution, so that the solution may be reused in different settings.
- ▶ The pattern is not a detailed specification.

Implementation :Issues

- ▶ Some aspects of implementation that are particularly important to software engineering.

These are:

- ▶ 1. *Reuse*: Most modern software is constructed by reusing existing components or systems. When you are developing software, you should make as much use as possible of existing code.

- 
- ▶ 2. *Configuration management*: During the development process, many **different versions of each software component** are created.
 - ▶ If you don't keep track of these versions in a configuration management system, you are liable to include the wrong versions of these components in your system.

- ▶ **3. *Host-target development*** : Production software does not usually execute on the same computer as the software development environment.
- ▶ Rather, you develop it on one computer (the host system) and execute it on a separate computer (the target system).

1. Reuse:

- ▶ Software reuse is possible at several different levels
- ▶ 1. *The abstraction level*: At this level, you don't reuse software directly but rather use knowledge of successful abstractions.
- ▶ 2. *The object level*: At this level, you directly reuse objects from a library rather than writing the code yourself
- ▶ To implement this type of reuse, you must find appropriate libraries and discover if the objects and methods offer the functionality that you need.

- ▶ 3. *The component level*: Components are collections of objects and object classes that operate together to provide related functions and services.
- ▶ You often must **adapt and extend the component** by adding some code of your own.
- 4. *The system level*: At this level, you **reuse entire application systems with some modification**

This function usually involves some kind of configuration of these systems. This may be done by adding and modifying code (if you are reusing a software product line)

2. *Configuration Management:*

- ▶ Configuration management is the name given to the general process of **managing a changing software system**.
- ▶ The aim of configuration management is to support the system integration process so that all developers can access the project code and documents in a controlled way, find out **what changes have been made**.

- ▶ There are 3 fundamental configuration management activities:
- ▶ 1. *Version management*, where support is provided to **keep track of the different versions of software components**.
- ▶ Version management systems include facilities to coordinate development by several programmers.
- ▶ They stop one developer from overwriting code that has been submitted to the system by someone else.

- ▶ 2. *Problem tracking*, where support is provided to allow users to report bugs and other problems, and to allow all developers to see who is working on these problems and when they are fixed.
- ▶ 3. *Release management*, where new versions of a software system are released to customers. **Release management is concerned with functionality of new releases** and organizing the software for distribution.

3. *Host - target development*

- ▶ IDE (Integrated Development Environment):
- ▶ An IDE is a **set of software tools that supports different aspects of software development** within some common framework and user interface.
- ▶ A general-purpose IDE is a framework for hosting software tools that **provides data management** facilities for the software being developed and **integration mechanisms** that allow tools to work together. The best-known general-purpose IDE is the **Eclipse**

Design with UML diagrams

- ▶ UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems.
- ▶ You can use models to do the following things:
 - ▶ Visually represent a system that you want to build
 - ▶ Communicate your vision of a system to customers and colleagues
 - ▶ Develop and test the architecture of a system
 - ▶ Use the UML diagrams to direct code generation

Types of diagrams in UML

Structural diagrams

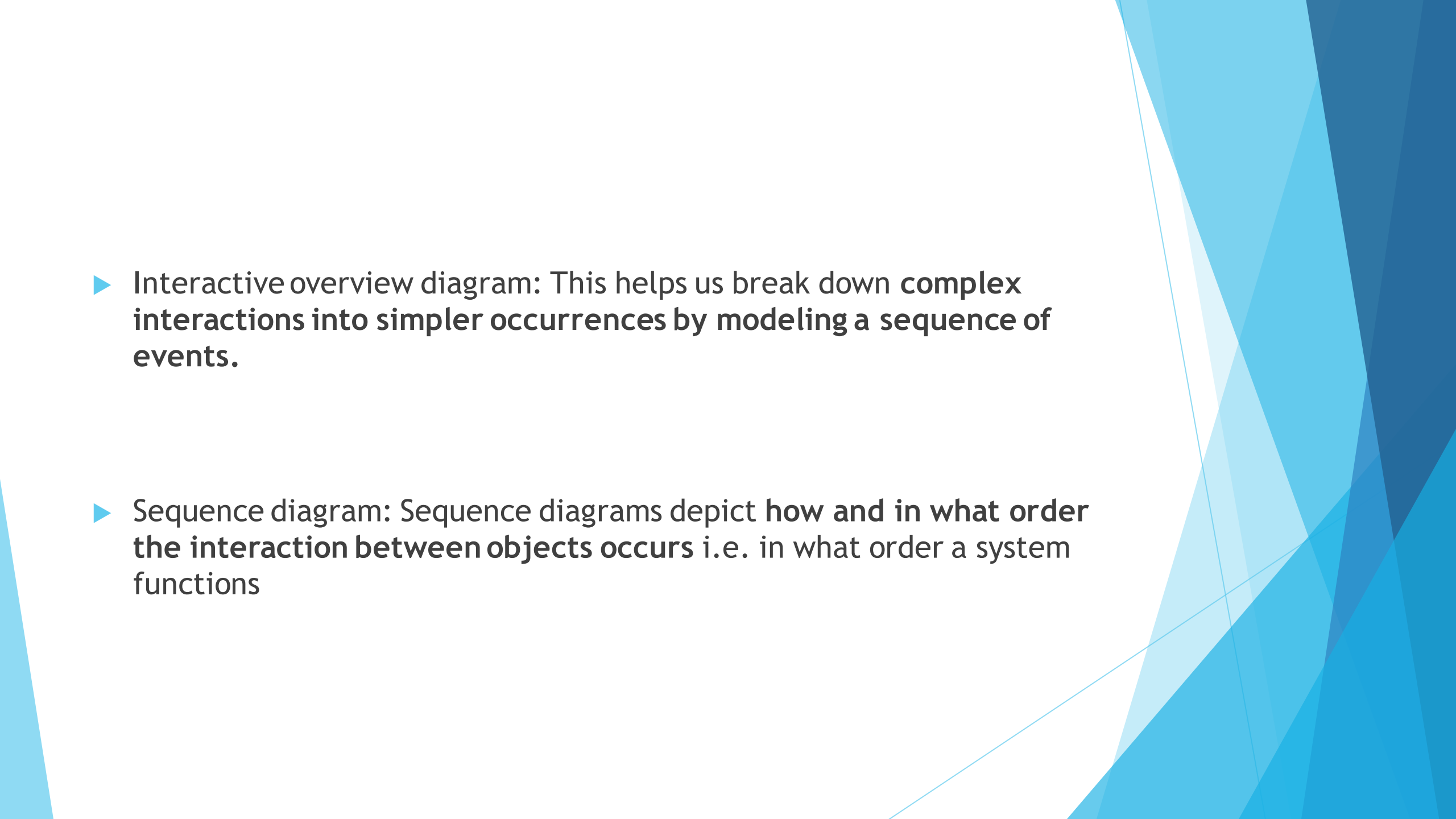
- ▶ Component diagram: These diagrams become essential when we design and build complex systems. A **complex system is visualized by breaking it down into smaller components and demonstrating how the components interact.**
- ▶ Deployment diagram: Deployment Diagrams show **how hardware and software are represented in a system.** These diagrams represent performance, scalability, and maintainability that are useful for systems engineers

- ▶ Object diagram: The object diagram represents **relationships between classes and their individual instances of classes at a point in time**. As object diagrams depict the behavior after objects have been instantiated.
- ▶ Class diagram: Class diagrams display a **system's static structure, including classes, their characteristics, and the connections between them**.
- ▶ Package diagram: A package diagram portrays the associations between **different packages**.

Behavior Diagrams

- ▶ **Activity diagram:** It illustrates the **flow of control in a system**. Activity diagrams visualize sequential, branched, or concurrent activities involved in the execution of a use case.
- ▶ **Use Case diagram:** It represents a high-level view of the system or part of the system, basically the **functional requirements and its interaction with external agents(actors)**.

- ▶ Timing diagram: A timing diagram is an inverted form of a sequence diagram that represents the **interaction between objects within a given time frame**.
- ▶ Communication diagram: Also known as a collaboration diagram shows **how objects connect through messages** within the architectural design of the system.


- 
- The background of the slide features an abstract design with overlapping geometric shapes in various shades of blue, ranging from light sky blue to deep navy blue. These shapes are primarily located on the right side of the slide, creating a modern, layered effect.
- ▶ **Interactive overview diagram:** This helps us break down **complex interactions into simpler occurrences by modeling a sequence of events.**
 - ▶ **Sequence diagram:** Sequence diagrams depict **how and in what order the interaction between objects occurs** i.e. in what order a system functions

Advantages:

- Provides standard for software development.
- Reducing of costs to develop diagrams of UML using supporting tools.
- Development time is reduced.
- The issues can be resolved fast by the developers
- Has large visual elements to construct and easy to follow

VERIFICATION & VALIDATION

- ▶ **VALIDATION** : Process of examining whether or not the software **satisfies the user requirements**. It usually is carried out at the **end of SDLC** . If the software matches requirements for which it was made, it is validated.
- ▶ **VERIFICATION** : Process of confirming if the software is developed adhering to the proper specifications and methodologies.

Error / Mistake	Defect / Bug/ Fault	Failure
Found by 	Found by 	Found by 
Developer	Tester	Customer

Testing : Principles of Testing

- ▶ Testing shows presence of defects
- ▶ Exhaustive testing is not possible
- ▶ ***Pesticide paradox: Repeating the same test cases again and again will not find new bugs. So it is necessary to review the test cases and add or update test cases to find new bugs.***
- ▶ Testing is context dependent

Test case

<i>ID</i>	14
<i>Title</i>	Add customer
<i>Pre-Conditions</i>	Sign in with sales authorization
<i>Test Steps</i>	<ol style="list-style-type: none">1. Select the client module.2. Enter the customer information.3. Click "Add".
<i>Expected Results</i>	A message appears in the program's status bar. The message reads "New customer added".

TYPES OF TESTING(IMP)

- ▶ 1. UNIT TESTING
- ▶ 2. INTEGRATION TESTING
- ▶ 3. SYSTEM TESTING
- ▶ 4. ACCEPTANCE TESTING
- ▶ 5. REGRESSION TESTING
- ▶ 6. STRESS TESTING
- ▶ 7. PERFORMANCE TESTING

UNIT TESTING

- ▶ A Unit is a smallest testable portion of system(module) or application which can be compiled, linked, loaded, and executed.
- ▶ It is often done by programmer by using sample input and observing its corresponding outputs.

2. INTEGRATION TESTING

- ▶ The objective is to take unit tested components and build a program structure that has been dictated by design
- ▶ Integration testing is testing in which a group of components are combined to produce output.
- ▶ Integration testing is of four types:
 - ▶ (i) Top down
 - ▶ (ii) Bottom up
 - ▶ (iii) Sandwich
 - ▶ (iv) Big-Bang

Top Down

- ▶ In this approach, testing begins at the top level of the application's hierarchy and gradually moves down to lower levels.
- ▶ The main or top-level modules are tested first, and then the lower-level modules are tested

Bottom Up

- ▶ In these, low-level modules are tested first, and then high-level modules are tested.
- ▶ Low-level modules are tested first, and real modules are integrated incrementally to build the system from the bottom up

Big Bang Integration Testing:

- ▶ Big bang integration testing takes place when all or most of the components/modules are developed and then integrated simultaneously.
- ▶ The system is tested as a whole without a systematic integration of individual modules.

Sandwich (Hybrid) Integration Testing:

- ▶ Sandwich integration testing is a **combination of both top-down and bottom-up approaches**.
- ▶ It starts with testing some **critical modules** from the top down and some from the bottom up.
- ▶ These critical modules are usually those that are **complex or have a high impact on system behavior**.
- ▶ Once these critical modules are tested and integrated, the testing process continues with the remaining modules using a more traditional top-down or bottom-up approach

SYSTEM TESTING

- ▶ System testing is performed on a complete, integrated system.
- ▶ It allows checking system's compliance as per the requirements. It tests the overall interaction of components.
- ▶ It involves functionality, performance, reliability and security testing.
- ▶ • System testing most often the final test to verify that the system meets the specification.
- ▶ It evaluates both functional and non-functional need for the testing.
- ▶ 1. FUNCTIONALITY TESTING
- ▶ 2. SYSTEM PERFORMANCE TESTING
- ▶ 3. SECURITY & PORTABILITY

ACCEPTANCE TESTING

- ▶ Acceptance testing is a test conducted to find if **the requirements of a specification or contract are met as per its delivery.**
- ▶ Acceptance testing is basically done by the user or customer
- ▶ **ALPHA TESTING** : It is a type of acceptance testing which is done before the product is released to customers. It is typically done by **QA people.**
- ▶ **BETA TESTING** : The beta test is conducted at one or more customer sites by the **end-user of the software.**

REGRESSION TESTING

- ▶ Every time new module is added leads to changes in program. This type of testing make sure that whole component works properly even after **adding components to the complete program.**

STRESS TESTING

- ▶ In this we gives **unfavorable conditions** to the system and check how they perform in those condition.
- ▶ Example:
 - ▶ (a) Test cases that require maximum memory
 - ▶ (b) Test cases that may cause excessive disk requirements

7.PERFORMANCE TESTING

- ▶ It is designed to test the run-time performance of software within the context of an integrated system.
- ▶ It is used to test speed and effectiveness of program.
- ▶ Example: Checking number of processor cycles

Testing Approach

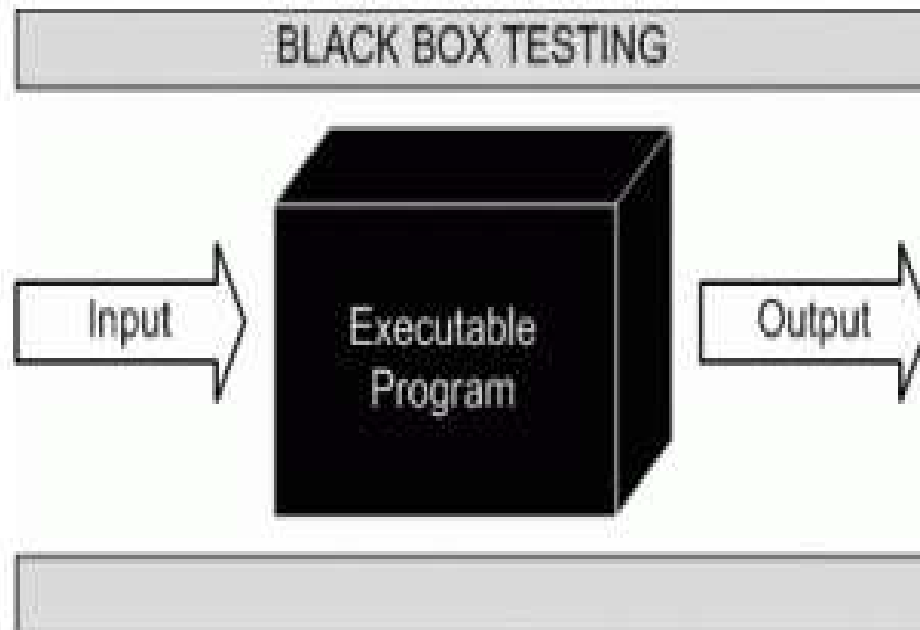
- ▶ Tests can be conducted based on two approaches:

1.BLACK BOX TESTING (FUNCTIONALITY TESTING, BEHAVIOURAL TESTING)

2.WHITE BOX TESTING (IMPLEMENTATION TESTING, STRUCTURAL TESTING)

BLACK BOX TESTING

- ▶ Black-box testing is a method of software testing that examines the **functionality of an application based on the specifications**. It is also known as Specifications based testing.
- ▶ The internal structure/design/implementation of the item being tested is not known to the tester.



- ▶ Following are some techniques that can be used for designing black box tests.
- ▶ **Equivalence Partitioning:** The input is divided into similar classes. If one element of a class passes the test it is assumed that all the class is passed.
- ▶ **•Boundary Value Analysis:** It is a software test design technique that involves the **determination of boundaries for input values and selecting values that are at the boundaries and just inside/ outside of the boundaries as test data.** That is input is divided into higher and lower values. If these values pass the test ,it is assumed that all values in between may pass too.

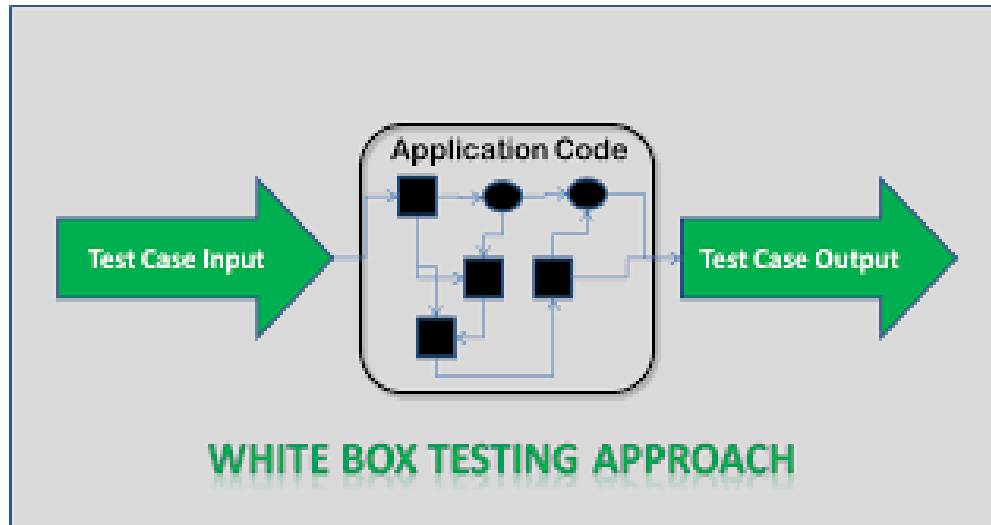
Example

- ▶ Let's say you're testing a simple login form where the username field accepts a minimum of 6 characters and a maximum of 20 characters. The password field has similar constraints, with a minimum of 8 characters and a maximum of 15 characters.

- ▶ **Cause-Effect Graphing:** In previous two methods ,only one input value at a time is tested . Cause(input) - Effect(output) is a testing technique where combinations of input values are tested in a systematic way

WHITE BOX TESTING

- ▶ It is conducted to test program and its implementation in order to improve code efficiency or structure.
- ▶ Design and structure of code is known to tester
- ▶ Programmers of the code conducts this test on the code



- ▶ The below are some of the white box testing techniques

1. Control flow testing :

- ▶ Set up test cases which covers all statements and branch conditions.
- ▶ The branch conditions are tested for both being true and false, so that all statements can be covered.

2. Data flow testing :

- ▶ Emphasizes to cover all data variables
- ▶ It tests where the variables were declared and defined and were used or changed

3. Basis path testing, :

- ▶ It is structured testing or white box testing technique used for designing test cases intended to examine **all possible paths of execution at least once**.
- ▶ Creating and executing tests for all possible paths results in 100% statement coverage and 100% branch coverage.

Test Automation, Test driven development

- ▶ Test automation is the process of using **automation tools to, execute tests, maintain test data and analyze test results to improve software quality**

eg. FWTS(firmware test suite)

It's like having a robot perform tasks to ensure that a program works correctly, without human intervention.

- ▶ In software testing, test automation is the use of software separate from the software being tested to control the execution of tests and the comparison of actual outcomes with predicted outcomes.

Approaches:

- ▶ **1. Graphical user interface testing.** A testing framework that generates user interface events such as **keystrokes and mouse clicks**, and observes the changes that result in the user interface, to validate that the observable behaviour of the program is correct.
- ▶ **2. API driven testing.** API-driven testing is a method of testing where use a **programming interface (API) of a software application to check if it behaves correctly**. This is often used to ensure that the building blocks (like classes, modules, or libraries) of the software work as expected and return the right results when given various types of input

- ▶ **Levels: 3 levels of automation**
- ▶ **Solid foundation(foundation level):** Unit testing is applied to testing individual parts of the code.
- ▶ **The service layer:** refers to testing the services of an application these services are anything that the application does in response to some input or set of inputs.
- ▶ **Top level:** UI testing involves fewer tests because even small changes in the user interface can easily break many tests, making them more difficult to maintain.

TEST DRIVEN DEVELOPMENT

- ▶ Test-driven development (TDD) is a software development approach that **focuses on writing tests before writing the actual code for a feature or functionality.**
- ▶ Here's how it works in a simplified manner:

Stages of TDD

- ▶ **Write a Test:** Before you start coding, you create a **test case that defines what you want your code to accomplish**. This test initially fails because there's no code to make it pass.
- ▶ **Write the Minimum Code:** You then write the **minimum amount of code necessary to make the test pass**. This might mean implementing just enough to satisfy the test, not the entire feature.
- ▶ **Run the Test:** You **run the test to check if it passes**. If it does, it means your code is working as intended. If it fails, you go back to step 2 and make adjustments to your code.

- ▶ **Refactor:** Once the test passes, you can refactor or improve the code to make it cleaner, more efficient, or more maintainable. Importantly, the test ensures that any changes you make do not break the existing functionality.
- ▶ **Repeat:** You repeat this cycle, writing a new test for the next piece of functionality or improvement, and then writing the code to make that test pass.

Advantages of TDD

- ▶ 1. It is a systematic approach that means you can be confident that your tests cover all of the code that has been developed and that there are no untested code sections in the delivered code.
- ▶ 2. It should be possible to understand what the program does by reading the tests.
- ▶ 3. It is argued that TDD leads to simpler code, as programmers only write code that's necessary to pass tests.
- ▶ 4. TDD can lead to better software design and architecture decisions, as it forces developers to think about how their code should work and how it fits within the larger system.

SECURITY TESTING

- ▶ It aims to find vulnerabilities that an attacker may exploit and to provide convincing evidence that the system is sufficiently secure.
- ▶ The **tests should demonstrate that the system can resist attacks** on its availability, attacks that try to inject malware, and attacks that try to corrupt or steal users' data and identity.
- ▶ Comprehensive security testing requires specialist knowledge of software vulnerabilities and approaches to testing that can find these vulnerabilities.
- ▶ One practical way to organize security testing is to adopt **a risk-based approach**, where you identify the common risks and then develop tests to demonstrate that the system protects itself from these risks.
- ▶ You may also use automated tools that scan your system to check for known vulnerabilities.

Risk-based approach

- ▶ In a risk-based approach, you start by **identifying the main security risks to your product**. To identify these risks, you use knowledge of possible attacks, known vulnerabilities, and security problems.
- ▶ Based on the risks that have been identified, you then **design tests and checks to see if the system is vulnerable**.
- ▶ It may be possible to construct automated tests for some of these checks, but others inevitably involve manual checking of the system's behavior and its files.
- ▶ Once you have identified **security risks**, you then **analyse them to assess how they might arise**.

DEV - OPS AND CODE MANAGEMENT

- ▶ DevOps (development + operations) integrates development, deployment, and support, with a single team responsible for all of these activities.
- ▶ Three development method adopted DevOps:
 - ▶ 1. **Agile software development** made building software faster, but the old way of releasing it had a slowdown between making it and putting it out there.

- ▶ 2. **Amazon** changed how they built their software. They divided it into smaller parts called services, and the team that created a service also took care of it. Amazon said this made their software more reliable, and a lot of people heard about it.
 - ▶ *Eg: Amazon Lambda: Lambda is a serverless computing service that lets you run code without managing servers, automatically scaling based on the number of incoming requests.*
 - ▶ *Amazon SNS (Simple Notification Service): SNS enables the sending of messages and notifications to a distributed set of subscribers, such as SMS, email, and mobile devices*
- ▶ 3. It became possible to release **software as a service**, running on a public or private cloud.
 - ▶ *eg. Zoom: Zoom provides a cloud-based video conferencing and communication platform that has become widely used for virtual meetings and webinars*


- 
- ▶ DevOps aims to creating a single team that is responsible for both development and operations. **Developers also take responsibility for installing and maintaining their software.**
 - ▶ • Creating a **DevOps team means bringing together a number of different skill sets**, which may include software engineering, UX design, security engineering, infrastructure engineering, and customer interaction.
 - ▶ • A successful DevOps team has a culture of mutual respect and sharing. **Team members should be encouraged to share their expertise with others and to learn new skills.** Developers should support the software services that they have developed.

Table 10.1 DevOps principles

Principle	Explanation
Everyone is responsible for everything.	All team members have joint responsibility for developing, delivering, and supporting the software.
Everything that can be automated should be automated.	All activities involved in testing, deployment, and support should be automated if it is possible to do so. There should be minimal manual involvement in deploying software.
Measure first, change later.	DevOps should be driven by a measurement program where you collect data about the system and its operation. You then use the collected data to inform decisions about changing DevOps processes and tools.

Table 10.2 Benefits of DevOps

Benefit	Explanation
Faster deployment	Software can be deployed to production more quickly because communication delays between the people involved in the process are dramatically reduced.
Reduced risk	The increment of functionality in each release is small so there is less chance of feature interactions and other changes that cause system failures and outages.
Faster repair	DevOps teams work together to get the software up and running again as soon as possible. There is no need to discover which team was responsible for the problem and to wait for them to fix it.
More productive teams	DevOps teams are happier and more productive than the teams involved in the separate activities. Because team members are happier, they are less likely to leave to find jobs elsewhere.

CODE MANAGEMENT

- ▶ Code management is like a set of rules and tools that help us **control how we work on a computer program as it gets bigger and changes over time.**
- ▶ We use code management to make sure that **when different people work on the program, their changes don't cause problems, and we can create different versions of the program.**
- ▶ These tools also help us turn the code into a working program and check it for errors automatically.

Fundamentals of source code management

- Source code management systems help us **control and organize a project's code**. They allow us to save different versions of parts or the whole project.
- This way, developers can work on their parts at the same time without causing problems for each other.
- They can also combine their work with what others have done
- ▶ The code management system provides a set of features that support four general areas:

- ▶ 1. *Code transfer* Developers take code into their personal file store to work on it; then they return it to the shared code management system.
- ▶ 2. *Version storage and retrieval* Files may be stored in several different versions, and specific versions of these files can be retrieved.
- ▶ 3. *Merging and branching* Developers can work on different parts of a project at the same time by creating separate branches. When they're done, they can put their changes together by merging them
- ▶ 4. *Version information:* Information about the different versions maintained in the system may be stored and retrieved.

Table 10.4 Features of source code management systems

Feature	Description
Version and release identification	Managed versions of a code file are uniquely identified when they are submitted to the system and can be retrieved using their identifier and other file attributes.
Change history recording	The reasons changes to a code file have been made are recorded and maintained.
Independent development	Several developers can work on the same code file at the same time. When this is submitted to the code management system, a new version is created so that files are never overwritten by later changes.
Project support	All of the files associated with a project may be checked out at the same time. There is no need to check out files one at a time.
Storage management	The code management system includes efficient storage mechanisms so that it doesn't keep multiple copies of files that have only small differences.

DEV - OPS AUTOMATION- Continuous Integration (CI)

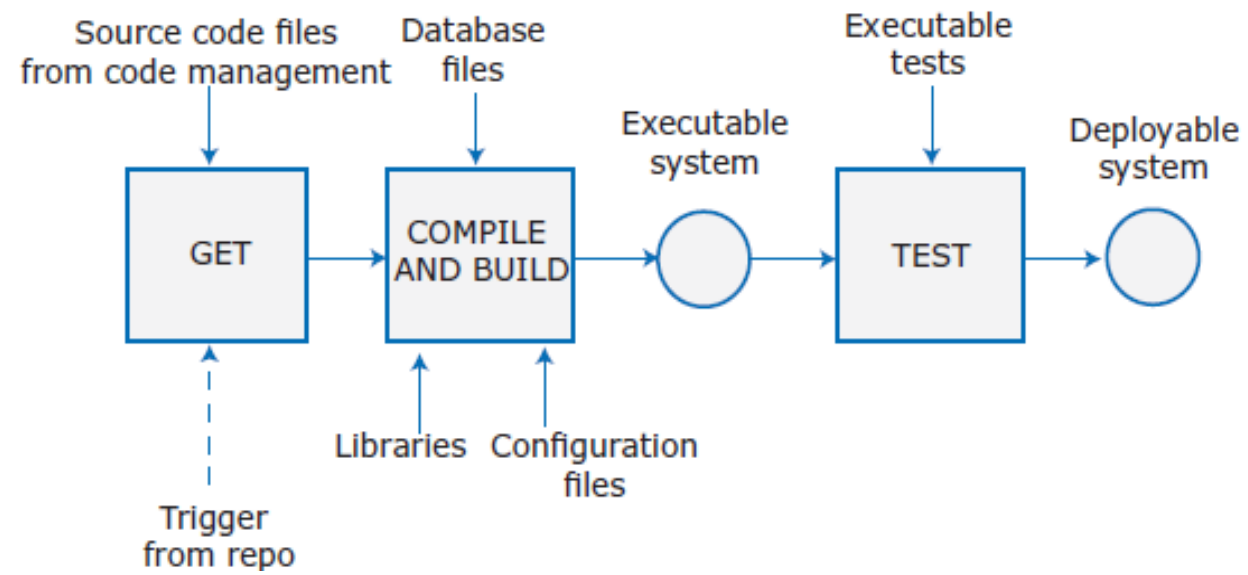
- ▶ Continuous Integration (CI) is a fundamental practice in DevOps that involves the constant and automated integration of code changes into a shared repository, typically multiple times a day. The primary goals of CI in the context of DevOps are:
- ▶ **Frequent Code Integration:** Developers regularly commit their code changes and these changes are then automatically integrated into a central repository.
- ▶ **Automated Builds:** As new code is integrated, an automated build process compiles the code, runs tests, and produces a build artifact (e.g., a compiled application or software package).

- ▶ **Automated Testing:** A suite of automated tests is run against the integrated code to ensure that new changes do not introduce defects or regressions.
- ▶ **Immediate Feedback:** Developers receive immediate feedback on whether their changes have passed the automated tests. If a problem arises, they can address it quickly.
- ▶ **Artifact Generation:** The CI process often results in the generation of deployable artifacts(compiled or packaged version of the software application) . These artifacts are then available for further testing and deployment

Advantages

- ▶ **Faster Development:** Developers can integrate and test their code continuously, which speeds up the development process.
- ▶ **Early Issue Detection:** Problems are identified and fixed early in the development cycle, reducing the cost and effort required for bug fixing.
- ▶ **Consistency:** CI ensures that the codebase is consistently built and tested, reducing configuration issues and variations between environments.
- ▶ **Increased Quality:** Automated testing helps maintain the overall quality of the software.
- ▶ **Fast Delivery:** With a CI/CD (Continuous Integration and Continuous Deployment) pipeline, code changes that pass CI tests can be automatically deployed to production, enabling rapid and reliable software deliver

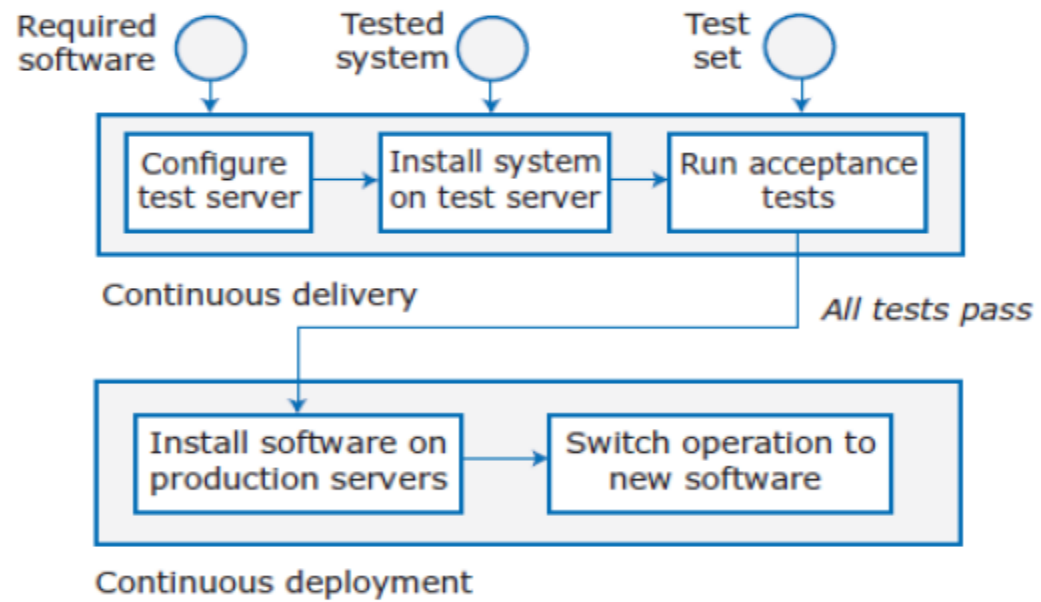
Figure 10.9 Continuous integration



Continuous Delivery and deployment

- ▶ Continuous delivery means that, after making changes to a system, you ensure that the changed system is ready for delivery to customers.
- ▶ This means that you have to test it in a production environment to make sure that environmental factors do not cause system failures or slow down its performance.

- ▶ Continuous Deployment (CD) is like having a super-efficient way to release software. It means that whenever developers make changes to the code, these changes are automatically tested and put into action without people.
- ▶ If the tests show everything's good, the changes are released to real users right away.
- ▶ This way, updates happen fast and it is an automatic process for getting new software features out to people.



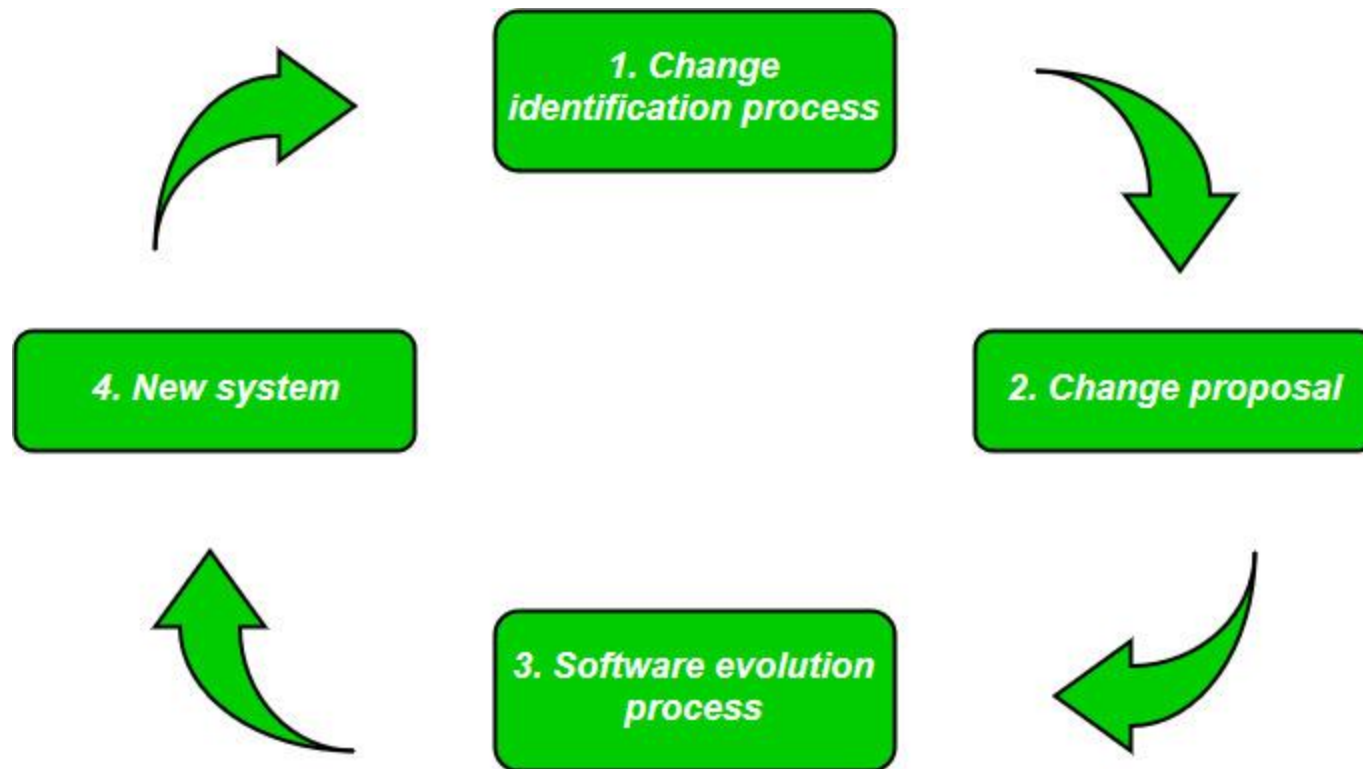
SOFTWARE EVOLUTION

- ▶ **Software Evolution** is a term which refers to the process of developing software initially, then timely updating it for various reasons, i.e., to add new features or to remove obsolete functionalities etc. The evolution process includes fundamental activities of change analysis, release planning, system implementation and releasing a system to customers.

Importance of Software Evolution

- ▶ **Functionality Check:** Software evaluation helps make sure that the software does what it's supposed to do. It's like making sure a TV remote actually changes channels and adjusts the volume.
- ▶ **Quality Assurance:** It's like checking food for freshness before eating. Software evaluation ensures the software is of good quality and doesn't have hidden issues or "bugs" that can cause problems.
- ▶ **Safety:** Software evaluation is like checking that a toy for children doesn't have small parts that could be dangerous. It helps ensure that the software won't harm your computer or your personal information.

- ▶ **User Satisfaction:** It's similar to testing a game before playing to see if you enjoy it. Software evaluation ensures that the software is user-friendly and provides a good experience.
- ▶ **Compliance:** Just like obeying traffic rules is necessary for safety, software evaluation ensures that the software complies with legal and industry standards.
- ▶ **Reliability:** Software evaluation is like checking the weather forecast before planning an outdoor event. It ensures that the software works reliably, so you can depend on it.
- ▶ **Cost-Effectiveness:** Software evaluation helps identify and fix issues early, preventing costly problems later. It's like fixing a small leak in a roof before it causes major damage

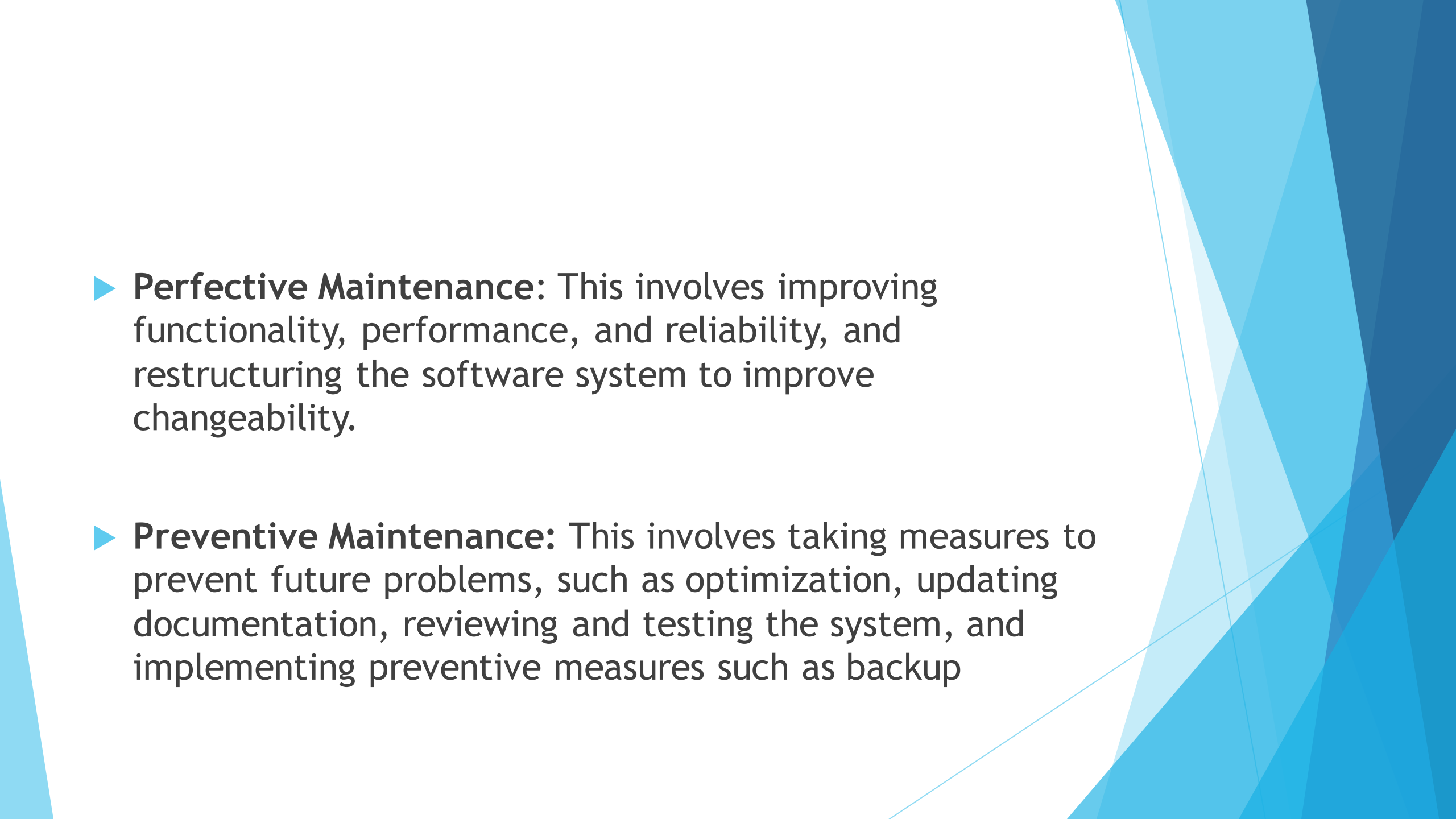


Software Maintenance

- ▶ **Software Maintenance** refers to the process of modifying and updating a software system after it has been delivered to the customer.
- ▶ This can include fixing bugs, adding new features, improving performance, or updating the software to work with new hardware or software systems.
- ▶ The goal of software maintenance is to keep the software system working correctly, efficiently, and securely, and to ensure that it continues to meet the needs of the users.

Types of Software Maintenance

- ▶ **Corrective Maintenance:** This involves fixing errors and bugs in the software system
- ▶ **Adaptive Maintenance:** This involves modifying the software system to adapt it to changes in the environment, such as changes in hardware or software, government policies, and business rules

- 
- The background of the slide features an abstract design with overlapping geometric shapes in various shades of blue, ranging from light sky blue to deep navy blue. These shapes are primarily located on the right side of the slide, creating a modern, tech-oriented aesthetic.
- ▶ **Perfective Maintenance:** This involves improving functionality, performance, and reliability, and restructuring the software system to improve changeability.
 - ▶ **Preventive Maintenance:** This involves taking measures to prevent future problems, such as optimization, updating documentation, reviewing and testing the system, and implementing preventive measures such as backup

Refactoring

- ▶ **Refactoring or Code Refactoring** is defined as systematic process of improving existing computer code, without adding new functionality or changing external behaviour of the code.
- ▶ It is intended to change the implementation, definition, structure of code without changing functionality of software.
- ▶ It improves extensibility, maintainability, and readability of software without changing what it actually does

- ▶ The goal of refactoring is not to add new functionality or remove an existing one. The main goal of refactoring is to make code easier to maintain in future and to fight technical debt.
- ▶ We do refactor because we understand that getting design right in first time is hard and also you get the following benefits from refactoring:

Code size is often reduced

Confusing code is restructured into simpler code

Re-Engineering

- ▶ Re-engineering, also known as reverse engineering or software re-engineering, is the process of analyzing, designing, and modifying existing software systems to improve their quality, performance, and maintainability.
- ▶ This can include updating the software to work with new hardware or software platforms, adding new features, or improving the software's overall design and architecture.

Refactoring vs Re-Engineering

- ▶ Reengineering takes place after a system has been maintained for some time, and maintenance costs are increasing.
- ▶ You use automated tools to process and reengineer a legacy system to create a new system that is more maintainable.
- ▶ Refactoring is a continuous process of improvement throughout the development and evolution process. It is intended to avoid the structure and code degradation that increases the costs and difficulties of maintaining a system.

Aspect	Refactoring	Reengineering
Goal	Improve the code's internal structure and design without changing its external behavior.	Make substantial changes to the software to achieve new functionality or meet new requirements.
Scope	Focused on small, incremental changes to specific portions of the codebase.	Involves a more extensive and comprehensive overhaul of the software, potentially affecting multiple modules or components.
Purpose	Enhance maintainability, readability, and reduce technical debt.	Address major architectural or structural deficiencies, adapt to new technologies, or implement significant new features.
Risk	Lower risk because changes are limited in scope and do not alter the software's fundamental behavior.	Higher risk due to the potential for introducing new issues, especially when making significant alterations to the software.
Frequency	A continuous and ongoing process throughout the software's lifecycle.	Typically done at specific milestones or when major changes are required.
Examples	Renaming variables, restructuring code, improving code comments, or extracting methods.	Replacing an old database system with a new one, migrating a monolithic application to microservices, or implementing a new feature requiring substantial changes.