SWE-417 SOFTWARE REENGINEERING

Software Engineering Department

Sir Syed University of Engineering & Technology

Week No. 3 & 4

CONTENTS

- Overview
  - Legacy system
  - Impact Analysis
  - Refactoring
  - Program Comprehension
  - Software Reuse

LEGACY SYSTEM

- A legacy software system is an old program that continues to be used because it still meets the users' needs, in spite of the availability of newer technology or more efficient methods of performing the task.
- A legacy system includes outdated procedures or terminology, and it is very difficult for new developers to understand the system.
- A legacy system falls in the Phase out stage of the software evolution model.

LEGACY SYSTEM

- Organizations continue to use legacy systems because those are vital to them and the systems significantly resist modification and evolution to meet new and constantly changing business requirements.
- Legacy systems are typically associated with the following characteristics: Aging Technology, Critical Business Functions, Limited Support, High Maintenance Costs, Risk and Vulnerability, Performance Issues etc.

LEGACY SYSTEM

To manage legacy systems, a number of options are available:

- **Freeze**
- **Outsource**
- **Carry on Maintenance**
- **Discard and redevelop**
- **Wrap**
- **Migrate**

IMPACT ANALYSIS

- Definition 1: Impact analysis is the task of estimating the parts of the software that can be affected if a proposed change request is made.
- The outcome of impact analysis can be used when planning for changes, making changes, and tracking the effects of changes in order to localize the sources of new faults.
- Definition 2: Impact analysis is a systematic process of evaluating and understanding the potential effects, consequences, or implications of a change, decision, or event within a specific context
- The primary goal of impact analysis is to assess the potential outcomes, both positive and negative, to make informed decisions, plan for possibilities, and manage change effectively.

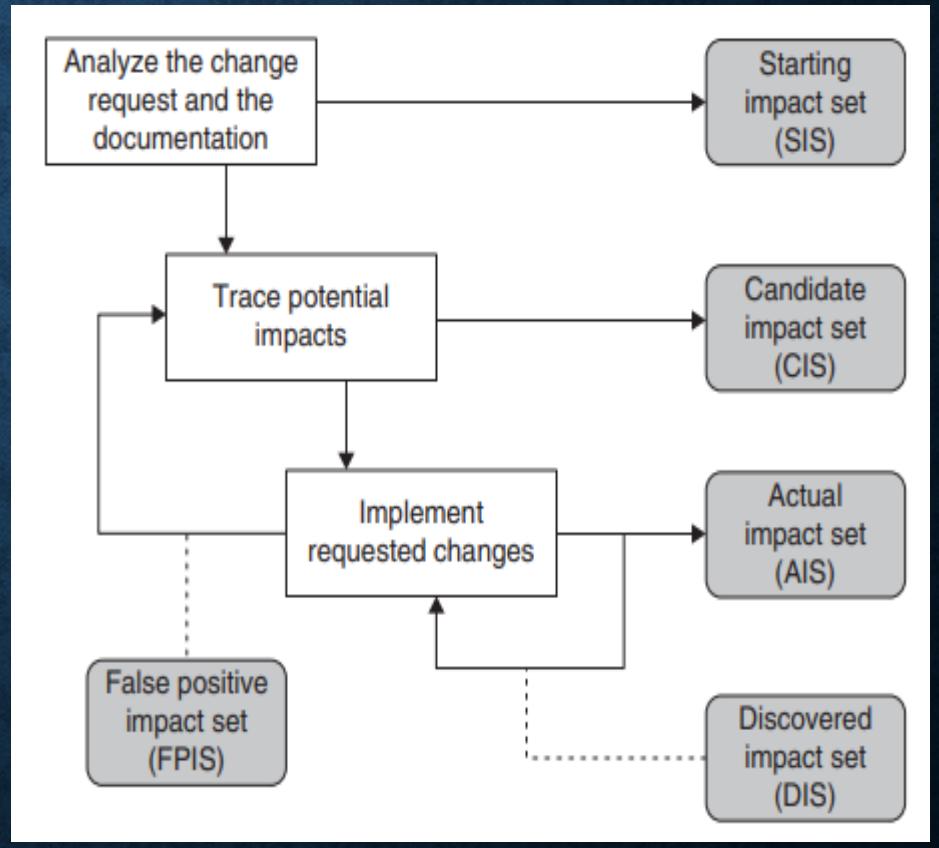
TYPES OF IMPACT ANALYSIS

- Impact analysis techniques can be categorized into two types/classes as follow:
  - Traceability analysis
  - Dependency analysis

- "Traceability impact analysis" and "dependency impact analysis" are two specific types of impact analysis techniques commonly used in software development, project management, and system engineering.

IMPACT ANALYSIS PROCESS

- Starting Impact Set (SIS): The initial set of objects (or components) presumed to be impacted by a software CR is called SIS.
- Candidate Impact Set (CIS): The set of objects (or components) estimated to be impacted according to a certain impact analysis approach is called CIS.
- Discovered Impact Set (DIS): DIS is defined as the set of new objects (or components), not contained in CIS, discovered to be impacted while implementing a CR.
- Actual Impact Set (AIS): The set of objects (or components) actually changed as a result of performing a CR is denoted by AIS.
- False Positive Impact Set (FPIS): FPIS is defined as the set of objects (or components) estimated to be impacted by an implementation of a CR but not actually impacted by the CR. Precisely, FPIS = (CIS ∪ DIS) \ AIS.



TYPES OF IMPACT ANALYSIS

- Traceability analysis:
  - **Purpose:** Traceability impact analysis is used to understand how a proposed change or update to a software system or project will affect related elements, requirements, and artifacts.
  - **Key Concepts:**
    - **Traceability Matrix:** A traceability matrix is a tool used in traceability impact analysis to map relationships between requirements, design specifications, test cases, and other project elements.
    - **Bidirectional Tracing:** This involves establishing links or traceability between various project elements, allowing for easy tracking of changes and their impacts.

TYPES OF IMPACT ANALYSIS

- **Process:**

1.1: Identify the proposed change or update.

1.2: Use the traceability matrix to trace the relationships between the affected requirements, design components, and other related project elements.

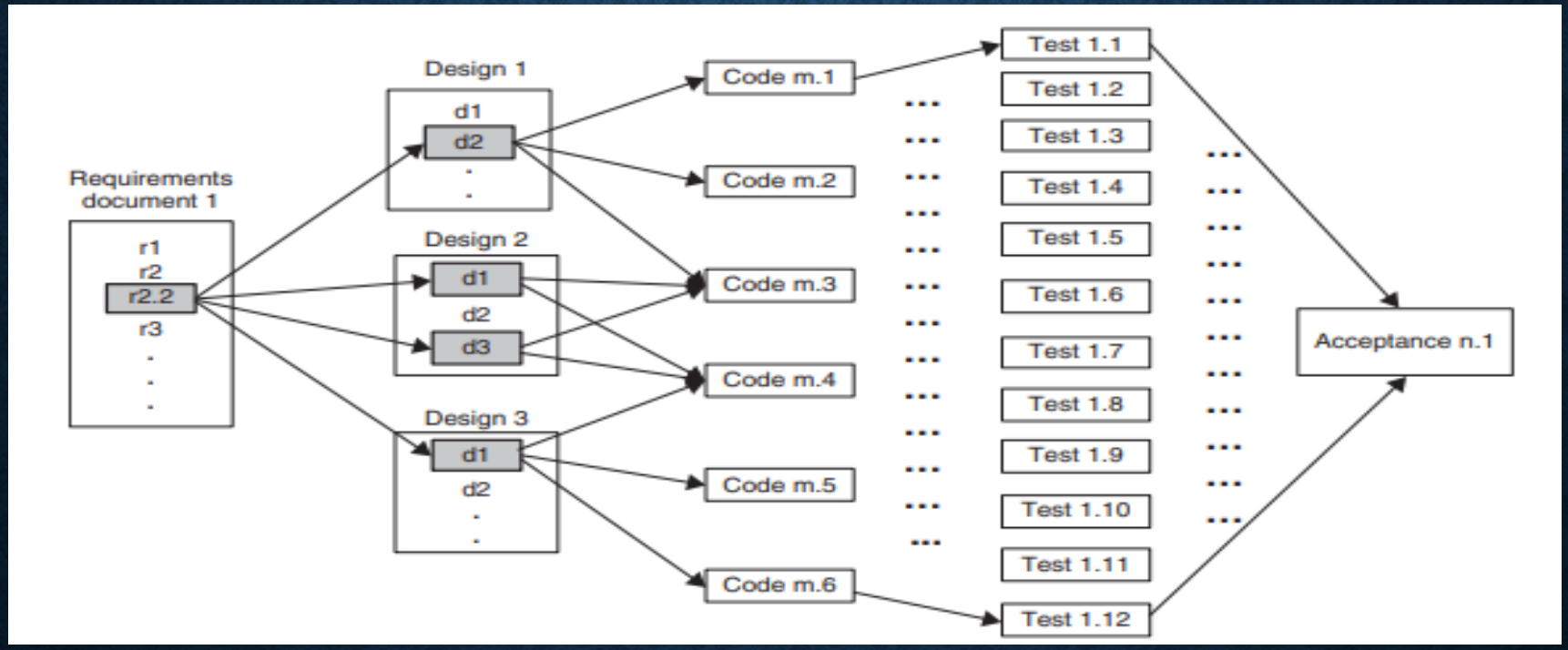
1.3: Analyze the impact by considering how changes in one area will affect linked elements.

1.4: Assess the consequences and determine necessary actions.

TYPES OF IMPACT ANALYSIS

- Use cases:
  - Estimating the effect of requirement changes on design, code, and tests.
  - Ensuring all requirements are covered in the implementation and testing.

TYPES OF IMPACT ANALYSIS



Traceability in software work products

EXAMPLE OF TRACEABILITY ANALYSIS

Scenario:

You are working on a healthcare management system, and the project team has proposed a change in the patient appointment scheduling feature to add automated appointment reminders via email and SMS.

Apply Traceability analysis process:

- Step 1: Identify the proposed change or update

The proposed change is to implement automated reminders for patients about their upcoming appointments. This includes sending notifications via email and SMS a few days before the appointment date.

EXAMPLE OF TRACEABILITY ANALYSIS

- Step 2: Use the traceability matrix to trace the relationships between the affected requirements, design components, and other related project elements.

Using the traceability matrix, you link this new requirement (automated reminders) to all related elements in the system:

Requirement ID	Requirement Description	Design Document ID	Design Description	Code Component(s)	Test Case ID(s)
R1	System must allow patients to schedule appointments.	D1	Design for the patient appointment scheduling workflow.	AppointmentScheduler.java, SchedulerController.java	TC1, TC2, TC3
R2	System must send email reminders to patients for upcoming appointments.	D2	Design for email notification flow after an appointment is scheduled.	EmailService.java, AppointmentNotification.java	TC4, TC5, TC6

EXAMPLE OF TRACEABILITY ANALYSIS

Requirement ID	Requirement Description	Design Document ID	Design Description	Code Component(s)	Test Case ID(s)
R3	System must send SMS reminders to patients for upcoming appointments.	D3	Design for SMS notification flow integrated into the scheduling system.	SMSService.java, AppointmentNotification.java	TC7, TC8, TC9
R4	System must allow customization of the reminder interval (e. g., 1 day before the appointment)	D4	Design to include reminder settings during appointment scheduling.	ReminderConfig.java, SchedulerSettings.java	TC10, TC11
R5	System must handle notification failures (retry logic, error logging).	D5	Design to incorporate error handling and retry mechanisms for notifications.	ErrorHandler.java, NotificationRetry.java	TC12, TC13, TC14

- Step 3: Analyze the impact by considering how changes in one area will affect linked elements.

EXAMPLE OF TRACEABILITY ANALYSIS

- Requirement Impact: The new reminder functionality needs to be integrated with the existing scheduling logic, meaning any updates to the reminder system should not break the appointment scheduling functionality.
- Design Impact: The flow of scheduling must now include a step for triggering reminders. This also introduces dependencies on external services (email, SMS providers), which could fail and need to be handled.
- Code Impact: Several modules will be affected, including the core appointment scheduling logic, which needs to integrate with the notification system. Additionally, third-party APIs (e.g., email and SMS providers) need to be incorporated.
- Testing Impact: The test cases will need to cover both positive and negative scenarios, ensuring reminders are sent only when appropriate and that errors in sending do not impact the core scheduling functionality.

EXAMPLE OF TRACEABILITY ANALYSIS

- Step 4: Assess the consequences and determine necessary actions

Consequences: The scheduling system now depends on third-party services (for email and SMS). If those services fail, appointments might not be updated or confirmed correctly. Furthermore, database changes might be needed to store reminder preferences for patients.

Necessary Actions:

- Update design documents to reflect changes in both scheduling and notification flows.
- Modify the scheduling module and introduce new components for email and SMS notifications. Add error-handling mechanisms for third-party failures.
- Write new test cases and update the existing ones to verify correct reminder functionality.

TYPES OF IMPACT ANALYSIS

- Dependency analysis:
  - **Purpose:** Dependency impact analysis focuses on understanding how interdependencies between various components, modules, or tasks within a project or system may be affected by changes or delays.
  - **Key Concepts:**
    - **Dependency Mapping:** Identifying and documenting the dependencies between project tasks, software modules, or components.
    - **Critical Path Analysis:** Determining the critical path in a project schedule, which is the longest sequence of dependent tasks that can delay the project's completion if delayed.

TYPES OF IMPACT ANALYSIS

- **Process:**

2.1: Identify the change or delay in one or more components, tasks, or modules.

2.2: Analyze how this change affects the dependencies between various elements.

2.3: Identify potential bottlenecks or risks introduced by the change using graph.

TYPES OF IMPACT ANALYSIS

- Use cases:
  - Identifying the parts of the system that rely on a particular module or service.
  - Assessing the ripple effect of changes across different layers of the software architecture.
  - Refactoring code by understanding the dependencies that may affect maintainability.

EXAMPLE OF DEPENDENCY ANALYSIS

Scenario:

You are working on a large e-commerce system, and the payment gateway module needs to be refactored due to performance issues.

Apply Dependency analysis process:

- Step 1: Identify the Change or Delay in Components, Tasks, or Modules

Proposed Change: The payment gateway module is being refactored to enhance performance, such as optimizing how the system processes payments, reducing the response time for transactions, and improving handling of concurrent transactions. Key areas that might be altered include: Transaction Processing Logic, & Data Management.

EXAMPLE OF DEPENDENCY ANALYSIS

Impact on Components:

- Order Processing Module: The order processing logic relies on successful payment confirmation to complete orders.
- Inventory Management System: Inventory is updated only after successful payment; any delay in payments could delay inventory updates.
- Notification Services: Customer notifications (e.g., payment confirmations, order shipped) depend on payment completion events.

- Step 2: Analyze How This Change Affects the Dependencies Between Elements

To analyze how the payment gateway refactor affects the system's dependencies, we examine the direct and indirect dependencies that exist within the system.

EXAMPLE OF DEPENDENCY ANALYSIS

- Direct Dependencies:
  - Order Processing Module → Payment Gateway Module
  - Inventory System → Order Processing
  - Notification Services → Payment Gateway
- Indirect Dependencies:
  - Analytics and Reporting → Payment Gateway → Order Processing
- Step 3: Identify Potential Bottlenecks or Risks Introduced by the Change
  - Performance Bottleneck in Upstream Systems: Faster payments might overwhelm the order processing system if it cannot handle the new speed efficiently.

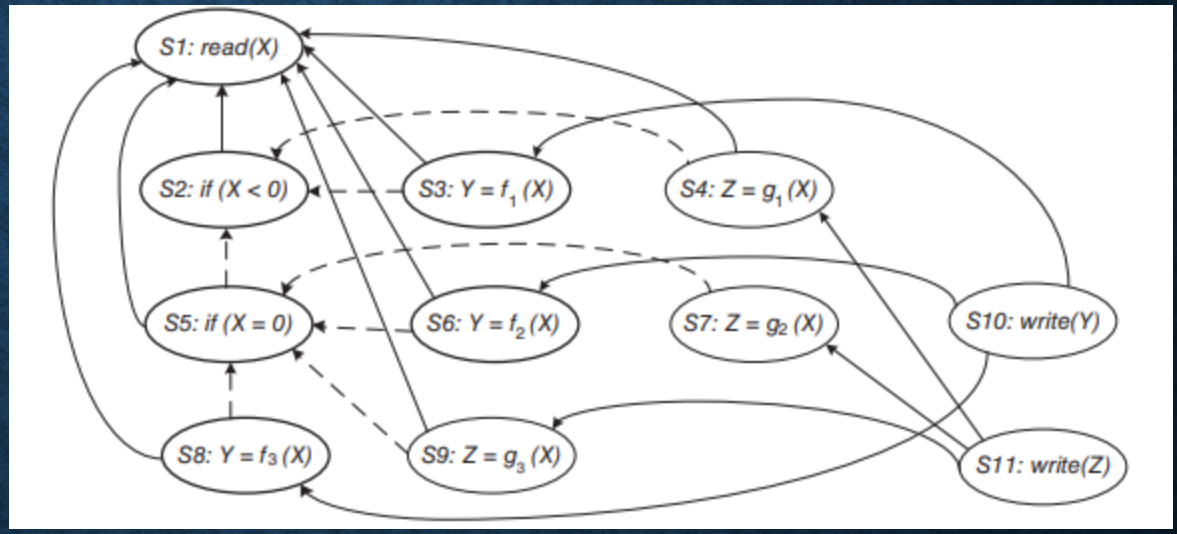
EXAMPLE OF DEPENDENCY ANALYSIS

- Concurrency Issues: Two simultaneous transactions for the same item could lead to inventory mismatches or double charges if not handled correctly.
- Delay in Customer Notifications: If the refactor changes the payment flow in a way that delays the confirmation of payments, customer notifications might be sent out late, leading to customer confusion or dissatisfaction.

EXAMPLE 2

```
    begin
S1:     read(X)
S2:     if (X < 0)
        then
S3:         Y = f1 (X);
S4:         Z = g1 (X);
        else
S5:         if (X = 0)
            then
S6:             Y = f2 (X);
S7:             Z = g2 (X);
            else
S8:             Y = f3 (X);
S9:             Z = g3 (X);
            end_if;
        end_if;
S10:    write(Y);
S11:    write(Z);
    end.
```

Example program



Program dependency graph of the program

TWO ADDITIONAL NOTIONS OF IMPACT ANALYSIS

- Two additional notions related to impact analysis are very common among practitioners:
  - Ripple effect
  - Change propagation.

IMPACT ANALYSIS

- Ripple effect
  - **Definition:** The ripple effect refers to the unintended and often unforeseen consequences of a change in a software system, project, or process that ripple through the system, affecting various interconnected components.
  - **Characteristics:**
    - It often results from unplanned or unexpected side effects of changes, which can lead to issues, defects, or disruptions in other parts of the system.
    - The ripple effect can extend to dependencies and components that were not immediately obvious, requiring additional testing and corrections.

EXAMPLE: RIPPLE EFFECT

- **Scenario:** Consider a web application that has a registration feature and a user profile feature. The user registration form includes fields for username, email, and password. The user profile page displays the user's information, including their username.

Apply Ripple effect:

- **Change Request:** A user reports that their username is not displaying correctly on the user profile page. The development team investigates and identifies a minor bug in the code responsible for retrieving and displaying the username on the profile page.
- **Fix Implementation:** The development team proceeds to fix the bug in the code for the user profile feature. They make the necessary code adjustments to correctly retrieve and display the username.

EXAMPLE: RIPPLE EFFECT

- **Unintended Consequence:** After deploying the bug fix, they discover an unintended consequence. Now, whenever a new user registers on the application, their username is not displayed correctly in the user profile. The registration and user profile features were indirectly connected through the shared code for username handling.
- **Ripple Effect:** The seemingly minor fix in the user profile code caused a ripple effect by unintentionally affecting the user registration feature. Users who registered after the fix were experiencing issues with their displayed usernames.

EXAMPLE: RIPPLE EFFECT

Note:

- In this example, the ripple effect is evident. A change made to one part of the system had unintended consequences that affected a different part of the system, and these consequences were not immediately apparent.
- This highlights the importance of thorough testing and impact analysis when making changes to software systems to prevent and address such unintended side effects.

TWO ADDITIONAL NOTIONS OF IMPACT ANALYSIS

- **Change Propagation:**
  - **Definition:** Change propagation is a controlled and deliberate process of applying planned changes to ensure that those changes are consistently and intentionally distributed to all affected components within a system, project, or process.
  - **Characteristics:**
    - It is a systematic approach that involves identifying the scope of changes, their impact, and ensuring that changes are uniformly and accurately applied throughout the system.
    - Change propagation is often a part of planned updates, maintenance, or development processes.

EXAMPLE OF CHANGE PROPAGATION

- **Scenario:** A project manager is overseeing the development of a complex software application. During the project, a decision is made to change the underlying database management system, as the current one no longer meets the project's requirements.

Apply Change Propagation:

- **Identifying the Need for Change:** The project team realizes that the current database management system is not suitable for the project's long-term scalability and performance needs. They decide to change to a different, more robust database system.

EXAMPLE OF CHANGE PROPAGATION

- Planning the Database Change: The project manager creates a change plan, which includes:
  - The scope of the change: Migrating from the old database system to the new one.
  - Impact analysis: Understanding how this change will affect the project's timeline, resources, and deliverables.
  - Affected areas: Identifying all project components that rely on the current database system.

EXAMPLE OF CHANGE PROPAGATION

- **Applying the Database Change:** The database change is systematically propagated throughout the project:
  - The codebase is updated to use the new database system's APIs.
  - Data migration scripts are developed to transfer data from the old database to the new one.
  - Any project components, such as reporting modules, that rely on the database are modified to work with the new system.
- Testing and Validation: Before proceeding with the database change, the project team conducts thorough testing to ensure the new database system is compatible with the existing software, and that data migration can be performed without data loss or corruption.

EXAMPLE OF CHANGE PROPAGATION

- **Testing and Validation (Again):** After applying the change, the team conducts comprehensive testing to ensure that the project functions correctly with the new database system. This includes verifying that data integrity is maintained.
- **Documentation and Communication:** The change propagation process includes documenting the changes made, communicating with all team members about the transition, and training personnel on how to work with the new database system.
- **Monitoring and Follow-Up:** The project manager continues to monitor the project to ensure that the change has been successfully propagated and that the project's performance and scalability are improved as intended.

REFACTORING

- Refactoring is the process of making a change to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.
- The **primary goal** of refactoring is to enhance code quality, maintainability, readability, and efficiency, while reducing technical debt and making future development and maintenance easier.
- Refactoring, which aims to improve the internal structure of the code, achieve through the removal of duplicate code, simplification, making code easier to understand, help to find defects and adding flexibility to program faster.

REFACTORING

- In an agile software methodology, such as eXtreme Programming (XP), refactoring is continuously applied to:
  - Make the architecture of the software stable;
  - Render the code readable; and
  - Make the tasks of integrating new functionalities into the system flexible.

PROGRAM COMPREHENSION

- Program comprehension, in the context of software development, refers to the process of understanding and gaining insight into existing software systems, codebases, or programs.

- It involves analyzing, interpreting, and making sense of the structure, behavior, and functionality of software, whether written by oneself or by others.

PROGRAM COMPREHENSION

- Program understanding or comprehension is “the task of building mental models of an underlying software system at various abstraction levels, ranging from models of the code itself to ones of the underlying application domain, for software maintenance, evolution and re-engineering purposes.”
- A mental model describes a programmer’s mental representation of the program to be comprehended.
- Program comprehension deals with the cognitive processes involved in constructing a mental model of the program.

PROGRAM COMPREHENSION

- A common element of such cognitive models is generating hypotheses and investigating whether they hold or must be rejected.
- Hypotheses are a way to understand code in an incremental manner.
- After some understanding of the code, the programmer forms a hypothesis and verifies it by reading code.
- By continuously formulating new hypotheses and verifying them, the programmer understands more and more code and in increasing details.

PROGRAM COMPREHENSION

- Several strategies can be used to arrive at relevant hypotheses such as:
  - bottom up (starting from the code).
  - top down (starting from high-level goal).
  - opportunistic combinations of the two.
- A strategy is formulated by identifying actions to achieve a goal.

SOFTWARE REUSE

- Software reuse was introduced by Dough McIlroy in his 1968 seminal paper.
- Software reuse is the practice of using existing software components, modules, libraries, or frameworks to create new software applications, rather than starting development from scratch.
- The primary goal of software reuse is to leverage existing assets to improve development efficiency, reduce development costs, and enhance software quality.

SOFTWARE REUSE

Key concepts and aspects of software reuse include:

- **Reusable Components**
- **Code Libraries**
- **Design Patterns**
- **Frameworks**
- **Maintainability, Consistency, Development Efficiency**
- **Cost Savings**
- **Quality Improvement**

SOFTWARE REUSE

- Reusable assets can be either reusable artifacts or software knowledge.
- Capers Jones identified four types of reusable artifacts:
  - Data reuse, involving a standardization of data formats,
  - Architectural reuse, which consists of standardizing a set of design and programming conventions dealing with the logical organization of software,
  - Design reuse, for some common business applications, and
  - Program reuse, which deals with reusing executable code.

SOFTWARE REUSE

- Software reuse of previously written code is a way to increase:
  - Software development productivity.
  - Quality of the software.
- The cost savings during maintenance as a consequence of reuse are nearly twice the corresponding savings during development.
- Reusability is a property of a software assets that indicates the degree to which it can be reused.

SOFTWARE REUSE

- For software component to be reusable, it must exhibit the following characteristics that directly encourage its use is similar situation:
  - Environmental independence - The components can be reused irrespective of the environment from which they were originally captured.
  - High cohesion - The components that implement a single operation or set of related operations.
  - Loose coupling - The components that have minimal links to other components.

SOFTWARE REUSE

- Benefits of Reuse
  - Increased reliability.
  - Reduced process risk.
  - Increase productivity.
  - Standards compliance.
  - Improve maintainability.
  - Reduction in maintenance time and effort.

SUMMARY

- A legacy system is an old or outdated system still in use, often because it is essential to business operations.
- Impact analysis is the process of evaluating the potential consequences of changes in the software system, such as adding new features, refactoring, or fixing bugs.
- While refactoring enhances long-term maintainability, it must be done carefully to avoid introducing new bugs or performance issues.
- Code reuse with sharing libraries, APIs, or utilities.