Chapter 1- Introduction and Software Processes

Topics covered

- ✧ Basic definition
- ✧ Role of Management in Software Development
- ✧ Software Products
- ✧ Essential attributes of good software
- ✧ Challenges for Software Engineering Practices
- ✧ Software Engineering Diversity
- ✧ Software Life Cycle
- ✧ Software Process Model and its Types

1.1 Basic Definition

- What is software?

  Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.

- What is software engineering?

  Software engineering is an engineering discipline that is concerned with all aspects of software production from initial conception to operation and maintenance.

- The IEEE [IEE17] has developed the following definition for software engineering:

  The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

1.2 Role of Management in Software Development

Factor	Role
People	Software development requires good managers. The manager who can understand the psychology of the people and provide good leadership. After having a good manager, project is in safe hands. It is the responsibility of a manager to manage, motivate, encourage, guide and control the people of his/her team.
Product	What is delivered to the customer, is called a product. It may include source code, specification document, manuals, documentation etc. Basically, it is nothing but a set of deliverables only.
Process	Process is the way in which we produce software. It is the collection of activities that leads to (a part of) a product. An efficient process is required to produce good quality products. If the process is weak, the end product will undoubtedly suffer, but an obsessive over reliance on process is also dangerous.
Project	A proper planning is required to monitor the status of development and to control the complexity. Most of the projects are coming late with cost overruns of more than 100%. In order to manage a successful project, we must understand what can go wrong and how to do it wright.

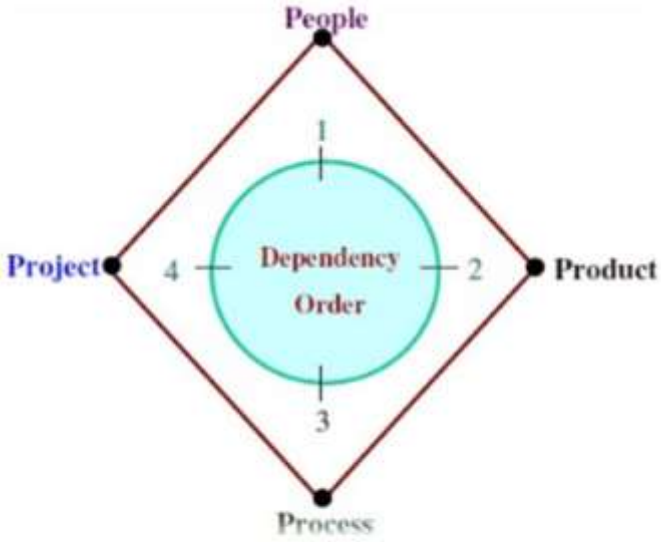


Figure 1.1 4Ps of Software Engineering

1.3 Software products

✧ Generic products

- Stand-alone systems that are marketed and sold to any customer who wishes to buy them.
- Examples – PC software such as graphics programs, project management tools; CAD software; software for specific markets such as appointments systems for dentists.

✧ Customized products

- Software that is commissioned by a specific customer to meet their own needs.
- Examples – embedded control systems, air traffic control software, traffic monitoring systems.

1.4 Essential attributes of good software

Product characteristic	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable and compatible with other systems that they use.

1.5 Challenges for Software Engineering Practices

1 ✧ Heterogeneity

- Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.

2 ✧ Business and social change

- Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software.

3 ✧ Security and trust

- As software is intertwined with all aspects of our lives, it is essential that we can trust that software.

1.6 Software Engineering Diversity

- There are many different types of software system and there is no universal set of software techniques that is applicable to all of these. Rather, a diverse set of software engineering methods and tools has evolved.
- The software engineering methods and tools used depend on the type of application being developed, the requirements of the customer and the background of the development team. There are many different types of application, including:

1. **Stand-alone applications**
2. **Interactive transaction-based applications**
3. **Embedded control systems**
4. **Batch processing systems**
5. **Entertainment systems**
6. **Systems for modeling and simulation**
7. **Data collection systems**
8. **Systems of systems**

Software Engineering Diversity : Application types

- ✧ Stand-alone applications
  - These are application systems that run on a local computer, such as a PC. They include all necessary functionality and do not need to be connected to a network.
- ✧ Interactive transaction-based applications
  - Applications that execute on a remote computer and are accessed by users from their own PCs or terminals. These include web applications such as e-commerce applications.
- ✧ Embedded control systems
  - These are software control systems that control and manage hardware devices. Numerically, there are probably more embedded systems than any other type of system.
- ✧ Batch processing systems
  - These are business systems that are designed to process data in large batches. They process large numbers of individual inputs to create corresponding outputs.

Software Engineering Diversity : Application types

- ✧ Entertainment systems
  - These are systems that are primarily for personal use and which are intended to entertain the user.
- ✧ Systems for modeling and simulation
  - These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects.
- ✧ Data collection systems
  - These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing.
- ✧ Systems of systems
  - These are systems that are composed of a number of other software systems.

1.7 Software Life cycle

- The goal of Software Engineering is to provide models and processes that lead to the production of well-documented maintainable software in a manner that is predictable.
- In the IEEE standard Glossary of Software Engineering Terminology, the **software life cycle** is: The period of time that starts when a software product is conceived and ends when the product is no longer available for use.
- The software life cycle typically includes a requirement phase, design phase, implementation phase, test phase, installation and check out phase, operation and maintenance phase, and

1.8 Software Process Model

- **Software process** is defined as the structured set of activities that are required to develop the software system.
- Many different software processes but all involve:

  Specification – defining what the system should do;

  Design and implementation – defining the organization of the system and implementing the system;

  Validation – checking that it does what the customer wants;

  Evolution – changing the system in response to changing customer needs.

- A **software process model** is an abstract representation of a

1.9 Types of Software process models

There are many types of software process models that development teams use across different industries. Here are some examples of typical software process models you can use to outline your development process:

1. The waterfall model
2. Incremental development model
3. Reuse-oriented Model
4. Boehm's Spiral Model
5. Agile methods

1.9.1The waterfall model

- Waterfall model is the earliest software development life cycle (SDLC) approach that was used for software development. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use.
- In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases. All phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases.
- In Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model".

The waterfall model

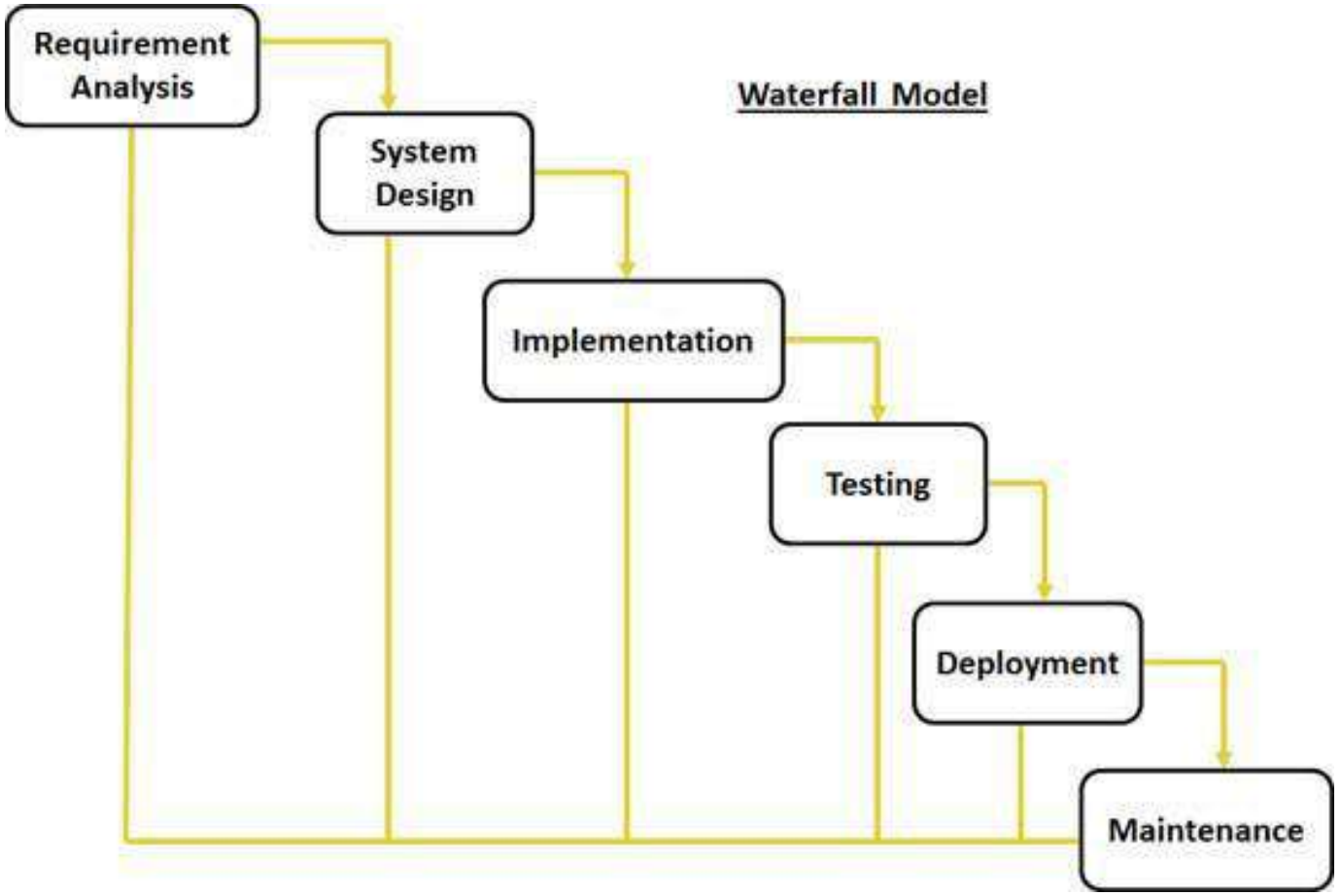


Figure 1.2: Waterfall Model

Waterfall model phases

Phase	Description
Requirement Gathering and analysis	All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.
System Design	The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.
Implementation	With inputs from system design, the system is first developed in small programs called units, which are integrated (وحدات متكاملة) in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.
Integration and Testing	All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
Deployment of system	Once the functional and non-functional testing is done, the product is deployed in the customer environment (تثبيت البرنامج للاستخدام) or released into the market.
Maintenance	There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

1.9.2 Incremental Development Model

- Incremental model uses the linear sequential approach with the iterative nature of prototyping. Incremental development is based on the idea of developing an initial implementation, delivering it to user for feedback and improving it through several versions of product releases until an acceptable system is developed.
- In incremental model the whole requirement is divided into various builds.

  During each iteration, the development module goes through the requirements, design, implementation and testing phase.

  Each subsequent release of the module adds function to the previous

Incremental Development Model

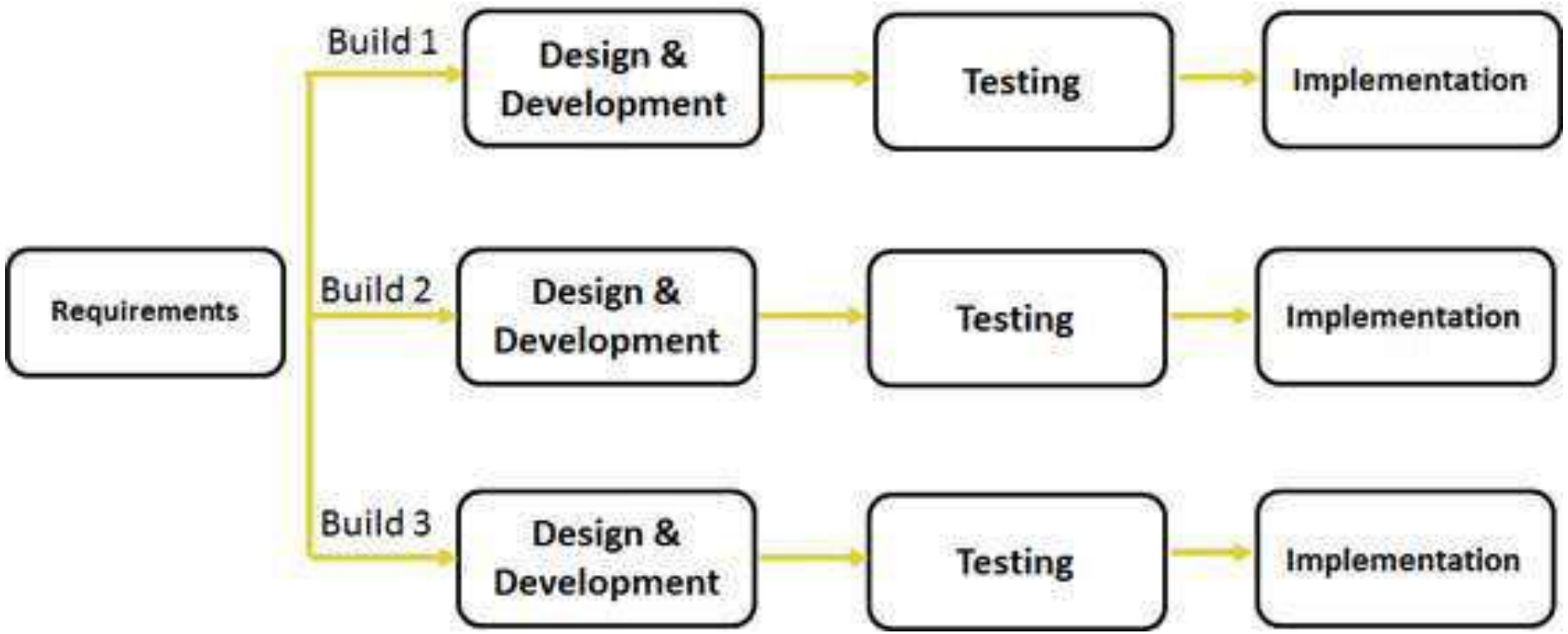


Figure 1.3: Incremental Development Model

1.9.3 Reuse-oriented Model

- The **reuse-oriented development** is where they reuse programming or software in previous projects or existing software components.
- Reuse-oriented development is based on systematic reuse where systems are integrated from existing components.
- This development can reduce the overall cost of software deve e of the proce en refin

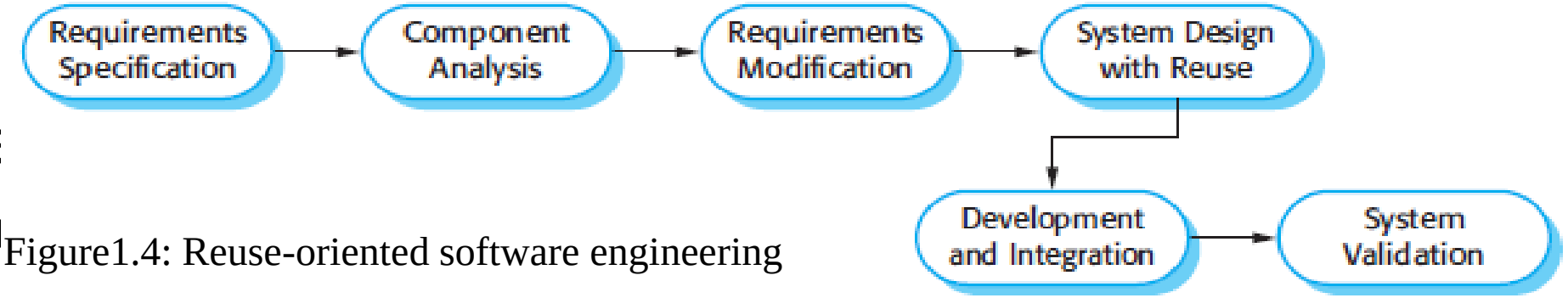


Figure1.4: Reuse-oriented software engineering

Reuse-oriented Model phases

1. **Requirement Specification:** All possible requirements of the system to be developed are captured in this phase.

2**. Component Analysis:** According to given requirement, component is selected to implement that requirement specification. That is not possible the selected component provide the complete functionality, but that is possible the component used provide some of the functionality required.

3. **Requirement Modification**: Information about component that is selected during component analysis is used to analysis requirement specification. Requirements are modified according to available components. Requirement modification is critical then component analysis activity is reused to find relative solution.

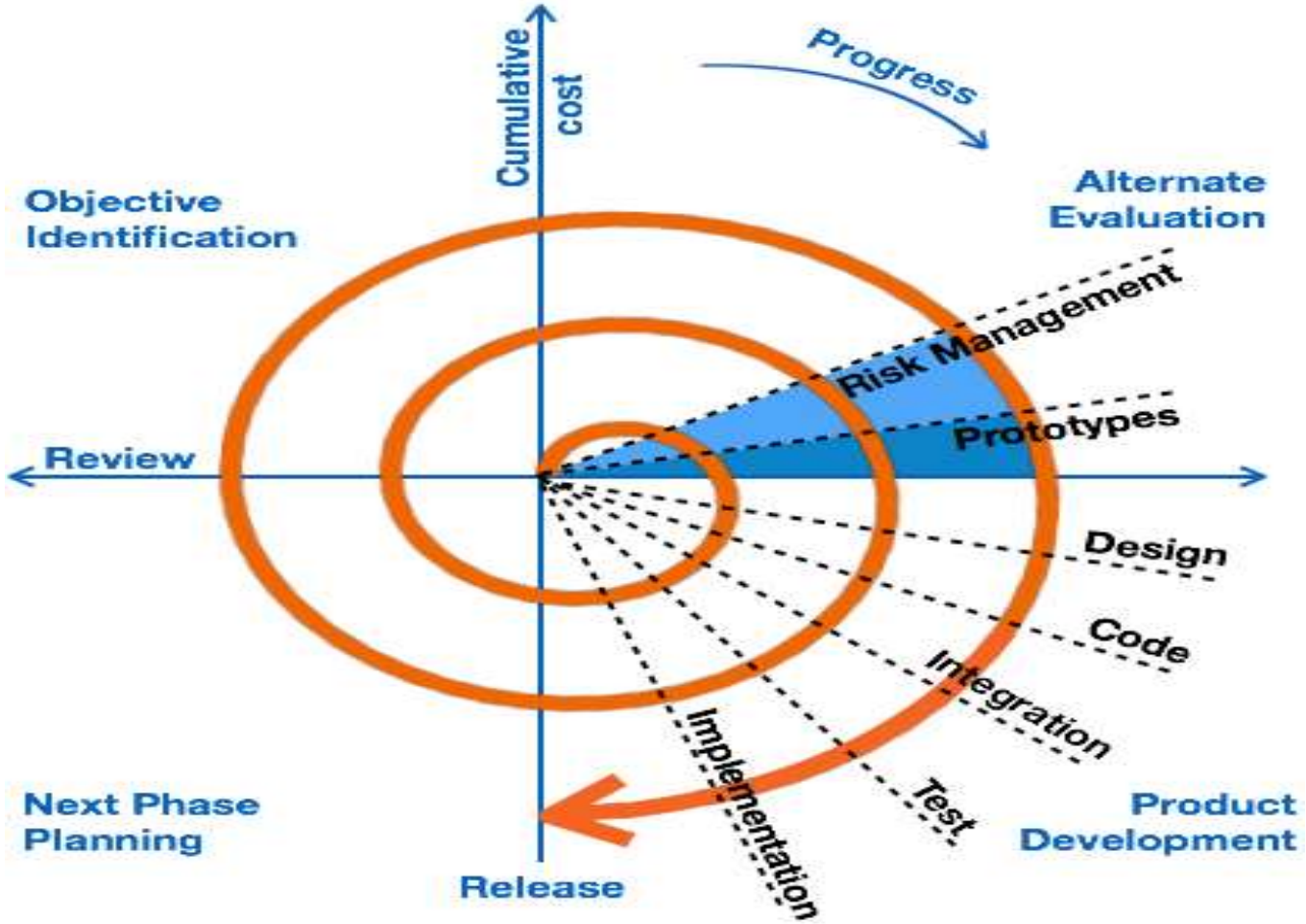
4. **System design with reuse:** During this stage the design of the system is build. Designer must consider the reused component and organize the framework. If reused component is not available then new software is develop.

5. **Development and Integration**: Components are integrated to develop new software. Integration in this model is part of development rather than separate activity.

1.9.4 Boehm's Spiral Model

- A risk-driven software process framework (the spiral model) was proposed by Boehm.
- Process is represented as a spiral rather than as a sequence of activities with some backtracking from one activity to another.
- Each loop in the spiral represents a phase in the process.
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required.
- Risks are explicitly assessed and resolved throughout the process.

Boehm's spiral model of the software process



Explanation of Spiral Model Activities

Activity	Description
Objective Identification	Requirements are gathered from the customers and the objectives are identified, elaborated, and analyzed. Project risks are identified. Alternative strategies, depending on these risks, may be planned.
Alternate Evaluation	In this quadrant, all the proposed solutions are analyzed and any potential risk is identified, analyzed, and resolved. The Prototype is built for the best possible solution.
Product development	The identified features are developed and verified through testing. At the end of the stage, the next version of the software is available.
Next Phase Planning	The project is reviewed and a decision made whether to continue with a further loop of the spiral. If it is decided to continue, plans are drawn up for the next phase of the project.

1.9.5 Agile methods

- Agile model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.
- Agile Methods break the product into small incremental builds. These builds are provided in iterations. At the end of each iteration a working product is displayed to the customer.
- In Agile development less time is spent on planning and more focus on the features need to be developed. There is feature driven development and the team adapts to the changing

Agile methods

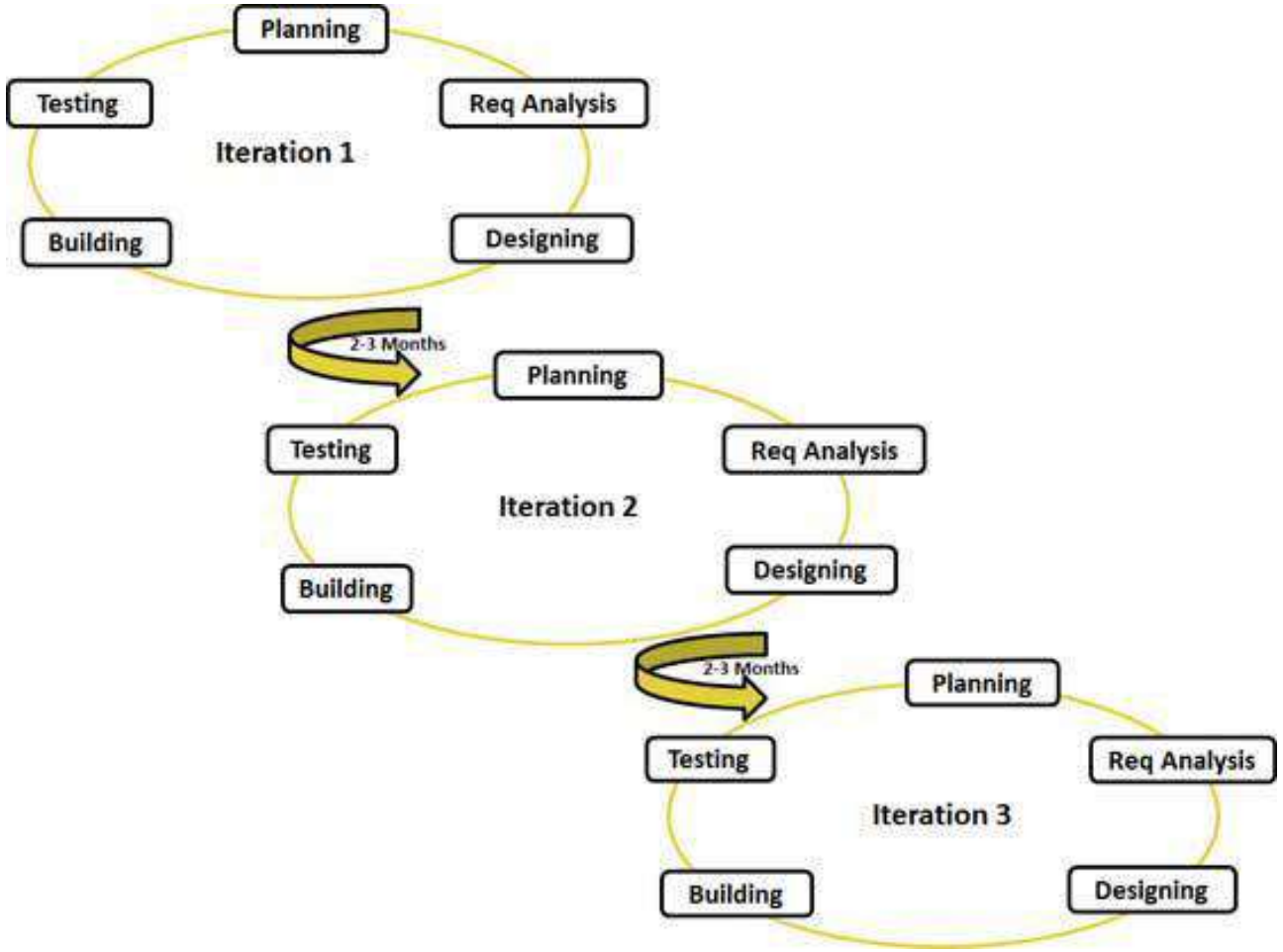


Figure 2.5: Graphical Illustration of the Agile Methods

1.9.6 Comparison of Different Software Process Models

Process model	Approach	Advantages	Disadvantages	When to Apply
Waterfall	Linear sequential	Clear documentation, easy to manage	Inflexible, limited feedback, high risk	Simple, small projects with clear requirements
Incremental	Linear Sequential with Iterative	Easier to test and debug, more flexible, simple to manage risk.	Can be time-consuming, expensive, requires a high level of collaboration	Projects with changing requirements
Reuse-oriented	Reuse-based	Reduce total cost, low risk factor, save lots of time and effort.	Requires well-defined components, may not fit all projects	Projects that necessitate a high level of flexibility, scalability, maintainability
Spiral	Iterative, risk-driven	Highlights risk management, flexible, adaptive	Time-consuming, can be expensive	Large, complex, high-risk projects
Agile	Iterative	Flexible, adaptive, continuous feedback	Requires active user involvement, can be chaotic	Complex projects with fluctuating requirements

Key points

- ✧ Software engineering is an engineering discipline that is concerned with all aspects of software production.
- ✧ Essential software product attributes are maintainability, dependability and security, efficiency and acceptability.
- ✧ The high-level activities of specification, development, validation and evolution are part of all software processes.
- ✧ There are many different types of system and each requires appropriate software engineering tools and techniques for their development.
- ✧ Software processes are the activities involved in producing a software system. Software process models are abstract representations of these processes.

Key points

- General process models describe the organization of software processes. Examples of these general models include the ‘waterfall’ model, incremental development, and reuse-oriented development.
- The spiral model is a systems development lifecycle (SDLC) method used for risk management that combines the iterative development process model with elements of the Waterfall model.
- Agile methods are incremental development methods that focus on rapid development, frequent releases of the software, reducing process overheads and producing high-quality code. They involve the customer directly in the development process.

Review Questions

1. Define: Software, Software Engineering, Software life cycle, Software process.
2. What are the 4P's of Software Engineering?
3. Differentiate between Generic product and Customized product.
4. What are the essential attributes of good software?
5. List down the challenges of software engineering practices, activities of Software process.
6. Explain in detail: Waterfall Model, Incremental development model, Reuse-oriented model, Boehm's spiral model, Agile method. What are the Advantages and Disadvantages?

Chapter 2 – Requirements Engineering

Topics covered

- ✧ Requirement Engineering
- ✧ Crucial Process Steps of Requirement Engineering
- ✧ Types of Requirements
- ✧ User and System Requirements
- ✧ Categories of Metrics
- ✧ Software Requirements Specification (SRS) Document
- ✧ Requirements Gathering Techniques

2.1 Requirements Engineering

- ✧ The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.
- ✧ The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.
- ✧ Requirements engineering is one of the most crucial activity in this creation process. Without well-written requirements specifications, developers do not know what to build, customers do not know what to expect, and there is no way to validate that the built system satisfies the requirements.

2.2 Crucial Process Steps of Requirement Engineering

✧ The requirements engineering consists of four steps

Process Step	Description
Requirements elicitation	This is also known as gathering of requirements. Here requirements are identified with the help of customer and existing systems process if available.
Requirements analysis	The requirements are analyzed in order to identify inconsistencies, defects, omissions, and also resolve conflicts if any.
Requirements documentations	It is the foundation for the design of the software. The document is known as software requirements specification (SRS).
Requirements review	The review process is carried out to improve the quality of the SRS.

✧ The primary output of requirements engineering is requirements specifications. If it describes both hardware and software, it is a system requirements specifications. If it describes only software, it is a software requirements specifications.

Crucial Process Steps of Requirement Engineering

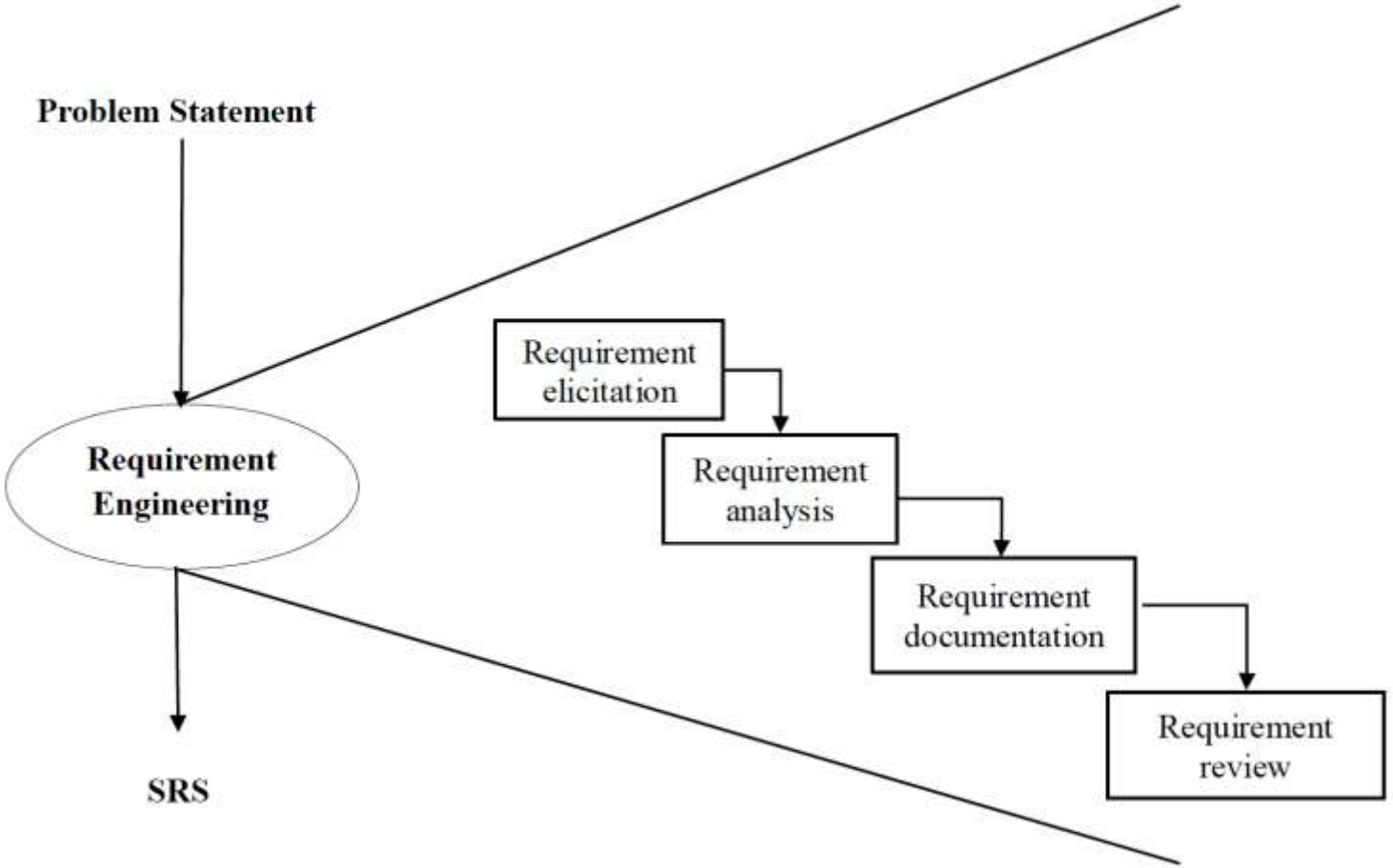


Figure 2.1: Crucial Process Steps of Requirement Engineering

2.3 Types of Requirements

- ✧ Functional requirements
  - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
  - May state what the system should not do.
- ✧ Non-functional requirements
  - Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
  - Often apply to the system as a whole rather than individual features or services.

Functional Requirements

- Describe functionality or system services.
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do.
- Functional system requirements should describe the system services in detail.

Non-Functional Requirements

- These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a particular IDE, programming language or development method.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.

Non-Functional Classifications

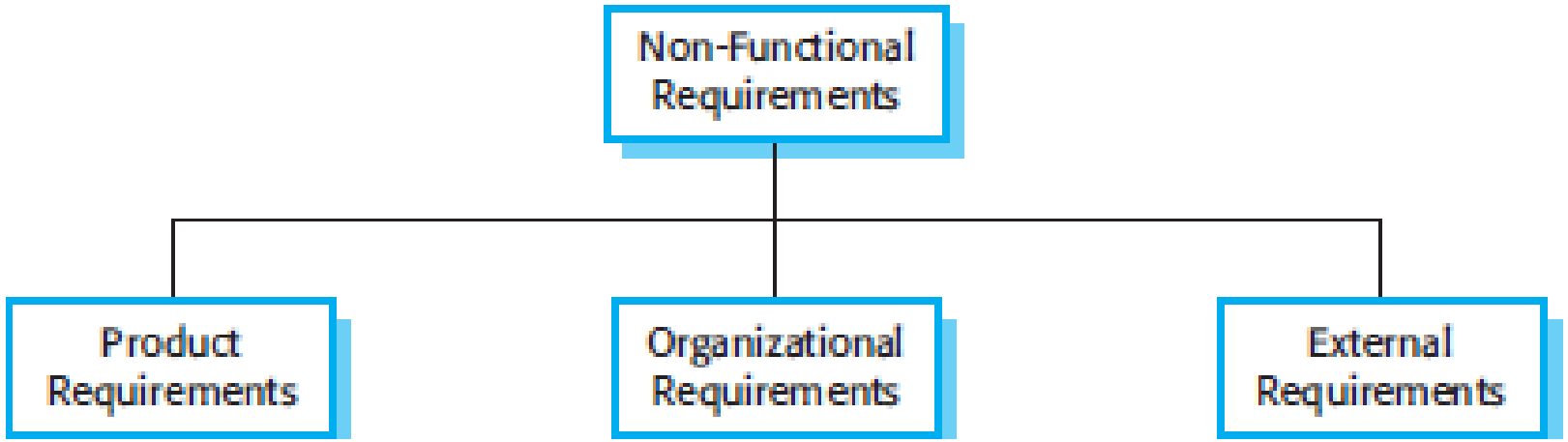


Figure 2.2: Types of Non-Functional Requirement

Non-Functional Classifications

- ✧ Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- ✧ Organisational requirements
  - Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
- ✧ External requirements
  - Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

2.4 User and System Requirements

- ***User requirement*** are written for the users and include functional and non-functional requirement. User requirements should specify the external behavior of the system with some constraints and quality parameters.
- ***System requirement*** are derived from user requirement. They are expanded form of user requirements.
  - ***A measure*** provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of the product or process".
  - ***Measurement*** is the act of determine a measure.
  - The ***metric*** is a quantitative measure of the degree to which a system, component, or process possesses a given attribute.

2.5 Categories of Metrics

- **Product metrics**: describe the characteristics of the product such as size, complexity, design features, performance, efficiency, reliability, portability, etc.

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Categories of Metrics

- **Process metrics**: describe the effectiveness and quality of the processes that produce the Software product. Examples are:
  - Effort required in the process
  - Time to produce the product
  - Effectiveness of defect removal during development
  - Number of defects found during testing
  - Maturity of the process
- **Project metrics:** describe the project characteristics and execution. Examples are:
  - Number of software developers
  - Staffing pattern over the life cycle of the software
  - Cost and schedule
  - Productivity

2.6 Software Requirements Specification (SRS) Document

- The SRS is a specification for a particular s/w product, program, or set of programs that performs certain functions in a specific environment.
- The SRS serve as contract document between customer and developer. SRS reduces the probability of the customer being disappointed with the final product.
- **Nature of the SRS**: The basic issues of that SRS writer(s) shall address the following:
    1. Functionality
    2. External interface
    3. Performance
    4. Attributes

Characteristics of a Good SRS

- The SRS should be:
  - Correct
  - Unambiguous
  - Complete
  - Consistent
  - Rank for importance and/ stability
  - Verifiable
  - Modifiable
  - Traceable

Organization of the SRS

- The (IEEE) has published guidelines and standards to organize an SRS document.
- Different projects may require their requirements to be organized differently but still first two

The template to organize and draft the SRS for any project.

2.7 Requirements Gathering Techniques

- Some requirements gathering techniques may be beneficial in one project but may not be in other.
- The usefulness of a technique is determined by its need and the kind of advantages.
- Following are some popular requirements gathering techniques:

Brainstorming

Description

Brainstorming is human nature to solve any problem as an early thought process. It can be utilized to gather a good number of ideas from a group of people by sharing ideas to identify all possible solutions.

Document Analysis

Description

Usually followed where a system is already in place, so evaluating the documentation of a present system can assist to gather requirements for updating or replacing existing system.

Focus Group

Description

A focus group is a gathering of people who are customers or user representatives for a product to gain its feedback. The feedback can be collected about opportunities, needs, and problems to determine requirements or it can be collected to refine and validate the already elicited requirements.

Interface Analysis

Description

Interface for any software product are either be human or machine. Integration with external devices and systems is another interface. The user-centric design approaches are quite effective to collect and develop usable software.

Interview

Description

Interviews of users and stakeholders are important in creating wonderful software. Without knowing the expectations and goal of the stakeholders and users it is impossible to satisfy them. To understand the perspective of every interviewee it is important to properly collect their inputs. Like a good reporter, listening is a quality that assists an excellent analyst to gain better value through an interview.

Observation

Description

The observation covers the study of users in their workplace. By watching users, a process flow, pain points, awkward steps and opportunities can be determined by an analyst for improvement. Observation can either be passive or active. Passive observation provides better feedback to refine requirements on the same hand active observation works best for obtaining an understanding over an existing business process.

Prototyping

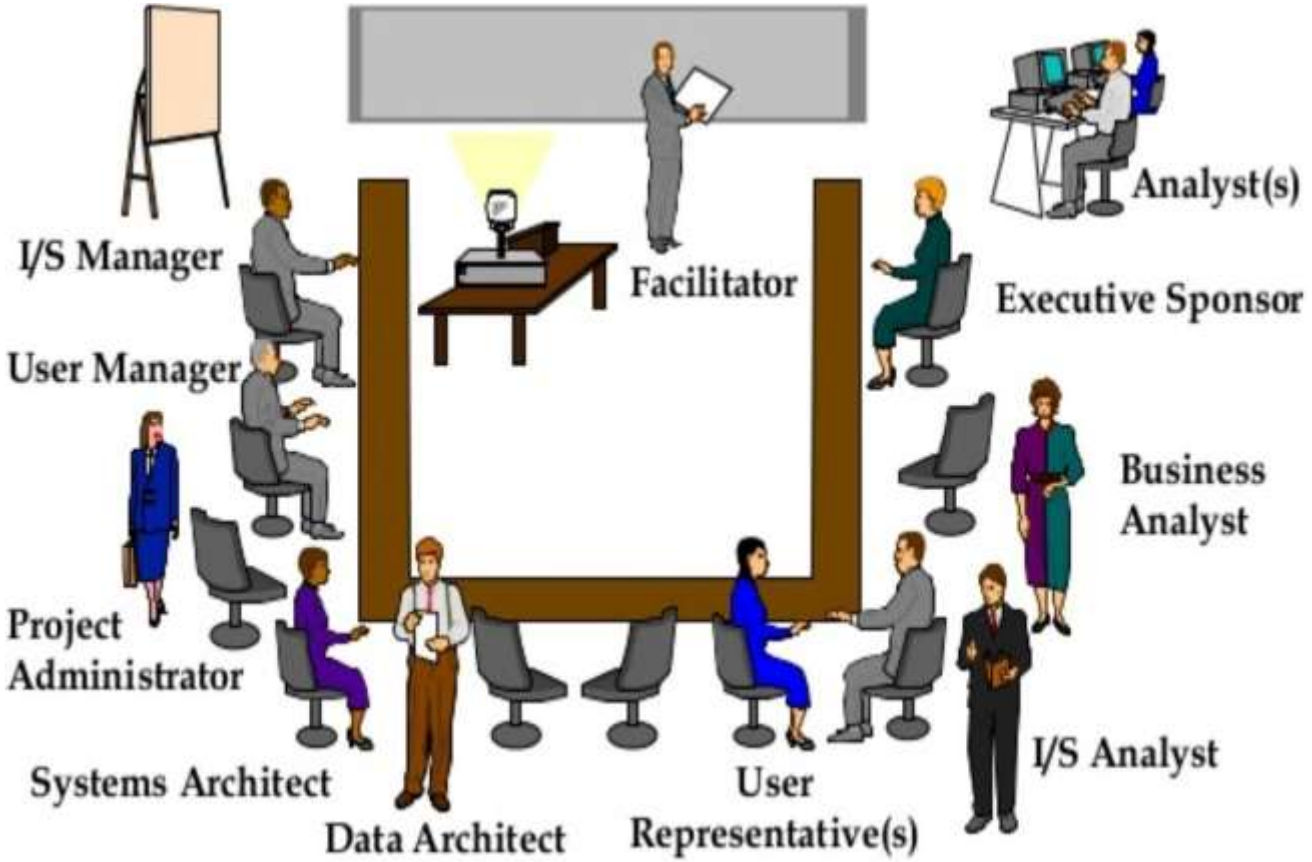
Description

Prototyping can be very helpful at gathering feedback. Low fidelity prototypes make a good listening tool. Many a times, people are not able to articulate a specific need in the abstract. They can swiftly review whether a design approach would satisfy their need. Prototypes are very effectively done with fast sketches of storyboards and interfaces.

Requirements Workshop

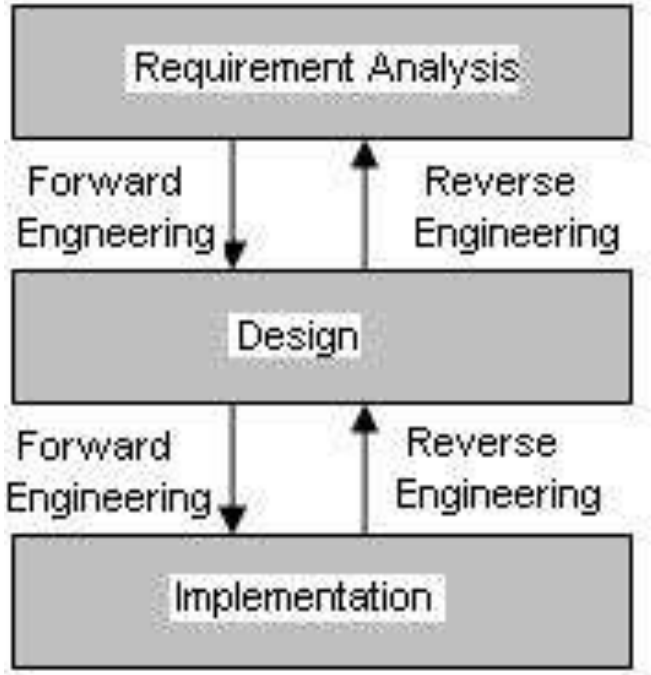
Description

Popularly known as JAD or joint application design, these workshops can be efficient for gathering requirements. The requirements workshops are more organized and structured than a brainstorming session where the involved parties get together to document requirements.



Reverse Engineering

Description
When a legacy project does not have enough documentation, reverse engineering can determine what system does? It do not determine what features went wrong with the system and what a system must do.



Survey

Description

When gathering information from many people: too many to interview with time constraints and less budget: a questionnaire survey can be used. The survey insists the users to choose from the given options agree / disagree or rate something.

Key points

- Requirements for a software system set out what the system should do and define constraints on its operation and implementation.
- Functional requirements are statements of the services that the system must provide or are descriptions of how some computations must be carried out.
- Non-functional requirements often constrain the system being developed and the development process being used.
- They often relate to the emergent properties of the system and therefore apply to the system as a whole.

Review Questions

1. Define: requirements engineering, measure, measurement, metrics, SRS
2. Illustrate crucial process steps of requirement engineering with neat diagram
3. Differentiate Functional and Non-functional Requirements
4. Explain different classifications of Non-functional requirements
5. What are the different categories of metrics?
6. What are the basic issues addressed by SRS writers
7. List down the characteristics of good SRS
8. Explain various requirement gathering techniques.

Chapter 3 – System Modeling and Architectural Design

Topics covered

- ✧ System Modeling
- ✧ UML Diagram Types
- ✧ Graphical Models
- ✧ Software Architecture
- ✧ Architectural Design
- ✧ Architectural Views
- ✧ Application Architectures
- ✧ Transaction Processing Systems

3.1 System Modeling

- System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system.
- System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML).
- System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.

3.2 UML Diagram Types

- Activity diagrams, which show the activities involved in a process or in data processing .
- Use case diagrams, which show the interactions between a system and its environment.
- Sequence diagrams, which show interactions between actors and the system and between system components.
- Class diagrams, which show the object classes in the system and the associations between these classes.
- State diagrams, which show how the system reacts to internal and external events.

3.3 Graphical Models

1. Context Model: in this model, context diagram and activity diagram is used.
2. Interaction Model: in this model, use-case diagram and sequence diagram is used.
3. Structural Model: in this model, class diagram and object diagram is used.

3.3.1 Context Models

- Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries.
- Social and organisational concerns may affect the decision on where to position system boundaries.
- Architectural models show the system and its relationship with other systems.

Context Diagram {Not UML Diagram} for Android Blood-Bank and its Boundaries

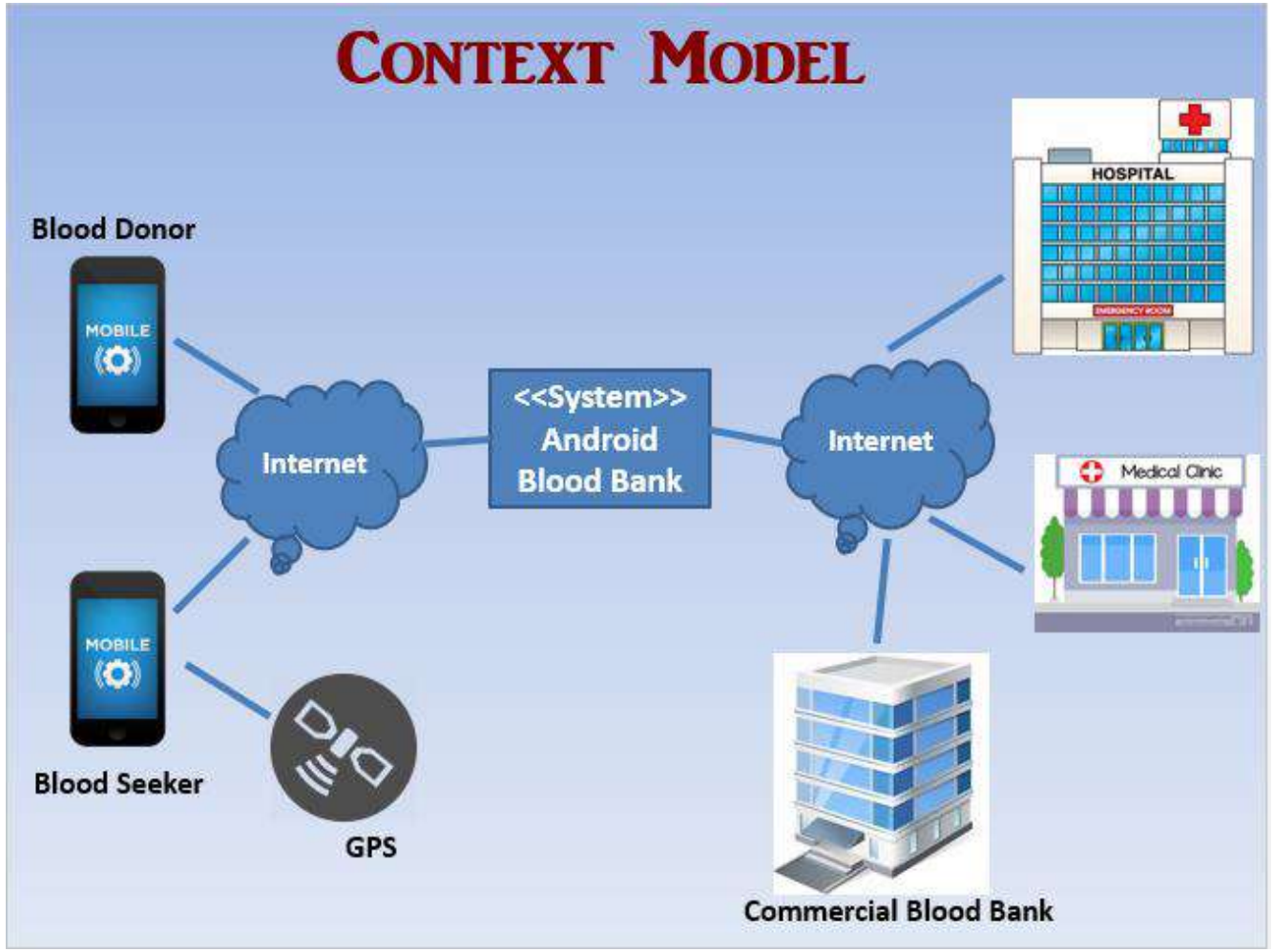


Figure 3.1: Context Diagram for Android Blood-Bank and its Boundaries

Context Models

- Activity diagrams are intended to show the activities that make up a system process and the flow of control from one activity to another.
- The start of a process is indicated by a filled circle; the end by a filled circle inside another circle.
- Rectangles with round corners represent activities, that is, the specific sub-processes that must be carried out.
- You may include objects in activity charts.
- UML activity diagrams are used to define major business process models.

UML Activity Diagram for Inventory System

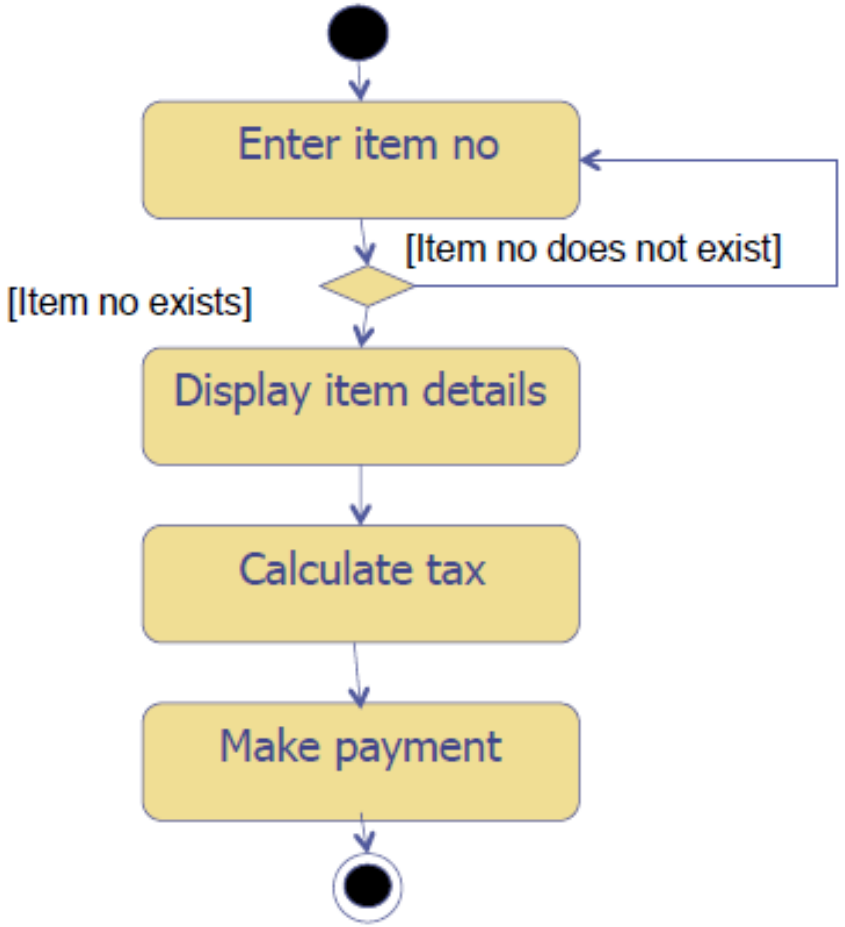


Figure 3.2: UML Activity Diagram for Inventory System

3.3.2 Interaction Models

- Modeling user interaction is important as it helps to identify user requirements.
- Modeling system-to-system interaction highlights the communication problems that may arise.
- Modeling component interaction helps us understand if a proposed system structure is likely to deliver the required system performance and dependability.
- Use case diagrams and sequence diagrams may be used for interaction modeling.

Use Case Modeling

- Use cases were developed originally to support requirements elicitation and now incorporated into the UML.
- Each use case represents a discrete task that involves external interaction with a system.
- Actors in a use case may be people or other systems.
- Use case diagrams give a fairly simple overview of an interaction so you have to provide more detail to understand what is involved. This detail can either be a simple textual description, a structured description in a table, or a sequence diagram.

Use-Case Diagram for ATM

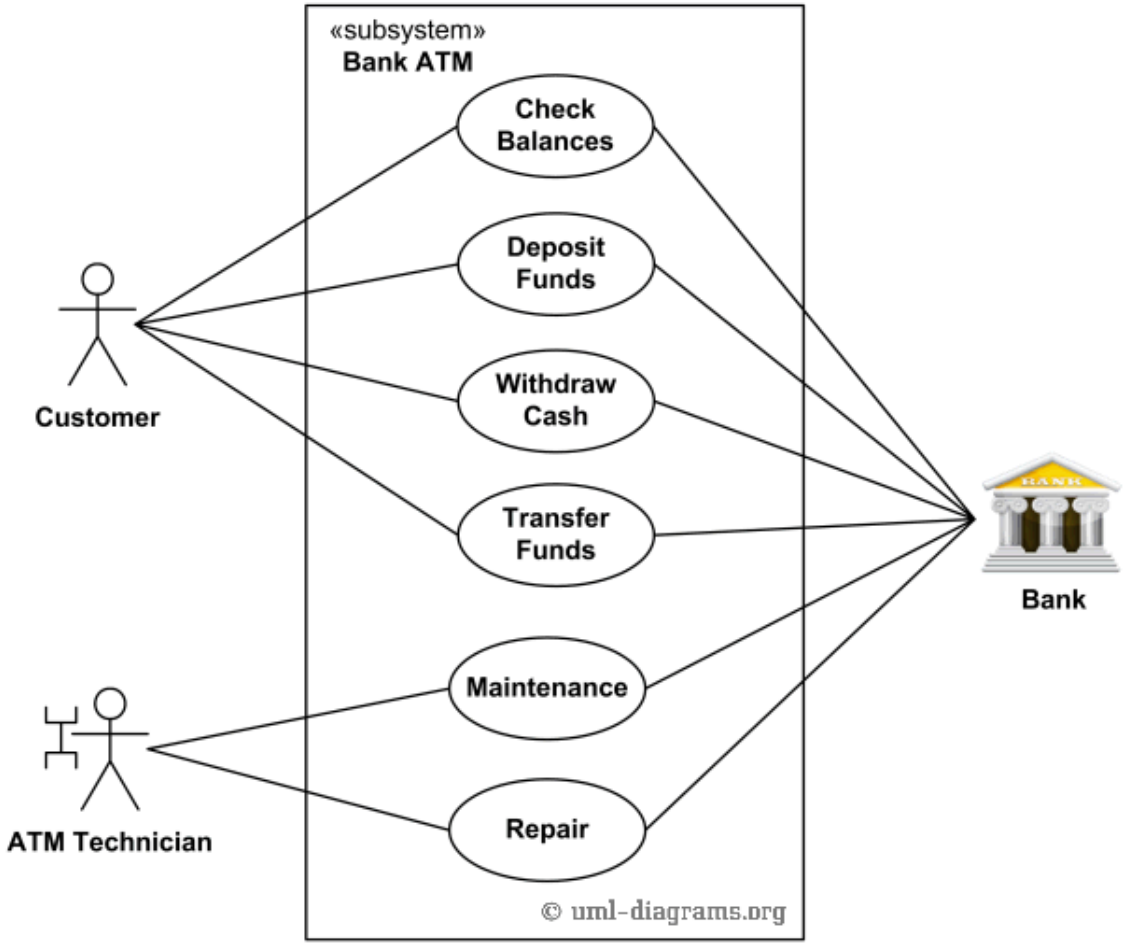


Figure 3.3: Use-Case Diagram for ATM

Sequence Diagrams

- Sequence diagrams are part of the UML and are used to model the interactions between the actors and the objects within a system.
- A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance.
- The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these.
- Interactions between objects are indicated by annotated arrows.

Sequence Diagram for ATM

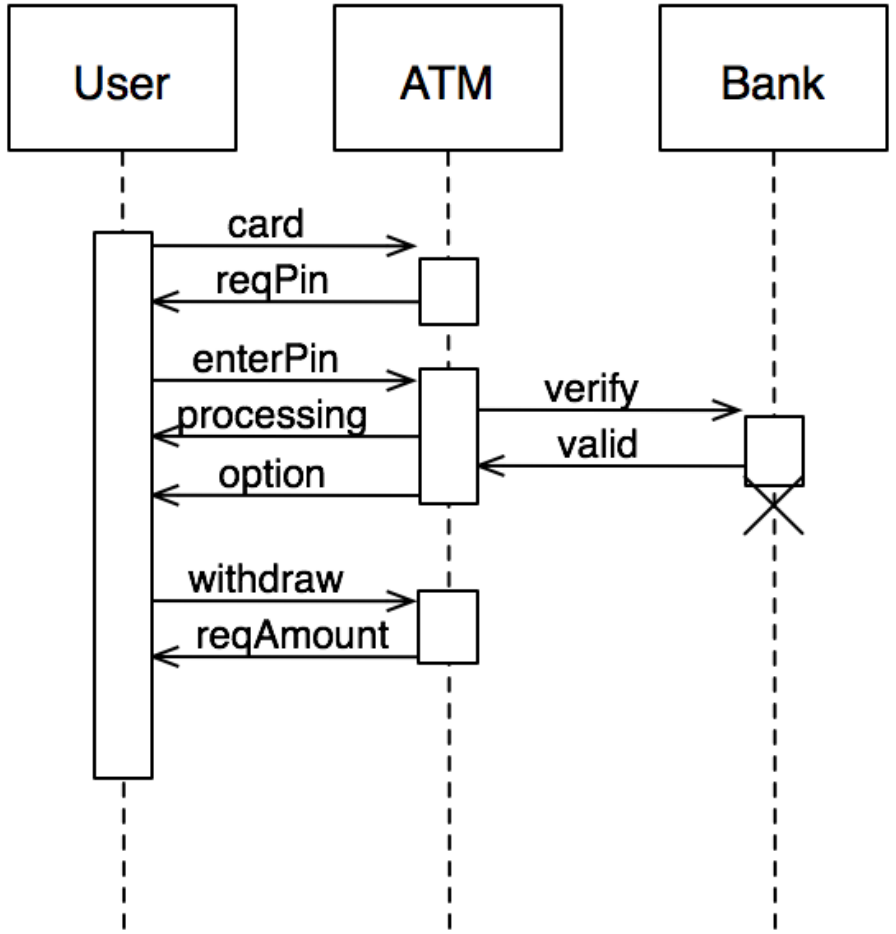


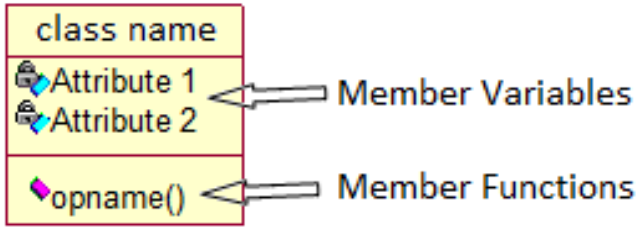
Figure 3.4: Sequence Diagram for ATM

3.3.3 Structural Models

- Structural models of software display the organization of a system in terms of the components that make up that system and their relationships.
- Structural models may be static models, which show the structure of the system design, or dynamic models, which show the organization of the system when it is executing.
- You create structural models of a system when you are discussing and designing the system architecture.

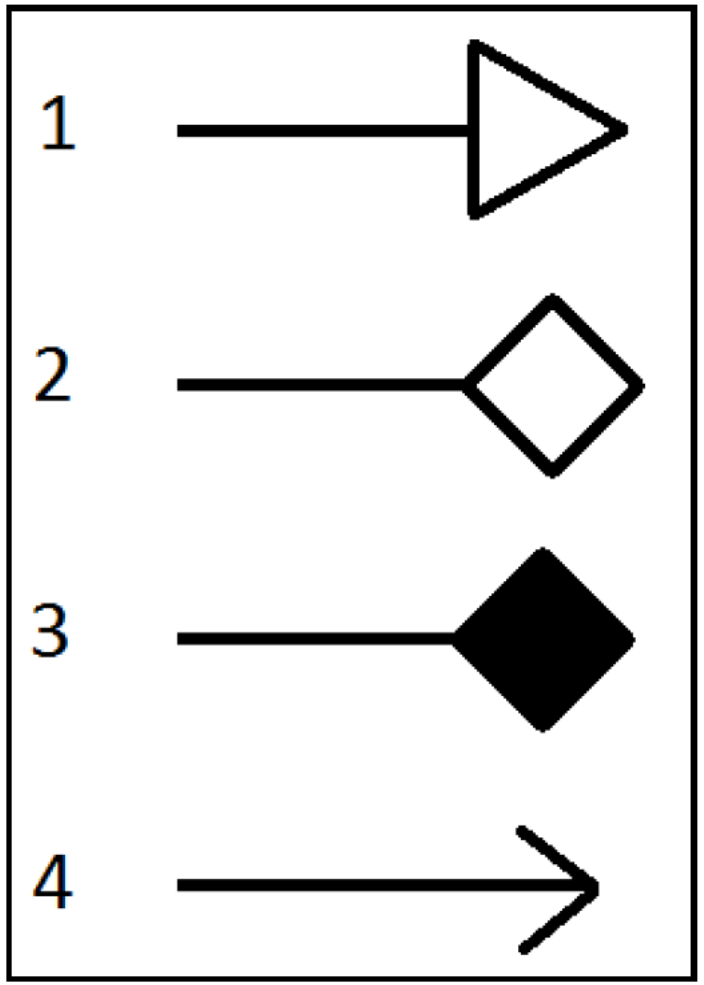
Class Diagrams

- Class diagrams are used when developing an object-oriented system model to show the classes in a system and the associations between these classes.
- Class diagram is an illustration of the relationships and source code dependencies among classes in the Unified Modeling Language (UML).
- **Class Diagram Notations:** UML class is represented by the diagram shown below that is divided into three sections.
  - First section is used to name the class.
  - Second section is used to show the attributes / member variable of the class.
  - Third section is used to describe the operations performed by the class.




```
Class Students {
private:
  int ID;
  char * Name;
public:
  void Registration();
};
```

Types of Relationship Between the Classes



1. Shows **Generalization** between two classes to maintain parent-child relationship. (***is-a relation***)
2. Shows **Aggregation** between two classes to maintain containership. (***has-a relation***)
3. Shows **Composition** between two classes to maintain containership. (***has-a relation***)
4. Shows **Association** between two classes to maintain a simple connectivity.

Types of Relationship Between the Classes

- **Difference between Aggregation and Composition: (Has-a Relationship)**

- In **aggregation** contained class exist even if main class is deleted. Eg: **Car has Engine**, If car is broken still engine exist and can be used in another car.

- In **composition** contained class live or die on the basis of main class. Eg: **Folder has file**, If you delete a folder all files will be deleted as well.

- **Generalization: (Is-a Relationship)**
  - Generalization is used in class diagrams to deal with most powerful concept of object orientation that is **Inheritance**. Generalization is drawn between two classes to show Parent-Child relation.
  - In modeling systems, it is often useful to examine the classes in a system to see if there is scope for generalization. If changes are proposed, then you do not have to look at all classes in the system to see if they are affected by the change.
  - The lower-level classes are subclasses that inherit the attributes and operations from their superclass. These lower-level classes then add more specific attributes and operations.

Class Diagram Showing Different Type of Relationships

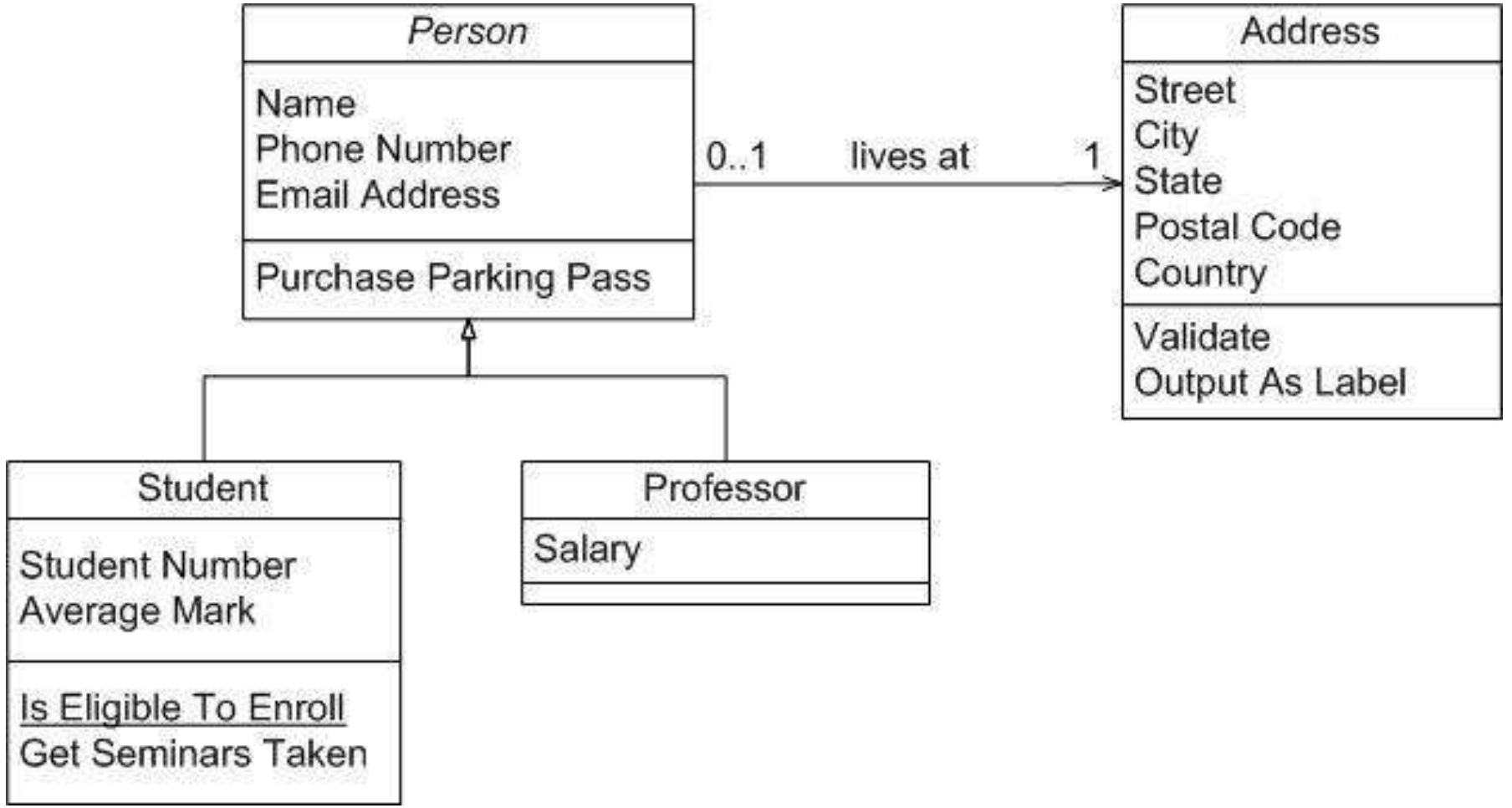


Figure3.5: Class Diagram Showing Different Type of Relationships

3.4 Architectural Design

- Architectural design is concerned with understanding how a software system should be organized and designing the overall structure of that system
- Architectural design is the first stage in the software design process.
- It is the critical link between design and requirements engineering, as it identifies the main structural components in a system and the relationships between them.
- .The output of the architectural design process is an architectural model that describes how the system is organized as a set of communicating components.

3.4.1 Use of Architectural Models

- ✧ As a way of facilitating discussion about the system design
  - A high-level architectural view of a system is useful for communication with system stakeholders and project planning because it is not cluttered with detail. Stakeholders can relate to it and understand an abstract view of the system. They can then discuss the system as a whole without being confused by detail.
- ✧ As a way of documenting an architecture that has been designed
  - The aim here is to produce a complete system model that shows the different components in a system, their interfaces and their connections.

3.5 Architectural Views

- ✧ Each architectural model only shows one view or perspective of the system.
  - It might show how a system is decomposed into modules, how the run-time processes interact or the different ways in which system components are distributed across a network. For both design and documentation, you usually need to present multiple views of the software architecture.
- ✧ Krutchen in his well-known 4 +1 view model of software architecture, suggests that there should be four fundamental architectural views, which can be linked through common use cases or scenarios (shown in Fig: 3.6 – Next slide)

4 + 1 View Model of Software Architecture

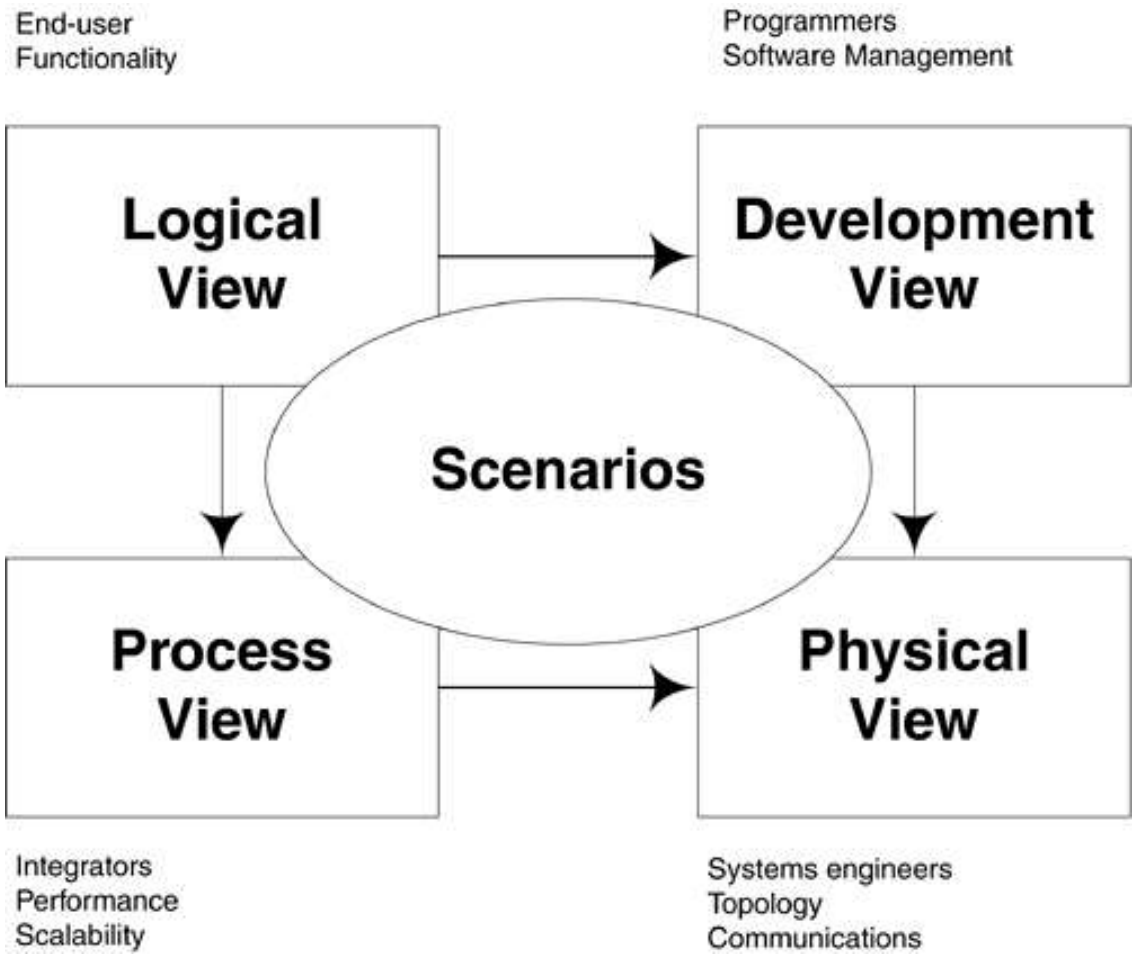


Figure 3.6: 4+1 View Model

4 + 1 View Model of Software Architecture

- **A logical view**, which shows the key abstractions in the system as objects or object classes.
- **A process view**, which shows how, at run-time, the system is composed of interacting processes.
- **A development view**, which shows how the software is decomposed for development.
- **A physical view**, which shows the system hardware and how software components are distributed across the processors in the system.
- Related using use cases or scenarios (+1)

3.6 Application Architectures

- Application systems are designed to meet an organizational need.
- As businesses have much in common, their application systems also tend to have a common architecture that reflects the application requirements.

Use of Application Architectures

- As a starting point for architectural design.
- As a design checklist.
- As a way of organizing the work of the development team.
- As a means of assessing components for reuse.
- As a vocabulary for talking about application types.

3.7 Transaction Processing Systems (TPS)

- TPS is a type of information system that collects, stores, modifies and retrieves the data transactions of an organization.

  For example - airline reservation systems, electronic transfer of funds, bank account processing systems.

- From a user perspective a transaction is:

  Any coherent sequence of operations that satisfies a goal;

  For example - find the times of flights from London to Paris.

- Users make asynchronous requests for service which are then processed by a transaction manager.

3.7.1 Application Architecture of TPS

- Transaction processing systems are usually interactive systems in which users make asynchronous requests for service.
  - Asynchronous means that you do not halt all other operations while waiting for the web service call to return.
- Figure 5.2 illustrates the conceptual architectural structure of TPS.
  - First a user makes a request to the system through an I/O processing component. The request is processed by some application specific logic. A transaction is created and passed to a transaction manager, which is usually embedded in the database management system. After the transaction manager has ensured that the transaction is properly completed, it signals to the application that processing has finished.

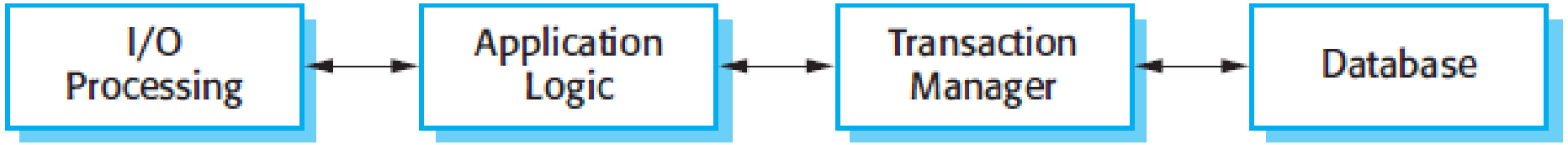


Figure 3.7: Architecture of TPS

3.7.2 The Software Architecture of an ATM System

- An example of a transaction is a customer request to withdraw money from a bank account using an ATM, Figure 5.3. This involves getting details of the customer's account, checking the balance, modifying the balance by the amount withdrawn, and sending commands to the ATM to deliver the cash. Until all of these steps have been completed, the transaction is incomplete and the customer accounts database is not changed.

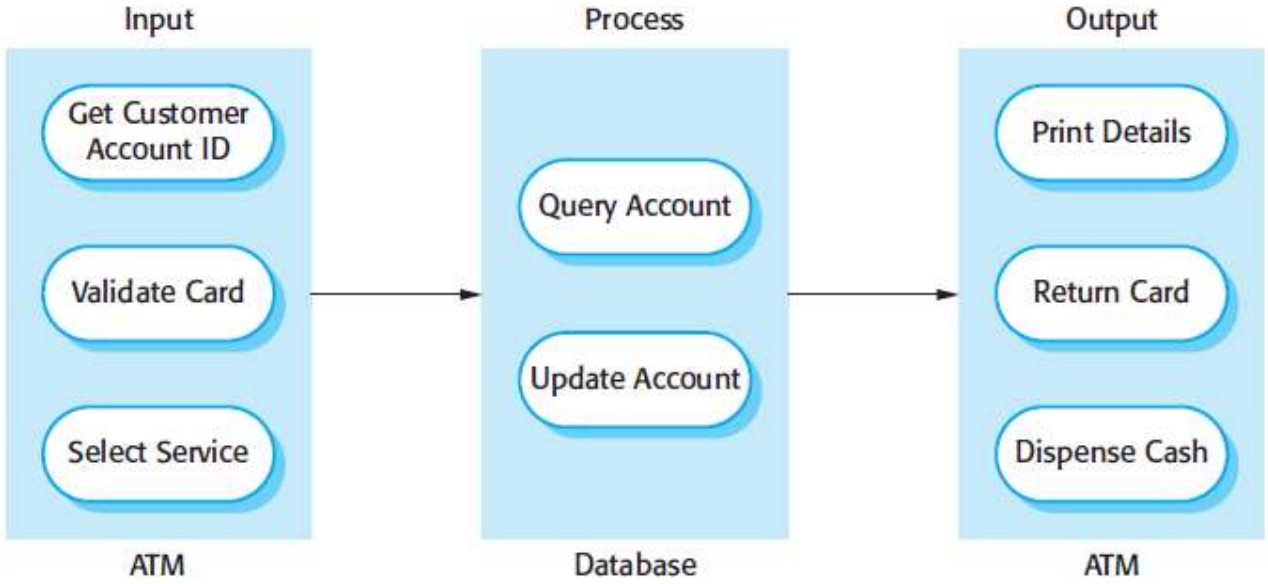


Figure 3.8: Architecture of ATM

Key Points

- A model is an abstract view of a system that ignores system details. Complementary system models can be developed to show the system's context, interactions, structure and behavior.
- Context models show how a system that is being modeled is positioned in an environment with other systems and processes.
- Use case diagrams and sequence diagrams are used to describe the interactions between users and systems in the system being designed. Use cases describe interactions between a system and external actors; sequence diagrams add more information to these by showing interactions between system objects.
- Structural models show the organization and architecture of a system. Class diagrams are used to define the static structure of classes in a system and their associations.

Key Points

- A software architecture is a description of how a software system is organized.
- Architectural design decisions include decisions on the type of application, the distribution of the system, the architectural styles to be used.
- Architectures may be documented from several different perspectives or views such as a conceptual view, a logical view, a process view, and a development view.
- TPS is a type of information system that collects, stores, modifies and retrieves the data transactions of an organization.

Review questions

1. Define: system modeling, Activity diagram, Use case diagram, sequence diagram, class diagram, state diagram
2. Explain various graphical models
3. Draw Use-case and Sequence diagram for ATM
4. Difference between Aggregation and Composition.
5. What is Architectural design? What are the uses of architectural model?
6. Illustrate 4+1 view model of software architecture with neat diagram
7. List down the uses of application architecture.
8. What is TPS? Explain application architecture of TPS with neat diagram.

Chapter 4 – Design and Implementation

Topics Covered

- ✧ Design and Implementation
- ✧ An Object-Oriented Design Process
- ✧ Context and Interaction Models
- ✧ Implementation Issues
- ✧ Open Source Development

4.1 Design and Implementation

- Software design and implementation is the stage in the software engineering process at which an executable software system is developed.
- Software design and implementation activities are invariably inter-leaved.
  - Software design is a creative activity in which you identify software components and their relationships, based on a customer's requirements.
  - Implementation is the process of realizing the design as a program.

4.2 An Object-Oriented Design Process

- Structured object-oriented design processes involve developing a number of different system models.
- They require a lot of effort for development and maintenance of these models and, for small systems, this may not be cost-effective.
- However, for large systems developed by different groups design models are an important communication mechanism.

Process Stages

- There are a variety of different object-oriented design processes that depend on the organization using the process.
- Common activities in these processes include:
  - Define the context and modes of use of the system;
  - Design the system architecture;
  - Identify the principal system objects;
  - Develop design models;
  - Specify object interfaces.

4.3 Context and Interaction Models

- A system context model is a structural model that demonstrates the other systems in the environment of the system being developed.
- An interaction model is a dynamic model that shows how the system interacts with its environment as it is used.

4.4 Implementation Issues

- Software engineering includes all of the activities involved in software development from the initial requirements of the system through to maintenance and management of the deployed system. A critical stage of this process is, of course, system implementation, where you create an executable version of the software.
- Implementation may involve developing programs in high- or low-level programming languages or tailoring and adapting generic, off-the-shelf systems to meet the specific requirements of an organization.

Implementation Issues

- Focus here is not on programming, although this is obviously important, but on other implementation issues that are often not covered in programming texts:
  - 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.
  - Configuration management During the development process, you have to keep track of the many different versions of each software component in a configuration management system.
  - 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).

4.4.1 Reuse

- From the 1960s to the 1990s, most new software was developed from scratch, by writing all code in a high-level programming language.
  - The only significant reuse or software was the reuse of functions and objects in programming language libraries.
- Costs and schedule pressure mean that this approach became increasingly unviable, especially for commercial and Internet-based systems.
- An approach to development based around the reuse of existing software emerged and is now generally used for business and scientific software.

Reuse Levels

- ✧ The abstraction level
  - At this level, you don’t reuse software directly but use knowledge of successful abstractions in the design of your software.
- ✧ The object level
  - At this level, you directly reuse objects from a library rather than writing the code yourself.
- ✧ The component level
  - Components are collections of objects and object classes that you reuse in application systems.
- ✧ The system level
  - At this level, you reuse entire application systems.

4.4.2 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, and compile and link components to create a system.

Configuration Management Activities

- **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.
- **System integration**, where support is provided to help developers define what versions of components are used to create each version of a system. This description is then used to build a system automatically by compiling and linking the required components.
- **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.

4.4.3 Host-Target Development

- Most software is developed on one computer (the host), but runs on a separate machine (the target).
- More generally, we can talk about a development platform and an execution platform.
  - A platform is more than just hardware.
  - It includes the installed operating system plus other supporting software such as a database management system or, for development platforms, an interactive development environment.
- Development platform usually has different installed software than execution platform; these platforms may have different architectures.

4.5 Open Source Development

- Open source development is an approach to software development in which the source code of a software system is published and volunteers are invited to participate in the development process
- Its roots are in the Free Software Foundation (www.fsf.org), which advocates that source code should not be proprietary but rather should always be available for users to examine and modify as they wish.
- Open source software extended this idea by using the Internet to recruit a much larger population of volunteer developers. Many of them are also users of the code.

4.5.1 Open Source Systems

- The best-known open source product is, of course, the Linux operating system which is widely used as a server system and, increasingly, as a desktop environment.
- Other important open source products are Java, the Apache web server and the mySQL database management system.

Apache HTTP Server (web server)

Blender (3D graphics and animation package)

GNOME (Linux desktop environment)

GNU Compiler Collection (GCC, a suite of compilation tools for C, C++, etc)

Moodle (virtual learning system)

Firefox (web browser based on Mozilla)

MySQL (database)

OpenOffice.org (office suite, including word processor, spreadsheet, and presentation software)

Perl (programming/scripting language)

Python (programming/scripting language)

4.5.2 Open Source Business

- More and more product companies are using an open source approach to development.
- Their business model is not reliant on selling a software product but on selling support for that product.
- They believe that involving the open source community will allow software to be developed more cheaply, more quickly and will create a community of users for the software.

4.5.3 Open Source Licensing

- A fundamental principle of open-source development is that source code should be freely available, this does not mean that anyone can do as they wish with that code.
  - Legally, the developer of the code (either a company or an individual) still owns the code. They can place restrictions on how it is used by including legally binding conditions in an open source software license.
  - Some open source developers believe that if an open source component is used to develop a new system, then that system should also be open source.
  - Others are willing to allow their code to be used without this restriction. The developed systems may be proprietary and sold as closed source systems.

Key Points

- Software design and implementation are inter-leaved activities. The level of detail in the design depends on the type of system and whether you are using a plan-driven or agile approach.
- The process of object-oriented design includes activities to design the system architecture, identify objects in the system, describe the design using different object models and document the component interfaces.
- A range of different models may be produced during an object-oriented design process. These include static models (class models, generalization models, association models) and dynamic models (sequence models, state machine models).
- Component interfaces must be defined precisely so that other objects can use them. A UML interface stereotype may be used to define interfaces.

Key Points

- When developing software, you should always consider the possibility of reusing existing software, either as components, services or complete systems.
- Configuration management is the process of managing changes to an evolving software system. It is essential when a team of people are cooperating to develop software.
- Most software development is host-target development. You use an IDE on a host machine to develop the software, which is transferred to a target machine for execution.
- Open source development involves making the source code of a system publicly available. This means that many people can propose changes and improvements to the software.

Review questions

1. Write short notes on Object-Oriented Design Process. List its common activities.
2. Define: system context model and interaction
3. List and explain various implementation issues.
4. What are the different levels of software reuse
5. What is meant by configuration management? List down its activities.
6. Write short notes on Host-target development.
7. Write briefly about open source development system.

Chapter 5 – Software Testing and Maintenance

Topics Covered

- ✧ Software verification and validation
- ✧ Software Testing
- ✧ Stages of Testing
- ✧ Black Box Testing (BBT)
- ✧ White Box Testing (WBT)
- ✧ Software Maintenance and its types
- ✧ The Software Maintenance Process
- ✧ Maintenance Costs and Prediction
- ✧ Software RE-Engineering

5.1 Software Verification and Validation

✧ Verification and validation are the processes in which we check a product against its specifications and the expectations of the users who will be using it.

✧ Verification:

"Are we building the product right”

“Does the product meet system specifications?”

✧ Validation:

"Are we building the right product”

“Does the product meet user expectations?”

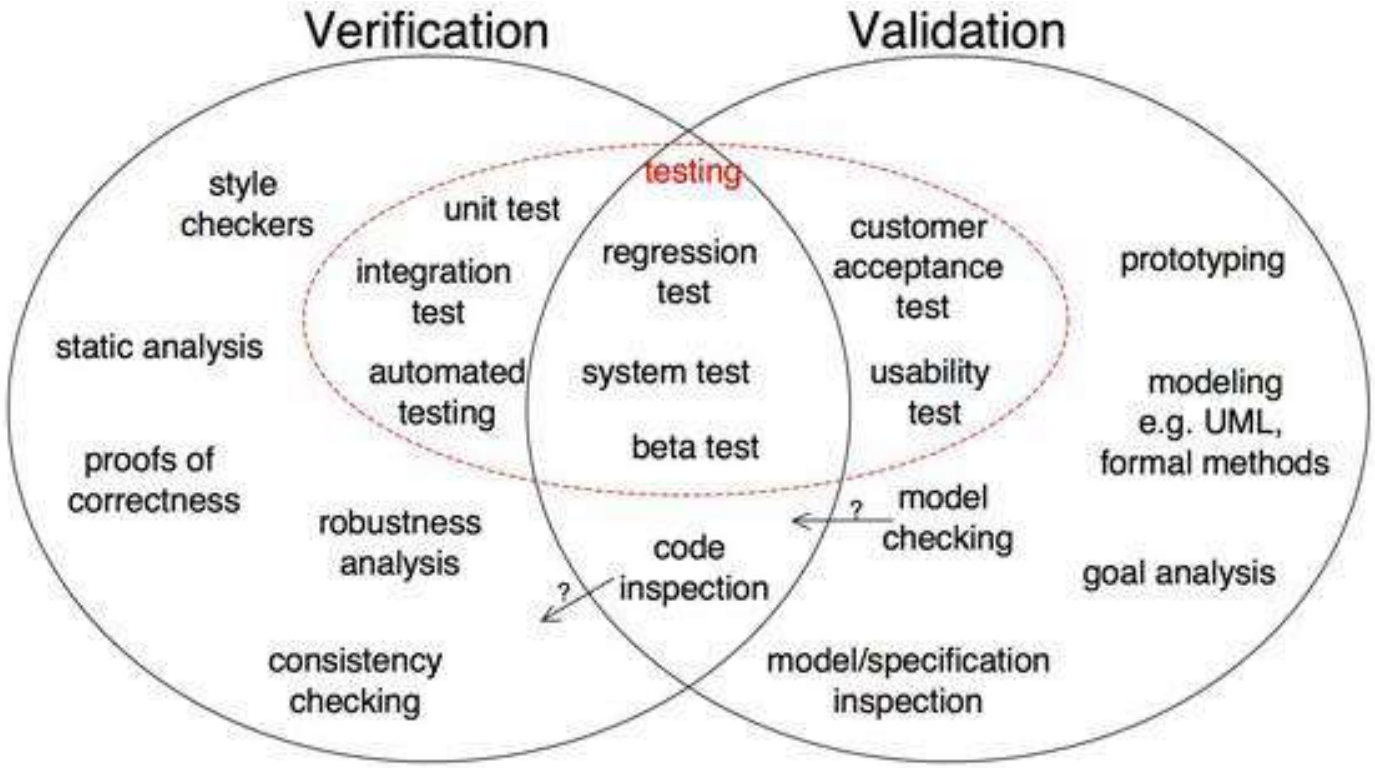


Figure 7.1: Scope of V & V in software engineering

5.2 Software Testing

- Testing is the process of executing a program with the intent of finding errors.

5.2.1 Why should we Test?

“Testing can reveal the presence of errors NOT their absence”

- ✧ Software testing is an expensive activity but launching of software without testing may lead to cost much higher than the cost of testing itself.
- ✧ Testing assure to developers and customers that the software meets its requirements.
- ✧ It discovers the situations where behavior of the software is incorrect, undesirable or does not match to its specification.
- ✧ Testing identifies undesirable system behavior such as system crashes, unwanted interactions with other systems, incorrect computations and data corruption.

5.2.2 Who should do the Testing?

- Testing requires the developers to find errors from their software.
- It is difficult for software developer to point out errors from own creations.

5.2.3 What should we Test?

- Many organizations have made a distinction between development and testing phase by making different people responsible for each phase.
  - ✧ We should test the program's responses to every possible input. It means, we should test for all valid and invalid inputs.
  - ✧ A strategy should develop test cases for the testing of small portion of program and also test cases for complete system or function.

5.3 Stages of Testing

- **Development testing**, where the system is tested during development to discover bugs and defects.
- **Release testing**, where a separate testing team test a complete version of the system before it is released to users.
- **User testing**, where users or potential users of a system test the system in their own environment.

5.3.1 Development Testing

- Development testing includes all testing activities that are carried out by the team developing the system.
  - **Unit testing**, where individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods.
  - **Component testing**, where several individual units are integrated to create composite components. Component testing should focus on testing component interfaces.
  - **System testing**, where some or all of the components in a system are integrated and the system is tested as a whole. System testing should focus on testing component interactions.
- One of the popular development testing techniques is White Box Testing (WBT) that can be applied at the unit, integration and system levels of the testing process. (WBT is discussed separately)

5.3.2 Release Testing

- Release testing is the process of testing a particular release of a system that is intended for use outside of the development team.
- The primary goal of the release testing process is to convince the supplier of the system that it is good enough for use.
- Release testing shows that the system delivers its specified functionality, performance and dependability, and that it does not fail during normal use.
- Release testing is usually a Black Box Testing (BBT) process where tests are only derived from the system specification. (BBT is discussed separately)

5.3.3 User Testing

✧ User or customer testing is a stage in the testing process in which users or customers provide input and advice on system testing.

Types of User Testing

✧ Alpha testing

- Users of the software work with the development team to test the software at the developer's site.

✧ Beta testing

- A release of the software is made available to users to allow them to experiment and to raise problems that they discover with the system developers.

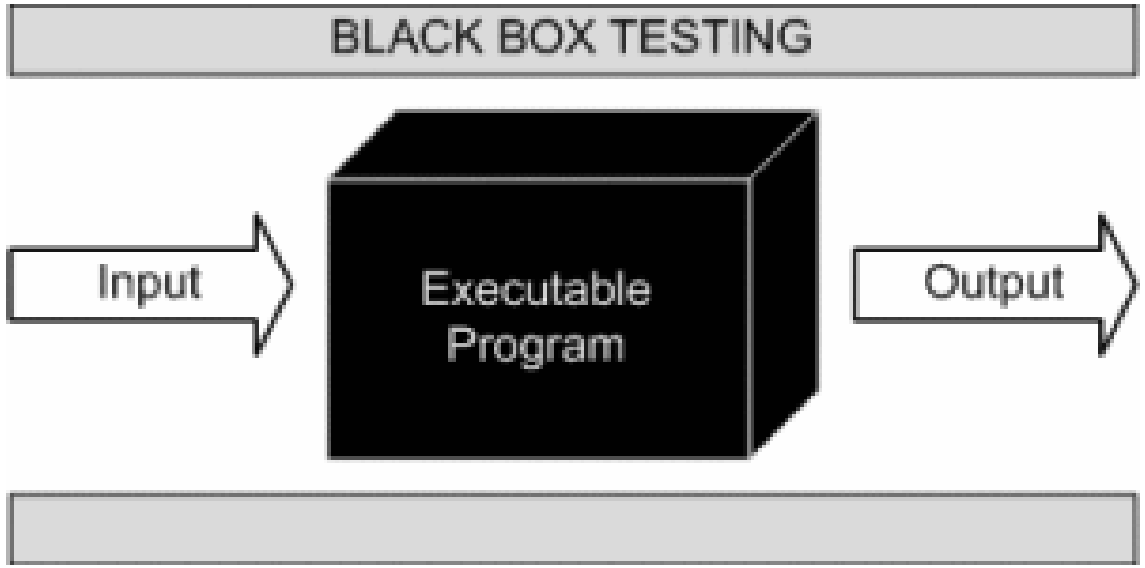
✧ Acceptance testing

- Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment. Primarily for custom (be-spoke) systems.

5.4 Black Box Testing (BBT)

- ✧ Black-box testing, also called behavioral testing, focuses on the functional requirements of the software.
- ✧ BBT enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program.
- ✧ As shown in Figure7.2 (next slide) BBT is concerned only with the possible inputs and their desired output regardless of the developmental details.
- ✧ BBT is performed by a separate testing team not the developer their self hence it is one type of release testing.

Black Box Testing (BBT)



There are further many types of BBT but two of the most commonly used types are as follows:

- Equivalence Partitioning
- Boundary Value Analysis

Equivalence Partitioning

- Equivalence partitioning is a black-box testing method that divides the input domain of a program into classes of data from which test cases can be derived.
- **Examples**: If a code of calculator has to test using BBT then possible partitioning of input test data are:

Test Cases	Possible Valid Results	Possible Invalid Results
Test case 1: 2 + 2 = ?	2 + 2 = 4	2 + 2 = 0
Test case 2: 2.52 + 4.36 = ?	2.52 + 4.36 = 6.88	2.52 + 4.36 = 6
Test case 3: 'a' + 345 = ?	'a' + 345 = error	'a' + 345 = 345
Test case 4: 23 – 45 = ?	23 – 45 = -22	23 – 45 = 22
Test case 5: 43 x 2.4 = ?	43 x 2.4 = 103.2	43 x 2.4 = 86
Test case 6: 10 / 0 = ?	10 / 0 = error	10 / 0 = 0
Test case 7: 45 mod 4 = ?	45 mod 4 = 1	45 mod 4 = 11

Boundary Value Analysis

- Boundary value analysis enhances the performance of equivalence partitioning because it leads to the selection of test cases at the "edges" of the class.
- **Examples**: In boundary value analysis testing test cases are generated as shown in Figure 7.3

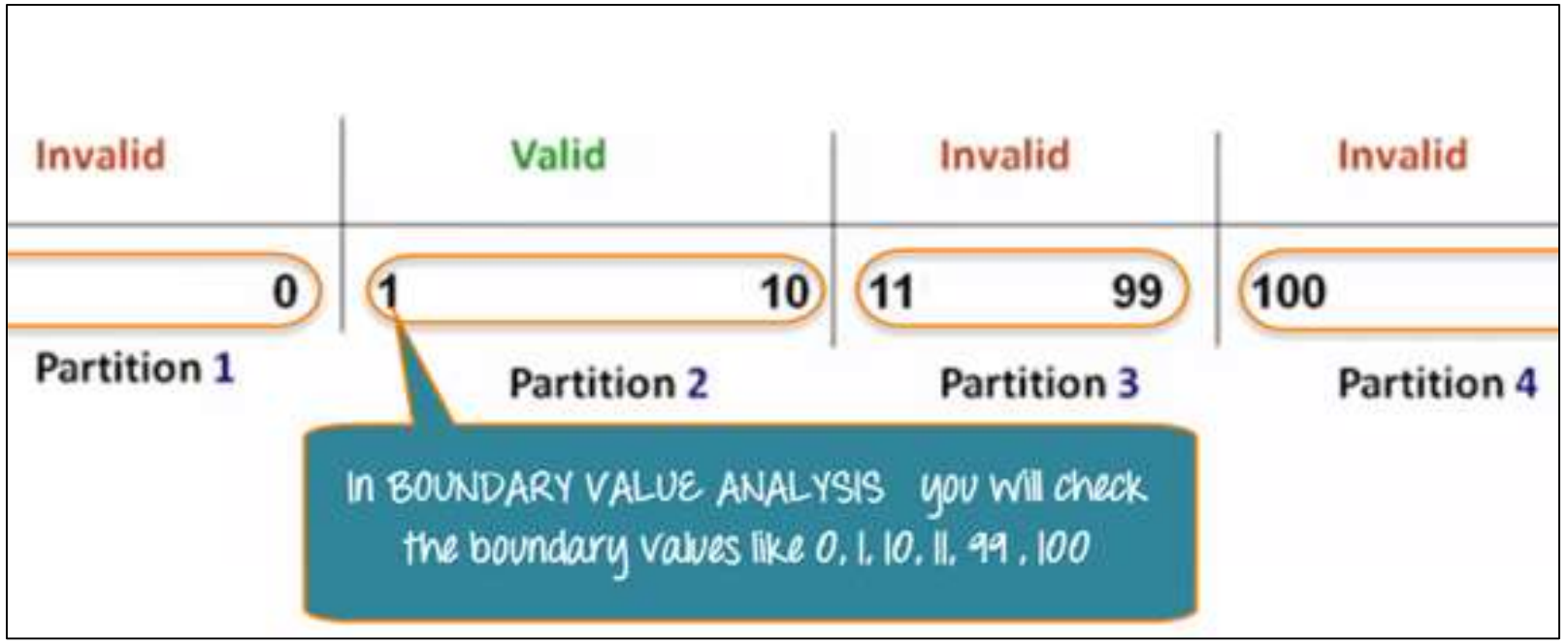


Figure 7.3: Boundary Value Analysis ranges

5.5 White Box Testing (WBT)

- ✧ White-box testing also called glass-box testing. In WBT test cases are designed using the control structure of the procedural design. Using white-box testing, software engineers can derive test cases that:
- ➢ Guarantee that all independent paths within a module have been exercised at least once.
- ➢ Exercise all logical decisions on their true and false sides
- ➢ Execute all loops at their boundaries and within their operational bounds
- ➢ Exercise internal data structures to ensure their validity.

White Box Testing (WBT)

- ✧ WBT can be applied at any level of development testing.
- ✧ In WBT at first code is converted into descriptive code by assigning the numbers against each significant step in the code.
- ✧ Then using the Cyclomatic complexity correctness of the program can be assured.
- ✧ **Cyclomatic complexity**: is a useful metric for predicting those modules that are likely to be error prone. It can be used for test planning as well as test case design.

Methods used in WBT

1. Cyclomatic complexity: corresponds to number of regions of the flow graph or possible independent paths.
2. Cyclomatic complexity: V(G), for a flow graph, is defined as V(G) = E - N + 2
3. Cyclomatic complexity, V(G), for a flow graph, is also defined as V(G) = P + 1

Independent path introduces at least one new set of processing statements or condition.

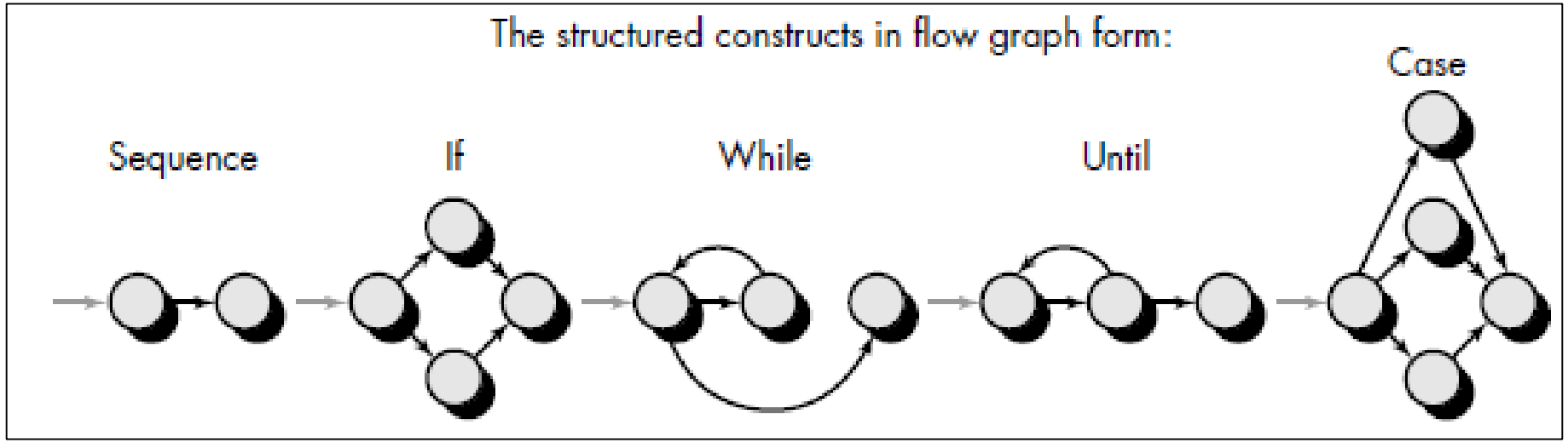
***E** is the number of flow graph edges*

***N** is the number of flow graph nodes.*

***P** is the number of predicate nodes in the flow graph.*

Methods used in WBT

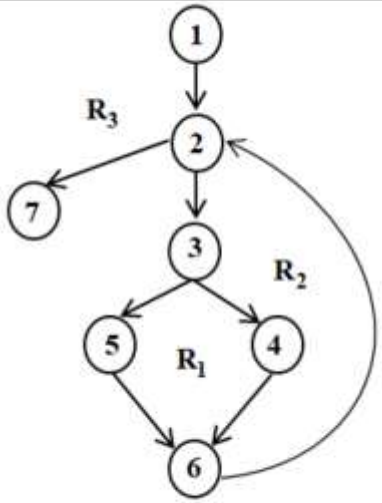
Flow Graph construction:



Methods used in WBT

Example-1:

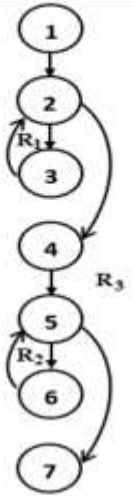
Descriptive Code	FlowGraph	

y = 5
Input (value of z) ①
for(int x = 0 ; x <= z ; x = x+y) ②
if(x%10==0) ③
Print("value is Even")
Print("value is Mulriple of Ten") ④
else
Print("value is Multiple of Five") ⑤
endif ⑥
endfor ⑦
		

Cyclometic Complexity		
Possible Paths	1-2-3-4-6-2-7 1-2-3-5-6-2-7 1-2-7 Total = 3	Regions = 3
E-V+2	8 – 7 + 2 = 3	
No of Predicate Nodes + 1	2 + 1 = 3	

Methods used in WBT

Example-2

<table>
<tr><th>Descriptive Code</th><th colspan="2">FlowGraph</th></tr>
<tr><td>A[3] = {1, 3, 5}
B[3] = {2, 4, 6} } 1
C[3]
for(i = 0, i < 3, i ++) 2
C[i] = A[i] + B[i] 3
end-for 4
for(i = 0, i < 3, i ++) 5
Print(C[i]) 6
end-for 7</td><td colspan="2"> 
</td></tr>
<tr><td colspan="3">Cyclometic Complexity</td></tr>
<tr><td>Possible Paths</td><td>1-2-3-2-4-5-6-5-7
1-2-4-5-6-5-7
1-2-4-5-7
Total = 3</td><td>Regions = 3</td></tr>
<tr><td>E-V+2</td><td colspan="2">8 – 7 + 2 = 3</td></tr>
<tr><td>No of Predicate Nodes + 1</td><td colspan="2">2 + 1 = 3</td></tr>
</table>

5.6 Software Maintenance

- Modifying a program after it has been put into use.
- The term is mostly used for changing custom software. Generic software products are said to evolve to create new versions.
- Maintenance does not normally involve major changes to the system's architecture.
- Changes are implemented by modifying existing components and adding new components to the system.

5.7 Problems During Maintenance

1. Often the program is written by another person or group of persons.
2. Often the program is changed by person who did not understand it clearly.
3. Program listings are not structured.
4. High staff turnover.
5. Information gap.
6. Systems are not designed for change

5.8 Types of Maintenance

- Maintenance to repair software faults

  Changing a system to correct deficiencies in the way meets its requirements.

- Maintenance to adapt software to a different operating environment

  Changing a system so that it operates in a different environment (computer, OS, etc.) from its initial implementation.

- Maintenance to add to or modify the system's functionality

  Modifying the system to satisfy new requirements.

5.9 Software Maintenance Process

- Once particular maintenance objective is established, the maintenance personnel must first understand what they are to modify.
- They must then modify the program to satisfy the maintenance objectives.
- After modification, they must ensure that the modification does not affect other portions of the program.
- Finally, the must test the program. These activities can be accomplished in the four phases as shown in Figure 8.2.

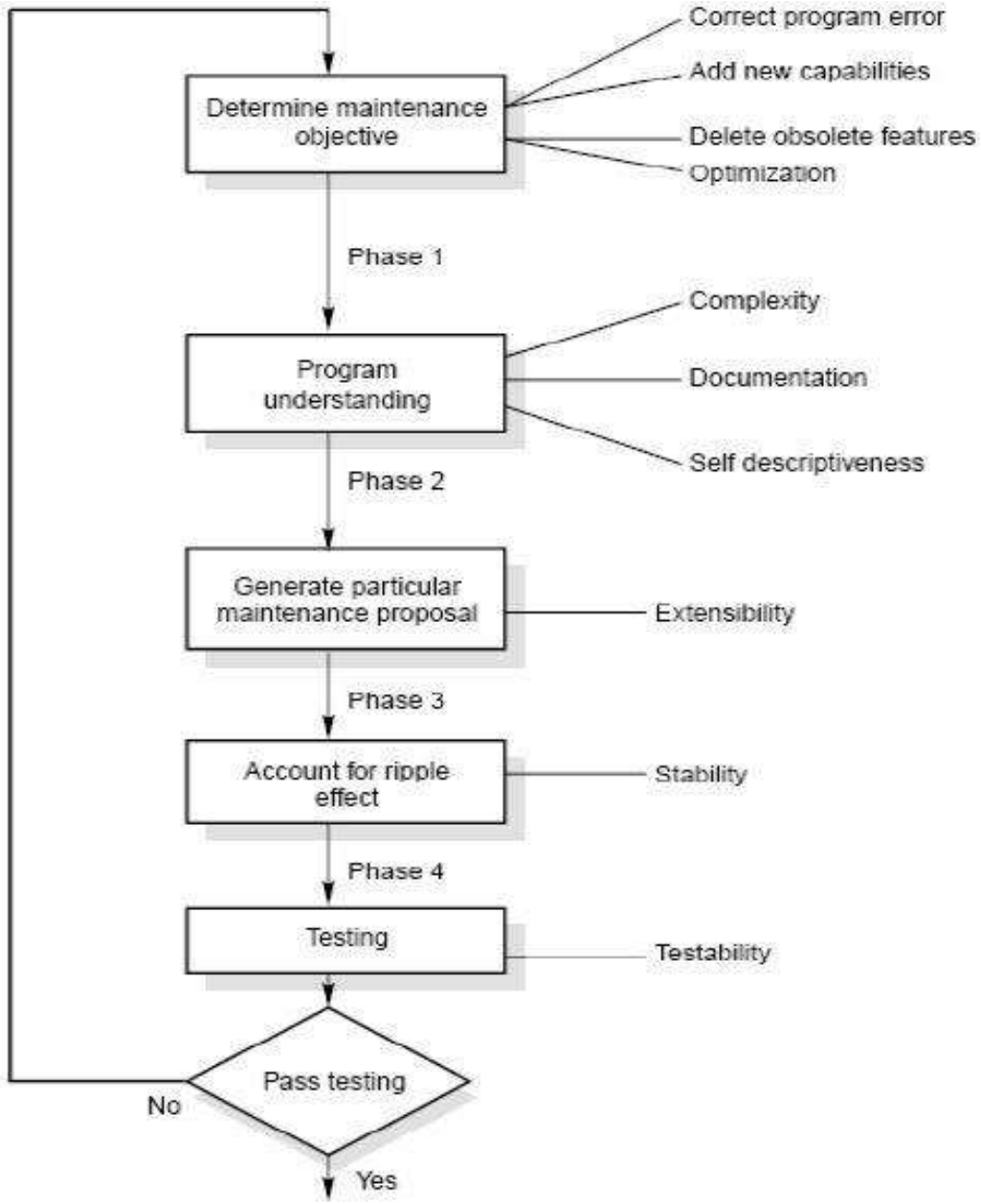
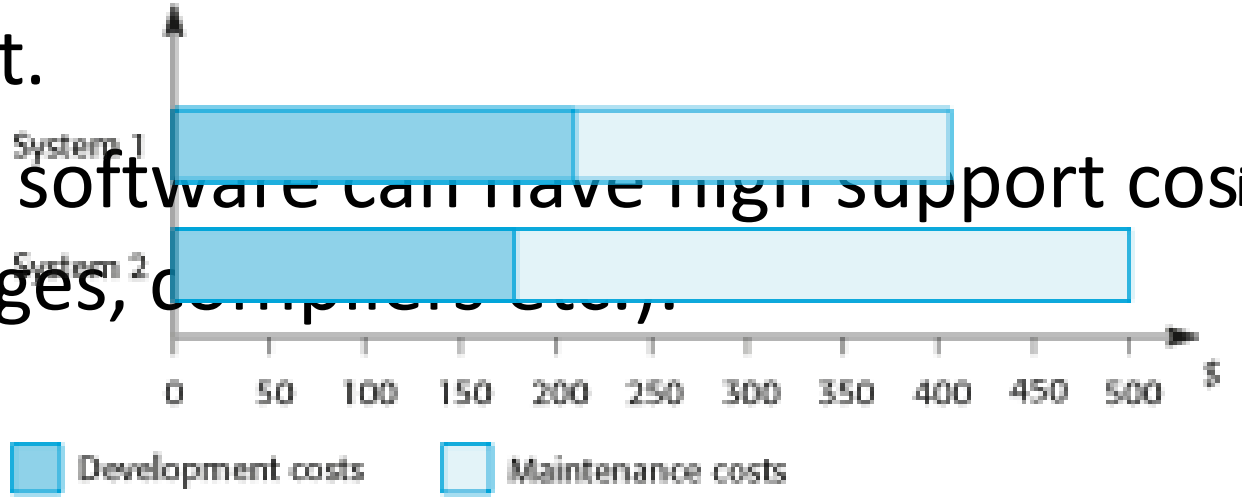


Figure 8.2: Software Maintenance Process

5.10 Maintenance Costs

- Usually greater than development costs (2 to 100 times depending on the application).
- Affected by both technical and non-technical factors.
- Increases as software is maintained. Maintenance corrupts the software structure so makes further maintenance more difficult.
- Ageing software can have high support costs (e.g. old languages, compilers etc.).



Development and Maintenance Costs

5.10.1 Maintenance Cost Factors

- Team stability

  Maintenance costs are reduced if the same staff are involved with the organization.

- Contractual responsibility

  The developers of a system may have no contractual responsibility for maintenance so there is no incentive to design for future change.

- Staff skills

  Maintenance staff are often inexperienced and have limited domain knowledge.

- Program age and structure

  As programs age, their structure is degraded and they become

5.11 Maintenance Prediction

- Maintenance prediction is concerned by identifying the parts of the system that may cause problems.
- More and more changes degrades the system and reduces its maintainability.

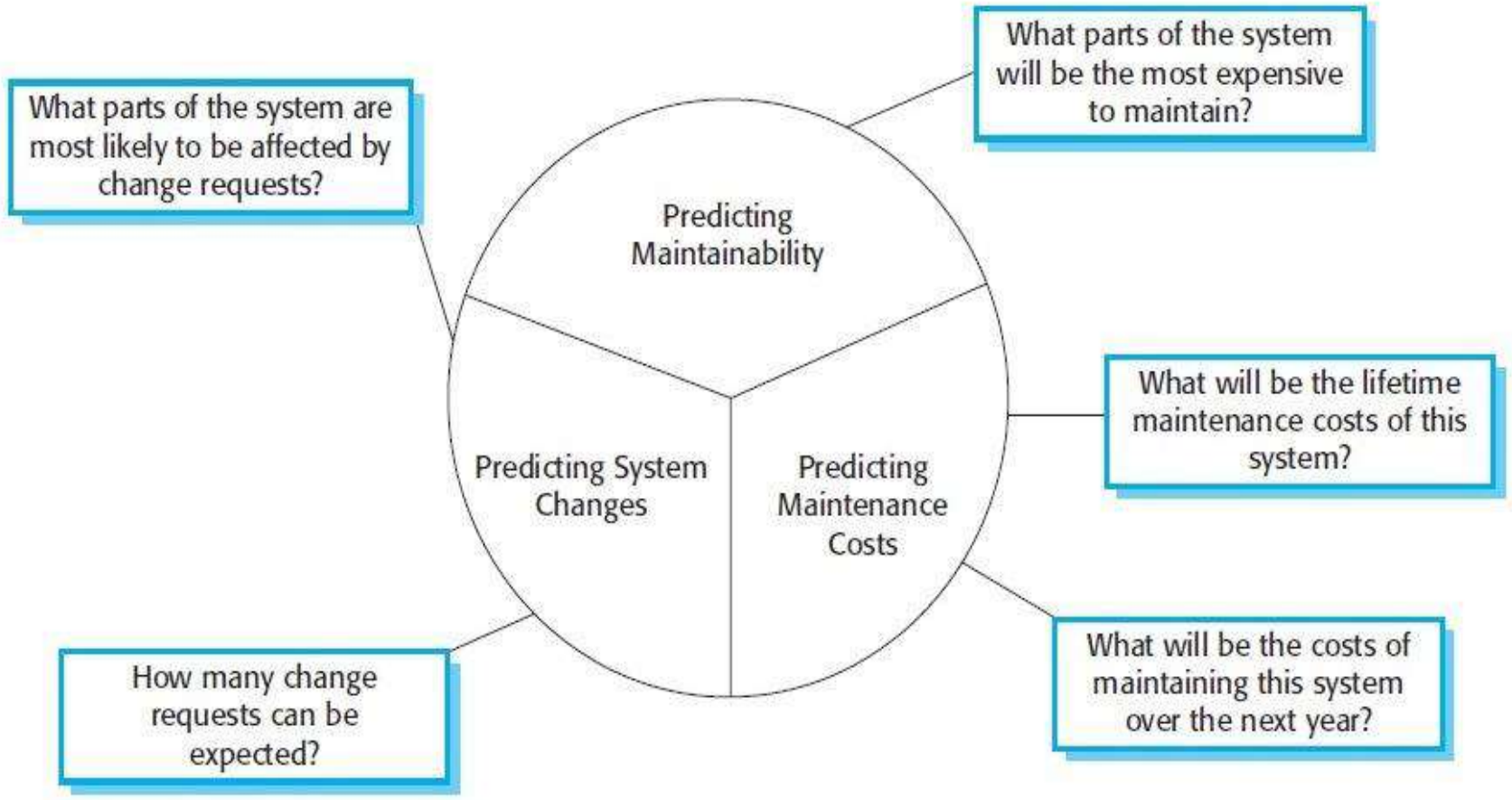


Figure 8.4: Maintenance Prediction

5.12 Software Re-Engineering

- Software re-engineering is concerned with taking existing legacy systems and re-implementing them to make them more maintainable without changing its functionality.
- The critical distinction between re-engineering and new software development is the

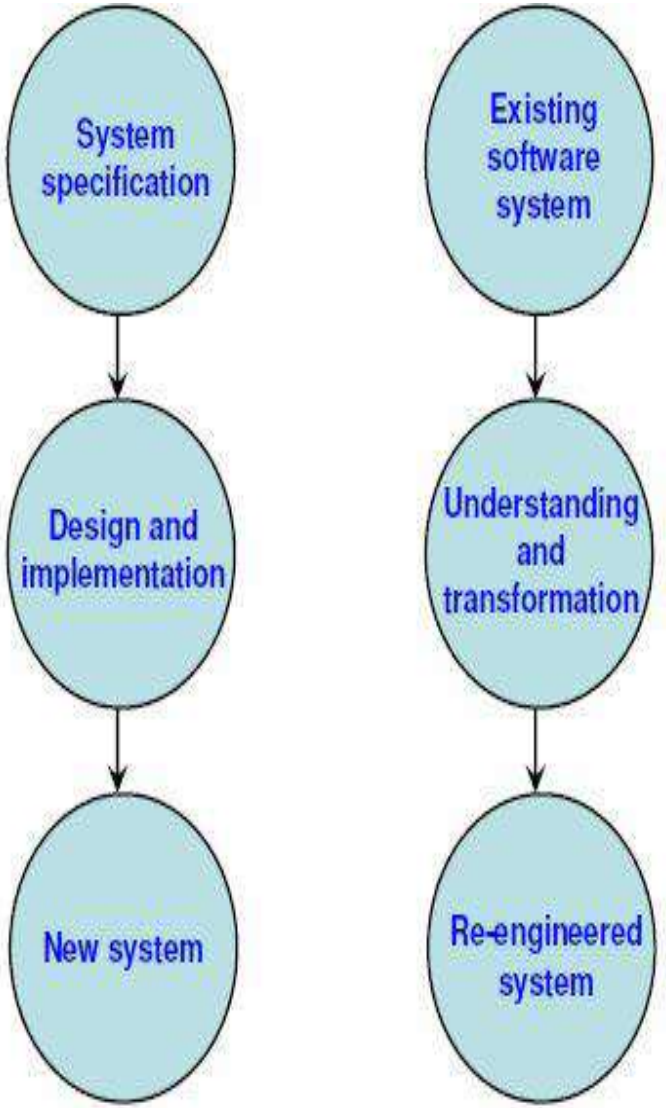


Figure 8.5: Comparison of new software development with re-engineering

Software Re-Engineering

- As a part of this re-engineering process, the system may be re-documented or restructured.
- It may be translated to a more modern programming language, implemented on existing hardware technology. Thus software re-engineering allows translate source code to new language, restructured old code, migrate to a new platform (such as client-server), capture and then graphically display design information, and re-document poorly documented systems.
- The cost of re-engineering depend on the extent of the work that is carried out. Other factors affecting costs are: the quality of the software, tool support available, extent of data conversion, availability of expert staff.
- The alternative to re-engineering a software system is to develop that system using modern software engineering techniques. Where systems are very badly structured this may be the only viable option as the re-engineering costs for these systems are likely to be high.

5.12.1 Reengineering process activities

- Source code translation

  Convert code to a new language.

- Reverse engineering

  Analyse the program to understand it;

- Program structure improvement

  Restructure automatically for understandability;

- Program modularisation

  Reorganise the program structure;

- Data reengineering

  Clean-up and restructure system data

5.12.2 Advantages of Re-Engineering

- Reduced risk

  There is a high risk in new software development. There may be development problems, staffing problems and specification problems.

- Reduced cost

  The cost of re-engineering is often significantly less than the costs of developing new software.

5.12.3 Reengineering cost factors

✧ The quality of the software to be reengineered.

✧ The tool support available for reengineering.

✧ The extent of the data conversion which is required.

✧ The availability of expert staff for reengineering.

Key Points

- ✧ software application needs to be verified as well as validated for a successful deployment.
- ✧ Testing is the process of executing a program with the intent of finding errors.
- ✧ **Stages of Testing-**Development testing, Release testing and User testing
- ✧ Black-box testing, also called behavioral testing, focuses on the functional requirements of the software.
- ✧ White-box testing also called glass-box testing. In WBT test cases are designed using the control structure of the procedural design.

Key Points

- There are 3 types of software maintenance, namely bug fixing, modifying software to work in a new environment, and implementing new or changed requirements.
- Software re-engineering is concerned with re-structuring and re-documenting software to make it easier to understand and change.
- The business value of a legacy system and the quality of the application should be assessed to help decide if a system should be replaced, transformed or maintained.

Review questions

1. Differentiate between software verification and software validation.
2. Write short notes on the different stages of testing
3. What are the different types of user testing
4. Explain in detail about BBT(Black Box Testing)
5. What is meant by cyclomatic complexity?
6. List down the problems happened during maintenance
7. What are the different types of maintenance?
8. Illustrate software maintenance process with neat diagram
9. List down the factors that affects the maintenance cost.

Review questions

10. Write short notes on Maintenance prediction
11. What is meant by software reengineering. List down its advantage.
12. List the activities of Reengineering process.
13. Verify the following descriptive code using White Box Testing.

(a)

```
1  get ( x )
   while (x<100)  2
      val = x % 2  3
4  if ( val == 0 )
         print ("Even Number")  5
      else
         print ("Odd Number")  6
      end-if  7
   end-while  8
```

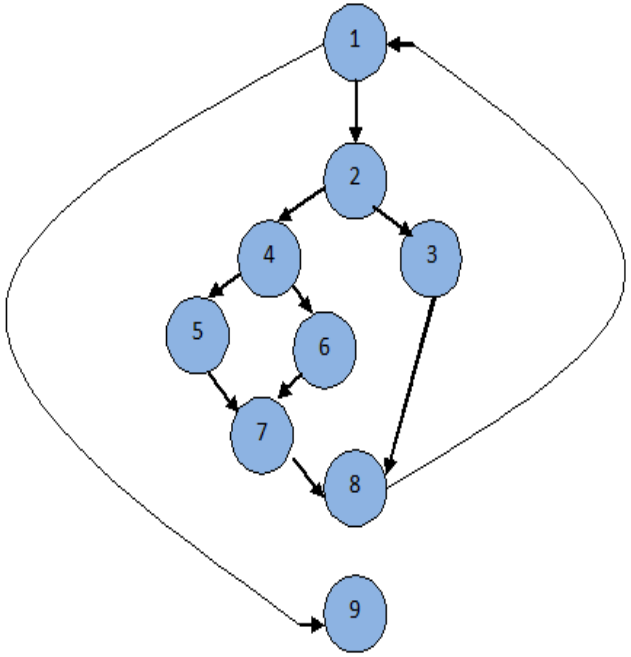
(b)

```
1  get ( A)
   get ( B )
   if ( A == B )  2
         print ("A and B are Equal")  3
   else if ( A > B)  4
         print ("A is Big")  5
   else if (A < B )  6
         print ("B is Big")  7
   end if  8
```

Review questions

14. Calculate cyclomatic complexity using all methods for the given control flow graph.

(a)



(b)