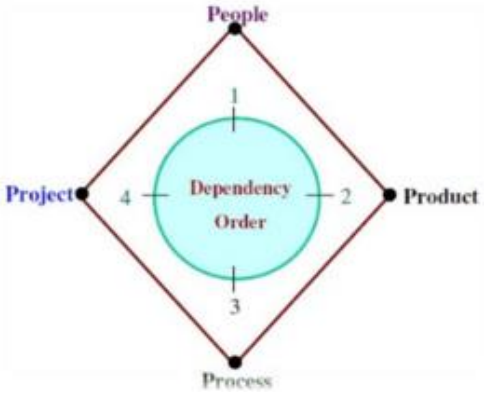
Chapter 1

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

Software engineering is a process of analyzing user requirements and then designing, building, and testing software applications.

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.



Software products

1. **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.
2. **Customized products**
   Software that is designed for a specific customer to meet their own needs.
   Examples – embedded control systems, air traffic control software, traffic monitoring systems.

Essential attributes of good software

1. **Maintainability**
   Software should be written in such a way so that it can evolve to meet the changing needs of customers.
   The ability to update it.
2. **Dependability and security**
   Malicious users should not be able to access or damage the system.
3. **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.
4. **Acceptability**
   Software must be acceptable to the type of users for which it is designed.

Challenges for Software Engineering Practices

1. **Heterogeneity**
   systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.
2. **Business and social change**
   They need to be able to change their existing software and to rapidly develop new software.
3. **Security and trust**
   The software must be secure and trustable.

Application types

1. **Stand-alone applications**
   They include all necessary functionality and do not need to be connected to a network.
2. **Interactive transaction-based applications**
   Applications that execute on a remote computer and are accessed by users from their own PCs or terminals.
   Note// These include web applications such as e-commerce applications.
3. **Embedded control systems**
   These are software control systems that control and manage hardware devices.
   Note// Numerically, there are probably more embedded systems than any other type of system.
4. **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.
5. **Entertainment systems**
   These are systems that are primarily for personal use and which are intended to entertain the user.
6. **Systems for modeling and simulation**
   These are systems that are developed by scientists and engineers to model physical processes or situations.
7. **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.
8. **Systems of systems**
   These are systems that are composed of a number of other software systems.

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.

Note// 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 sometimes retirement phase".

These activities involve the development of software from scratch until it is delivered to customers.

Software Process Model

Is 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.

Note// A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

Types of Software process models

1. The Waterfall Model.

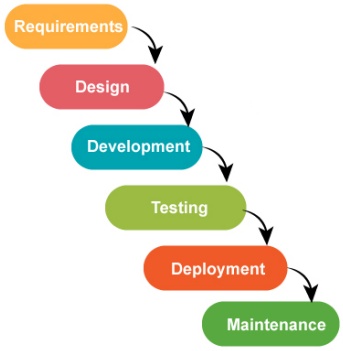
Plan-driven model. Separate and distinct phases of specification and development.

The waterfall model is also referred to as a linear-sequential life cycle model.

In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

In Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

Note// Waterfall model is the **earliest software development life cycle (SDLC)**. It is also referred to as a **linear-sequential life cycle 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.

Advantages & Disadvantages

Advantages	Disadvantages
Simple and easy to understand and apply.	No working software is produced until whole life cycle is not completed.
Easy to manage due to fix boundaries each phase with specific deliverables.	Not a good model for complex and object-oriented projects.
Phases are processed and completed one at a time.	Poor model for long and ongoing projects.
Works well for smaller projects where requirements are very well understood.	Not suitable for the projects where requirements are changing frequently.
Process and results are well documented.	It is difficult to measure progress within stages.
	Adjusting scope during the life cycle can end a project

2. Incremental Development Model

Specification, development and validation are interleaved. May be plan-driven or agile.

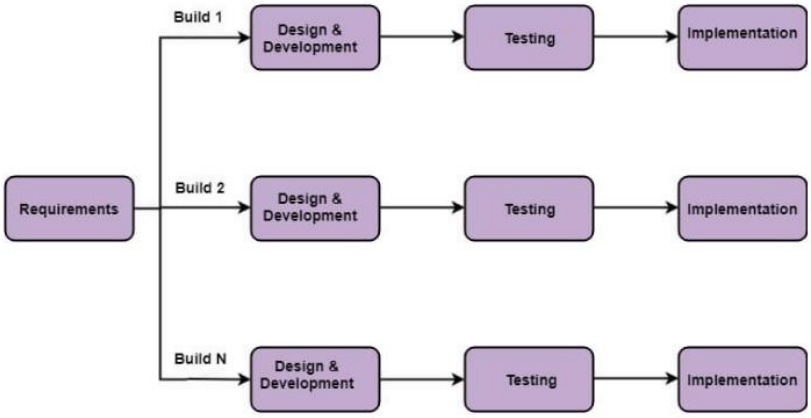
Incremental model uses the linear sequential approach with the iterative nature of prototyping.

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 release. The process continues till the complete system is ready as per the requirement.

Note// The key to use of incremental development model successfully is rigorous validation of requirements, and verification of each version of the software against the requirements.



Advantages & Disadvantages

Advantages	Disadvantages
At the end of each build, working product is produced that facilitates customer evaluation and feedback.	More resources (manpower) may be required.
Results are obtained periodically.	Although cost of change is lesser in this model, but changing requirements itself is not very suitable.
Parallel development can be planned and measured.	More management attention is required.
Less costly to change the scope/requirements.	Not suitable for smaller projects.
Testing and debugging during smaller iteration is easy.	Defining increments (builds) may require definition of the complete system.
Risks are identified and resolved during iteration; and each iteration is an easily managed milestone.	System architecture or design issues may arise, all requirements are not gathered at the beginning of life cycle.
	Ending of project is not very obvious.
	Project's progress is highly dependent upon the risk analysis phase.

3. Reuse-Oriented Model

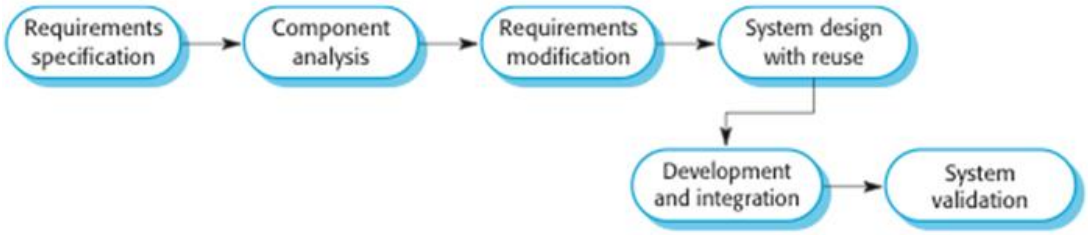
The system is assembled from existing components. May be plan-driven or agile.

Based on systematic **reuse** where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.

This often happens informally when people working on the project know of designs or code that is similar to what is required.

Note// Reuse is now the standard approach for building many types of business system.

the stages of “requirements specification” and “system validation” are same as in other models but intermediate stages in this model are different.



Explanation of Reuse-Oriented Model Phases

Phase	Description
Component Analysis	After getting user requirements a search is made for components to be implemented that are available in library as it is or with small variations. Usually, there is no exact match and already available components only provide some of the required functionality.
Requirements Modification	During this stage, the requirements are analyzed using information about the components that have been discovered. They are then modified to reflect the available components. Where modifications are impossible, the component analysis activity may be re-entered to search for alternative solutions.
System Design with Reuse	During this phase, the framework of the system is designed or an existing framework is reused. The designers take into account the components that are reused and organize the framework to cater for this. Some new software may have to be designed if reusable components are not available.
Development and Integration	Software that cannot be externally procured is developed, and the components and COTS (commercial off-the-shelf) systems are integrated to create the new system. System integration, in this model, may be part of the development process rather than a separate activity.

Note// In practice, most large systems are developed using a process that incorporates elements from all of these models.

4. Boehm's Spiral Model

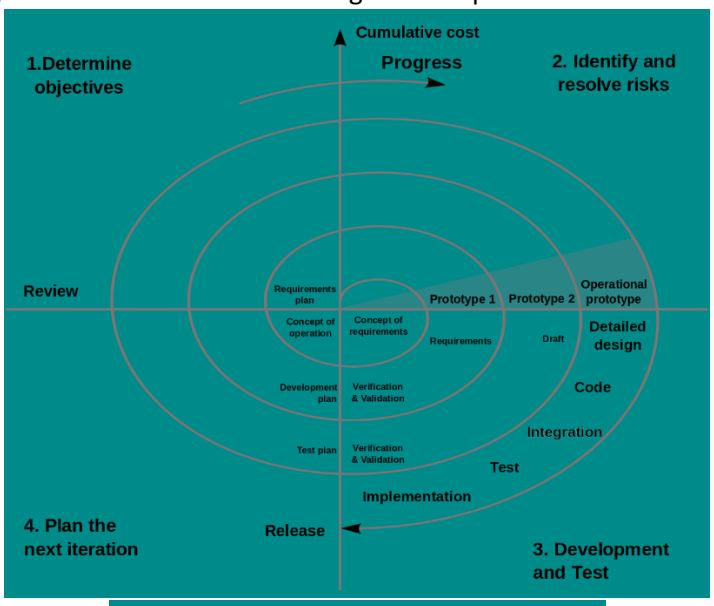
A risk-driven software process framework (the spiral model) was proposed by Boehm.

Process is represented as a spiral rather than as a sequence of activities with some backtracking from one activity to another.

Note// 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.



Explanation of Spiral Model Activities

Activity	Description
Determine Objective	Specific objectives for that phase of the projects are defined. Constraints on the process and the product are identified and a detailed management plan is drawn up. Project risks are identified. Alternative strategies, depending on these risks, may be planned.
Identify and resolve risk	For each of the identified project risks, a detailed analysis is carried out. Steps are taken to reduce the risk. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed.
Development and test	After risk evaluation, a development model for the system is chosen. For example, throwaway prototyping may be the best development approach if user interface risks are dominant. If safety risks are the main consideration, development based on formal transformations may be the most appropriate process, and so on. If the main identified risk is sub-system integration, the waterfall model may be the best development model to use.
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.

Advantages & Disadvantages

Advantages	Disadvantages
Changing requirements can be accommodated with the help of extensive use of prototypes.	Management is more complex.
Requirements can be captured more accurately.	End of project may not be known clearly.
Users see the system early.	Not suitable for small or low risk projects and could be expensive for small projects.
Development can be divided into smaller parts and more risky parts can be developed earlier which helps better risk management.	Process is complex
	Large number of intermediate stages requires excessive documentation

5. Agile model

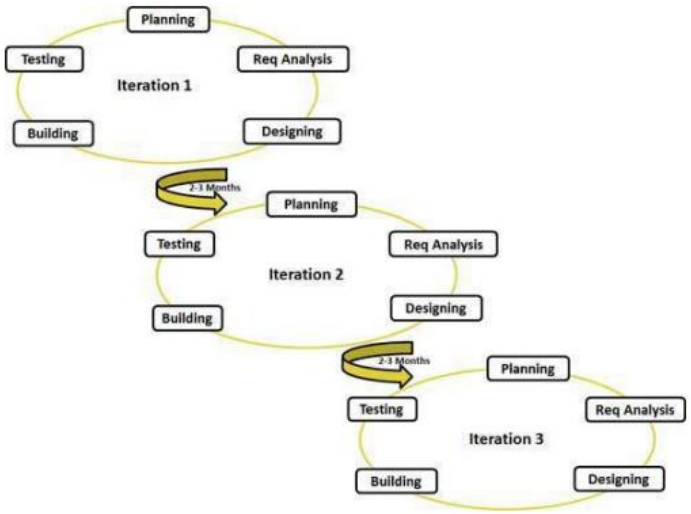
is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.

Note// 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.

Draw



Advantages & Disadvantages

Advantages	Disadvantages
Very realistic approach for software development.	Not suitable for handling complex projects.
Promotes teamwork and cross training.	More risk of sustainability, maintainability and extensibility.
Functionality can be developed rapidly and demonstrated.	Not suitable for small or low risk projects and could be expensive for small projects.
Resource requirements are minimum.	Without an overall plan and in absence of agile project manager practice may fail entirely.
Suitable for fixed or changing requirements.	Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
Minimal rules, documentation easily employed.	There is very high individual dependency, since there is minimum documentation generated.
Enables concurrent development and delivery within an overall planned context.	Transfer of technology to new team members may be quite challenging due to lack of documentation.
Little or no planning required.	
Easy to manage.	
Gives flexibility to developers.	

Chapter 2

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.

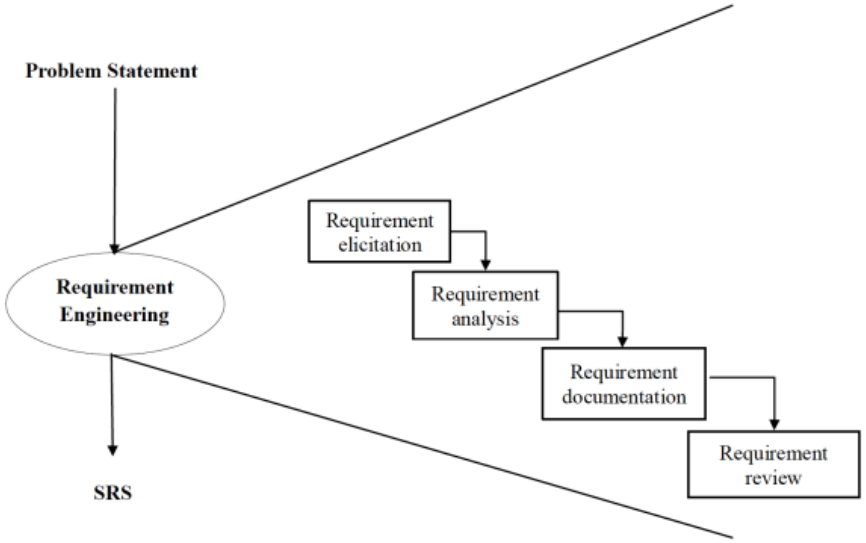
Note// Requirements engineering is one of the most crucial activity in this creation process.

Crucial Process Steps of Requirement Engineering

1. **Requirements elicitation**
   Here requirements are identified with the help of customer and existing systems process if available.
   This is also known as gathering of requirements.
2. **Requirements analysis**
   The requirements are analyzed in order to identify inconsistencies, defects, omissions, and also resolve conflicts if any.
3. **Requirements documentations**
   It is the foundation for the design of the software.
   Note// The document is known as **software requirements specification (SRS).**
4. **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 requirement specification**.
- If it describes only **software**, it is a **software requirement specification**.



Types of Requirements

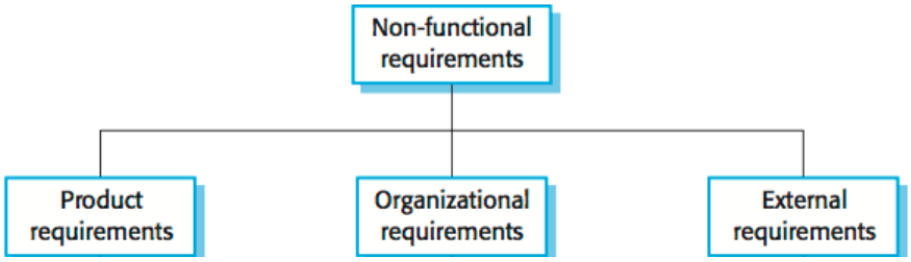
1. Functional requirements

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

2. Non-functional requirements

These define system properties and constraints (reliability, response time and storage requirements). Constraints are (I/O device capability, system representations, etc).
Note// 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



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

User and System Requirements

- **User requirement**
  are written for the users and include functional and non-functional requirement.
  Note// 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.
  - **metric** is a quantitative measure of the degree to which a system, component, or process possesses a given attribute.

Categories of Metrics

1. 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

2. 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.

3. 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

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.

Note// The SRS serve **as contract** document between customer and developer.

Note// 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
5. Design constraints imposed on an implementation.

Characteristics of a Good SRS

The SRS should be:

- Correct
- Unambiguous
- Complete
- Consistent
- Rank for importance and/ stability
- Verifiable
- Modifiable
- Traceable

Requirements Gathering Techniques

Note// Some requirements gathering techniques may be beneficial in one project but may not be in other.

1. **Brainstorming**
   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.
2. **Document Analysis**
   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.
3. **Focus Group**
   is a gathering of people who are customers or user representatives for a product to gain feedback.
4. **Interface Analysis**
   Interface for any software product are either be human or machine.
5. **Interview**
   Interviews of users and stakeholders are important in creating wonderful software.
6. **Observation**
   The observation covers the study of users in their workplace.
7. **Prototyping**
   Prototyping can be very helpful at gathering feedback.
8. **Requirements Workshop**
   Popularly known as JAD or Joint Application Design, these workshops can be efficient for gathering requirements.
9. **Reverse Engineering**
   When a legacy project does not have enough documentation, reverse engineering can determine what system does?
   Note// It do not determine what features went wrong with the system and what a system must do.
10. **Survey**
    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.

Chapter 3

Architectural Design

is concerned with understanding how a software system should be organized and designing the overall structure of that system.

Note// **Architectural design** is **the first stage** in the software design process.

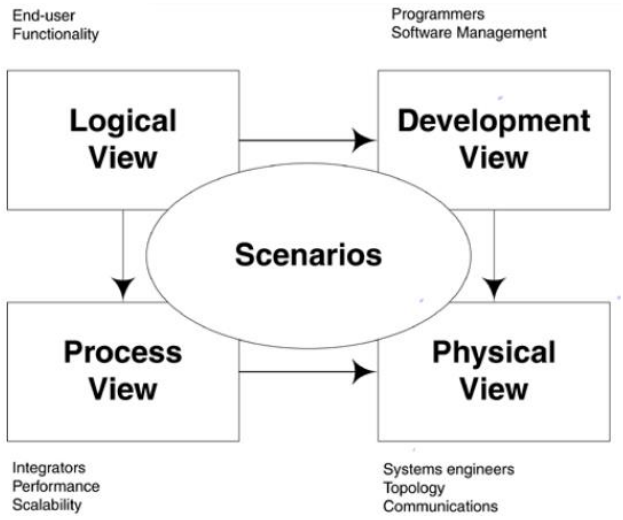
It is the critical **link** between design and requirements engineering, as it identifies the main structural components in a system and the relationships between them.

The **output of the architectural design** process is an **architectural model** that describes how the system is organized as a set of communicating components.

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.

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.

Application Architectures

Application systems are designed to meet an organizational need.

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.

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.

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.

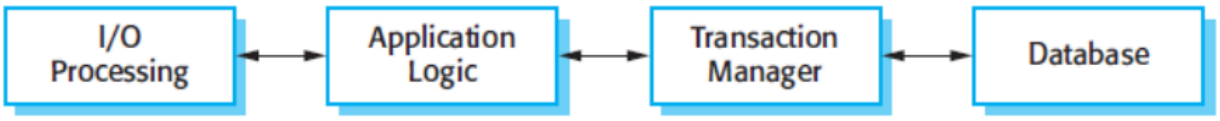


Figure 3.7: Architecture of TPS

The Software Architecture of an ATM System

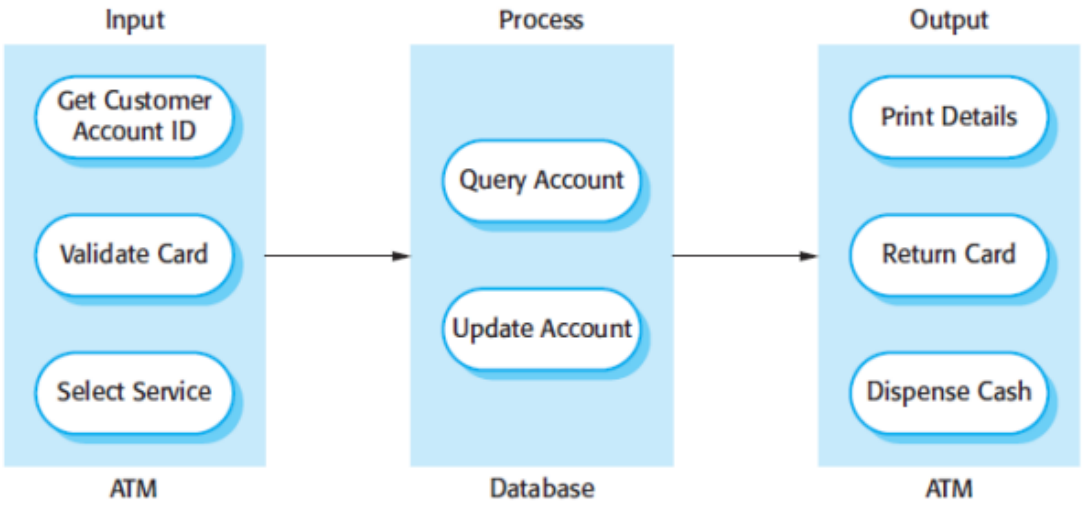


Figure 3.8: Architecture of ATM

Chapter 4

Implementation Issues

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.

- **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).**

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.

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.

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.

Host-Target Development

Most software is developed on one computer (**the host**), but runs on a separate machine (**the target**).

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.

Open-Source Development

the source code of a software system is published and volunteers are invited to participate in the development process.

Note// Its roots are in the Free Software Foundation (www.fsf.org).

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.

Note// The best-known open-source product is, of course, the Linux operating system.

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.

Chapter 5

Software Verification and Validation

1. **Verification**
   "Are we building the product right", "Does the product meet system specifications?"
2. **Validation**
   "Are we building the right product", "Does the product meet user expectations?"

Software Testing

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

Note// Testing can reveal the **presence of errors NOT their absence**.

Stages of Testing

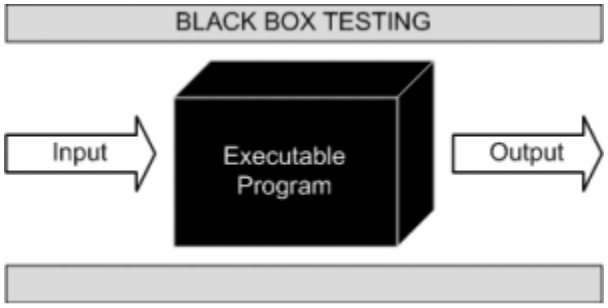
1. **Development testing**
   where the system is tested during development to discover bugs and defects.
   - **Unit testing**
     where individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods.
   - **Component testing**
     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.

   Note// One of the popular development testing techniques is **White Box Testing (WBT)**
2. **Release testing**
   where a separate testing team test a complete version of the system before it is released to users.
   Release testing shows that the system delivers its specified functionality, performance and dependability, and that it does not fail during normal use.
   Note// Release testing is usually a **Black Box Testing (BBT)** process
3. **User testing**
   where users or potential users of a system test the system in their own environment.
   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.

Black Box Testing (BBT)

also called behavioral testing, focuses on the functional requirements of the software.

Note// BBT enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program.



BBT is concerned only with the possible inputs and their desired output regardless of the developmental details.

Note// BBT is **performed by a separate testing team NOT** the developer their self hence it is **one type of release testing**.

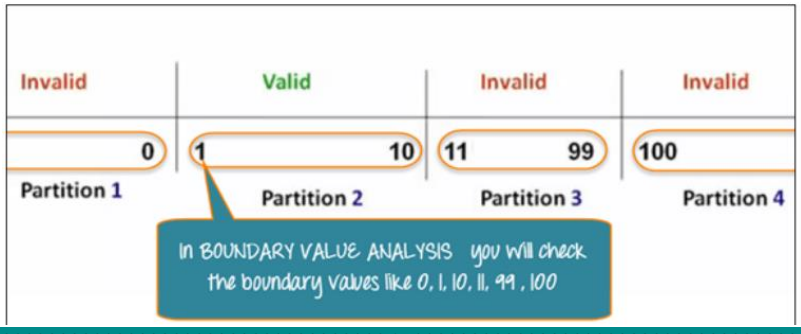
BBT Types

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

1. **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

2. **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



White Box Testing (WBT) //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.

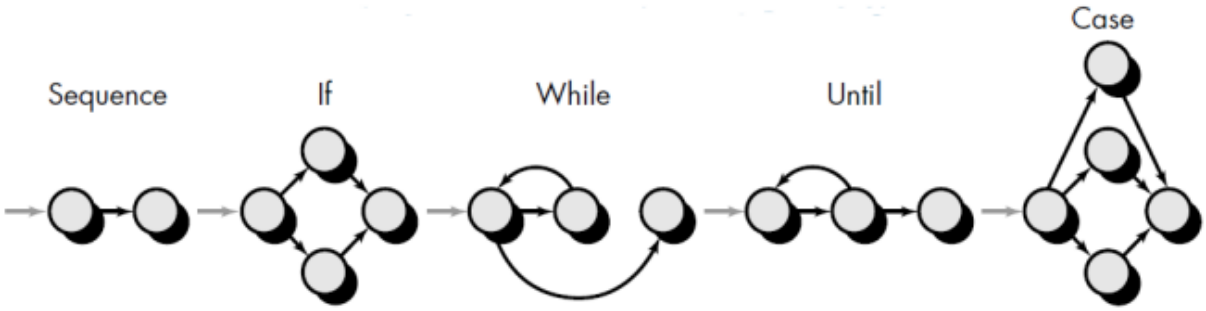
Note// WBT can be applied at **any** **level of development testing**.

1. In WBT at first code is converted into descriptive code by assigning the numbers against each significant step in the code.
2. Then using the **Cyclomatic complexity** correctness of the program can be assured.

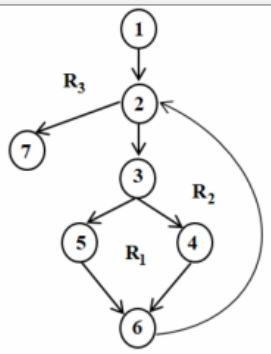
Cyclomatic complexity

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.

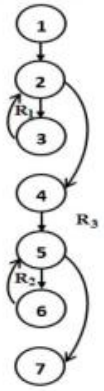
Flow Graph construction



Example 1

<table>
<tr><th>Descriptive Code</th><th colspan="2">FlowGraph</th></tr>
<tr><td>y = 5 ①
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</td><td colspan="2"> 
</td></tr>
<tr><td colspan="3">Cyclometic Complexity</td></tr>
<tr><td>Possible Paths</td><td>1-2-3-4-6-2-7
1-2-3-5-6-2-7
1-2-7
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>

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>

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

Types of Maintenance

1. Maintenance to repair software faults.
2. Maintenance to adapt software to a different operating environment.
3. Maintenance to add to or modify the system's functionality.

Maintenance Costs

Usually **greater** than development costs (2 to 100 times depending on the application).

Affected by both technical and non-technical factors.

Maintenance Cost Factors

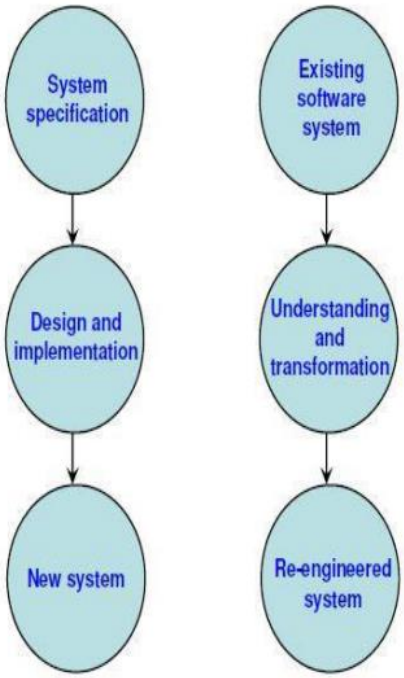
- Team stability
- Contractual responsibility
- Staff skills
- Program age and structure

Maintenance Prediction is concerned by identifying the parts of the system that may cause problems.

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 starting point for the development as shown in Figure



Reengineering process activities

- **Source code translation**
  Convert code to a new language.
- **Reverse engineering**
  Analyze the program to understand it.
- **Program structure improvement**
  Restructure automatically for understandability.
- **Program modularization**
  Reorganize the program structure.
- **Data reengineering**
  Clean-up and restructure system data.

Advantages of RE-Engineering

1. **Reduced risk**
   There is a high risk in new software development. There may be development problems, staffing problems and specification problems.
2. **Reduced cost**
   The cost of re-engineering is often significantly less than the costs of developing new software.

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.

Review questions

13. Verify the following descriptive code using White Box Testing.

(a)

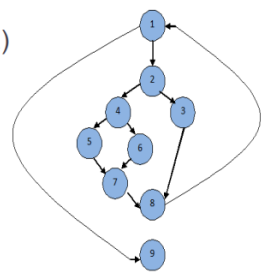
```
1 get ( x )
    while (x<100) 2
      val = x % 2 3
    4 if ( val == 0 )
        print ("Even Number") 5
      else
        print ("Odd Number") 6
      end-if 7
    end-while 8
```

(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
```

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

(a)



(b)