Software Engineering

2020-2021

الدراسات الأولية / المرحلة الثالثة / الفصل الأول

أستاذ المادة

م.د.فاتن عبدعلي داود

Introduction

What difference between Software(S/W) and Software engineering (S.E)?

Software (s/w): is a computer program which may be applied in any situation for specified set of procedural steps has been defined (e.g. algorithm).

A simple def. is: Software program = data structures + algorithms

Software engineering (S.E):

A simple def. is:

Software engineering = data structures + algorithms +documentation.

There are several definitions of **Software engineering** as follows:

First: Software engineering refers to a systematic procedure that is used in the context of a generally accepted set of goals for the analysis, design, implementation, testing and maintenance of software.

Second: S.E is an outgrowth of H.W and system engineering. It encompasses a set of three key elements: ***methods***, ***tools*** and ***procedures*** that enables the manager to control the process of s/w development and provide high quality software in a productive manner.

- ***Methods:*** consist of system and software requirements, planning, design of data structure and algorithm procedure, coding, testing and maintenance.
- ***Tools:*** used to implements each one of the previous methods.
- ***Procedures:*** used to define the implementation sequence for both methods and tools.

Third: S.E is the practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate and maintain them.

What is Software engineering?

- Software engineering is concerned with theories, methods, and tools which are needed to develop a good ***software product***.
- The objective of S.E is to produce a high quality software products with a finite amount of resources and to predicated schedule.

Software products: are software systems which delivered to a customer with the documentation that describes how to install and use the system. Software products may be developed for a particular customer or may be developed for a general market.

There are two kinds of 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 editing, graphics programs, project management tools; CAD software_ software for specific markets such as appointments systems for dentists.*

2. 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.*

Software Applications:

1. **System software:** such as compilers, editors, file management utilities
2. **Application software:** stand-alone programs for specific needs.
3. **Engineering/scientific software:** such as automotive stress analysis, molecular biology, orbital dynamics etc.
4. **Embedded software** resides within a product or system. (key pad control of a microwave oven, digital function of dashboard display in a car)
5. **Product-line software** focus on a limited marketplace to address mass consumer market. (word processing, graphics, database management)
6. **WebApps** (Web applications) network centric software. As web 2.0 emerges, more sophisticated computing environments is supported integrated with remote database and business applications.
7. **AI software** uses non-numerical algorithm to solve complex problem. Robotics, expert system, pattern recognition game playing.

Software—New Categories

- **Open world computing**— distributed computing due to wireless networking. (How to allow mobile devices, personal computer, enterprise system to communicate across vast network).

- **Net sourcing**—the Web as a computing engine. How to architect simple and sophisticated applications to target end-users worldwide.

- **Open source**—"free" source code open to the computing community. Software for: Data mining, Grid computing, Cognitive machines and nanotechnologies.

Why Software is Important?

- The economies of ALL developed nations are dependent on software.
- More and more systems are software controlled (transportation, medical, tele-communications, military, industrial, entertainment,)
- Software engineering is concerned with theories, methods and tools for professional software development.

Goals of software engineering are :

1- Low cost of production.
2- High performance.
3- Portability.
4- High reliability.
5- Low cost of maintenance.
6- Delivery on time.

Essential attributes of well- engineering software

Product characteristic	Description
1- 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.
2- Dependability & 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 accessor 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. This means that it must be understandable, usable and compatible with other systems that they use.

What are the costs of software engineering?

- Software is the most expensive component of many computer-based systems, a reliable approach to cost estimation is important for the continued success of a S/w development organization.

- Roughly 60% of costs are development costs, 40% are testing costs.

- Costs vary depending on:
  (1) the type of system being developed
  (2) the requirements of system attributes such as performance and system reliability.

- Distribution of costs depends on the development model that is used.

What are software engineering methods?

- Structured approach to software development which includes: system models, notation, rules, design advice, and process guidance.

- **Module descriptions:** Graphical models that should be produced.
- **Rules:** Constraints applied to system models.
- **Recommendations:** Advice on good design practice.
- **Process guidance:** What activities to follow and how to do so.

Software process: Is a structured set of activities required to develop a software system whose goal is the production or evolution of the software system.

There are generic activities in all software processes:

1. ***Specification:*** the process of establishing what services are required and identifying the constraints (e.g. cost and time) on the operation of the system and its development.

2. ***Development*** (design and implementation): the process of converting the system specification into an executable system.

Software design: design a software structure that realizes the specification.

What design methods?

Is a systematic approach of software design that is usually documented as a set of graphical models:

- Data- flow model.
- Entity- relation- attribute model.
- Structural model.
- Object models.

Implementation: translate the designed software structure into an executable program.

- This activity contains the programming and debugging process which translating a design into a program and removing errors from that program.
- Programming is a personal activity; there is no "best" generic programming process.
- Programmers carry out some program testing to discover faults in the program and remove these faults in the debugging process; as shown in Figure **2.1**.

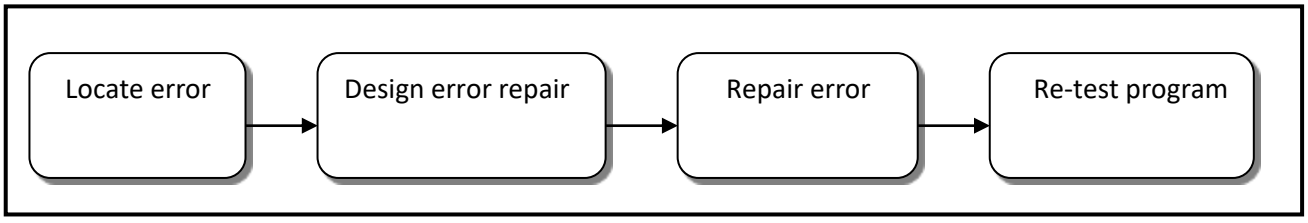

Figure (2.1): The debugging process.

3. ***Validation:*** checking that the software is what the customer wants.
Validation is intended to show that a system conforms to its specification and meets the requirements of the customer. Involves ***checking***, ***review processes***, and ***system testing***.

- ***System testing*** involves executing the system with test cases that are derived from the specification of the real data to be processed by the system. System testing process contains several stages as shown in Figure **2.2**, they are :

1-**Unit testing:** individual components are tested.

2- **Module testing:** collections of related components are tested.

3- **Sub-system testing:** modules are integrated into sub-systems and tested.

4- **System testing:** integrated system as a whole is tested.

5- **Acceptance testing:** final test with real data for customer/client acceptance.

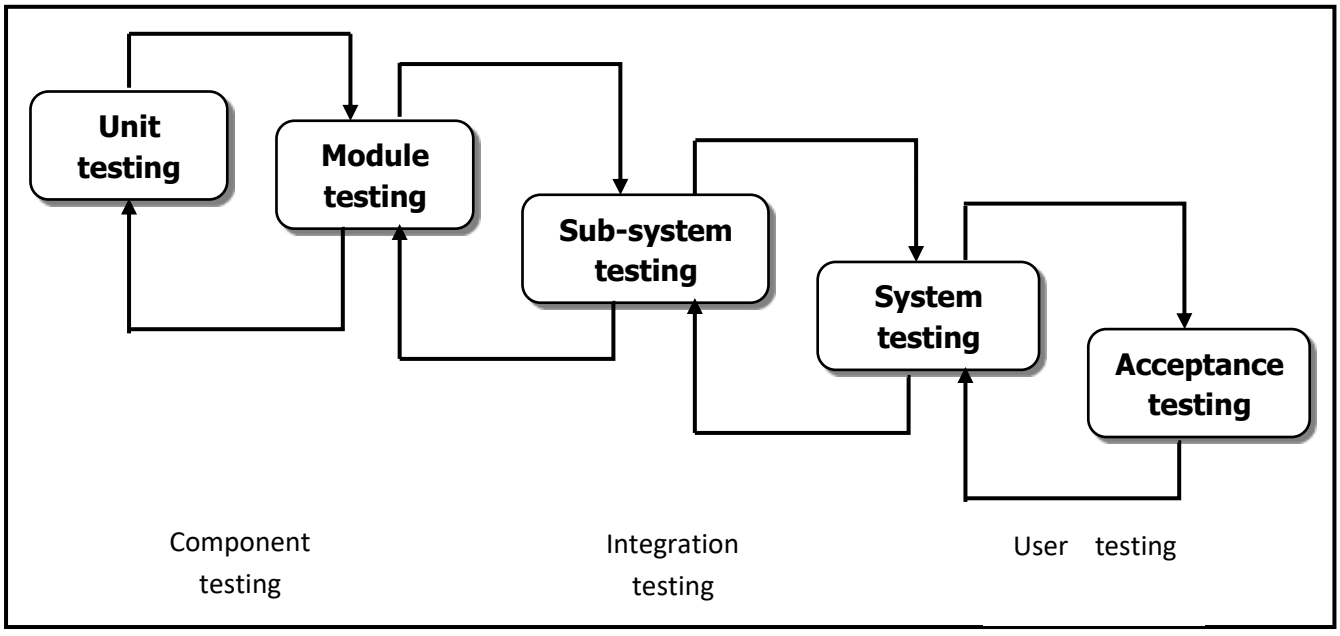

Figure (2.2): The testing process.

4. ***Evolution:*** changing the software in response to changing in demands. The software evolves to meet changing customer needs. Evolution is concerned with modifying the system after it is in use; as shown in Figure **2.3**.

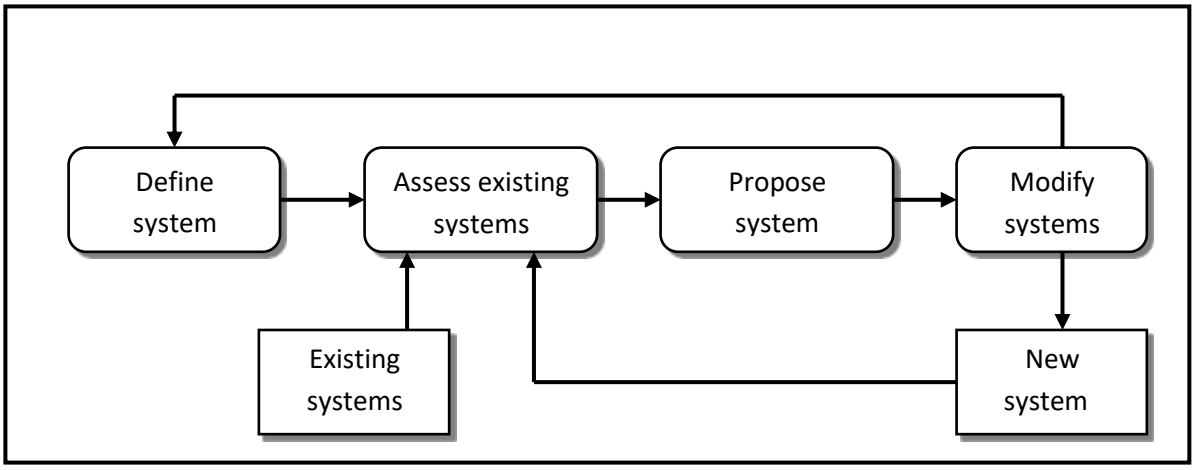

Figure (2.3): The system evolution.

Software Process

What is a Software process?

A process is a collection of activities, actions and tasks that are performed when some work product is to be created. It is *not a rigid prescription* for how to build computer software. Rather, it is an adaptable approach that enables the people doing the work to pick and choose the appropriate *set of work actions* and tasks.

Five Activities of a Generic Process framework as follows:

1. **Communication:** communicate with customer to understand objectives and gather requirements

2. **Planning:** creates a "map" defines the work by describing the tasks, risks and resources, work products and work schedule.

3. **Modeling:** Create a "sketch", what it looks like architecturally, how the constituent parts fit together and other characteristics.

4. **Construction:** code generation and the testing.

5. **Deployment:** Delivered to the customer who evaluates the products and provides feedback based on the evaluation.

- These five framework activities can be used to all software development regardless of the application domain, size of the project, complexity of the efforts etc, though the details will be different in each case.

- For many software projects, these framework activities are applied iteratively as a project progresses. Each iteration produces a software increment that provides a subset of overall software features and functionality.

What is a requirement?

- It may range from a high-level abstract statement of a service that the system should provide or a system constraint to a detailed mathematical functional specification.
- All of the process models for software development include activities aimed at capturing the requirements to understanding what the customer wants and what users expect the system will do.

•The requirements may be developing to:

- Replace an existing manual system.
- Enhance or extend an existing software system.
- Develop a new system unrelated to existing systems.

Types of Requirements

User requirements:

1- Statements in natural language (NL) plus diagrams of the services the system provides and its operational constraints.
2- Should describe functional and non-functional requirements so that they understand by system users who do not have detailed technical knowledge.
3- User requirements are defined using NL, tables, and diagrams.
4- Written for customers.
5- ***Requirements readers*** **(*Client managers*, *System end-users* and *System Architects*).**

System requirements:

1- Structured documents setting out detailed description of the system services. Serve as a basis for designing the system.
2- May be used as part of the system contract.
3- System requirements may also be expressed using system models.
4- Written as a contract between client and contractor.
5- ***Requirements readers*** ***(System end-users, System architects and Software developers).***

What is Requirements definition ?

1- A high-level abstract description of requirements.
2- Is a statement, in a natural language (NL) plus diagrams, of what services the system is expected to provide and the constraints under which it must operate.
3- It is generated using customer-supplied information.
4- It should only specify the external behavior of the system; it should not be concerned with system design characteristics.

5- Should be written in such a way that it is understand able by customers without knowledge of specialized notations.

What is Requirements Specification ?

1- A detailed description of what the system should do.
2- It is a statement in a more formal notation which sets out the system services in more details.
3- It is a structured document which is sometimes called a ***functional specification***, should be precise. It may serve as a contract between the system buyer and software developer.
4- It is usually presented with the systems models developed during requirements analysis.
5- It should include all necessary information about what the system must do and all constraints on its operation.
6- It is restates the requirements in technical terms appropriate for the development of the system design.
7- Should be written by the requirements analysts.

Software specification: is an abstract description of the software which is a basis for design and implemented this specification may add further detail to the requirements specification. It is a document for the design team rather than the system customer.

Requirements engineering (RE)

- 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.

The requirements engineering Process

This process is the set of activities that lead to the production of the requirements definition and requirements specification, as shown in Figure **3.1**.

There are four principal stages in this process:

1- Feasibility study

- A feasibility study decides whether or not the proposed system is worthwhile.
- A short focused study that checks:

- If the system contributes to organisational objectives;
- If the system can be engineered using current technology and within budget;
- If the system can be integrated with other systems that are used.

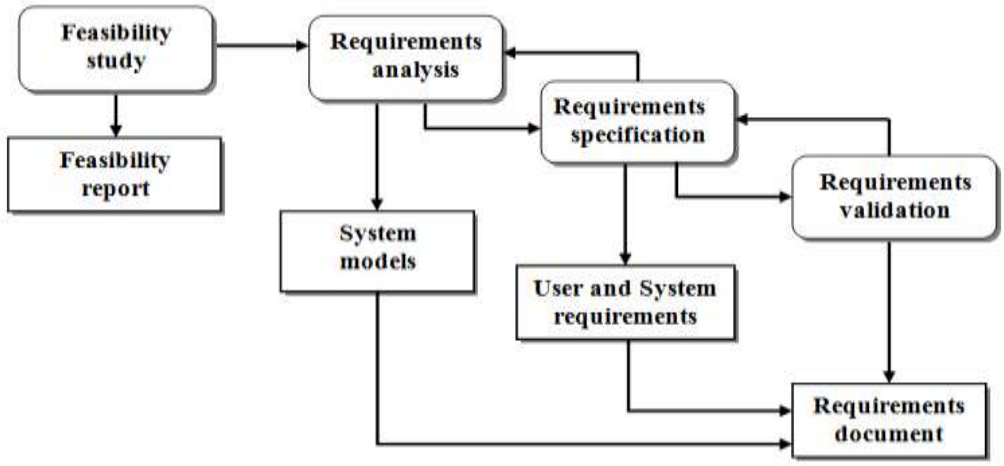

Figure 3.1: The requirements engineering process.

Feasibility study implementation

- Based on information assessment (what is required), information collection and report writing.
- Questions for people in the organisation :
  - What if the system wasn't implemented?
  - What are current process problems?
  - How will the proposed system help?
  - What will be the integration problems?
  - Is new technology needed? What skills?
  - What facilities must be supported by the proposed system?

2- Requirements analysis

- The process of deriving the system requirements through observation of existing systems, discussions with potential users and procures, task analysis and so on.
- Help the analyst understand the system to be specified.
- System prototypes may also be developed to help understand the requirements.
- Involves technical staff working (e.g. end-users, managers, engineers involved in maintenance, domain experts) with customers to find out about the application domain, the services that the system should provide and the system's operational constraints.

3- Requirements specification

- **Definition**
  - Is the activity of translating the information gathered during the analysis activity into a document that defines a set of requirements.
  - These should accurately reflect what the customer wants.
  - This document must be written so that it can be understood by the end-user and the system customer.

➢ Specification

- A detailed and precise description of the system requirements is set out to act as a basis for a contract between client/customer and software developer.
- The creation of this document is usually carried out in parallel with some high-level design.
- During the creation of this document, errors in the requirements definition are discovered. It must be modified to correct these problems.

Note: The requirement analysis continues during definition and specification and new requirements are change and should be placed under control of a configuration management system.

4- Requirements validation

- Concerned with demonstrating that the requirements define the system that the customer really wants.
- Requirements error costs are high so validation is very important (fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error).

Requirements checking

- **Validity Checks:** Does the system provide the functions which best support the customer's needs?
- **Consistency Checks:** Are there any requirements conflicts?
- **Completeness checks:** Are all functions required by the customer included?
- **Realism Checks:** Can the requirements be implemented given available budget and technology?
- **Verifiability Checks:** Can the requirements be checked?

Requirements validation techniques

1- Requirements reviews

- Systematic manual analysis of the requirements.
- Regular reviews should be held while the requirements definition is being formulated.
- Both client and contractor staff should be involved in reviews.
- Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage.

2- Prototyping

- Using an executable model of the system to check requirements.

3- Test-case generation

- Requirements should be testable.

- If a test is difficult or impossible to design, this usually means that the requirement will be difficult to implement and should be reconsidered.

Requirements management

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development.

- Requirements are incomplete and inconsistent:

- New requirements emerge during the process as business needs change and a better understanding of the system is developed;
- Different viewpoints have different requirements and these are often contradictory.

Stable requirements: It is derived from the core activity of the customer organisation (e.g. a hospital will always have doctors, nurses, etc), may be derived from domain models.

Volatile requirements: Requirements which change during development or when the system is in use (e.g. in a hospital, requirements derived from health-care policy).

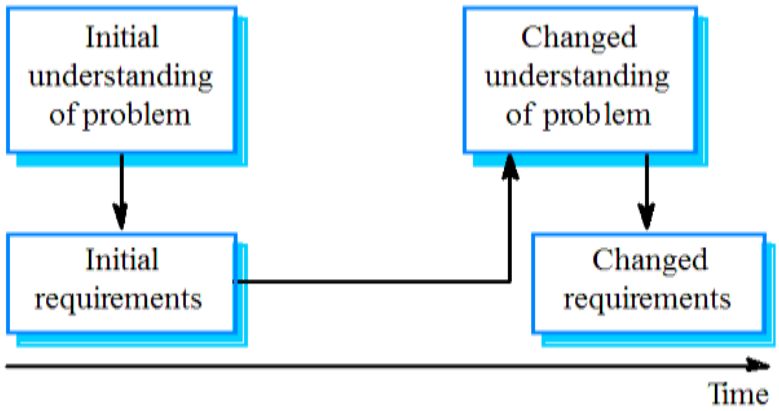

Figure 3.2: The requirements evolution process.

Requirements management planning

- During the requirements engineering process, you have to plan:

- **Requirements identification** (how requirements are individually identified);
- **A change management process** (the process followed when analysing a requirements change);
- **Traceability policies** (the amount of information about requirements relationships that is maintained);
- **CASE tool support** (the tool support required to help manage requirements change).

Requirements change management

- Should apply to all proposed changes to the requirements.

- **Principal stages:**
  1- **Problem analysis** (Discuss requirements problem and propose change);

  2- **Change analysis and costing** (Assess effects of change on other requirements);

  3- **Change implementation** (Modify requirements document and other documents to reflect change).

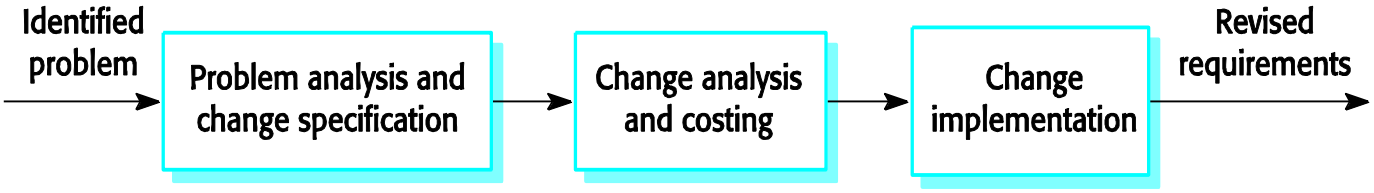

Figure 3.3: The requirements change management.

Functional & Non-Functional Requirements

System requirements may be either ***Functional*** or ***Non-functional*** requirements**.**

1- Functional requirements

- Statements of services the system should provide,
- How the system should react to particular inputs,
- How the system should behave in particular situations.

2- Non-functional requirements

- Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.

3- Domain requirements

- Requirements that come from the application domain of the system and that reflect characteristics of that domain.
- They may be functional or non-functional requirements.

1- Functional Requirements (FR)

Are statements of services the system should provide, how the system should react to particular inputs, and how the system should behave in particular situation.

The major attributes of FR are:

1- Describe functionality or the system services (Functional system requirements should describe the system services in detail).
2- Describe an interaction between the system and its environment.
3- Depend on the type of software, expected users and the type of system where the software is used.
4- **FR** are independent of the implementation of a solution, they do not place constraints on how the solution will be developed.
5- Functional user requirements may be high-level statements of what the system should.
6- Problems arise whenever functional requirements are not precisely stated (**Ambiguous Requirements*).***
7- Ambiguous requirements may be interpreted in different ways by developers and users.

In principle, the functional requirements should be both complete and consistent.

- ***Completeness:*** all services required by the user should be defined and should be includes descriptions of all facilities required.
- ***Consistency:*** there should be no conflicts in the descriptions of the system facilities.

In practice, for large systems, it is mostly impossible to produce a complete and Consistent requirements document because of overall complexity and varying views.

2- Non-Functional Requirements (NFR)

Are constraints on the services or functions offered by the system such as timing constraints
on the development process, standards, etc.

The major attributes of NFR are:

1- Define system properties and constraints, e.g. of system properties are performance, reliability, response time and storage requirements. Constraints are I /O device capability, system representations, etc.

2- Process requirements may also be specified mandating a particular CASE toolset, programming language and a description of the process model which should be followed.

Customers impose these process requirements for two reasons:

- ***System quality:*** In general, a good process leads to a good product.
- ***System maintainability:*** Process requirements may be imposed so that the development methods used to design and implement the system are compatible with those to be used for system maintenance.

3- **NFR** may be more critical than functional requirements. If non-functional requirements are not met, the system is useless.

4- **NFR** arise through user needs, because of budget constraints, organizational policies, safety, and privacy and so on.

Types of Non-Functional requirements:

NFR can be classified depending on how they have been derived:

A- Product requirements: These are requirements which specify that the delivered product must behave in a particular way, e.g. execution speed; reliability; portability and usability requirements. This type of NFR may be derived directly from user needs.

B- ***Organizational requirements:*** These are requirements which are a consequence of organizational policies and procedures, e.g. Process standards used, Implementation requirements such as programming language or design method used, etc. This type of NFR may be derived from both the customer's and developer's organization.

C- ***External requirements:*** requirements arising from factors that are external to the system and its development process, such as examples :

interoperability requirements which define how the system interacts with systems in order organizations; ***legislative*** requirements which must be followed to ensure that the system operates within the law; and ***ethical*** requirements which may be placed on the system to ensure that it will be acceptable to its users and the general public.

The following **Figure 4.1** shows different types of Non-functional requirements.

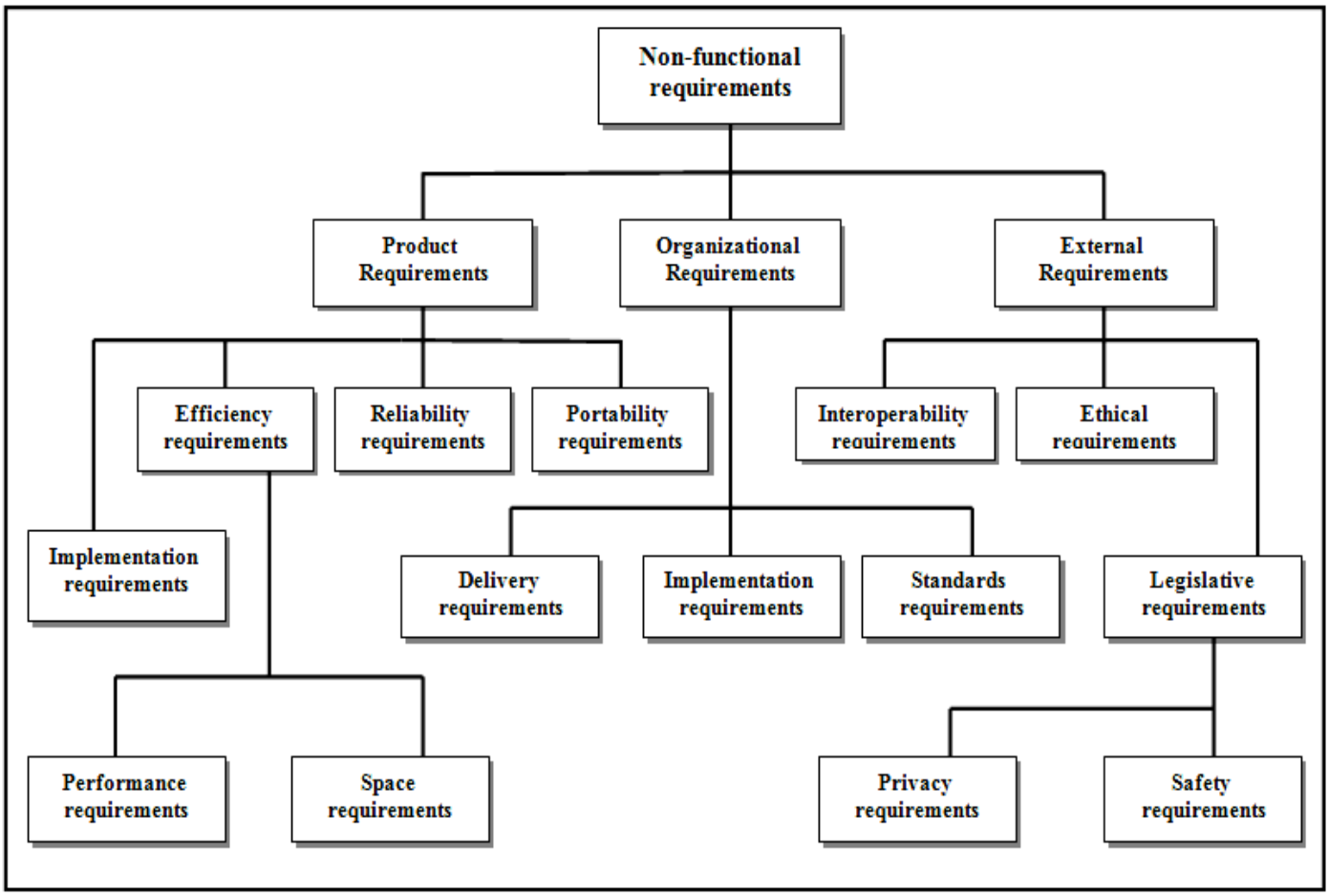

Figure 4.1: Types of Non-functional requirements.

- **NFR** may be very difficult to state precisely but imprecise requirements are routinely
  hard to verify.

- The best way to ensure that NFR are verifiable is to express them quantitatively based on a ***measure*** or ***metric*** that can be objectively tested.

Table 4.1 shows a number of possible measures that may be used to specify non-functional system properties.

- **The system goals** are helpful to developers as they convey the intentions of the system users. The system should be easy to use by experienced controllers and be organized in such a way that user errors are minimized.

- Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day.

- Conflicts between different non-functional requirements are common in complex systems.

- Optimizations along the lines of several different goals might contradict each other; need to priorities goals!

Table 4.1: Metrics for specifying NFR properties.

Property	Measure
Speed	Processed transactions/second User / Event response time Screen refresh time
Size	K Bytes Number of RAM chips
Ease of use	Training time Number of help frames
Reliability	Mean time of 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

Software design process

It is the process of implementing software solutions to one or more set of problems. One of the important parts of software design is the software requirements analysis (SRA). It is a part of the software development process that lists specifications used in software engineering. Software design includes both low-level component and algorithm design and high level, architecture design as shown in Figure (5.1).

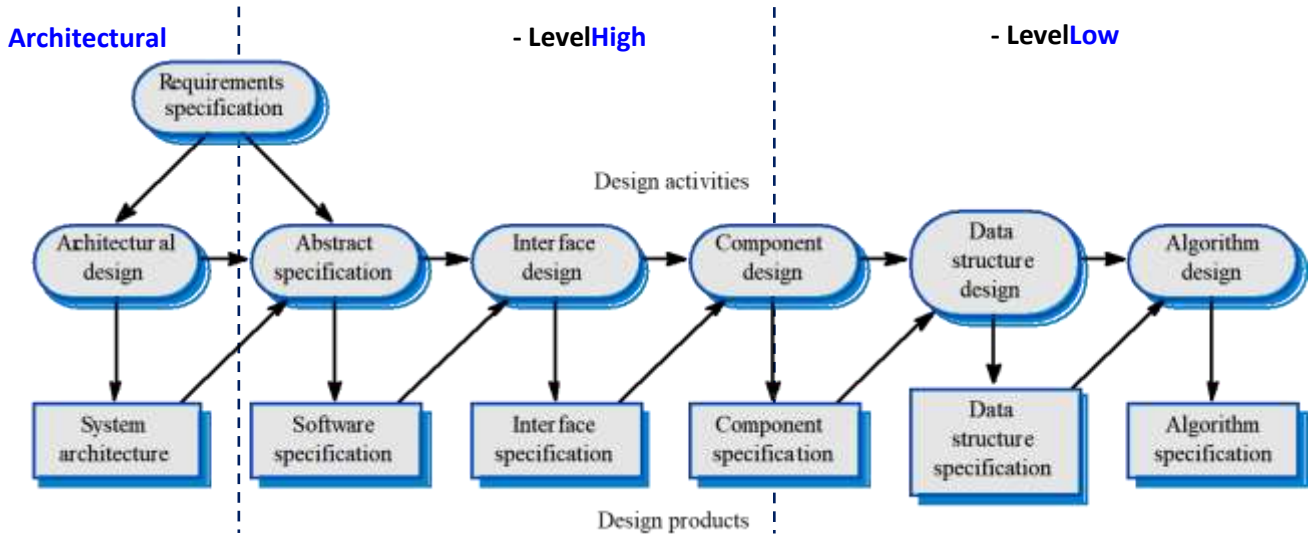

Figure 5.1: The typical Phases of design process.

- **Architectural design:** Identify sub-systems
- **Abstract specification:** Specify sub-systems
- **Interface design:** Describe sub-system interfaces
- **Component design:** Decompose sub-systems into components
- **Data structure design:** Design data structures to hold problem data
- **Algorithm design:** Design algorithms for problem functions

Software design can be considered as creating a solution to a problem in hand with available capabilities. The main difference between ***software analysis*** and ***design*** is that the output of a software analysis consists of smaller problems to solve. Also, the analysis should not be very different even if it is designed by different team members or groups.

The design focuses on the capabilities, and there can be multiple designs for the same problem depending on the environment that solution will be hosted. Sometimes the design depends on the environment that it was developed for, whether it is created from reliable frameworks or implemented with suitable design patterns. When designing software, **two important factors to consider are its *Security* and *Usability***.

Requirements and Design

- In principle, requirements should state ***what*** the system is to do and the design should describe ***who*** it accomplishes this.
- *In practice, requirements and design are inseparable; because:*

1- System architecture may be designed to structure the requirements.
2- The system may inter-operate with other systems that generate design requirements.
3- The use of a specific design may be a domain requirement.

The Software Requirements Document (SRD)

- SRD is the official statement of what is required of the system developers.
- Should include both the requirements **definition** and **specification**.
- It is not a design document. It should set out what the system is to do rather than how it should accomplish this.

There are six requirements which SRD should satisfy:

1- Specify external system behavior.
2- Specify implementation constraints.
3- Needs to be easy to change.
4- Serve as reference tool for maintenance.
5- Recode forethought about the life cycle of the system, i.e. predict change.
6- Characterize acceptable responses to unexpected events.

There are different users of the SRD ,such as:

1. **System customers:** Specify the requirements and read them to check that they meet their needs. They specify changes to the requirements.
2. **Managers:** Use the requirements document to plan a bid for the system and to plan the System development process.
3. **System engineers:** Use the requirements to understand what system is to be developed.
4. **System test engineers:** Use the requirements to develop validation tests for the system.
5. **System maintenance engineers:** Use the requirements to help understand the system and the relationship between its parts.

The best organization of SRD is as a series of chapters with the detailed specification perhaps presented as an appendix to the document.

A generic structure for SRD is shown in the following Table 5.1.

Table 5.1: The structure of a software requirement document.

Chapter	Description
Introduction	Describe the system functions and explain how it will work with other systems. It should describe objectives of the organization commissioning the software.
Glossary	Define the technical terms used in the document. No assumptions should be made about the experience of the reader.
System models	Should set out one or more system models showing the relationships between the system components and the system and its environments. These might include object and data-flow models.
Functional Requirements definition	Should be described the services provided for the user, may use natural language, diagrams or other notations that are understandable by customers.
Non-functional requirements definition	Describe the constraints that imposed on the software and the restrictions on the freedom of the designer. This might include details of specific data representation, response time and so on.
System evolution	Should describe the fundamental assumptions on which the System is based and anticipated changes due to HW evolution, Changing user needs, and so on.
Requirements specification	Should describe the functional requirements in more detail. If necessary , further detail may also be added to the non-functional requirements, for example interfaces to other systems may be defined.

- **There are three major problems with *Requirements Definitions Document* (RDD) written in natural language (NL):**
  1. **Lake of clarity:** It is very difficult to use language in a precise and unambiguous way without making the document difficult to read.
  2. **Requirements confusion:** Functional requirements, non-functional requirements, system goals and design information may not be clearly distinguished.
  3. **Requirements amalgamation:** Several different requirements may be expressed together as a single requirement.

- **There are three major problems with *Requirements Specification Document* (RSD) written in natural language (NL):**
  1. **Ambiguity:** The readers and writers of the requirement must interpret the same words in the same way. Natural language (NL) is inherently ambiguous, so this is very difficult.
  2. **Over-flexibility:** The same thing may be said in a number of different ways in the specification.
  3. **Lack of modularization:** NL structures are inadequate to structure system requirements.

Software Development Life Cycle (SDLC):

SDLC is a series of **phases** that provide a common understanding of the software building process. How the software will be realized and developed from the business understanding and requirements elicitation phase to convert these business ideas and requirements into functions and features until achieve the customer needs.

The good software engineer should have enough knowledge on **how** to choose the SDLC model based on the project context and the business requirements.

Therefore, it may be required to choose the right SDLC model according to the specific concerns and requirements of the project to ensure its success.

Software Process Models

What is a process model?

A process model is a simplified representation of a software process. It is a set of ordered tasks, involving activities, constraints; and Resources.

- The process of building a software product, called a **software life-cycle**.
- ***Life cycle -*** describes the life of the software from conception through its implementation, delivery, use and maintenance.

Why we need Software process?

1- Common understanding of the activities, resources and constraints involved in software development.

2- Creating processes helps to find inconsistencies, Redundancies; and Omissions.

Software Process Models :

A. Waterfall model
B. V- Shaped model
C. Evolutionary Prototyping Model
D. Iterative and Incremental Model
E. Spiral Model (e.g. Reuse-based development)
F. Agile development

A- The Waterfall Model

The waterfall model is the classic lifecycle model – it is widely known, understood and commonly used. Is the first process model to be introduced and followed widely in Software Engineering to ensure success of the project.

- The whole **SDLC** is divided into separate process phases, e.g., "***requirement phase***", "***design phase***", etc. (see Figure 6.1). These phases correspond to the four stages of the fundamental software process activities (as explained in ***Lecture-1***).

- A phase takes place in sequence to another.
- Each activity is completed before the next starts.

In theory:

- Each phase produces documents that are: ***Verified*** and ***validated***; Assumed to be ***complete***.
- Each phase depends on the document of the previous stage to proceed → it has to wait for the completion of previous stage.

In practice:

- The phases overlap and feedback to each other (see the feedback arrow in the diagram).

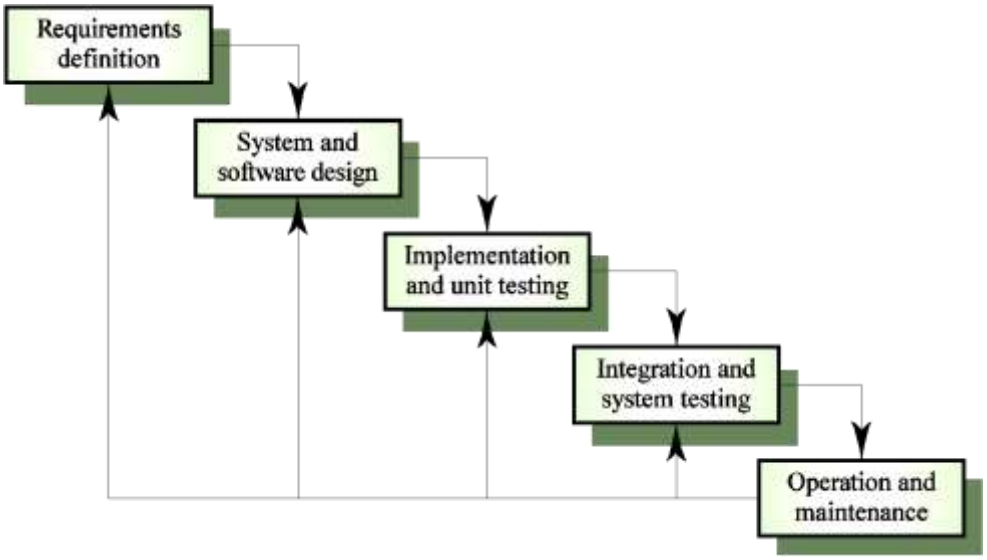

Figure 6.1: The waterfall model lifecycle.

Waterfall Model Phases:

1- Requirements analysis and definition

- All possible requirements of the system to be developed are captured in this phase.
- Requirements then are analyzed for their validity and the possibility of incorporating the requirements in the system to be development is also studied.
- Finally, a ***Requirement Specification*** document is created which serves the purpose of guideline for the next phase of the model.

2- System and software design

- System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.
- The system design specifications serve as input for the next phase of the model.

3- Implementation and unit testing

- On receiving system design documents, the work is divided in modules/units and actual coding is started.
- The system is first developed in small programs called units, which are integrated in the next phase.
- Each unit is developed and tested for its functionality referred to as Unit Testing.
- Unit testing mainly verifies if the modules/units meet their specifications.

4- Integration and system testing

- The units are integrated into a complete system during Integration phase and tested to check if all units/modules coordinate between each other and the system as a whole behaves as per the specifications.
- After successfully testing the software, it is delivered to the customer.

5- Operation and maintenance

- This phase of "The Waterfall Model" is virtually never ending phase (Very long).
- Generally, problems with the system developed (which are not found during the development life cycle) come up after its practical use starts, so the issues related to the system are solved after deployment of the system.
- Not all the problems come in picture directly but they arise time to time and needs to be solved- referred as Maintenance.

Features of a Waterfall Model

1. A waterfall model is easy to follow. It can be implemented for any size project.
2. Process stages can be iterative. Every stage has to be done separately at the right time so you cannot jump stages.
3. Early and frequent validation of software system. Testing is done at every stage.
4. Documentation is produced at every stage of a waterfall model allowing people to understand what has been done.
5. Works well when quality is more important than cost or schedule.

Waterfall Model Advantages:

1. Easy to explain to the users.
2. Structures approach.
3. Stages and activities are well defined.
4. Helps to plan and schedule the project.
5. Verification at each stage ensures early detection of errors/misunderstanding.
6. Each phase has specific deliverables.

Waterfall Model Disadvantages:

1. If requirements may change, the Waterfall model may not work.
2. Very difficult to go back to any stage after it finished.
3. Is only appropriate when the requirements are well-understood; and changes will be fairly limited during the design process.
4. Difficult to estimate time and cost for each stage of the development process.
5. Constant testing of the design is needed.

When to use the Waterfall Model

- Requirements are very well known.
- Product definition is stable.
- Technology is understood.
- New version of an existing product.
- Porting an existing product to a new platform.

B- V-Shaped Model

It is an extension of the waterfall model that emphasizes the verification and validation of the product. Testing of the product is planned in parallel with a corresponding phase of development to form the typical V - shape as shown in the following Figure (7.1).

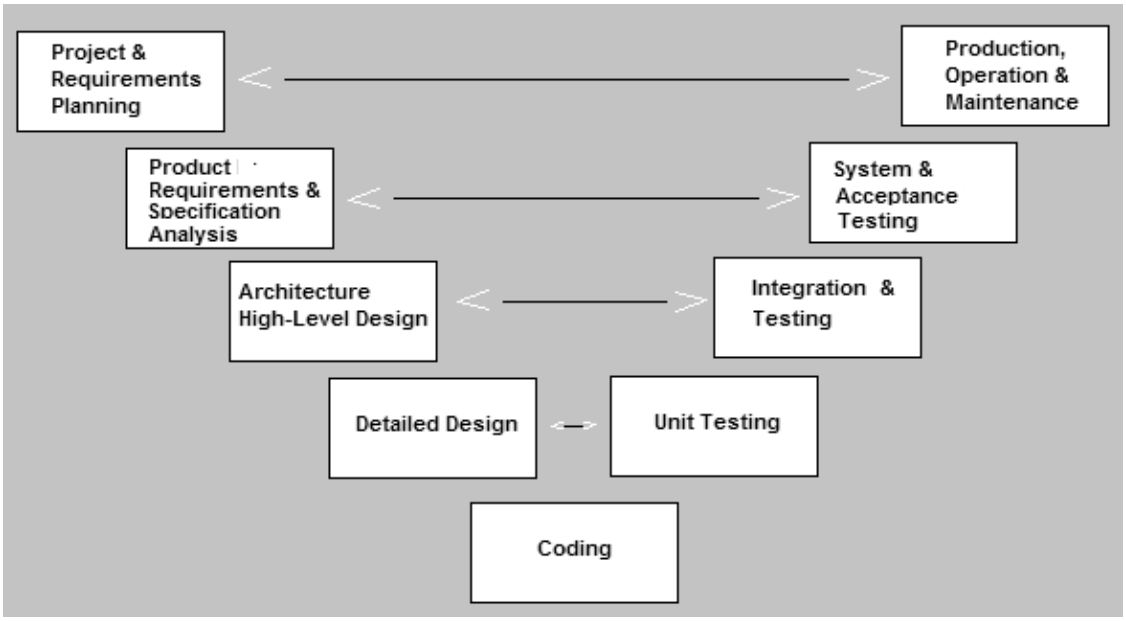

Figure 7.1: The V-Shaped model lifecycle.

V-Shaped Steps:

1- **Project and Requirements Planning** – allocate resources

2- **Product Requirements and Specification Analysis** – complete specification of the software system

3- **Architecture or High-Level Design** – defines how software functions fulfill the design

4- **Detailed Design** – develop algorithms for each architectural component

5- **Production, operation and maintenance** – provide for enhancement and corrections

6- **System and acceptance testing** – check the entire software system in its environment

7- **Integration and Testing** – check that modules interconnect correctly

8- **Unit testing** – check that each module acts as expected

9- **Coding** – transform algorithms into software

V-Shaped Model Advantages:

1. Simple and Easy to use.
2. Works well for where requirements are easily understood.
3. Verification and validation of the product in the early stages of product development.
4. Higher chance of success over the waterfall model due to the development of test plans early on during the life cycle.
5. Each phase has specific deliverable and must be testable.
6. Project management can track progress by milestones.

V-Shaped Model Disadvantages:

1. Very inflexible, like the waterfall model.
2. Does not easily handle concurrent events or iterations (i.e. phases).
3. The software is developed during the implementation phase, so no early prototypes of the
   software are produced.
4. Does not easily handle dynamic changes in requirements.
5. The model doesn't provide a clear path for problems found during testing phases.
6. Costly and required more time, in addition to a detailed plan.

When to use the V-Shaped Model

- Requirements are well defined, clearly documented and fixed.
- Product definition is stable.
- Technology is not dynamic and is well understood by the project team.
- There are no ambiguous or undefined requirements.
- The project is short.

C- Evolutionary development model

- Evolves an initial implementation with user feedback → ***multiple versions until the Final version*** (**see Figure 8.1**).

- This model is based on the idea of rapidly developing initial software implementation from very abstract specifications.

- Each program version **inherits** the best features from earlier versions. Each version is refined based upon **feedback** from the user to produce a system which satisfies the customer needs. At this point, the system may be **delivered** or it may be **re-implemented** using a more structured approach to enhance **robustness** and **maintainability**.

- ***Specification***, ***development*** and ***validation*** activities are concurrent with strong feedback between each.

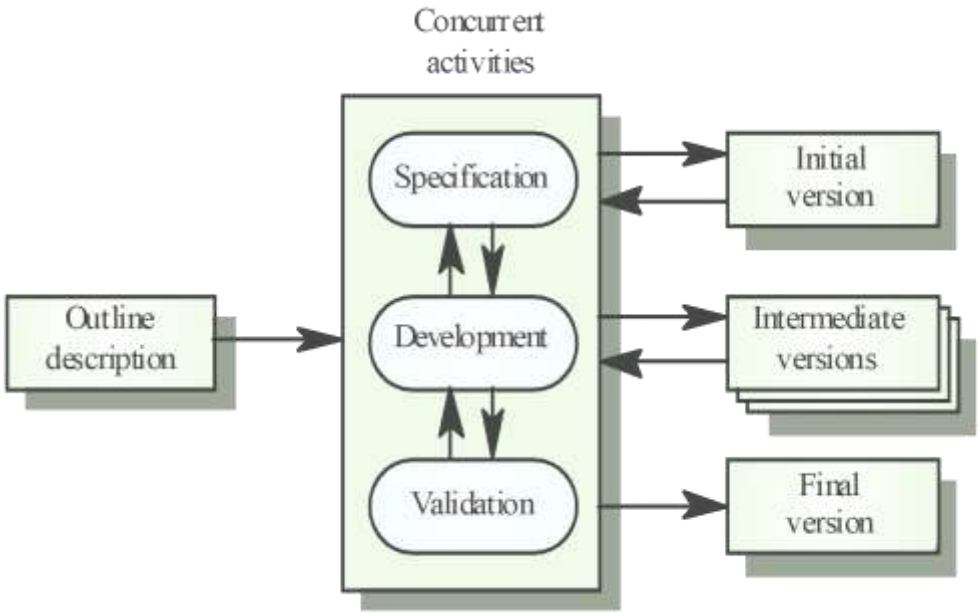

Figure 8.1: The Evolutionary Development Life Cycle.

➢ Two fundamental types:

1. Evolutionary Prototyping:

- Objective is to work with customers and to evolve a final system from an initial outline specification.

- Should starts with well-understood requirements and add new features as proposed by the customer (iterative incorporation of user feedback) as shown in **Figure 8.2.**

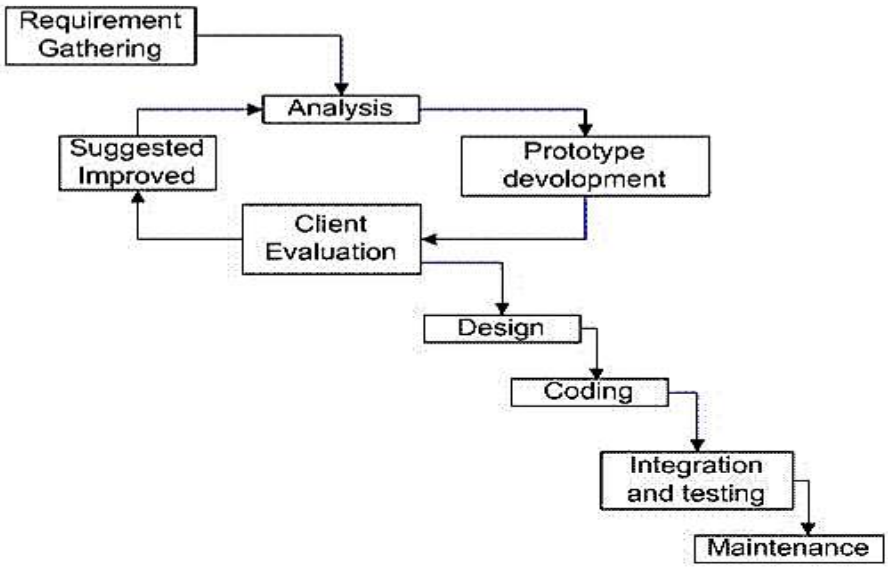

Figure 8.2: Evolutionary Prototyping Model.

2. Throw-away Prototyping:

- Objective is to understand the system requirements and develop a better requirement definition for the system.
- Should starts with poorly understood requirements to clarify what is really needed.
- **Prototypes** that are eventually discarded rather than becoming a part of the finally delivered software. (see Figure 8.3)

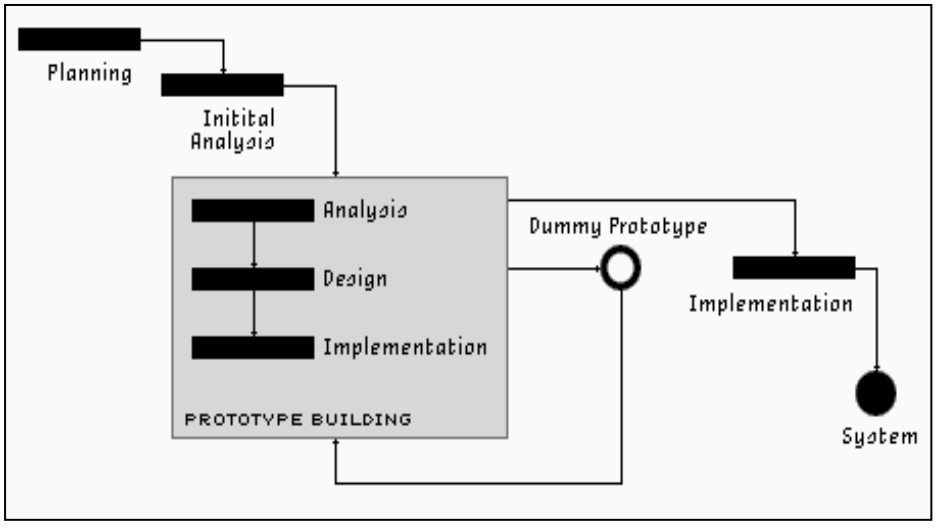

Figure 8.3: Throw-away Prototyping Model.

Advantages of Evolutionary Model:

1. Customer involvement in the process to meet the user requirement.
2. Improved and increased user involvement.
3. Reduced time and costs, but this can be a disadvantage if the developer loses time in developing the prototypes.
4. Early and frequent testing to identify problems and lower risk.
5. Applicable for :
- small or medium-size interactive systems;
- parts of large systems (e.g. the user interface);

Evolutionary Model Problems:

1.Lack of process visibility.
2.It is difficult to measure progress and produce documentation reflecting every version of the system as it evolves.
3. Insufficient analysis. User confusion of prototype and finished system.
4. Developer misunderstanding of user objectives.
5. Excessive development time of the prototype.
6.Production of good quality software requires highly skilled and motivated programmers.
7. It is costly to implement the prototypes

When to use the Evolutionary model

1- It used to visualize some component of the software to limit the gap of misunderstanding the customer requirements by the development team.

2- It used when you are developing a system has user interactions.

3- It used in "short- lifetime" systems.

D- Incremental model :

A method of software development where the product is designed, implemented and tested incrementally until the product is finished. It involves both ***development*** and ***maintenance***. The product is defined as finished when it satisfies all of its requirements. **This model combines the elements of the waterfall and evolutionary model** (which are then applied in an **iterative manner**).

Incremental model lifecycle:

- The product is decomposed into a number of components, each of which is designed and built separately. Each component is delivered to the client when it is complete.

- The incremental model can applies the ***waterfall model*** incrementally as illustrated in Figure 9.1.

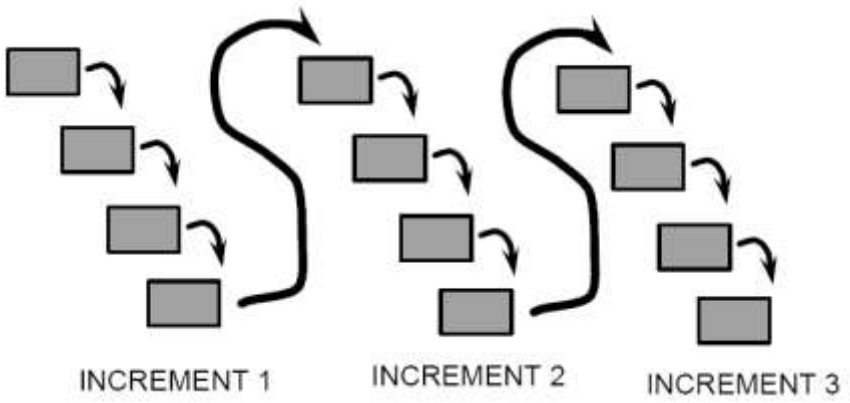

Figure 9.1: Each increment is a mini-waterfall.

- **Multiple development cycles take place here,** making the life cycle a "multi-waterfall" cycle. **Cycles** are divided up into smaller, more easily managed modules.

- **Each module passes through the requirements**, design, implementation and testing phases. A working version of software is produced during the first module, so you have working software early on during the software life cycle.

- **Each subsequent release of the module adds function to the previous release**. The process continues till the complete system is achieved (see Figure 9.2).

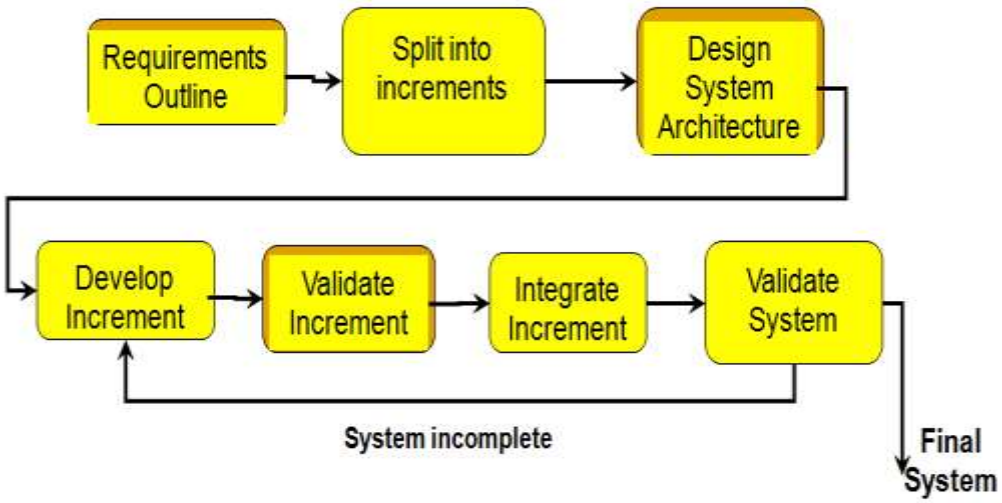

Figure 9.2: The incremental model life-cycle

Features of Incremental Model

1- Prioritizes the services to be provided by the system.
2- Maps these requirements to ***Increment*** based on priority.
3- Freezes requirement for the current *Increment.*
- Requirements for the later increments can evolve concurrently.

4- Each ***Increment*** release is a working system:
- Allows user to experiment.
- Can be put into service right away.

Advantages of Incremental Model:

1. Initial product delivery is faster with Lower cost.
2. Core product is developed first (i.e. main functionality is added in the first increment).
3. Regression testing should be conducted after each iteration. During this testing, faulty elements of the software can be quickly identified because few changes are made within any single iteration.
4. It is generally easier to test and debug than other methods of software development because relatively smaller changes are made during each iteration.
5. With each release a new feature is added to the product.
6. Customer can respond to feature and review the product.
7. Risk of changing requirement is reduced.
8. Work load is less.

Disadvantages of Incremental Model:

1. Requires good analysis.
2. Resulting cost may exceed the cost of the organization.
3. Each phase of an iteration is rigid and do not overlap each other.
4. As additional functionality is added to the product, problems may arise related to system architecture which were not evident in earlier prototypes.
5. Hard to map requirement into small increments (< 20,000 lines of code).
6. Hard to define the basic services that are shared by all subsequent increments.

When to use the Incremental model:

1- This model can be used when the requirements of the complete system are clearly defined and understood.
2- Major requirements must be defined; however, some details can evolve with time.
3- There is a need to get a product to the market early.
4- A new technology is being used.
5- Resources with needed skill set are not available.
6- There are some high risk features and goals.

E- The Spiral Model :

- The **Spiral Model** was designed to include the best features from the **Waterfall** and **Prototyping Models**, and introduces a new component - ***Risk-assessment***.

- The term **'Spiral'** is used to describe the process that is followed as the development of the system takes place. Similar to the **Prototyping Model**, an initial version of the system is developed, and then repetitively modified based on input received from customer evaluations.

-With each iteration around the spiral (beginning at the centre and working outward).

Risk assessment is included as a step in the development process as a means of evaluating each version of the system to determine whether or not development should continue.
If the customer decides that any identified risks are too great, the project may be halted.

For example, if a substantial increase in cost or project completion time is identified during one phase of risk assessment, the customer or the developer may decide that it does not make sense to continue with the project, since the increased cost or lengthened timeframe may make continuation of the project impractical or unfeasible.

The lifecycle of Spiral Model is including the following steps (see **Figure 10.1**):

1. **Project Objectives:** Specific objectives for the phase are identified**.** Similar to the system conception phase of the **Waterfall Model**. Objectives are determined, possible obstacles are identified and alternative approaches are weighed.

2. **Risk Assessment:** Possible alternatives are examined by the developer, and associated risks /problems are identified. Resolutions of the risks are evaluated and weighed in the consideration of project continuation. Sometimes prototyping is used to clarify needs.

3. **Engineering & Production:** Detailed requirements are determined and the software piece is developed.

4. **Planning and Management:** The customer is given an opportunity to analyze the results of the version created in the Engineering step and to offer feedback to the developer.

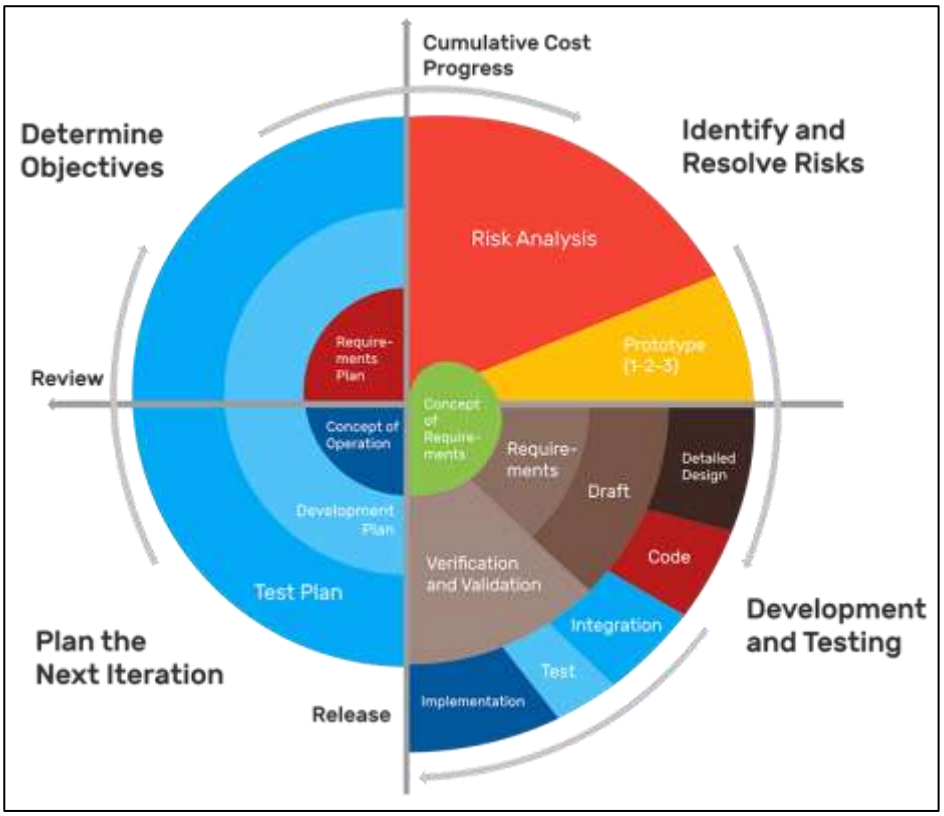

Figure 10.1: The Spiral model life-cycle

Each cycle contains the following tasks:

1. Determining objectives.
2. Specifying constraints.
3. Generating alternatives.
4. Identifying risks.
5. Resolving risks.
6. Developing next-level product.
7. Planning next cycle.

Spiral Model Advantages:

1. **Realism:** the model accurately reflects the iterative nature of software development on projects with unclear requirements.
2. **Flexible:** incorporates the advantages of the waterfal and rapid prototyping methods.
3. **Comprehensive** model decreases risk. Critical high-risk functions are developed first.
4. Users see the system early because of rapid prototyping tools.
5. Users can be closely tied to all lifecycle steps.
6. Early and frequent feedback from users.

7. Good project **visibility**.

Spiral Model Weaknesses

1. Time spent for evaluating risks too large for small or low-risk projects.
2. Time spent planning, resetting objectives, doing risk analysis and prototyping may be excessive.
3. The model is poorly understood by non-technical management, hence not so widely used.
4. Needs technical expertise in risk analysis to really work.
5. Spiral may continue indefinitely.
6. Developers must be reassigned during non-development phase activities.
7. Complicated model, needs professional management. High administrative overhead.
8. May be hard to define objective, verifiable milestones that indicate readiness to proceed through the next iteration.

When to use Spiral Model?

1. The project is large.
2. The requirements are unclear and complex.
3. When changes may require at any time.
4. Large and high budget projects.