Chapter 1

Software Process and Requirements

[Prepared by Amir GC and shishir Rai]

1.1 Software crisis

Software crisis is the production of failed and challenged software as a result of

- Introduction of powerful computer hardware
- When larger and complex software were ordered
- Software built is over budget, late delivery unreliable, difficult to maintain properly.

It became clear that individual approaches to program development did not scale up to large and complex system. This term was purposed in 1968.

1.2 Software characteristics

To gain an understanding of software, it is important to examine the characteristics of software. Software is a logical rather than a system element.

Main characteristics of software are:

2 Software is developed and engineered; it is not manufactures in the classical sense.
3 Software doesn't wear out: i.e. it is maintainable with the introduction of new hardware.
4 A software component should be designed and implemented so that it can be reused in many different programs.
5 Software should have all required functionality and performance for user.

1.3 Software quality attributes

A. Runtime System Qualities

Runtime System Qualities can be measured as the system executes.

Functionality: the ability of the system to do the work for which it was intended.

Performance: the response time, utilization, and throughput behavior of the system. Not to be confused with human performance or system delivery time.

Security: a measure of system's ability to resist unauthorized attempts at usage or behavior modification, while still providing service to legitimate users.

Availability (Reliability quality attributes falls under this category): the measure of time that the system is up and running correctly; the length of time between failures and the length of time needed to resume operation after a failure.

Usability: the ease of use and of training the end users of the system. Sub qualities: learnability, efficiency, affect, helpfulness, control.

Interoperability: the ability of two or more systems to cooperate at runtime

B. Non-Runtime System Qualities

Non-Runtime System Qualities cannot be measured as the system executes.

Modifiability: the ease with which a software system can accommodate changes to its software

Portability: the ability of a system to run under different computing environments. The environment types can be either hardware or software, but is usually a combination of the two.

Reusability: the degree to which existing applications can be reused in new applications.

Integrability: the ability to make the separately developed components of the system work correctly together.

Testability: the ease with which software can be made to demonstrate its faults

1.4 Software process models

Software process model is a simplified representation of a software process. Each process model represent a process from a particular perspective, so only provide only partial information about that process.

Main process model of software are:

1. The water fall model
2. Incremental development
3. Reuse-oriented software engineering
4. Spiral model

1.4.1 The water fall model

These takes the fundamental process activities of specification, development, validation, development, ad evolution and represent them as separate process phases such as requirements specification, software design, implementation, testing, and so on.

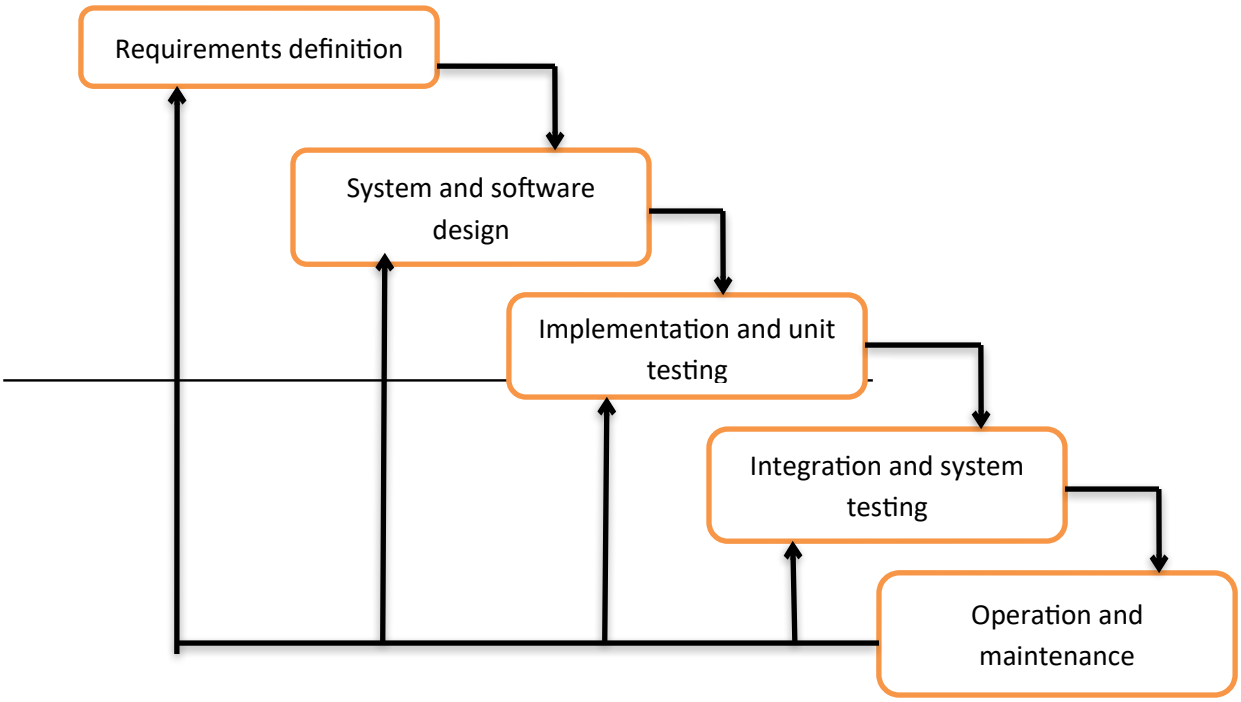


Fig 1.1 the water fall model

Because of the cascade from one phase to another is known as the water fall model. We should plan and schedule all of the process activities before starting works on them. Principle stages of the water fall model are listed below:-

i. Requirements analysis and definition:
   - The system's services constraints and goals are established by consulting with the user.
   - They are defined in detail and serve as a system specification.

ii. System and software design:
   - This process gives the requirements to either hardware or software systems by establishing an overall system architecture.
   - Involves identifying and describing the fundamental software system abstractions and their relationships.

iii. Implementing and unit testing:
   - At this stage software design is realized as a set of programs or program units.
   - Unit testing involves verifying that each unit meets its specification.

iv. Integration and system testing:
   - Individual program units or programs are integrated and tested as a complete system to ensure that software requirements have been made.

v. Operation and maintenance:
   - Longest phase
   - System is put into practical use.
   - Maintenance involves correcting errors which were not discovered earlier.

Advantages

- ✓ Reflect systematic way of software process

- ✓ Useful for larger system engineering project

Disadvantages

- ✓ Inflexible partitioning of the project into distinct stages.
- ✓ Difficult to respond to changing customs requirements.

1.4.2 Incremental development model

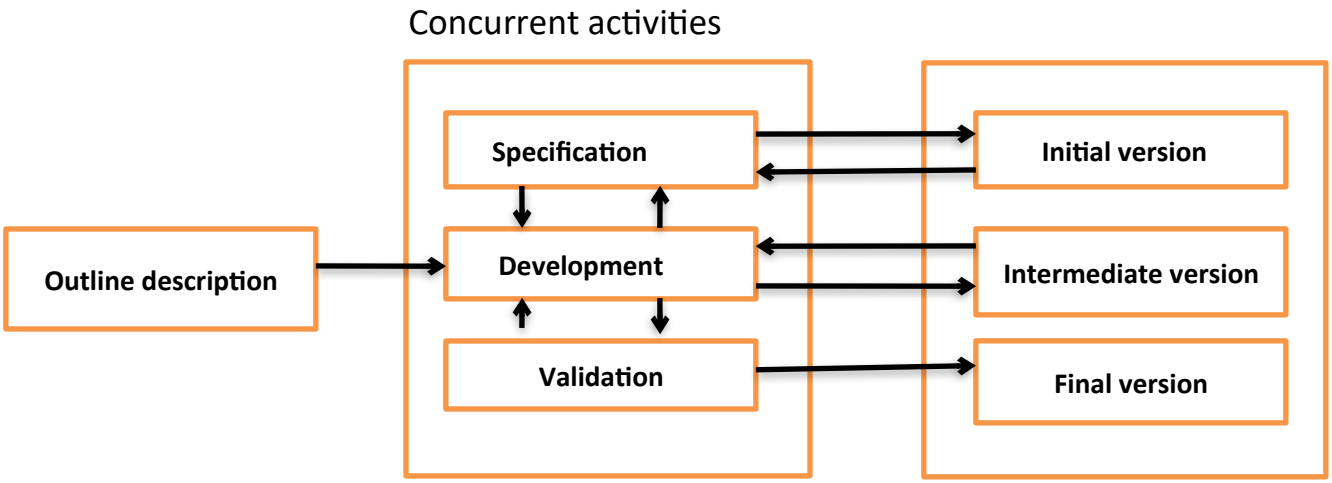


Fig 1.2 Incremental development

- ❖ Incremental development is based on the idea of developing an initial implementation, exposing this to user comment and evolving through several versions until and complete system has been developed.
- ❖ Interleaves the activity of specification, development & validation
- ❖ Developed as a series of version (increments) with each version adding functionalities to the previous one.

  Advantages

  - ✓ The cost of accommodating changing customer requirements is reduced.
  - ✓ It is easier to get customer feedback on the development work that has been done.
  - ✓ More rapid delivery and development of useful software to the customer is possible, even if all of the functionality has been included.

Disadvantages

- ✓ Hard to identify common facilities that are needed by all increments as requirement are not defined in detail at early stage.
- ✓ Difficult when replacement system is being developed as increments do not have full functionalities
- ✓ Conflicts arises with the pro current model of organization where complete system specification is part of the system development

1.4.3 Reuse oriented software engineering model

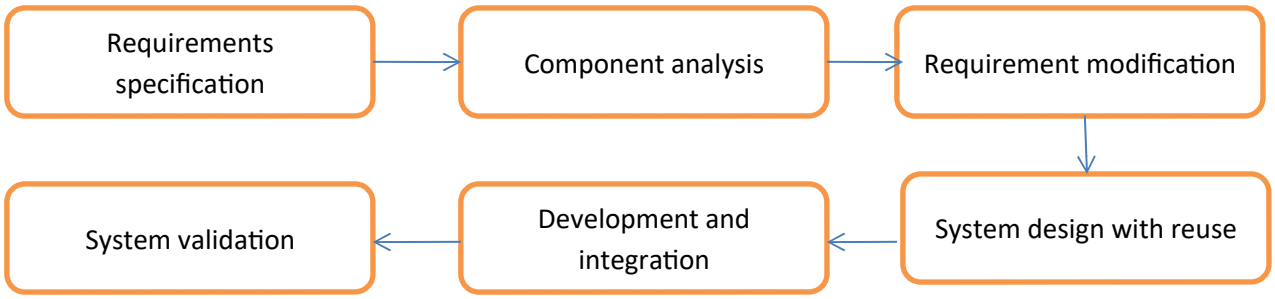


Fig 1.3 Reuse-oriented software engineering

The principle stages are

I. Requirement specification:
   - ➢ same as of water fall model

II. component analysis:
   - ➢ A search is made for components to implement that requirement specification.
   - ➢ Usually there is no exact match
   - ➢ Components which are discovered may only provide some of functionalities required.

III. Requirement modification:
   - ➢ The requirements are analyzed using the information about the components that have been discovered.
   - ➢ They are the modified to reflect the available components

IV. System design with reuse:

- Framework of the system is designed or existing frame work is reused.
- Some new software are designed if reusable components are not available.

V. Development and integration
- Software that cannot be externally procured is developed
- And the components and COTS (commercial-of-the-shelf) are integrated to create the new system.

There are three types of software component that may be reused:

a. Web services that are developed according to services and which are available for remote invocation
b. Collection of object that are developed as a package to be integrated with a component frame work such as .NET or J2EE.
c. Stand-alone software system that are configured for use in a particular environment.

Advantages

- ✓ Reduce the amount of software to be developed
- ✓ Reduces cost and risk.
- ✓ Fast delivery of the software.

Disadvantages

- ✓ May lead to a system that does not meet the real necessary of the user requirement.
- ✓ Some control over the system evolution is lost as new as new versions of the reusable components are not under the control of the organization using them.

1.4.4 Spiral model

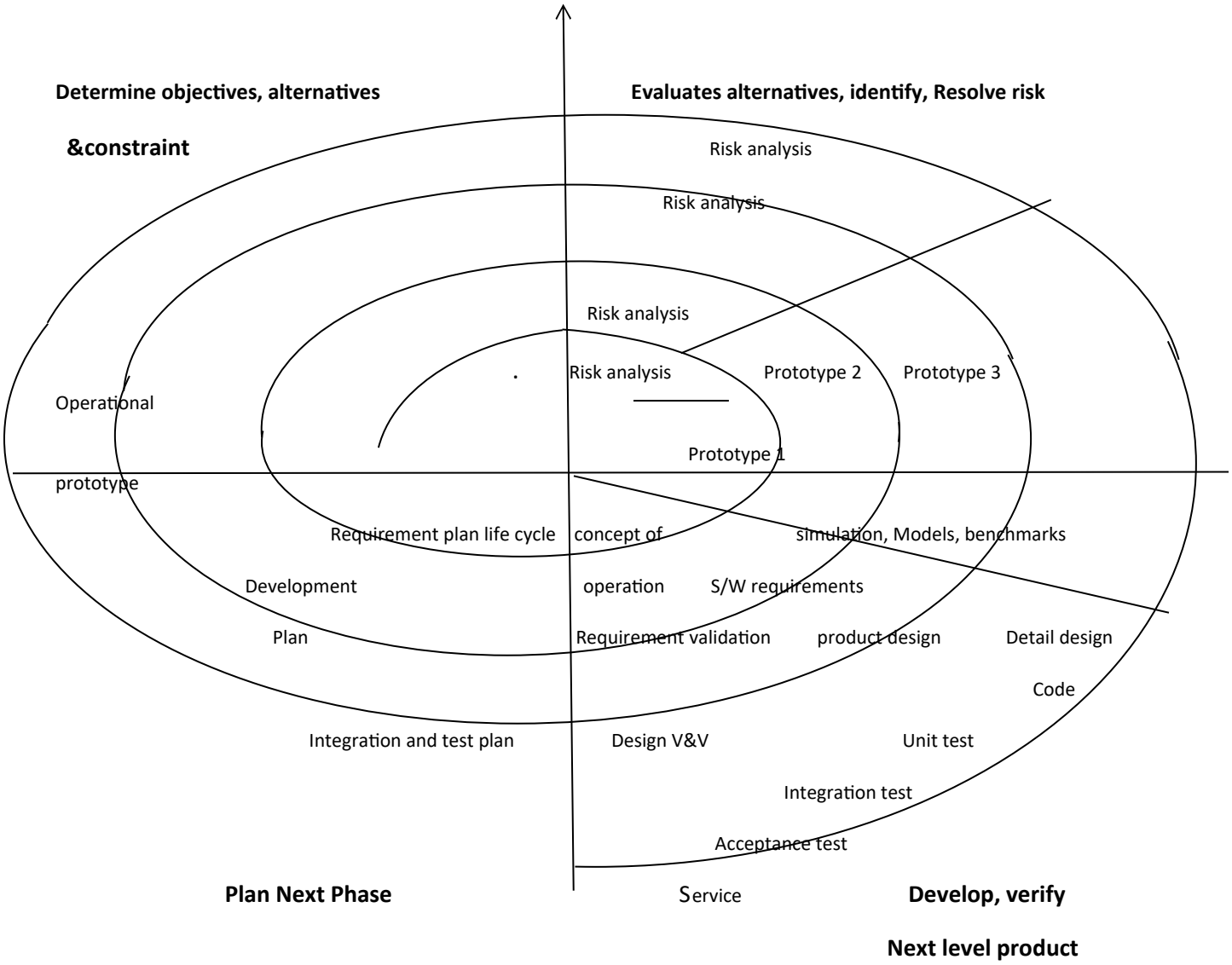


Fig 1.4 Spiral Model of Software process

In this model process is represented as spiral rather than a sequence of activities with some backtracking from one activity to another. Main principal stage of spiral model are as follow:

1. Objective setting:

- Specific object for that phase of the project are defined
- Constraints and product are identified and detail management plot is drawn up.
- Project risks are identified.
- Strategies are planned to minimize risk

2. Risk assessment and reduction:
    - For each of identified risks, a detailed analysis is carried out and step are taken to reduce them.
3. Development and validation
    - After a risk evaluation, a development model is chosen.
    - For eg . if user interface risk are dominant then prototyping may be throw away
    - If safety risk are main dominant then development based on formal transformation is chosen
    - If risk is about sub system integration ,then waterfall model is best to use
4. Planning:
    - Project is reviewed and decision is made whether to continue if it is decided then further plan for next phase are drawn up.

Advantages

- ✓ Explicit recognition of the risk.
- ✓ Flexibility to manage requirement and control changes
- ✓ Features for large business and complicated project
- ✓ Compromises both water fall model and prototype model

Disadvantages

- ✓ Not suitable for smaller project
- ✓ Not suitable for changes that happen frequently

1.5 Process iteration

The Waterfall model has dominated software development for many years, but iteration of processes is catching in. There are now a number of well-established iterative development process models that can be classified according to the levels where iteration is applied. Iteration can improve validation and verification by allowing earlier quality feedback. Moreover, there seems to be a secret marriage between teamwork and iteration. Altogether, from a SPI (software process Iteration) point of view, changing to an iterative development process model could very well raise your professional standards in software development.

- ❖ Parts of the process are repeated as system requirement evolve.
- ❖ System design & implementation work must be reworked to implement the change requirement.
- ❖ It is alternative approach to S/w development.
- ❖ Makes the system that can do all to do little more.
- ❖ Minimize the risk of building wrong product .e.g. building a table instead of chair.
- ❖ Several development process use iteration in high level or level or both.

Development process that support process iteration:

- Incremental development process
- Spiral development

- During iteration process turns of iteration should me marked strictly.
- Effective iteration means optimizing the number of turns which requires the right stop criteria.

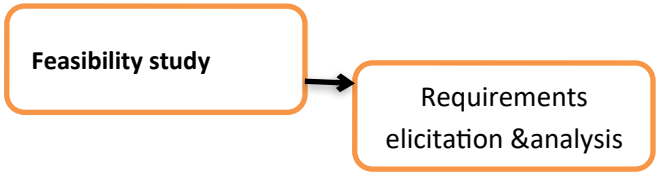
1.6 Process Activities

A software process is a set of related activities that leads to the production of a software product. This may involves the development of software from a scratch in a standard programming language like java or C. There are many different S/W processes but all must include four activities they are:

a. Software specification
b. Software design and implementation
c. Software validation
d. Software evolution

A. software specification:

Software specification or requirements engineering process of the understanding & defining what services are required from the system and identifying the constraints on the system development.



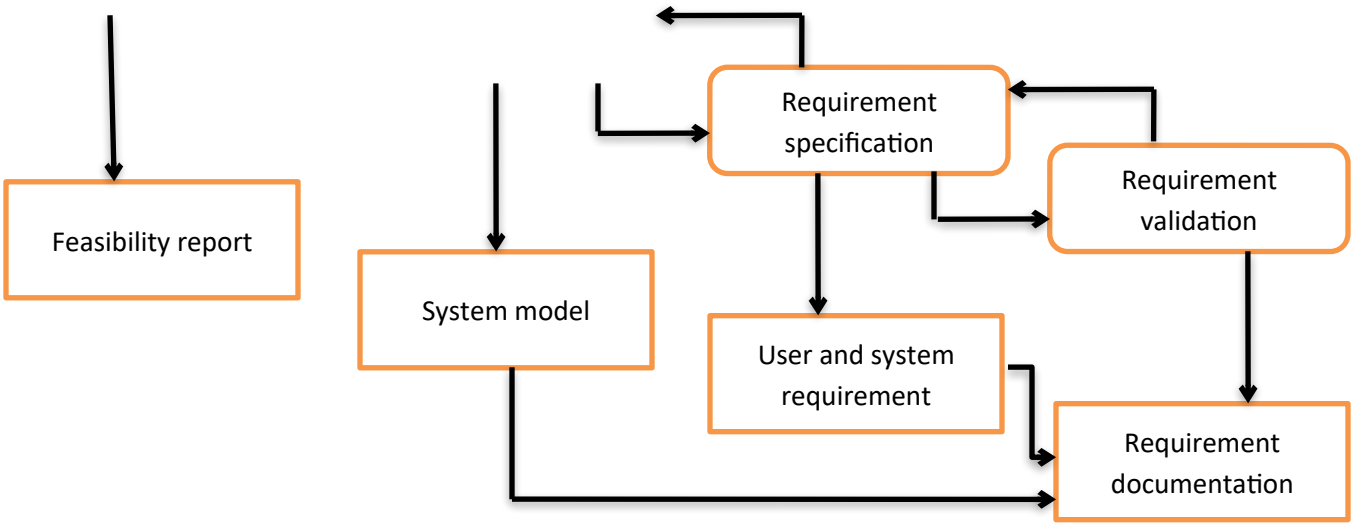


Fig 1.5The requirement engineering process

The four main activities in engineering requirement process:

- Feasibility study:
  - An estimate is made whether the user need may be satisfied using current software and hardware technologies.
  - The study also considers whether the purposed system is cost-effective from business point of view.
  - It should be quick and cheap.
  - Should provide information to decide whether or not to go ahead with more detail analysis.
- Requirement elicitation and analysis:
  - Derivation of the system requirements by observing the existing system, discussion with potential users & procurers, task analysis.
  - Involve development of one or more system models & prototype.
  - Helps us to understand the system to be specified.
- Requirement specification
  - Activity of translating the information gathered during the analysis activity into document that defines the set of requirement.
  - Two types of requirement are: i) user requirement b) system requirement

- Requirement validation
  - Checks the realism, consistency, completeness of requirements.
  - Errors in the requirements are discovered & modified to correct these problems.

B. **Software design and implementation**

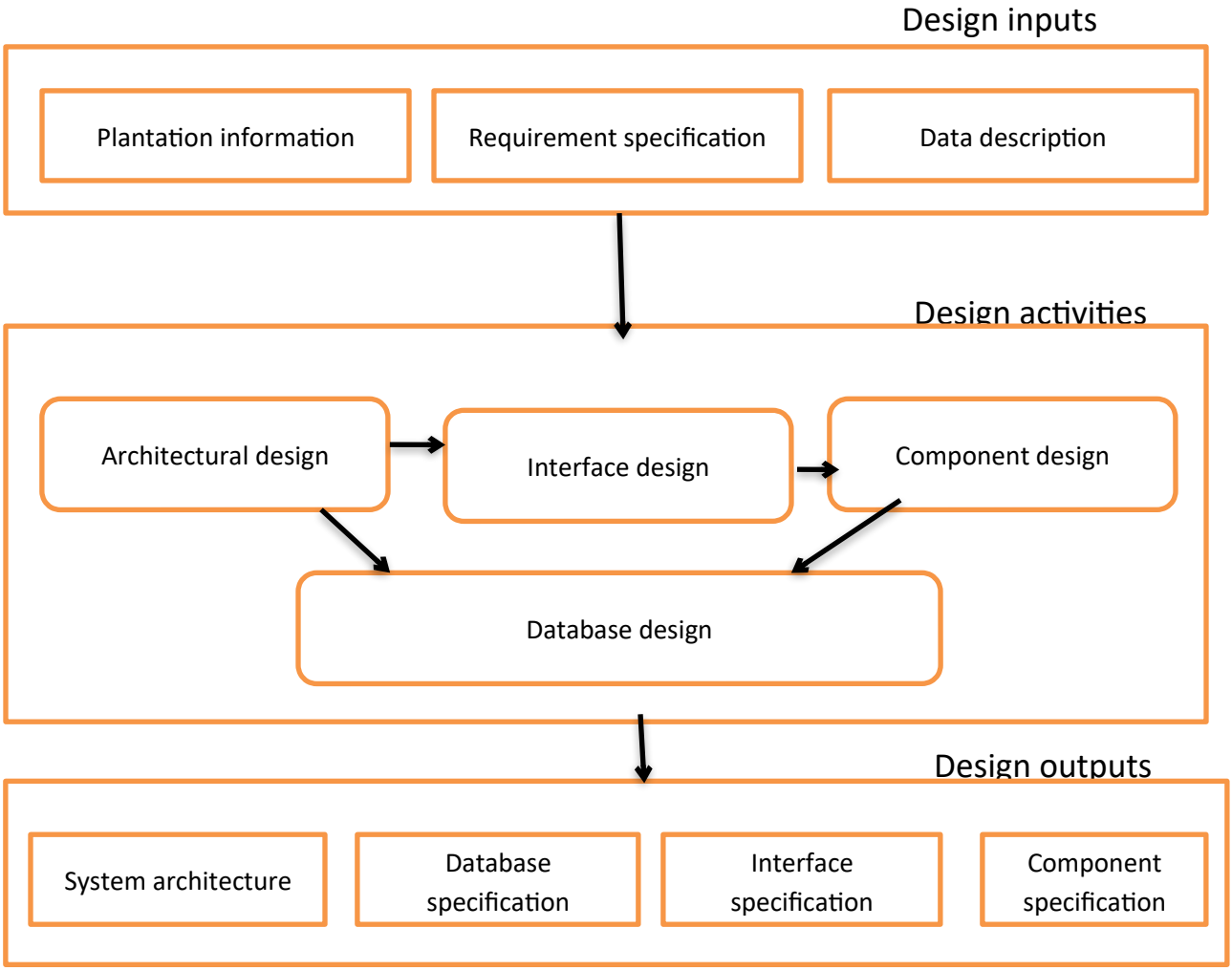


Fig 1.6 software design process

Four main activities that may be part of design process are

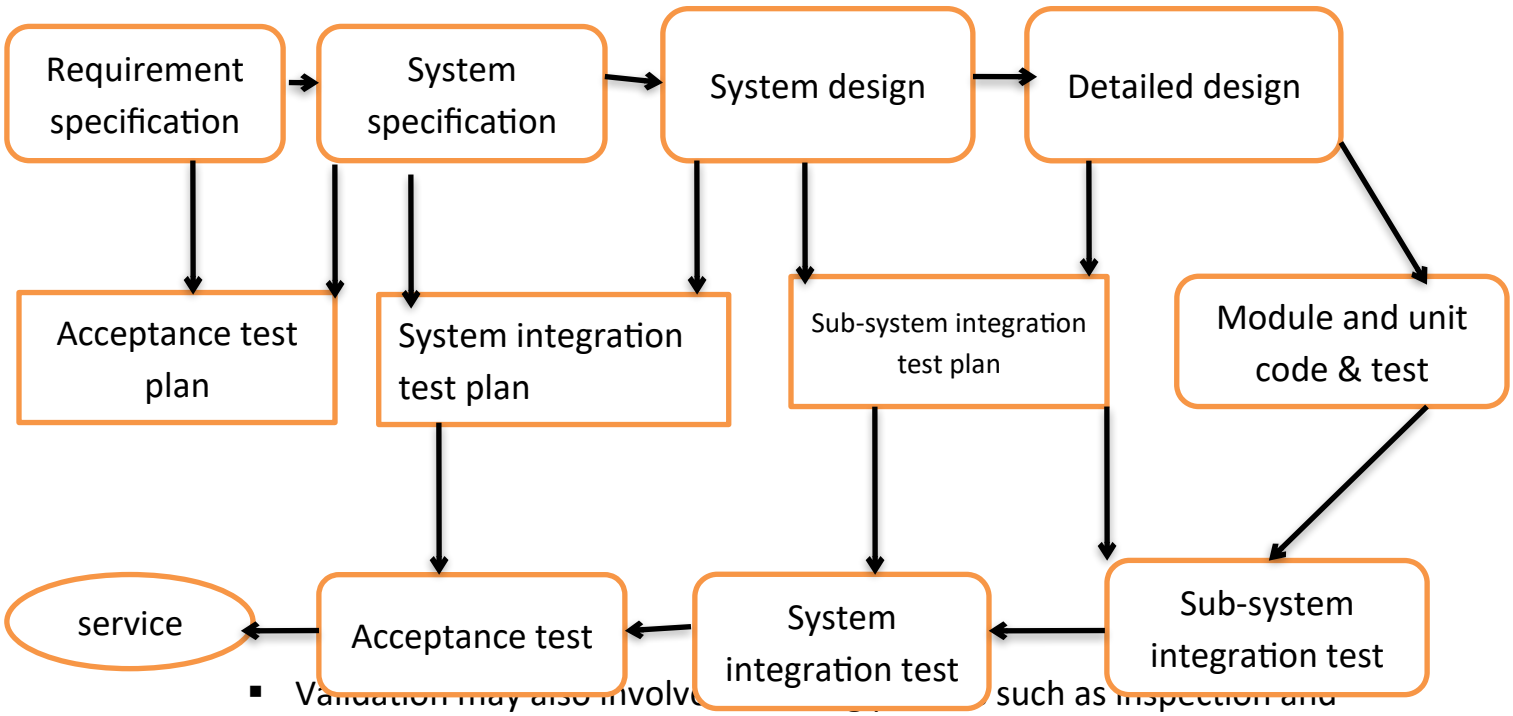
- Architectural design
  - Identification of the overall system.
  - Identify the relationship between principal component
- Interface design
  - Define the interfaces between system components.

- ❖ Component design
  - Here we take each system component and design how it will operate.
  - It may be the list of changes to be made to a reusable component or a detailed design model.
- ❖ Database design
  - Design the system data structure and how they are to be represented in a database.

C. System validation

Software validation & verification is intended to show that both that a System meets both specification and expectation of system customer.

Figure below is of testing phase of plan-driven software process



- ▪ Validation may also involve [illegible] such as inspection and review at each step of software process. Main process in software testing and validation are as follow

- ❖ Development testing
  - Component making up the system are tested by the people developing the system.
  - Each component is tested separately.

- Component may be simple entities such as function, object and class.

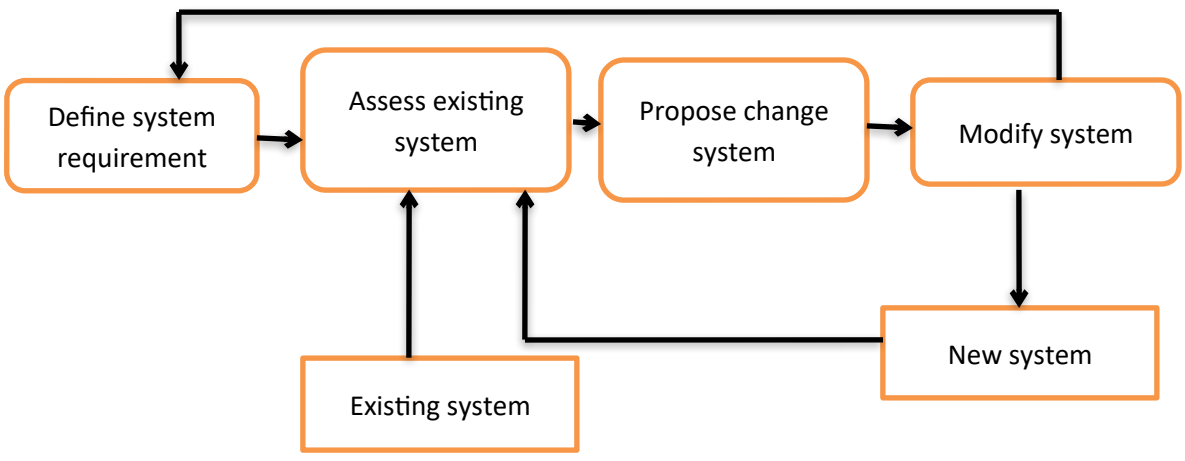
❖ System testing
- System component are integrated to create a complete system.
- Concerned with finding error that happens due to components and component interface problem.

❖ Acceptance testing
- Is the final stage in testing before the system is accepted for operational use?
- The system is tested with data supplied by a customer.
- May reveal errors and requirements problems.

- Alpha testing: some time acceptance testing is known as alpha testing. Custom system is developed for single client. It continues until the client and developer agreed that the system is acceptable.
- Beta testing: involves delivering the system to multiple clients. They report the problem to the developer. After this developer modify it and release the system.

D. **Software evolution**

It is very expensive to make changes to hardware design but changes can be made to software at any time during or after the development in cheaper in correspondence to hardware change. Software engineering is a evolutionary process where software is continually changed over its life time with response to changing requirements and user needs.



1.7 Computer-aided software engineering(CASE)

- CASE tools are programs that are used to support software engineering process. These tools therefore include design editors, data dictionaries, compilers, debuggers, system building tools, etc.
- CASE tools provide process support by automating some process activities and by providing information about the software that is being developed.
- Assist in development and maintenance of software
- Developed in 1970's to speed up the s/w build up process
- Allows rapid development of software to cope with the increasing speed of market demand.

Classification of CASE tools

a. Business system planning
   - Information engineering tools
   - Process modeling and management tools

b. Project management
   - Project planning tools
   - Risk analysis tools
   - Project management tools
   - Requirement tracing tools

c. Programming tools
   - Integrating and testing tools
   - Client /server tools

d. Maintenance tools
   - Requirement engineering tools

Specific examples:

- ✓ with class-object oriented design & code generation
- ✓ oracle designer/200-integrated CASE environment

1.8 Functional and non-functional requirements

Functional requirements
Functional requirements specify the product capabilities, or things that a product must do for its users. The functional requirements specify what the product must do. They relate to the actions that the product must carry out in order to satisfy the fundamental reasons for its existence. Functional requirement must fully describe the actions that the intended product can perform. They describe the relationship between the input and output of the system.

Non-functional requirements
Non-functional requirements define system properties such as reliability, performance, security, response time and storage requirements and constraints like Input output device capability, system representations.
Non-functional requirements are more critical than functional requirements. A system user can usually find ways to work around a system function that doesn't really meet their needs but if the non-functional requirements are not met, then the system will be useless.
They describe various quality factors, or attributes, which affect the functionality's effectiveness.

Functional	Nonfunctional
Product features	Product properties
Describe the work that is done	Describe the character of the work
Describe the actions with which the work is concerned	Describe the experience of the user while doing the work
Characterized by verbs	Characterized by adjectives

1.9 User requirements

User requirements are high level statements, in a natural language with diagrams, of what the system should do and the constraints under which it must operate.
User requirements should describe functional and non-functional requirements in such a way that they are understandable by system users who don't have detailed technical knowledge.
User requirements are defined using natural language, tables and diagrams as these can be understood by all users.

1.10 System requirements

They are more precise than user requirements.
They are more detailed descriptions of the software system's functions services and operational constraints.
The system requirements document should define exactly what is to be implemented.
They may be incorporated into the system contract between the system buyer and the software developers so as to define how the system should work.

They may be defined or illustrated using system models.

1.11 Interface specification

Interface specification describes the behavior of some software unit such as function or class.
Interface specification is an important part of any design process which describes the interfaces between the components in the design.
It is required so that objects and sub functions can be designed in parallel.
It is used to document the design of future software components and the correct usage of an existing component.

1.12 The software requirements documents

The software requirements document sometimes called the software requirements specification or SRS is an official statement of what the system developers should implement.
It should include both the user requirements for a system and a detailed specification of the system requirements.
The requirements document states 'what the software will do'. It does not state 'how the software will do it'.
The main purpose of a requirements document is to serve as an agreement between the developers and customers on what the application will do.

The characteristics of a good software requirements document are

1. **Complete**:A complete requirements specification must precisely define all the real world situations that will be encountered and the capability's responses to them. It must not include situations that will not be encountered or unnecessary capability features.
2. **Consistent:** System functions and performance level must be compatible and the required quality features (reliability, safety, security, etc.) must not contradict the utility of the system. For example, the only aircraft that is totally safe is one that cannot be started, contains no fuel or other liquids, and is securely tied down.
3. **Correct:** The specification must define the desired capability's real world operational environment, its interface to that environment and its interaction with that environment.
4. **Modifiable:** Related concerns must be grouped together and unrelated concerns must be separated. Requirements document must have a logical structure to be modifiable.
5. **Ranked:** Ranking specification statements according to stability and/or importance is established in the requirements document's organization and structure. The larger and more complex the problem addressed by the requirements specification, the more difficult the task is to design a document that aids rather than inhibits understanding.
6. **Testable:** A requirement specification must be stated in such a manner that one can test it against pass/fail or quantitative assessment criteria, all derived from the specification itself and/or referenced information. Requiring that a system must be "easy" to use is subjective and therefore is not testable.
7. **Traceable:**Each requirement stated within the SRS document must be uniquely identified to achieve traceability. Uniqueness is facilitated by the use of a consistent and logical scheme for assigning identification to each specification statement within the requirements document.

8. **Unambiguous:** A statement of a requirement is unambiguous if it can only be interpreted one way. This perhaps, is the most difficult attribute to achieve using natural language. The use of weak phrases or poor sentence structure will open the specification statement to misunderstandings.
9. **Valid:** To validate a requirements specification all the project participants, managers, engineers and customer representatives, must be able to understand, analyze and accept or approve it. This is the primary reason that most specifications are expressed in natural language.
10. **Verifiable:** In order to be verifiable, requirement specifications at one level of abstraction must be consistent with those at another level of abstraction. Most, if not all, of these attributes are subjective and a conclusive assessment of the quality of a requirements specification requires review and analysis by technical and operational experts in the domain addressed by the requirements.

1.13 Feasibility study

A feasibility study is a short, focused study that is done earlier in requirement engineering process and is carried out to select the best system that meets performance requirements.
The main aim of the feasibility study activity is to determine whether it would be financially and technically feasible to develop the product.
The feasibility study activity involves the analysis of the problem and collection of all relevant information relating to the product such as the different data items which would be input to the system, the processing required to be carried out on these data, the output data required to be produced by the system as well as various constraints on the behavior of the system.
It should be relatively cheap and quick.

1.14 Requirements elicitation and analysis

After an initial feasibility study, the next stage of the requirements engineering process is requirements elicitation and analysis.
It is an iterative process that can be represented as a spiral of activities – requirements discovery, classification and organization, negotiation with prioritization and requirements specification.
In this process the software engineers work with the customers and system end users to find out about the application domain, what services the system should provide, the required performance of the system hardware constraints.

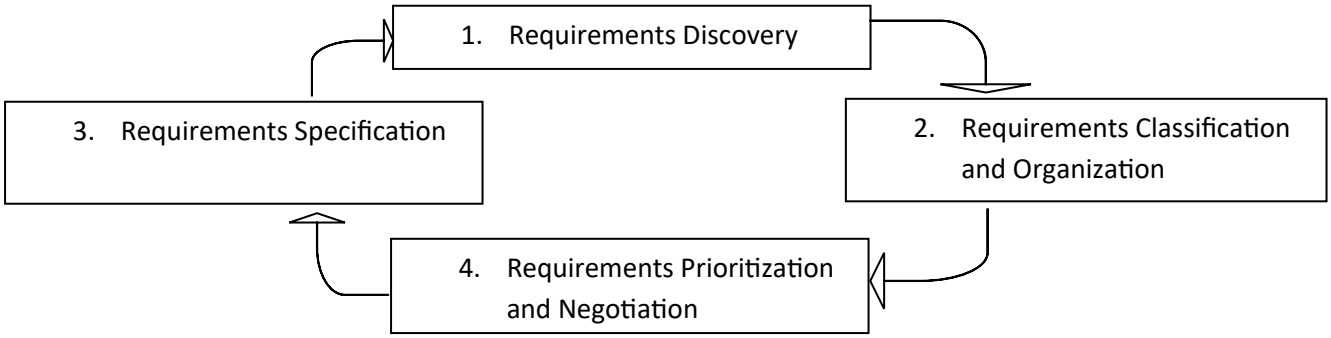


Fig. The requirements elicitation and analysis process

1. **Requirements discovery**

This is the process of interacting with stakeholders of the system to discover their requirements. A system stakeholder is anyone who should have some direct or indirect influence on the system requirements.

2. **Requirements classification and organization**

The discovered unstructured collection of requirements are then classified and structured properly. The most common way of grouping requirements is to use a model of the system architecture to identify sub-systems and to associate requirements with each sub-system.

3. **Requirements prioritization and negotiation**

Inevitably, when multiple stakeholders are involved, requirements will conflict. So the prioritization of the requirements is necessary. Stakeholders have to meet and negotiate to resolve differences and agree on compromising requirements.

4. **Requirements specification**

Finally the requirements are documented and written in a requirements document.

Requirements elicitation is a **difficult process** for several reasons:

a. Stakeholders often don't know what they want from a computer system in most general terms, they may make unrealistic demands because they don't know what is and isn't feasible.
b. Stakeholders in a system naturally express requirements in their own terms and with implicit knowledge of their own work. Requirements engineers, without experience in the customer's domain may not understand these requirements.
c. Different stakeholders have different requirements and they may express these in different ways. Requirements engineers have to discover all potential sources of requirements and find the commonalities and conflict.
d. Political factors may influence the requirements of a system. Managers may demand specific system requirements because these will allow them to increase their influence in the organization.
e. The economic and business environment in which the analysis takes place is dynamic. It inevitably changes during the analysis process. The importance of particular requirements may change. New requirements may emerge from new stakeholders who were not previously consulted.

1.15 Requirements validation and management

Requirements validation is the process of checking that requirements actually define the system that the customer really wants.

It overlaps with analysis as it is concerned with finding problems with the requirements.

It is important because errors in a requirements document can lead to extensive rework costs when these problems are discovered during development or after the system is in service.

The process of requirements validation includes different checks.

1. **Validity and verifiability:** Verification and validation is not the same thing.

Validation: Are we building the right product?

Verification: Are we building the product right?
Validation: The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.
Verification: The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. For verifiable software, we must be able to write a set of tests that can demonstrate that the delivered system meets each specified requirement.
Validation ensures that the product actually meets the user's needs, and that the specifications were correct in the first place, while verification is ensuring that the product has been built according to the requirements and design specifications.
Validation ensures that "you built the right thing". Verification ensures that "you built it right".

2. **Consistency:** Requirements in the document should not conflict. There should not be contradictory constraints or different descriptions of same system function.
3. **Completeness:** The requirements document should include requirements that define all functions and the constraints intended by the system user.
4. **Realism:** The requirements should be checked to ensure that they can actually be implemented under constraints such as time and money.

The main problem with requirement validation is that the requirements change continuously during requirements elicitation.
Requirements validation techniques:
Requirement reviews: The requirements are analyzed systematically by a team of reviewers who check for errors and inconsistencies.
Prototyping: An executable model of the system in question is used to check the validity.
Test-case generation: Requirements should be testable. If a test for a requirement is difficult or impossible to design, this usually means that the requirements will be difficult to implement and should be reconsidered.

Requirement management is the process of managing changing requirements during the requirement engineering process by documenting, analyzing, tracing, and agreeing on requirements and then controlling change and communicating to relevant stakeholders.
It is the process of understanding and controlling changes to the system requirements.

Requirement management planning: Planning is important during requirements management.

a. Requirements identification: Each requirement should be uniquely identified.
b. A change management process: Process followed when analyzing a requirement change and the impact and cost of changes.
c. Traceability policies: These policies define the relationships between each requirement and between the requirements and the system design that should be recorded. The traceability policy should also define how these records should be maintained.
d. Tool support: Requirements management involves the processing of large amounts of information about the requirement. Tools that may be used range from specialist requirements management systems to spreadsheets and simple database systems.

A good requirement engineering process consists of 4 main phases.

1. Feasibility study
2. Requirements elicitation and analysis
3. Requirements specification
4. Requirements validation

Past IOE question

- What is software crisis? Explain with the help of an example..
- Describe spiral model for software development. What are its advantages and disadvantages?
- Explain requirement management process with necessary illustration.
- What are the advantages and disadvantages of the water fall process? List out various model of the software development. Explain the limitation of waterfall model in detail.
- Explain soft requirement specification (SRS).what are the good characteristics of good SRS document?