Module-1: Introduction

What is software?

A program is a set of instructions that performs a specific task. Software is a set of programs that accomplish a collective functionality.

Properties of software differ from the properties of physical constructs. The abstract and intangible nature of the software makes it different. Software could be complex, difficult to understand and expensive to change depending on the type of software being developed. Depending on the context of its operation different types of software require different approaches for development. It is also possible that software can fail. The failure of software could be due to:

1) Increase in demand and

2) Low expectation

What is Software Engineering?

Software Engineering is an engineering discipline that involves the education of building software using appropriate theories and methods to solve problems bearing in mind organizational and financial constraints. While doing so, all aspects of software production are considered and not just technical process of development. Also, project management and the development of tools, methods etc. to support software production should also be considered.

Why is software engineering required?

Software engineering principles involve techniques and practices that are used and tested. It is very much required to be able to produce reliable and trustworthy systems economically and quickly. It is usually cheaper, in the long run, to use software engineering methods and techniques for software systems.

Software products can be categorised into:

1. Generic products: These are 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 commissioned by a specific customer to meet their own needs. Such products are generally used to solve a problem in a specific domain.

Examples – embedded control systems, air traffic control software, traffic monitoring systems.

The attributes of good software are:

Maintainability –Every software developed should be able to meet the changing needs of customers.

Dependability and security- Software developed for any situation should not cause physical or economic damage in the event of system failure. Care should be taken that malicious users should not be able to access or damage the system.
Efficiency – Depending on the context, the developed software would be evaluated based on its responsiveness, processing time and memory utilisation
Acceptability- Any software developed should be understandable, usable and compatible with other systems in the context that work in unison with it.

Software crisis

It is obvious to note that it is very difficult to write efficient software within the right time specified. This could be due to various reasons such as

1. Heterogeneity –increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.

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

3. Security and trust also plays a very important role in developing a software module. As software is intertwined with all aspects of our lives, it is essential that we can trust that software. Hence appropriate security measures need to be taken to safeguard the software from external attack.

Software Engineering Ethics

Software development does not just involve technical skills, but also includes ethics. The team involved in the software development is expected to be honest and ethically responsible for the software at all times. The following factors need to be addressed during, before and after software development.

a) Confidentiality: It is the employers‟ responsibility to maintain confidentiality of the employee information and it is the employee responsibility to maintain the secrecy and confidentiality involved in the development of software.

b) Competence: It is the ethical responsibility of an employee to accept work that matches his technical skills. He should be competent enough to carry out the task assigned to him.

c) IPR (Intellectual Property Rights) – Very tricky, but very essential. It has to be taken care that when reusing existing components for software development, the components is available for use as per IPR.

d) Computer misuse- Using the computer provided by the employer for personal use is misconduct and misuse of the system. Watching videos, playing songs, browsing social networking sites all mark the misuse of computer.

Software process activities

Any software developed generally follows the following software process activities:

1. Software specification – Involves gathering requirements for the development of software

2. Software development- involves the implementation of the software.

3. Software validation – verifying if the software developed meets its requirements.

4. Software evolution- making sure that the software can be modified with respect to change in requirements over time.

Case studies

Insulin pump control system:

This is a system that collects data from a blood sugar sensor and calculates the amount of Insulin required to be injected. Calculation is based on the rate of change of blood sugar levels. It is programmed to send signals to a micro-pump to deliver the correct dose of insulin. This system is considered as a safety-critical system as low blood sugars can lead to brain malfunctioning, coma and death; and high-blood sugar levels have long-term consequences such as eye and kidney damage. Hence the software involved should function just as expected. The architecture of the system is shown in Fig 1.

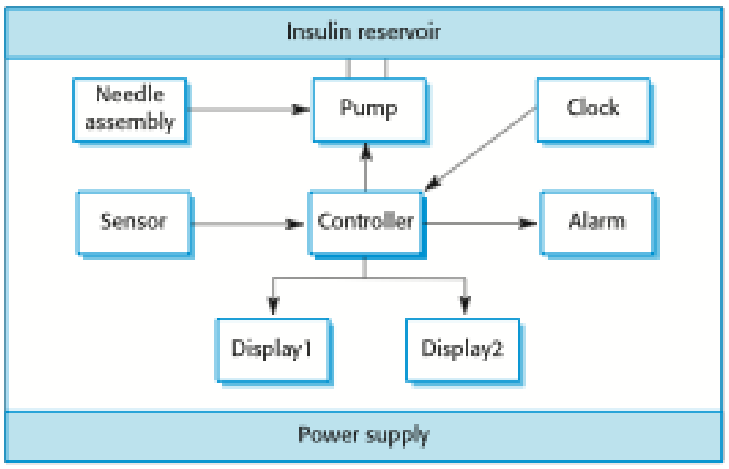


Fig 1: Architecture of insulin pump control system

High-level requirements specifications can be written as follows

1. The system shall be available to deliver insulin when required.
2. The system shall perform reliably and deliver the correct amount of insulin to counteract the current level of blood sugar.
3. The system must therefore be designed and implemented to ensure that the system always meets these requirements.

MHC-PMS (Mental Health Care-Patient Management System)

This is a system which makes use of a centralized database of patient information. When the local systems have secure network access, they use patient information in the database but they can download and use local copies of patient records when they are disconnected. Most mental health patients do not require dedicated hospital treatment but need to attend specialist clinics. Fig 2 shows the architecture of this system.

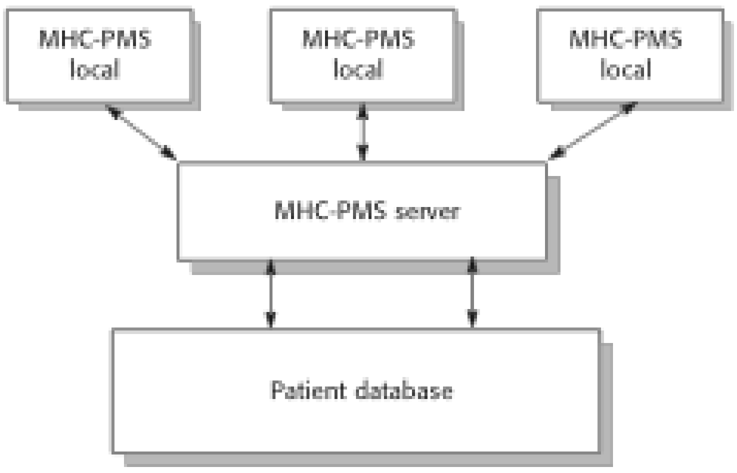


Fig 2: Architecture of MHC-PMS

The system is expected to support the following functionality:

1. Individual care management
2. Patient monitoring
3. Administrative reporting
4. Other Aspects that need to be considered other than functional are:

Privacy - patient information is confidential. At most care need to be taken to protect the privacy of the patient and his illness details.

Safety - prescribe the correct medication to patients.

Wilderness weather station

The government of a country with large areas of wilderness decides to deploy several hundred weather stations in remote areas.

Weather stations collect data from a set of instruments that measure temperature and pressure, sunshine, rainfall, wind speed and wind direction. The weather station includes a number of instruments that measure weather parameters such as the wind speed and direction, the ground and air temperatures, the barometric pressure and the rainfall over a 24-hour period. Each of these instruments is controlled by a software system that takes parameter readings periodically and manages the data collected.

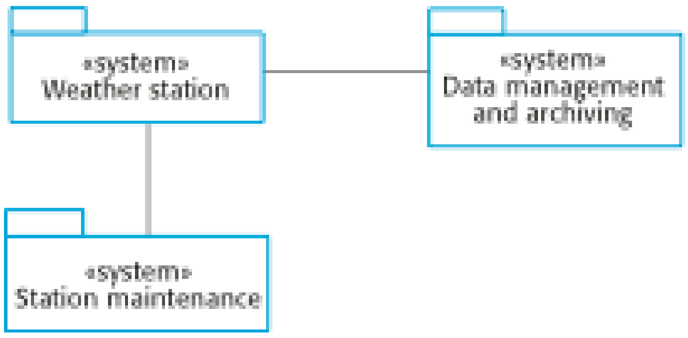


Fig 3.Weather information system

The weather station system: This is responsible for collecting weather data, carrying out some initial data processing and transmitting it to the data management system.

The data management and archiving system: This system collects the data from all of the wilderness weather stations, carries out data processing and analysis and archives the data.

The station maintenance system: This system can communicate by satellite with all wilderness weather stations to monitor the health of these systems and provide reports of problems.

Software Processes

Software processes are a integral part of software process models. Various process activities that are carried out during every phase of development form the software activities. To define technically, A software process is a structured set of activities required to develop a software system. The activities by and large include:

1. Specification – defining what the system should do;

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

3. Validation – checking that it does what the customer wants;

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

Requirements engineering process

Requirements engineering process consists of the following activities:

a) Feasibility study – Technical and Financial feasibility of the project needs to be considered.

b) Requirements elicitation and analysis – Expectation from the system. The functionalities that the system is expected to provide.

c) Requirements specification – Defining requirements for every activity performed by the user or set of users and the expected outcome for each of these activities.

d) Requirements validation – Checking validity of the requirements. Verifying if the requirements specification contains the functionalities that the system is expected to perform.

The requirements engineering process is depicted in Fig 4.

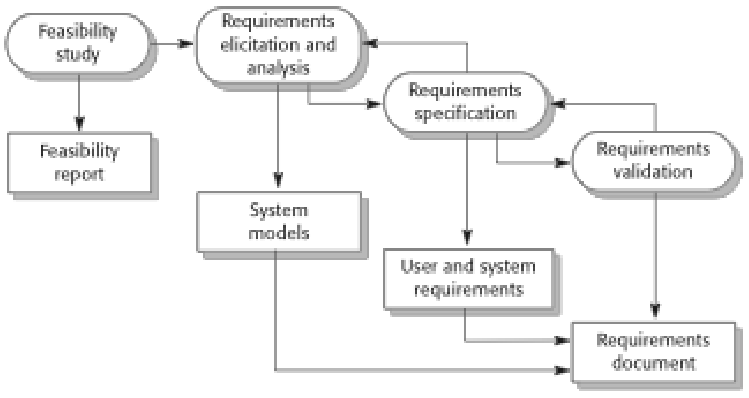


Fig 4: Requirements Engineering Process

Once the feasibility study has been completed a feasibility report is generated. It is observed that the requirements elicitation and analysis, requirements specification and requirements validation are iterative activities. Based on the analysis it is required to decide on a system model that needs to be incorporated as a part of requirements document. The requirements specification activity gives the complete user and system requirements. The requirements validation process checks if the system performs the required functions and the requirement specification covers all functionalities to be provided by the system. The outcome of each activity is an input to the requirements document.

Software design and implementation

Once the requirements for specific software have been gathered the next activity is to design and develop the software. These activities could be performed individually or in parallel depending on the project. Each of these activities have a set of process activities to be carried out. They are described in detail below:

Software design: The major aim of this phase of software development is to design a software structure that realises the specification as given by the requirements specification document.

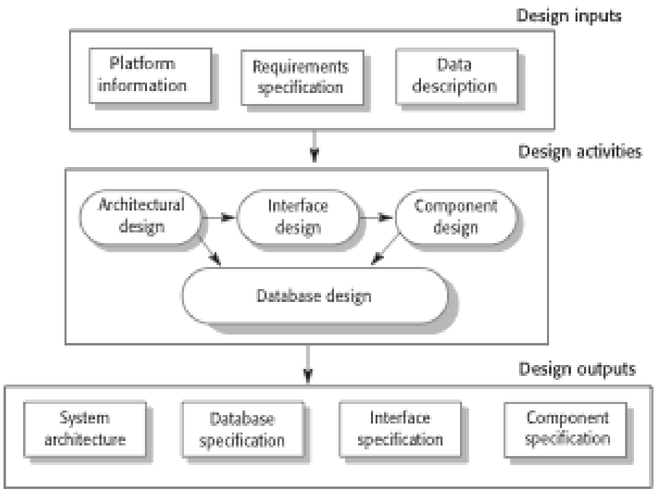


Fig 5: Design process and activities involved

Design activities involve Architectural design, interface design and component design. The system architecture and its components need to be defined and specified as a part of architectural design. The interface design involves the specification and working of interfaces for enables inter and intra component communication.

Implementation

Translate this structure into an executable program. This requires the use of a programming language. The choice of programming language is based on the complexity of the software, the requirements specification of the software, the reusability of components in the software etc.

Software validation

Validating the software involve checking for conformation of the system to its specification. This also involves checking and review processes and system testing.

System testing involves executing the system with test cases. The system testing involves Component testing and acceptance testing. Component testing involves testing of each component that is used to build the software. The interfaces between the components and component that is used to build the software. The interfaces between the components and the working of components with the existing system code needs to be tested. The process activities involved in testing and its iterations are shown in Fig 6. Every individual component is tested and then system is tested as a whole. Acceptance testing is done to determine if the system is functioning well in the domain of its application.

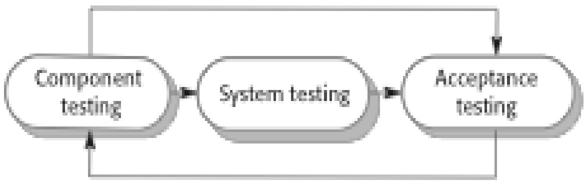


Fig 6: Testing activities

The various phases of testing are depicted below in Fig 7. It is a wrong notion that testing only begins after implementation is complete. As a fact, testing is carried out throughout the development process. The figure below depicts the outcome of each activity in the phase of testing.

Testing phases

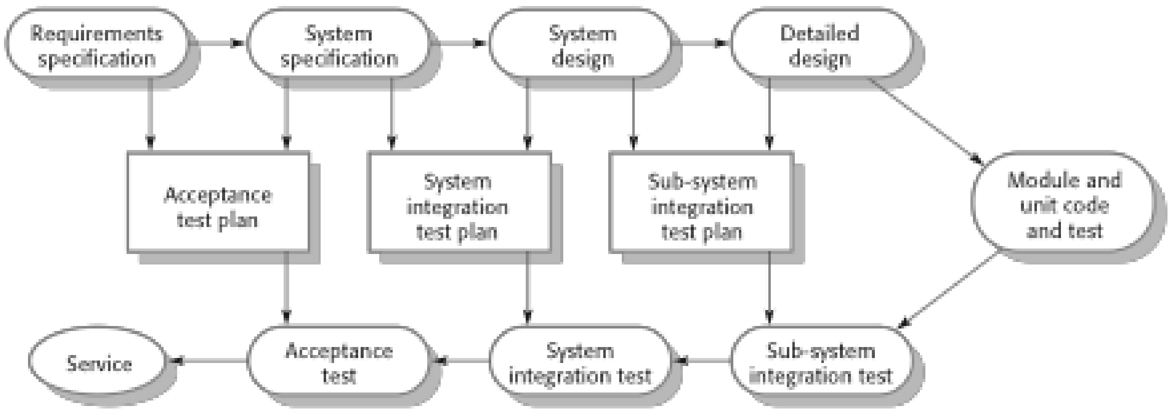

Requirements specification
System specification
System design
Detailed design
Acceptance test plan
System integration test plan
Sub-system integration test plan
Module and unit code and test
Service
Acceptance test
System integration test
Sub-system integration test

Fig 7: Phases of testing

As and when the requirements specification is complete the acceptance test plan is also parallel written because the acceptance testing involves testing of the software in its working domain. Acceptance test cases are written at this stage. The test plan gets its input from both requirements specification as well as system specification activities. The system integration test plan is an output of system specification and design. Workings of individual components, testing for interfaces between them are all a part of system integration testing. Once the detailed design is available the sub system integration test plan is also drafted. Later on the tests are accomplished by making use of the test plan. This is also called as V-model for test plan driven software development process.

Software Evolution

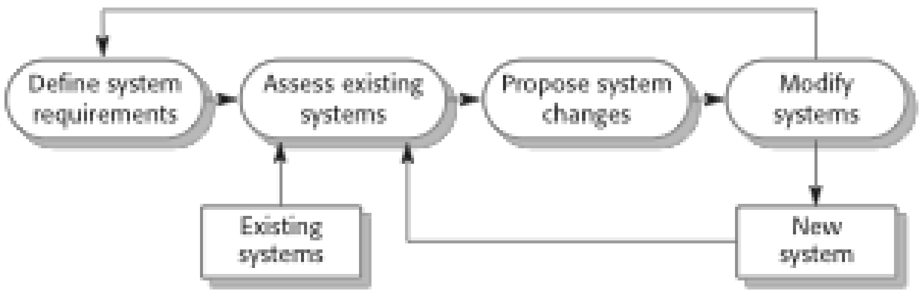


Fig 8: Activities in software evolution

It is obvious that software is inherently flexible and can change. As requirements change through changing business circumstances, the software that supports the business must also evolve and change. Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new. Fig 7 depicts the activities involved in software evolution. Existing system is continuously assessed based on the system requirements. If any change has been proposed the system needs to be modified and a new system is then released. This is an iterative process and continues until the required outcome has been obtained.

Software Process Model

A software process model is an abstract representation of a process. The process descriptions may also include:

1. Products, which are the outcomes of a process activity;
2. Roles, which reflect the responsibilities of the people involved in the process.
3. Pre- and post-conditions, which are statements that are true before and after a process activity has been enacted or a product produced.

Process activities

Processes are inter-leaved sequences of activities with the overall goal of specifying, designing, implementing and testing a software system. The four basic process activities of software development are organized differently in different development processes depending on the application being developed. The activities are: specification, development, validation and evolution. Each of these is explained in detail in the rest of the chapters.

Software specification

The specification of the software requires answering the following questions:

a) What are the services required by the software? – To get the functionalities of the software to be developed.

b) What are the constraints on system operation?- To get the domain information of the software to be developed.

Models

a) The waterfall model: This is a plan-driven model. There are separate and distinct phases of specification and development.

b) Incremental development: This is model where the specification, development and validation are interleaved. Incremental development may follow either plan- driven approach or agile approach.

c) Reuse-oriented software engineering: The system is assembled from existing components. This approach also may be plan-driven or agile.

There is no hard rule on the model chosen for developing a particular application. In practice, most large systems are developed using a process that incorporates elements from all of these models. Each of these models are described in detail below:

Water fall model

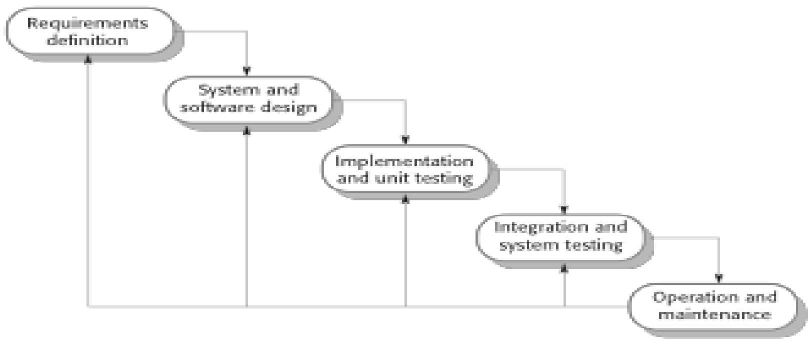

Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Fig 9: Waterfall model

The phases of Waterfall model depicted in Fig 9 are:

a) Requirements analysis and definition – Gathering the requirements and defining the functionality of each requirement. This could be done through various techniques discussed in the next section.

b) System and software design- Design and architecting the software as a whole. The components involved, the interface between them and other design considerations are depicted as diagrams. UML diagrams are most frequently used design representations.

c) Implementation and unit testing – Developing the system and testing each module one by one as a single unit. Every component developed or reused are individually tested.

d) Integration and system testing- The components are integrated together and the system is tested as a whole. The interfaces and communication modules between the systems are tested here.

e) Operation and maintenance- Once the system is put into use, the functionality of the system in the operational domain needs to be taken care of. The maintenance and operation of the system at the client end is important.

Problems with Waterfall model

The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements. The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.

Incremental development and delivery

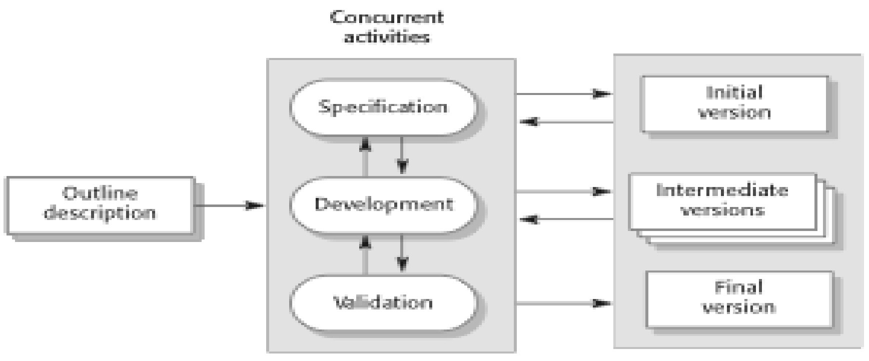


Fig 10: Process in incremental development

Incremental development involves developing the system in increments and evaluating each increment before proceeding to the development of the next increment. This is the incremental approach is done by user/customer proxy.

Incremental delivery involves deploying an increment for use by end-users. This is a more realistic evaluation about practical use of software. It is difficult to implement for replacement systems as increments have less functionality than the system being replaced.

Advantages and disadvantages

Cost of accommodating changing customer requirements is reduced. Customer feedback on the development is obtained as and when an increment is ready to be released.

Rapid delivery and deployment of useful software can be accomplished by making use of this approach.

The Problems with this approach is that the process is not visible. Also, it is inevitable that system structure tends to degrade as new increments are added.

Reuse-oriented software engineering

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

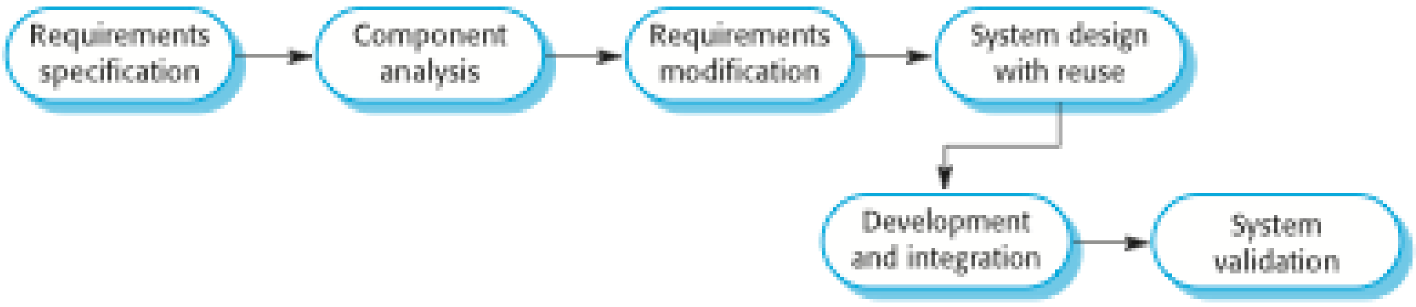
Process stages

Component analysis;

Requirements modification;

System design with reuse;

Development and integration.


Requirements specification
Component analysis
Requirements modification
System design with reuse
Development and integration
System validation

Software specification

The process of establishing what services are required and the constraints on the system's operation and development.

Requirements engineering process

Feasibility study

• Is it technically and financially feasible to build the system?

Requirements elicitation and analysis

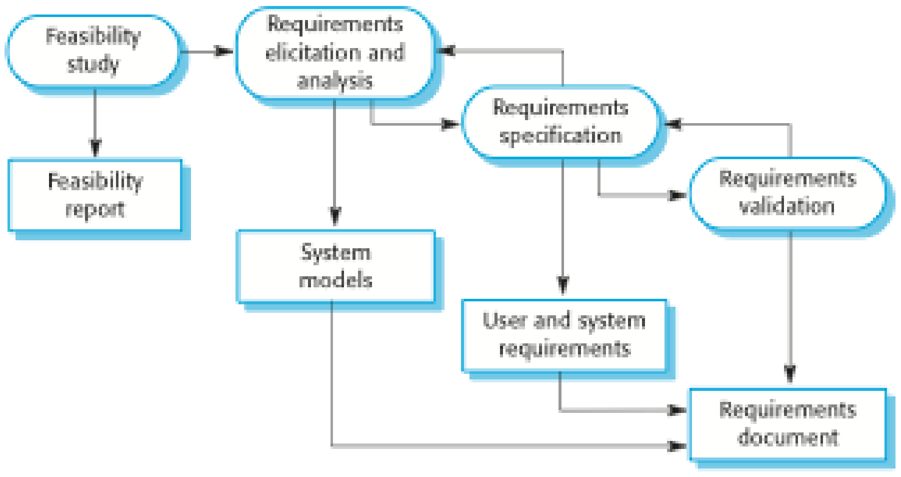
• What do the system stakeholders require or expect from the system?

Requirements specification

• Defining the requirements in detail

Requirements validation

• Checking the validity of the requirements



Software design and implementation

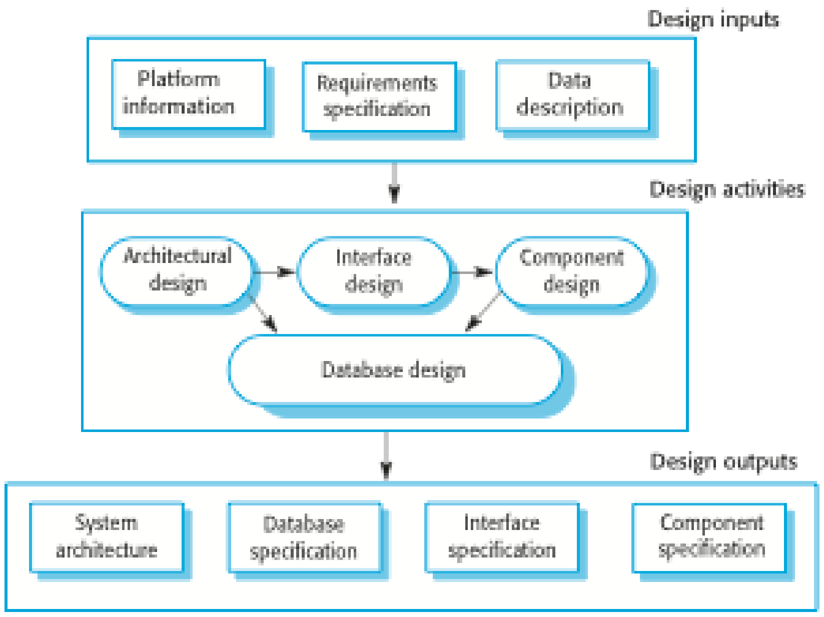
The process of converting the system specification into an executable system.

Software design

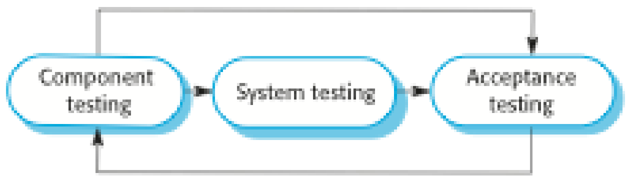
Design a software structure that realises the specification;

Implementation

Translate this structure into an executable program



Stages of testing



Development or component testing

Individual components are tested independently; Components may be functions or objects or coherent groupings of these entities.

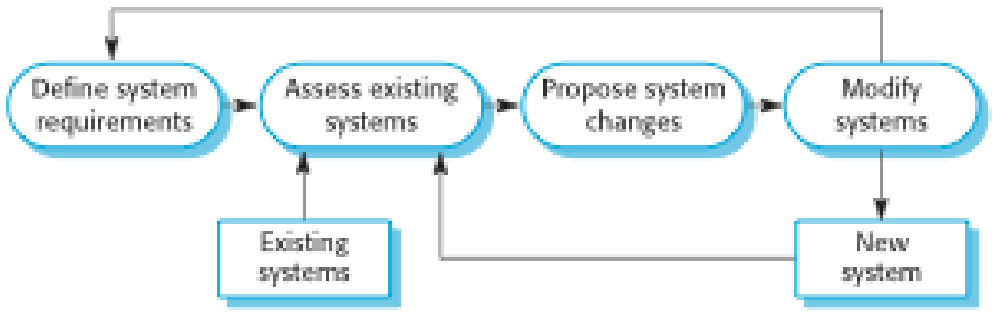
System testing

Testing of the system as a whole. Testing of emergent properties is particularly important.

Acceptance testing

Testing with customer data to check that the system meets the customer's needs.

System Evolution



Software prototyping

A prototype is an initial version of a system used to demonstrate concepts and try out design options. A prototype can be used in:

The requirements engineering process to help with requirements elicitation and validation;

In design processes to explore options and develop a UI design;

In the testing process to run back-to-back tests.

Chapter 2 Software Processes

Benefits of prototyping

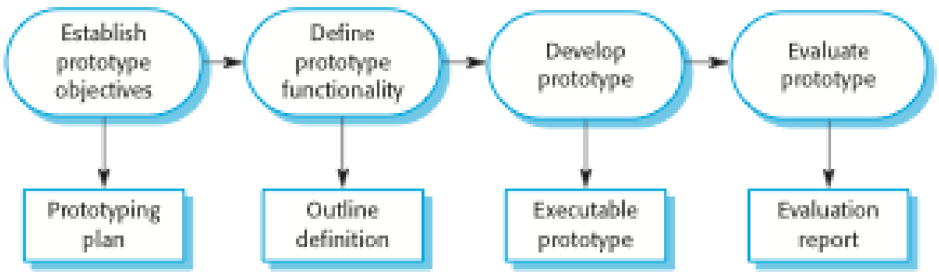
Improved system usability.

A closer match to users' real needs.

Improved design quality.

Improved maintainability.

Reduced development



Throw-away prototypes

Prototypes should be discarded after development as they are not a good basis for a production system:

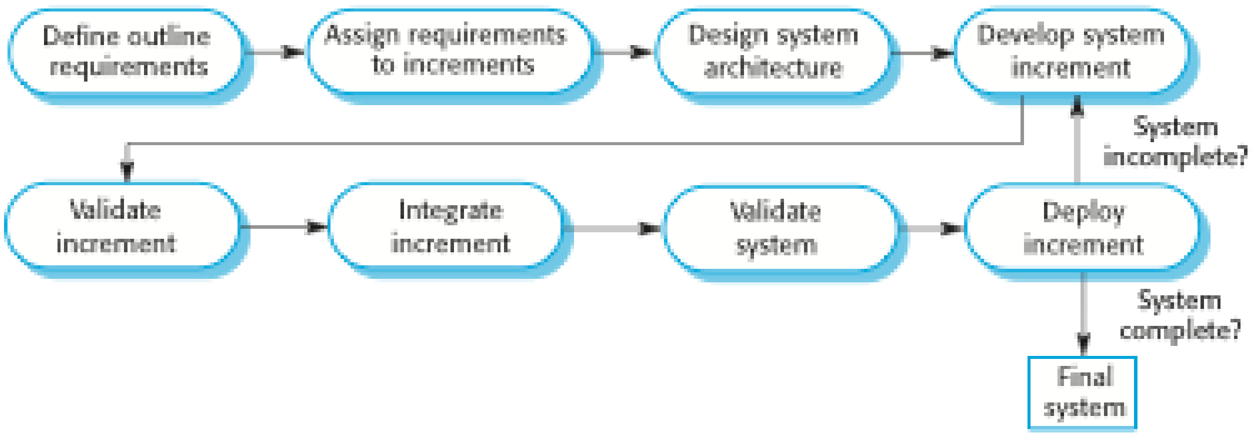
It may be impossible to tune the system to meet non-functional requirements;

Prototypes are normally undocumented;

The prototype structure is usually degraded through rapid change;

The prototype probably will not meet normal organisational quality standards.

Incremental Delivery



Incremental delivery advantages

Customer value can be delivered with each increment so system functionality is available earlier.

Early increments act as a prototype to help elicit requirements for later increments.

Lower risk of overall project failure.

The highest priority system services tend to receive the most testing.

Incremental delivery problems

Most systems require a set of basic facilities that are used by different parts of the system. As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.

The essence of iterative processes is that the specification is developed in conjunction with the software. However, this conflicts with the procurement model of many organizations, where the complete system specification is part of the system development contract.

Boehm‟s spiral model

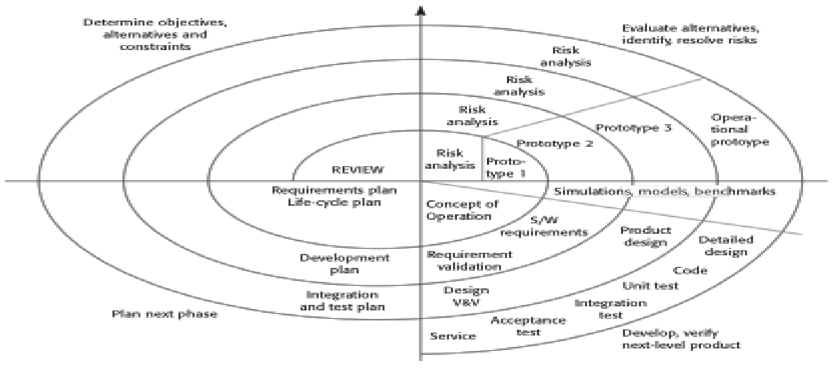


Fig 11: Boehm‟s spiral model

Fig 11 shows the Boehm‟s spiral model. Here the process is represented as a spiral rather than as a sequence of activities with backtracking. Each loop in the spiral represents a phase in the process. There are no fixed phases such as specification or design. The loops in the spiral are chosen depending on what is required. Risks are explicitly assessed and resolved throughout the process. Spiral model has been very influential in helping people think about iteration in software processes and introducing the risk-driven approach to development. In practice, however, the model is rarely used as published for practical software development.

Sectors of spiral model

The following are the sectors of the spiral model:

1. Objective setting -Specific objectives for the phase are identified.

2. Risk assessment and reduction-Risks are assessed and activities put in place to reduce the key risks.

3. Development and validation-A development model for the system is chosen

Which can be any of the generic models.

4. Planning-The project is reviewed and the next phase of the spiral is plan.

The Rational Unified Process (RUP)

A modern generic process derived from the work on the UML and associated process.

Brings together aspects of the 3 generic process models discussed previously.

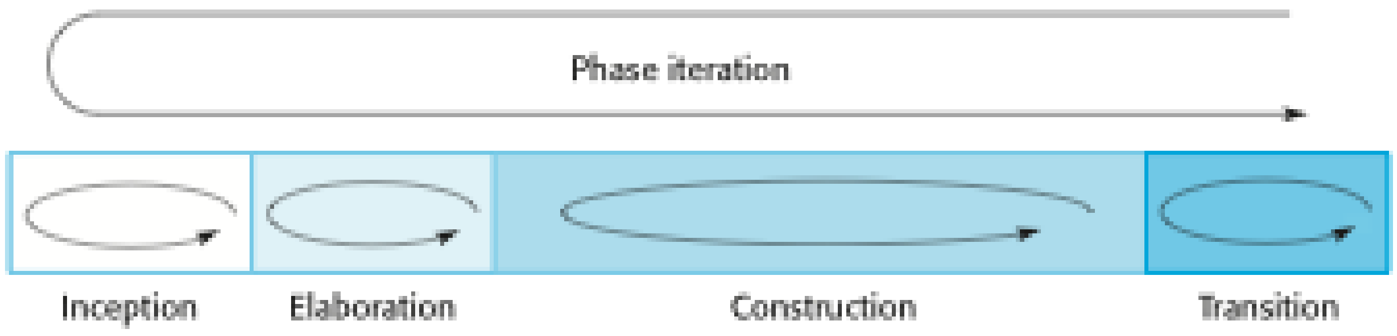
Normally described from 3 perspectives

A dynamic perspective that shows phases over time;

A static perspective that shows process activities;

A practical perspective that suggests good practice.

Phases in RUP



Inception: Establish the business case for the system.

Elaboration: Develop an understanding of the problem domain and the system architecture.

Construction: System design, programming and testing.

Transition Deploy the system in its operating environment.

RUP good practice

Develop software iteratively: Plan increments based on customer priorities and deliver highest priority increments first.

Manage requirements: Explicitly document customer requirements and keep track of changes to these requirements.

Use component-based architectures: Organize the system architecture as a set of reusable components.

Visually model software: Use graphical UML models to present static and dynamic views of the software.

Verify software quality: Ensure that the software meet's organizational quality standards.

Control changes to software: Manage software changes using a change management system and configuration management tools