Course Title : Software Engineering

Course no: CSC - 351 FM : 60+20+20

Credit hours: 3 PM : 24+8+8

Course Synopsis: Discussion on types of software, developing process and maintaining the software

Goal: This course introduces concept of software development paradigm and implementing these in real world.

Course Contents:

Unit 1: - 11Hrs.

1.1 Introduction to Software Engineering: Definition of software, software engineering, Comparing between other engineering and
8 software engineering.

1.2 System Engineering: Introduction to System, System properties,
9 system and their environment, system modeling.

1.3 Software Process: Introduction, software process model, process iteration, software specification, software design and implementation
30 software validation, software evolution.

1.4 Project Management: Introduction, management activities, project
28 planning, project scheduling, risk management.

Unit 2: - 12 Hrs.

2.1 Software Requirements: Introduction, Types of requirements, requirements engineering process: feasibility study, requirements elicitation and analysis, requirement validation, requirement management.

14

2.2 Software Prototyping: Introduction, prototyping in the software process, rapid prototyping techniques, user interface prototyping.

6

2.3 Formal Specification: Introduction, formal specification in software process, interface specification, behavioural specification.

Unit 3:

3.1 Architectural Design: Introduction, system structuring, control models, modular decomposition, domain specific architecture.

3.2 Object Oriented Design: Introduction, features of object oriented design, object oriented software engineering.

Unit 4:

4.1 Verification & Validation: Introduction, verification & validation planning, software inspection, cleanroom software development.

4.2 Software Testing: Introduction, types of testing, testing work benches.

4.3 Critical system validation: Introduction, formal methods and critical systems, reliability validation, safety assurance, security assessment.

4.4 Software Cost Estimation: Introduction, productivity, estimation techniques.

4.5 Software Reengineering: Introduction, source code translation, reverse engineering.

classmate

Date
Page

UNIT 1:

1.1

1.1) Introduction to Software Engineering

The software engineering is composed of two words; software and engineering.

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations. Software, when made for specific requirement is called software product.

Engineering on the other hand, is all about developing products, using well defined, scientific principles and methods. So we can define "software engineering as an engineering branch associated with the development of software products using well-defined scientific principle, methods and procedure. The outcome of software engineering is an efficient and reliable software product. Software engineers should adopt a systematic and organized approach to their work and use appropriate tools and techniques depending upon problem to be solved, the development constraints and resources available.

Without using software engineering proinciple, it would be difficult to develop large programs. In industry it is usually needed to develop large program to accomodate multiple functions. A problem with developing such large commercial programs is that the complexity and difficulty levels of the program increase exponentially with their sizes.

Software engineering helps to reduce this programming complexity. Software engineering principles use two important techniques to reduce problem complexity: Abstraction and decomposition.

The principle of abstraction implies that a problem can be simplified by omitting irrelevant details. In otherwords, the main purpose of abstraction is to consider only those aspects of the problem that are relevant for certain purpose and suppress other aspect that are not relevant for the given purpose. Once this simple problem is solved then omitted details can be taken into consideration to solve next lower level abstraction and so on.

The other approach to tackle problem complexity is decomposition. In this technique, a complex problem is divided into small problems and small problems are solved one by one. However, in this technique any random decomposition of a problem into small part will not help. The problem has to be decomposed such that each component of the decomposed problem can be solved independently and then the solution of different components can be combined to get full solution. A good decomposition of a problem should minimize the interaction among various components. If the different sub-components are interrelated, then the different component cannot be solved seperately and the desired reduction in complexity will not be realized.

classmate
Date
Page

Need of Software Engineering :

The need of software engineering arises because of higher rate of change in user requirement and environment on which software is working.

Large software: with the increase in size of software, it becomes more complex so software engineering has to give scientific process.

Scalability: If the software process were not based on scientific and engineering concepts, it would be easier to recreate new software then to scale an existing one

Cost: As hardware industry has shown its skills and huge manufacturing has lower down the price of computer and electronic hardware. But the cost of software remains high if proper process is not adopted.

Dynamic nature : The always growing and adapting nature of the software hugely depends upon the environment in which the user works. If the nature of software is always changing, new enhancement need to be done in the existing one. This is where software engineering plays a good role.

Quality management: Better process of software development provides better and quality software products.

Characteristics of software:

To gain the understanding of software, it is important to examine the characteristics of software that make it different from other thing that human being built. Software is logical component rather than physical element. Software has characteristics that are considerably different than those of hardware.

Software does not wear out: Hardware exhibits relatively high failure rate in its life cycle. The main reason for failure of hardware are commutative, effect of dust, temperature and other many environmental factor.

Unlike hardware, software is not suspectable to environmentable factor that cause hardware tear out. Undiscovered defect will cause high failure rate early in the life of program however these are corrected during its life cycle and software will undergo changes and maintainance. Software can be changed easily during its life cycle.

Although some similarities exhibits between software development and hardware manufacture, In both cases, high quality is achieved through good design but in software, development software component is improved at any point of its life cycle.

Attributes of good software:

Software products have a number of associated attributes that reflect the quality of software. These attributes are not directly concerned with what software does. They reflect their behaviour ~~with~~ while it is executive, the structure and organization of source program and associated documentation. The essential attributes for good software are as follows:

1. Maintainability:

Software should be written in such a way that it may evolve to meet the changing needs of customer. This is a critical attribute because software change in an inevitable consequence of changing business environment.

2. Dependability:

Software dependability has a range of characteristics including reliability, security, safety and availability. Dependable software shouldnot cause physical or economic damage in the event of system failure.

3. Efficiency:

Software shouldnot make wasteful use of system resources such as memory and processor's cycles. Efficiency therefore includes responsiveness, processing time, memory utilization, etc.

4. Usability:

Software must be usable with undue effort, by the type of user for whom it is designed. This means that it should have an appropriate user interface and adequate documentation.

Types of software product:

Generally, there are 2 fundamental type of software product:

a) Generic products
b) Customized products

market ma sajilai painxa

a) Generic products :-

These are stand alone systems that are produced by a development organization and sold on the open market to any customer which usable buy them.

Examples of this type of product include software for PCs such as databases, word processors, drawing package and project management tools.

market ma order garna parxa (सजिलै available हुन्न)

b) Customized products:-

These are system which are commissioned by a particular customer. A software contractors ~~by a particular~~ develop the software specially for that customer.

Example of this type of software include control system for electronic devices, system written to support a particular business process and air traffic control system, etc.

2068 (7)

Challenges facing software engineering:

i) Heterogenety challenge:-

Increasingly, systems are required to operate as distributed system across networks that include different types of computer and with different kind of support system. The heterogenety challenge is the challenge of developing technique to build dependable software which is flexible enough to cope with this heterogenety.

ii) Delivery challenge:-

Many traditional software engineering techniques are time consuming. The time they take is required to achieve software quality. However, business today must be responsive and change very rapidly. Their supporting software must change frequently rapidly. The delivery challenge is the challenge of shortening delivery times for large and complex system without compromising quality.

iii) Security and trust challenge:-

A software is interwined (§) with all aspects of our life, it is essential that we can trust that software. This is specially true for remote software system access through a web page or web service interface. We have to make sure that malicious users cannot attack our software and that information security is maintained.

Relation between software engineering and computer science:

Computer science is concerned with the theories and method that underlying computer and software system whereas software engineering is concerned with the practical problem of producing software. Some knowledge of computer science is essential for software engineers in the same way that some knowledge of physics is essential for electrical engineers. Computer science theory, however, is often must applicable to relatively small program. Elegant theories of computer science cannot always be applied to large, complex problems that require a software solution.

Relation between software engineering and system engineering:

System engineering is concerned with all aspects of the development and evolution of the computer system or other complex system where software plays major role. System engineer is therefore concerned with hardware development, policy and process design and system deployment, as well as software engineering.

1.2) System Engineering

System:-

A system is a well organized collection of inter-related components that work together according to plan to achieve central objectives. The component in system are inter dependent so failure in one component can propagate through the system and affect operation of other component.

For example:- Software can only operate if the processors can only carryout computation, if software system defining these computation has been successfully installed.

System properties:

1. Volume

The volume of system (total space ocupied) varies depending on how the components assemblies are arranged and connected.

2. Usability

This property reflects how easy it is to use the system. It depends on the system components, its operators and its operating environment.

3. Reliability

System Reliability depends on component reliability but unexpected interactions can cause new types of failure and therefore affect the reliability of the system.

4. Repairability

This property reflects how easy it is to fix a problem with the system once it has been discovered. It depends on being able to diagnose the problem, access the components that are faulty and modify or replace these components.

5. Security

The security of the system (its ability to resist attack) is a complex property that cannot be easily measured. Attack may be devised that werenot anticipated by the system designers and so may defeat built in safeguard

System Engineering :-

System Engineering is the activity of specifying, designing, implementing, validating and maintaining the system. System engineer are not just concerned with software but also with hardware, system interaction with users and its environment. They must think about the service that the system provides the constraint under which the system must be built and operated and the way in which system is used to fulfill its purpose.

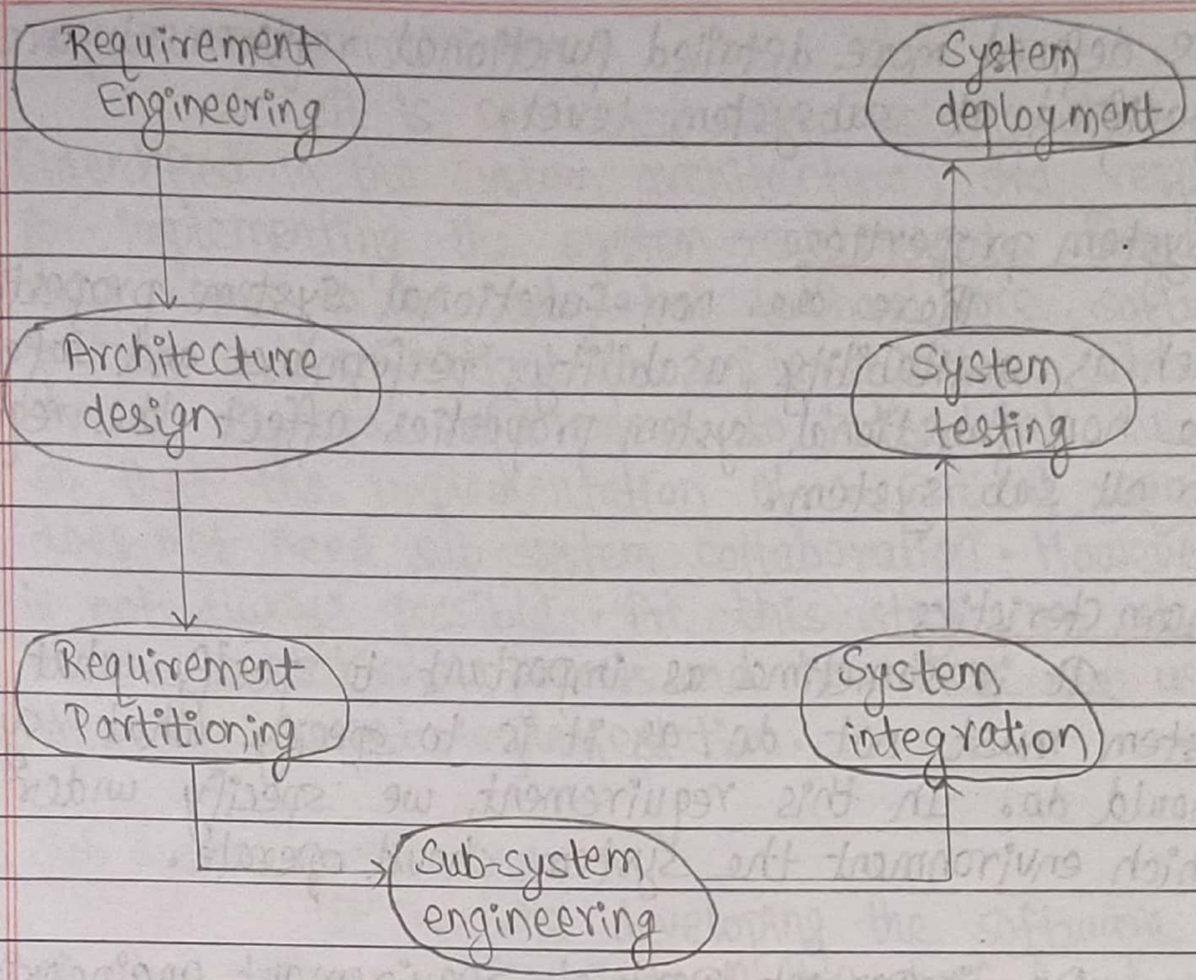


Fig:- The System development process

1. Requirement Engineering:

It is the process of refining, analyzing and documenting the high level and business requirement identified in the conceptual design. The system requirement definition (engineering) phase involves the consultation with the system customer and end user. The requirement engineering phase usually concentrated on driving three types of requirements:

i) Abstract functional requirement

The basic functions that the system must provide are defined more detailed functional requirement are specified at subsystem level.

ii) System properties

There are non-functional system properties such as availability, usability, performance and safety. The non-functional system properties affect the requirement for all sub system.

iii) Characteristics

It is sometime as important to specify what the system must not do, as it is to specify what system should do. In this requirement, we specify under which environment the system should operate.

An important part of requirement engineering phase is to establish a set of overall objective that the system should meet.

2. Architectural design:

It overlaps significantly with the requirement engineering process. The process involves establishing the overall architecture of the system, identifying the different system components and understanding the relationship between them.

3. Requirement Partitioning:

It is concerned with deciding which sub-system (identified in the system architecture) are responsible for implementing the system requirements. Requirements may have to be allocated to hardware, software or process and prioritized for implementation. Ideally, we should allocate requirement to individual sub-systems so that the implementation of a critical requirement does not need sub-system collaboration. However, this is not always possible. At this stage, we also decide on the operational process and how these are used in the requirement implementation.

4. Sub-system engineering:

It involves developing the software components of the system, configuring off-the-shelf hardware and software designing if necessary, special purpose hardware, defining the operational process for the system and re designing essential business process.

5. System integration:

It is the process of putting together the system elements to create a new system. Only then do the system properties become apparent.

6. System testing:

It is an extended activity where the whole system is tested and problems are exposed. The sub-system engineering and system integration phases are re-entered to repair these problems, tuned the performance of the system and implement new requirements. System testing may involve both testing by the system developer and acceptance / user testing by the organization that has procured the system.

7. System deployment:

It is the process of making the system available to its users, transferring data from existing systems and establishing communication with other system in the environment. The process culminates with a 'go live' after which user start to use the system support their work.

System and Environment:

Environment means everything with which system interacts. Systems are not independent but exists on environment. Systems function may be to change environment or sometimes environment affects the functioning of the system. For a program running on a single computer, the system environment might include any other programs running on the computer, the operating system, all the computer configuration setting and the computer's physical characteristics. The change in parameters of environment may affect the system.

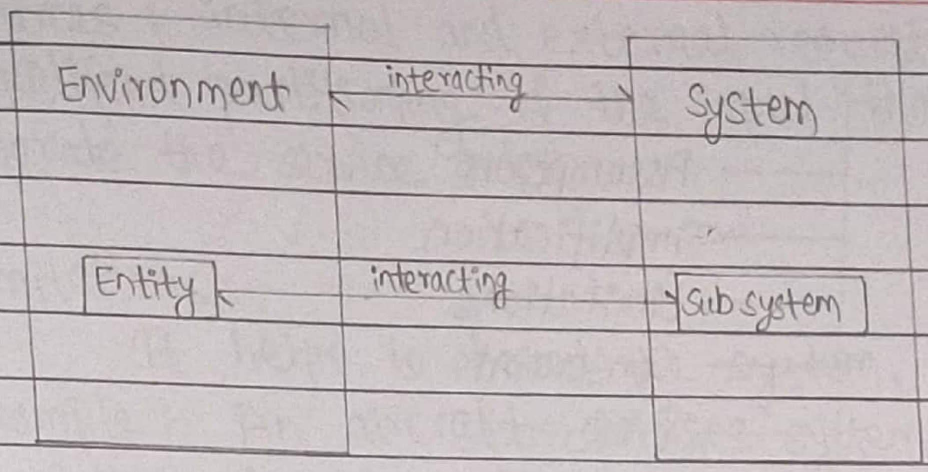


Fig :- System and environment

System Modeling :-

System modeling is the process of developing abstract model of a system, with each model presenting a different view or perspective of that system.

Features of Model:

1. Define the processes that serve the needs of view under consideration.
2. Represent the behaviour of the process and the assumption on which behaviour is based.
3. Explicitly define both exogenous and endogenous input to the model.
4. Represent all linkage that will enable the engineer to better understand the view.

To construct a model, the engineers should consider the number of restraining factor (controlling factor).

System Modeling	— Assumptions — Simplification — Limitations — Constraints — Preferences

Fig:- Restraining factor in system modeling

1. Assumptions:-
It enables a model to reflect the problem in a reasonable manner by reducing the number of possible permutation and variations. Example: representation of 3D human form. In this input domain, may be that the system engineer makes certain assumptions about the range of allowable human movement (legs cannot be wrapped around the torso) so that the range of input processing can be limited.

2. Simplification:-
It enables the model to be created in timely manner. Example:- A system engineer is modeling the needs of service organization and is working to understand the flow of information that produce a service order. Although a service order can be derived from many origins, the engineer categorizes only two

sources : internal and external request. This enables a simplified partitioning of the input that is required to generate the service order.

3. Limitations :-

It helps to bound the system.

Example :- An aircraft avionics system is being modeled for future aircraft. Since the aircraft will be two-engine design the monitoring domain for porpulsion will be modeled to accomodate a maximum of two engines and associated redundant system.

4. Constraints :-

It will guide the manner in which the model is created and the approach taken when the model is implemented.

Example: Suppose a system for the 3-D rendering describes previously is a single 64-bit based processor so computational complexity of problem must constraint to fit within processing bound imposed by the processor.

5. Preferences :-

It indicates the preferred architecture of all data, functions and technology.

1.3) Software Process

A software process is a set of related activities that leads to the production of software product. These activities may involve the development of software from scratch in standard programming language.

Software process are complex and like all intellectual a creative processes, rely on people making decisions and judgements. There is no ideal process and most organizations have developed their own software development process. Process have evolved to take advantage of the capabilities of the people in an organization and the specific characteristics of the systems that being developed. For some systems, such as critical systems, a very structured development process is required.

Software Process Model:

A software process model is an abstract representation of software process. Each process model represents a process from a particular perspective and thus provides only partial information about that process. There are various types of process model.

Waterfall Model

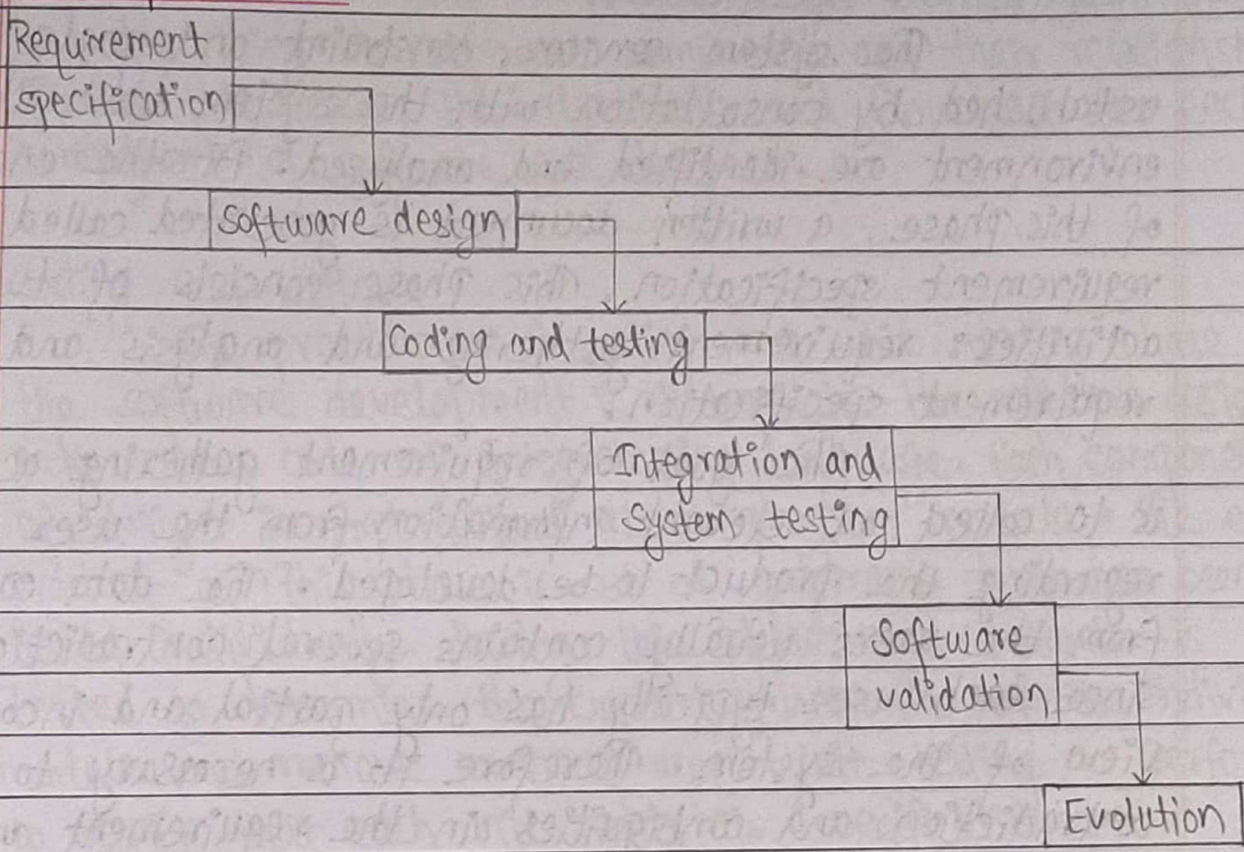


Fig:- Waterfall Model

In the waterfall model, the fundamental process activity: specification, development, validation and evolution are represented as seperate process phase such as requirement specification, software design, coding and testing, implementation and evolution. In waterfall model we complete one phase before going to next phases and we cannot return back to the previous phase.

1. Requirement specification:

The system services, constraint and goals are established by consultation with the system user. User's environment are identified and analyzed. At the end of this phase, a written document is generated called requirement specification. This phase consists of two activities: requirement gathering and analysis and requirement specification.

The goal of requirement gathering activity is to collect all relevant information from the user regarding the product to be developed. The data collected from the users usually contains several contradiction since each user typically has only partial and incomplete view of the system. Therefore it is necessary to identify contradiction and ambiguities in the requirement and resolve them through further discussion with the customer.

After that the requirement specification activity can start. During the activity, the user's requirement are systematically organized into software requirement specification document.

2. Software design:

The goal of the software design phase is to transform the requirement specification into a structure that is suitable for implementation in some programming language. In technical term, during the design phase the software architecture is derived from the software requirement specification document. Software design involves identifying and describing the software components and their relationship. During this phase, the algorithm and flowchart for each component of software are designed.

Coding and testing:

The procedure for coding and unit testing phase of the software development is to translate the software design into source code of programming language. Each component of design is implemented as a program module. The end product of this phase is set of programs that has been tested individually. During this phase, end module is tested to determine the current functioning of the individual module and remove error if any. Unit testing is performed to verify that each unit needs its specification.

Integration and System testing:

Integration of different module is undertaken, once they have been coded and tested individually. During the integration and system testing phase, the modules are integrated in a planned manner. The different modules making up a software product and almost never integrated into one shot. Integration is normally carried out incrementally over a number of steps. During each integration state, the partially integrated system is tested and shapes of previously planned modules are added with. Finally, after all the modules have been successfully integrated and tested, system testing is carried out. The goal of system testing is to ensure that the developed software confirms to its requirement specification. System testing is normally carried out in a planned manner according to the system test planned document. The system test planned document identifies all testing related activities that must be performed specify the schedule of testing and allocated required resources.

5. Software validation:

After testing the system software, software validation is done. Software validation is an important part of software process, apart from verification, debugging, and certification. Validation ensures that the software needs the quality standards set by customer and product meets customer requirement.

6. Evolution:

After a reasonable period of time, the system is evaluated to improve its efficiency and performance. During this phase, the system is changed or modified to adopt the changing requirement of user and organization.

Advantages of waterfall model:

- Easy to understand even by non-technical person, i.e. customers
- Each phase has well defined inputs and outputs.
- Easy to use as software development proceeds.
- Each stage has well defined deliverables.
- Helps project manager in proper planning of the project.

Disadvantages of waterfall model:

- Difficulty in accomodating change after the process is underway because of sequential nature.
- Inflexible partitioning of the project into distinct stages.
- This makes it difficult to respond the changing customer requirement.

This model is suitable when requirement are well understood.

Evolutionary Development Model:

Evolutionary development model is based on the idea of developing an initial implementation, exposing this to the user comment and refining it through many version until an adequate system has been developed. Specification, development and validation activities are interleaved than seperate with rapid feedback across activities. There are two fundamentally evolutionary development models.

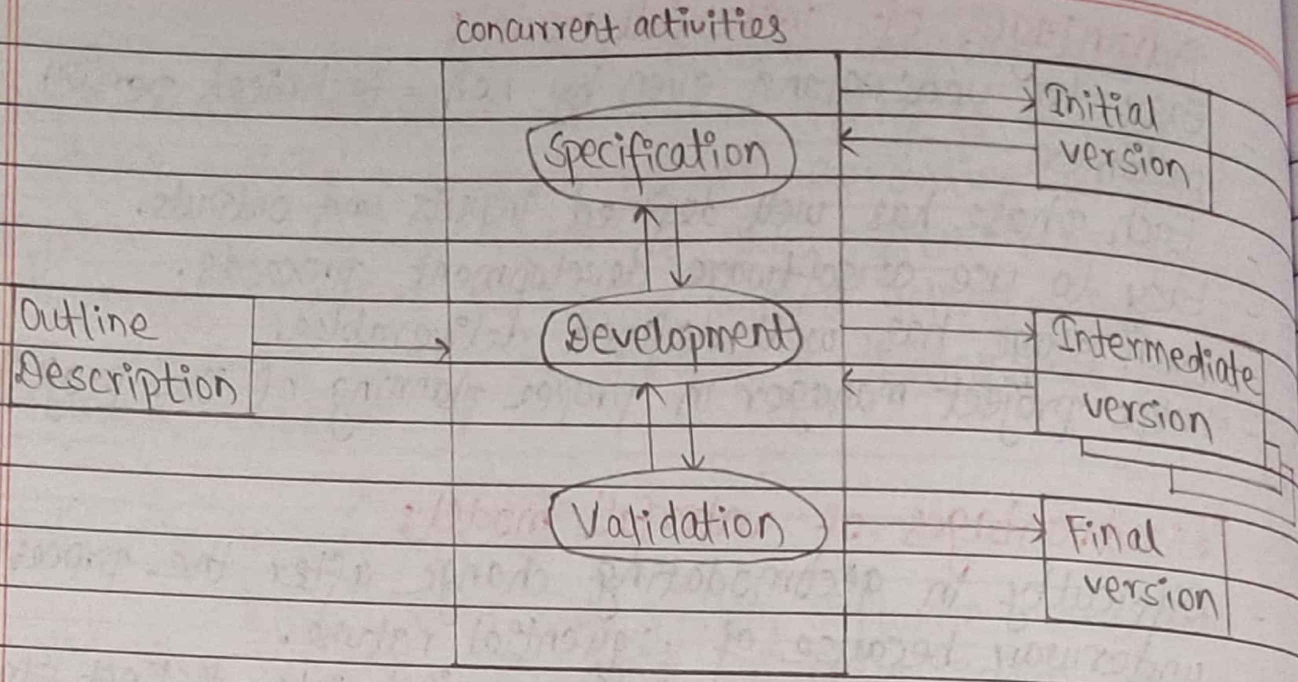


Fig:- Evolutionary Development.

a) Prototyping model (Throwaway prototype):
The main objective of prototype model is to understand system requirement. It can start with the poorly understand requirement.

b) Exploratory development model:
The main objective of this model is to work with customers and evolve a final system from an initial outline specification. The development starts with the part of the system that is understood. The system are expanded by adding new features purposed customers.

Necessity of Prototype:

There are several users of prototype and important purpose of prototype is to illustrate input data format, message, reports, dataflow, etc to users. This is valuable mechanism for gaining better understanding of customers. Another reason for developing prototype is that it is impossible to get a perfect product in first attempt. Many researcher and engineering advice that if we want to develop a good product. We must plan to develop first version and gained experience in developing the prototype and use this experience to develop a final product.

Advantages of evolutionary development model:

- An evolutionary approach is often more effective than waterfall model approaches in producing systems that meet the ~~inter~~ immediate needs of customers.
- The advantages of software process that is based on an evolutionary approach is that the specification can be developed incrementally.
- As users develop a better understanding of their problem, this can be reflected in software system.

Problems towards engineering and management perspective:

- The process is not visible: Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost effective to produce document that reflect every version of the system.

- Systems are often poorly structured: Continual change tend to corrupt the software structure. Incorporating software change becomes increasingly difficult and costly.

Reuse Oriented Software Engineering (Component based Software Engineering):-

In majority of software projects, there is some software reuse. Reuse oriented approaches rely on a large base of reusable software components and integrating frame work for the composition of these components. Sometimes, these components are systems in their own right (COTS or Commercial Off - the Shelf syst that may provide specific functionality such as text-formatti or numeric calculation. A general process model for reuse based development is shown in figure below. Here, we have requirement specification stage and validation stage as comparable with other process. The intermediate stages of component based model are different.

1. Requirement specification:-

In this stage, the system service constraints and goals are established by consultation with the system user. The system user requirements are identified and analyzed. After that document is generated called requirement specification.

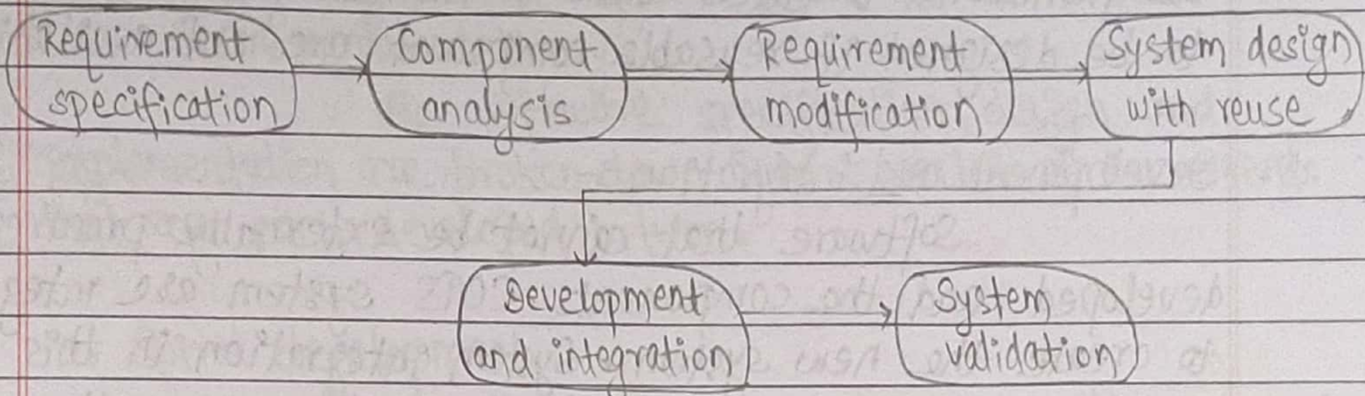


Fig:- Reuse-Oriented Software Engineering

2. Component analysis:-
In this stage, a search is made for component to implement that specification. Usually, there is no exact match and the components that may be used to provide some of the functionality required.

3. Requirement modification:-
During this stage, the requirements are analyzed using information about the components that have been discovered. They are then modified to reflect the available components where modification is impossible, the component analysis activity may be re-entered to search for alternative solutions.

4. System design with reuse:-
During this phase, the framework of the system is designed or an existing framework is reused. The designers take into account the components that are reused and organize

provide with what is needed

the framework to cater this. Some new feature may have to be designed if reusable component are not available.

5. Development and integration :-

to get sth specially after effort

Software that cannot be externally procured is developed, and the components COTS system are integrated to create the new system. System integration in this model may be part of the development process rather than seperate activity.

6. System validation :-

In this phase, the developed system is verified and checked whether it satisfy requirement specification properly or not. If the system do not specify user requirements, the new version of system should be re-designed.

Process iteration :-

The change is inevitable in all large software projects. The system requirement change as the business procuring the system response to external pressure. Management priorities change, as new technologies become available designs and implementation change. This means that the software process is not a one-off process; rather, the process activities are regularly repeated as the system is re-worked in response to change request.

The two process model that have been

explicitly designed to support process iteration are:

1. Incremental delivery :-
The software specification, design and implementation are broken down into a series of increments that are each developed in turn.

2. Spiral development :-
The development of the system spirals outwards from an initial outline through to the final developed system.

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

1. Incremental delivery:-
Incremental delivery is an approach that combines the advantages of waterfall model and evolutionary model.

- In an incremental development process, customer identify, in outline, the services to be provided by the system. They identify which of the services are most important and which are least important to them. A number of delivery increment are then defined with each increment providing a subset of the system functionality. The allocation of services to increment depends on the service priority with the highest priorty service delivered first.

- Once the system increments have been identified the requirements for the services to be delivered in the first increment are defined in detail, and that increment is developed. During development, further requirements analysis for later increment can take place, but requirements changes for the current increment are not accepted.

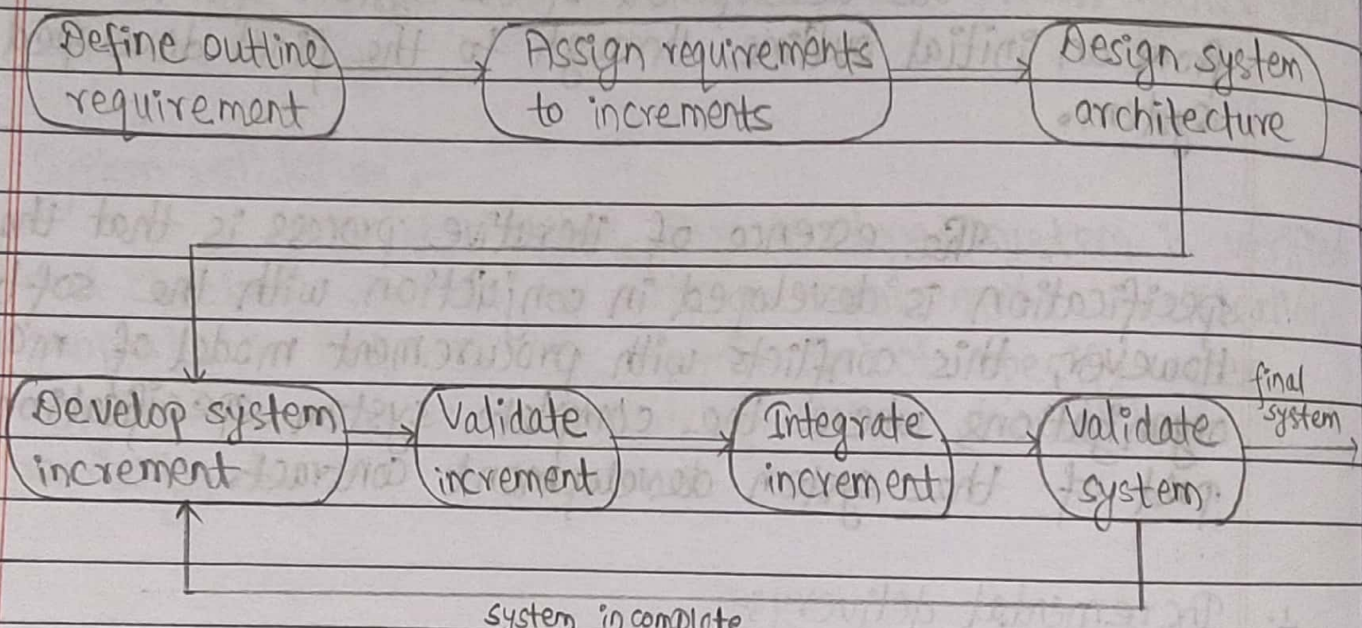


Fig:- Incremental delivery

- Once increment is completed and delivered customer can put it into services. This means that they take early delivery of part of system functionality. They can experiment with the system that helps them clearify their requirement for later increments and for later versions of the current increment. As new increment are completed, they are integrated with existing increments so that the system functionality improves with each delivered increment. The common services may be implemented early in the process or may be implemented incrementally as functionality is required by an increment.

Advantages of incremental delivery:

i) Customers do not have to wait until the entire system is delivered, before they can gain value from it. The first increment satisfy their most critical requirement so they can use software immediately.

ii) Early increment act as prototype, which helps to elicit requirements for later increments.

iii) Lower risk of overall project failure.

iv) The highest priority system services tends to receive the most testing.

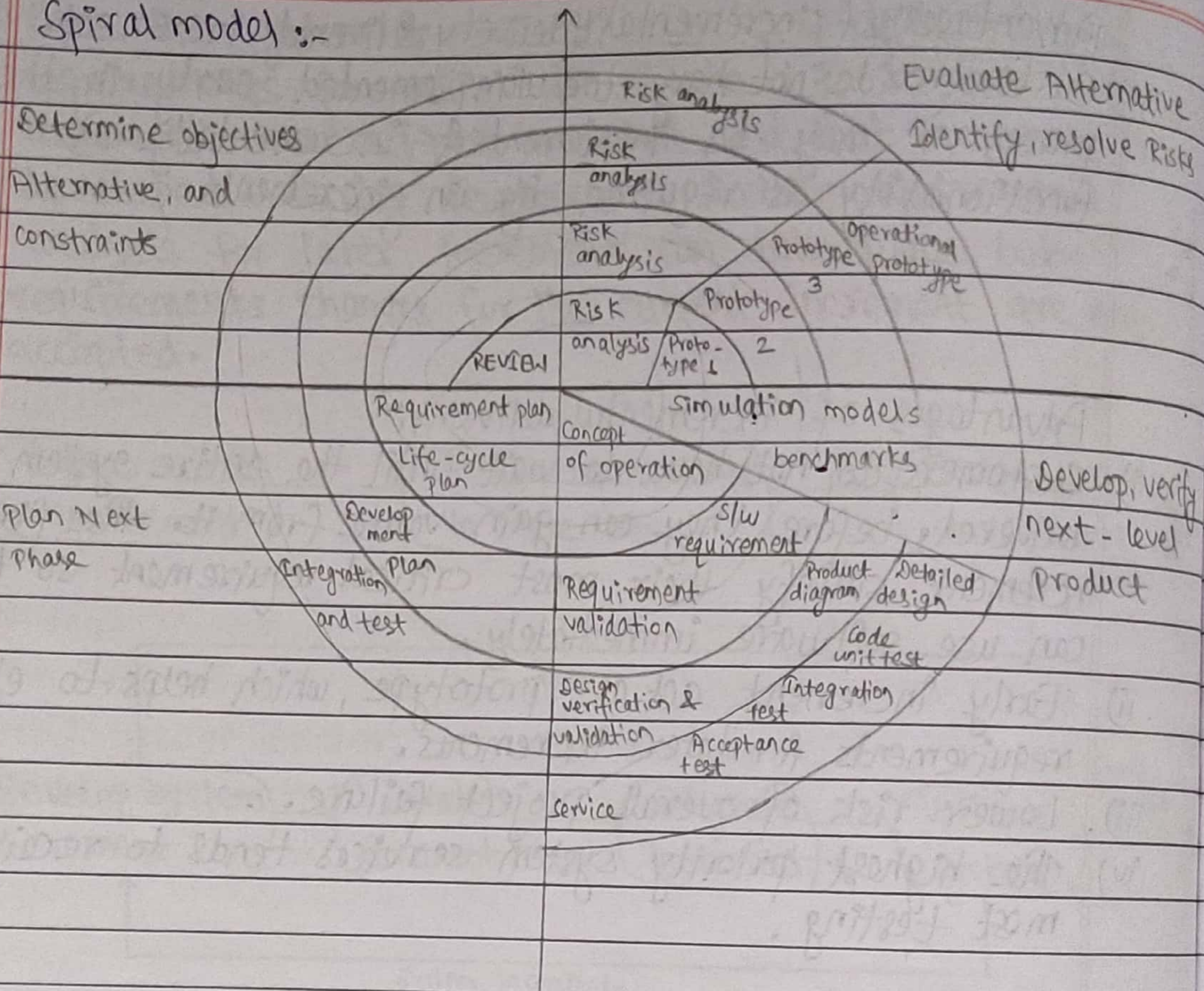
Disadvantages of incremental delivery:

i) As a product is delivered in parts, total development cost is higher.

ii) Well defined interfaces are required to connect modules developed with each phase.

iii) Testing of modules also result into overhead and increased cost.

2. Spiral model :-



- The spiral model was originally proposed by Boehm in 1988.
- In this model, process is represented as a spiral rather than as a sequence of activities with back tracking from one activity to another.
- Each loop in spiral represent a phase in the process.
- No fixed phases such as specification or design - loops in the spiral are choosen depending on what is required.
- Risks are explicitly assessed and resolved through the process.

Spiral model sector:
Each loop in the spiral model is splitted into four sector:

a) Objective setting:
Specific objectives for that phase of the project are defined. Constraints on the process and the product are identified and detailed management plan is drawn up. Project risk are identified throughly. Alternative strategies, depending on these risk may be planned.

b) Risk assessment and reduction:
For each identified project risk, a detailed analysis is carried out, steps are taken to reduce the risk. For example, if there is a risk that requirement are inappropriate, a prototype system may be developed.

c) Development and validation:
After a risk evaluation, a development model for the system is choosen. For example, if user interface risk are dominant, appropriate development model might be evolutionary prototyping. The waterfall model may be most appropriate development model if the main identified risk is subsystem integration.
If the safety risk are main consideration, development based on formal transformation may be the most appropriate and so on.

d) Planning:
The project is reviewed and decision is made whether to continue with a further loop of the spiral. If it is decided to continue, plans are drawn up for the next phase of the project.

Advantages

i) The model tries to resolve all possible risk involved in the project starting with the highest risk.
ii) End users get a chance to see the product early in life cycle.
iii) With each phase as product is refined of customer feedback, the model ensures a good quality product.
iv) The model makes use of techniques like reuse, prototyping and component based design.

Disadvantages:

i) The model requires expertise in risk management and excellent management skills.
ii) This model is not suitable for small project as cost of risk analysis may exceed the actual cost of the project.
iii) Different person involved in the project may find it complex to use.

CASE (Computer Aided Software Engineering):

Computer Aided Software Engineering (CASE) tools are software programs that automate or support the drawing and analysis of the system model and provide for the translation of system models into application programs. Some CASE tool also provide prototyping and code generation capabilities.

A CASE repository is a system developers database. It is a place where developer can store system models, detailed description and specification, and other products of system development.

CASE Facilities

To use the repository, the CASE tools provide some combination of the following facilities:

- Diagramming tools:

These are used to draw the system models required or recommended in most system development methodologies. Usually, the shapes on one system model can be linked to other models and detail descriptions.

- Dictionary tools:

Dictionary tools are used to record, delete, edit and output detailed documentation. The descriptions can be associated with shapes appearing on system models that were drawn with diagramming tools.

- Design tools:

Design tools can be used to develop system components such as inputs and outputs.

- Quality management tools:

Quality management tools analyze system model descriptions and specifications, and design for completeness, consistency and conformance to accepted rules of the methodologies.

- Documentation tools:

Documentation tools are used to assemble, organize and report on system models, descriptions and specifications and prototype that can be reviewed by system owner, users designer and builders.

- Design and code generation tools:
Design and code generation tools automatically generate database design and application programs on significant portions of those programs.

Forward engineering requires the systems analyst to draw system models, either from scratch or from templates. The resulting models are subsequently transformed into program code.

Reverse engineering allow a CASE tool to read existing program code and transform that code into a representative system model that can be edited and refined by the system analyst.

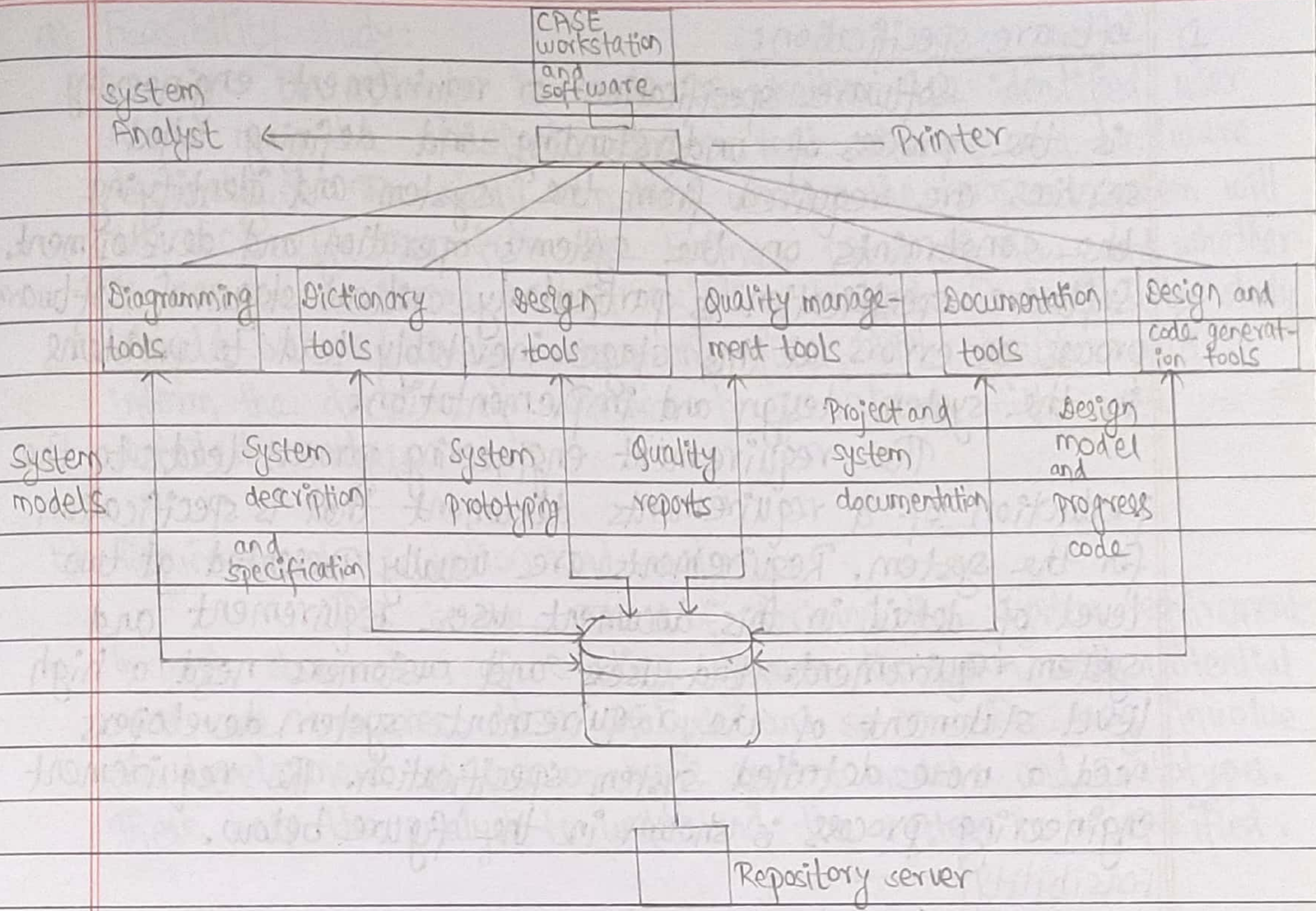


Fig :- CASE tools

Software activities :-

(or software specification)

Four basic software activities are requirement specification, development, validation, and evolution are organized differently in different software process model.

In the waterfall model, they are organized in sequence whereas in evolutionary development they are interleaved. How these activities are carried out depends on the type of software people and organization structures involved. There is no right or wrong way to organize these activities.

1) Software specification:

Software specification or requirement engineering is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development. Software specification is particularly critical stage of software process as errors at this stage inevitably lead to problems in the system design and implementation.

The requirement engineering process leads to the production of a requirements document that is specification for the system. Requirements are usually presented at two level of detail in this document user requirement and system requirement. End-users and customers need a high level statement of the requirement system developers need a more detailed system specification. The requirement engineering process is shown in the figure below.

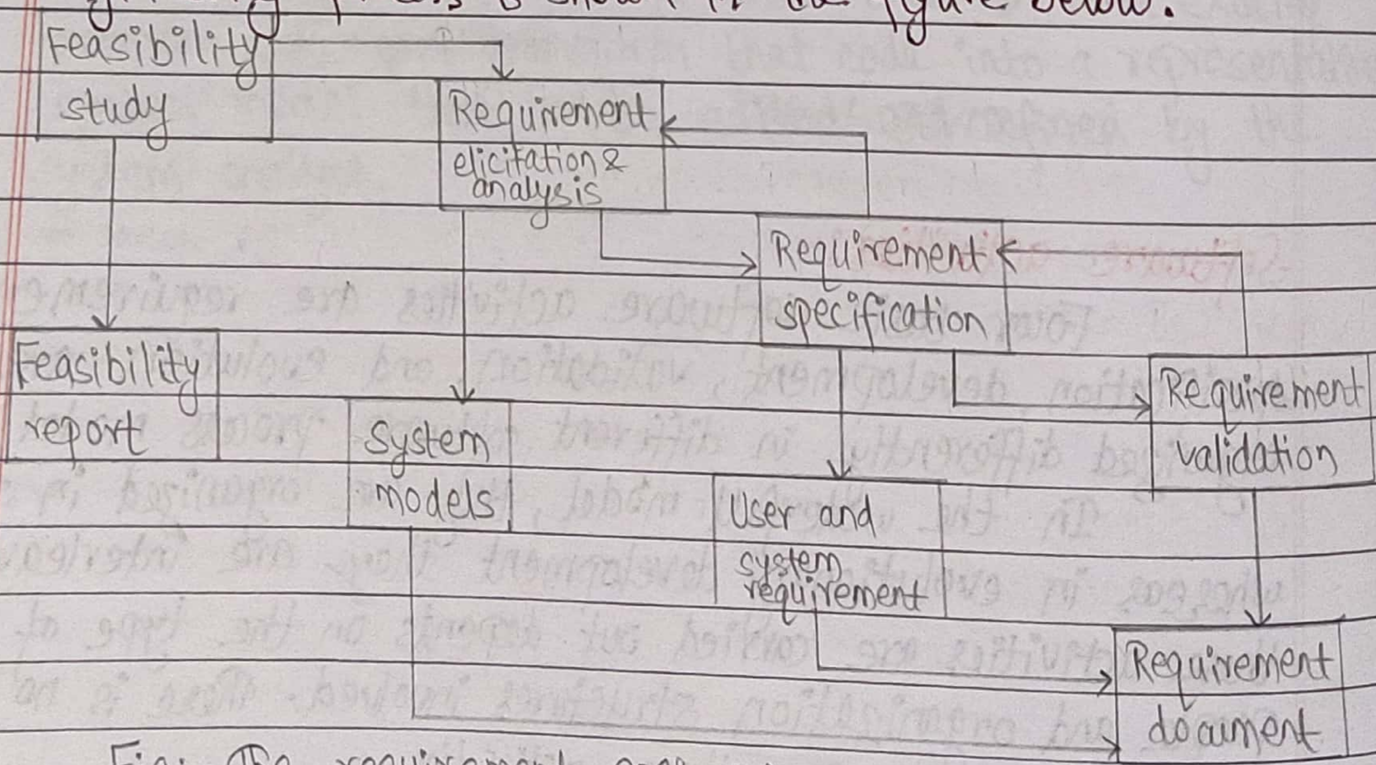


Fig:- The requirement engineering process

a) Feasibility study:

An estimate is made of whether the identified user needs may be satisfied using current software and hardware technologies. The study consider whether the proposed system will be cost effective from the business point of view and whether it can be developed within existing budget. A feasibility study should be relatively cheap and quick. The result should inform the decision of whether to go ahead with more detailed analysis.

b) Requirement elicitation and analysis:

This is the process of deriving the system requirements through observation of existing systems, discussion with potential users are procures, task analysis and so on. This may involve the development of one or more system models and prototypes. These help the analyst to understand the system to be specified.

c) Requirement specification:

The activity of translating the information gathered during the analysis activity into document that defines the set of requirement. Two types of requirement are included in this document: user requirement and system requirement. User requirement are abstract statements of the system requirements for the customer and user of the system. System requirement are more detailed description of the functionality to be provided

d) Requirement validation:

This activity checks the requirements for realism, consistency and completeness. During this process, error in the requirement document are discovered. It must then be modified to correct these problems.

The activities in the requirement process are not simply carried out in a strict sequence. Requirement analysis continues during definition and specification, and new requirements come to light throughout the process. Therefore, the activities of analysis, definition and specification are interleaved. In agile methods such as extreme programming requirements are developed incrementally according to user priorities, and the elicitation of the requirement comes from users who are the part of development plan.

2) Software design and implementation:

The implementation stage of software development is the process of converting software specification into an executable software. If an incremental approach is used, it may also involve refinement of the software specification.

A software design is the description of the structure of the software to be implemented, data models, interfaces between system component and may be algorithms used. The software designers develop the software design iteratively; they add formality, details and correct the design.

The design process may involve developing several models of the system at different level of abstraction. As a design is decomposed, errors and omissions in earlier stage are discovered. These feedback to allow earlier design models to be improved. The figure given below is a model of this process showing the design descriptions that may be produced at various stage of design. The diagram suggests that the stages of design process are sequential. Feedback from one stage to another and consequent design is inevitable in all, design process.

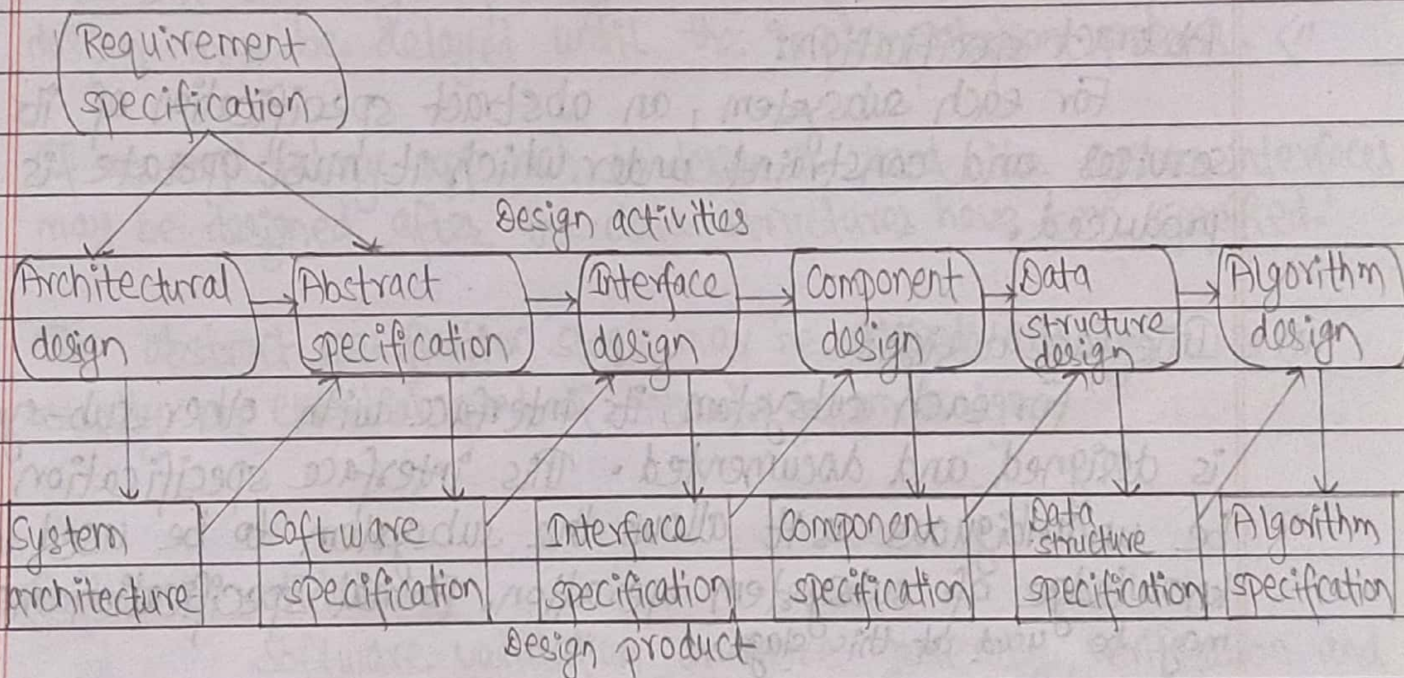


Fig:- General model of design process.

A specification for the next stage is the output of each design activity. This specification may be an abstract, formal specification that is produced to clarify the requirement or it may be specification of how part of the system is to be realized. As the design process continues, these specification becomes more detailed. The final result of the process are precise specification of the algorithm and data structures to be implemented.

The specific design process activities are:

i) Architectural design:
The subsystem making up the system and their relationships are identified and documented.

ii) Abstract specification:
For each subsystem, an abstract specification of its services and constraint under which it must operate is produced.

iii) Interface design:
For each subsystem, its interface with other sub-system is designed and documented. This interface specification must be unambiguous as it allows the subsystem to be used without knowledge of subsystem operation. Formal specification method may be used at this stage.

iv) Component design:
Services are allocated to components and interfaces of these components are designed.

v) Data structure design:
The data structures used in system implementation

are designed in detail and specified.

vi) Algorithm design:
The algorithms used to provide services are designed in detail and specified.

This is a general model of the design process and real practical processes may adapt it in different ways. Possible adaptation are:

i) The last two stage of design - data structure and algorithm design may be delayed until the implementation process.

ii) If an exploratory approach to design is used, the system interfaces may be designed after the data structures have been specified.

iii) The abstract specification stage may be skipped, although it is usually an essential part of critical system design.

3) Software validation:
Software validation or more generally, verification and validation is intended to show that a system confirms to its specification and that the system meets the expectation of customer buying the system. It involves checking processes, such as inspections and reviews, at each stage of software process from user requirements definition to program development. The majority of validation cost, however, are incured after

implementation when the operational system is tested.

The system should not be tested as a single, monolithic unit, expect for small programs. The following figure shows a three-stage testing process where system component are tested, the integrated system is tested and finally the system is testom with the customer data.

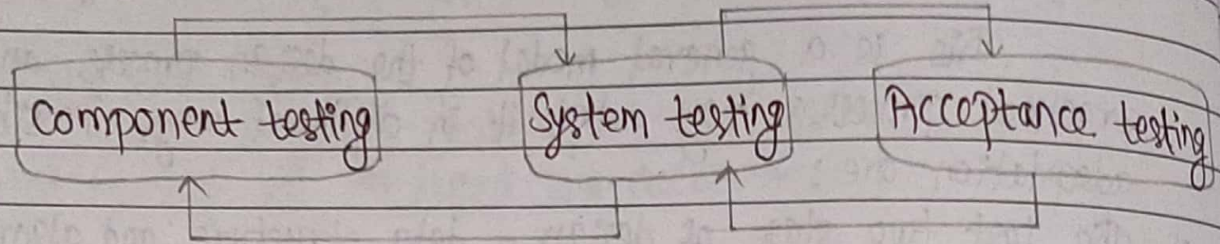


Fig:- The testing process

The stages in testing process are :

i) Component testing :

Individual components are tested to ensure that they operate correctly. Each component is tested independently, without other system components. Components may be simple entities such as function or object classes or may be coherent grouping of these entities.

ii) System testing :

The components are integrated to make up the system. This process is concerned with finding errors that results from unanticipated interactions between components and component interface problems. It is also concerned with validating that the system meets its functional and non-functional requirements and testing the emergent system properties.

iii) Acceptance testing:

This is the final stage in the testing process before the system is accepted for operational use. The system is tested with data supplied by the system customer rather than with simulated test data. Acceptance testing may reveal errors in the system requirement definition because the real data exercise the system in different way. Acceptance testing is sometime called alpha testing. The alpha testing process continues until the system developer and client agree that delivered system is acceptable implementation of the system requirements.

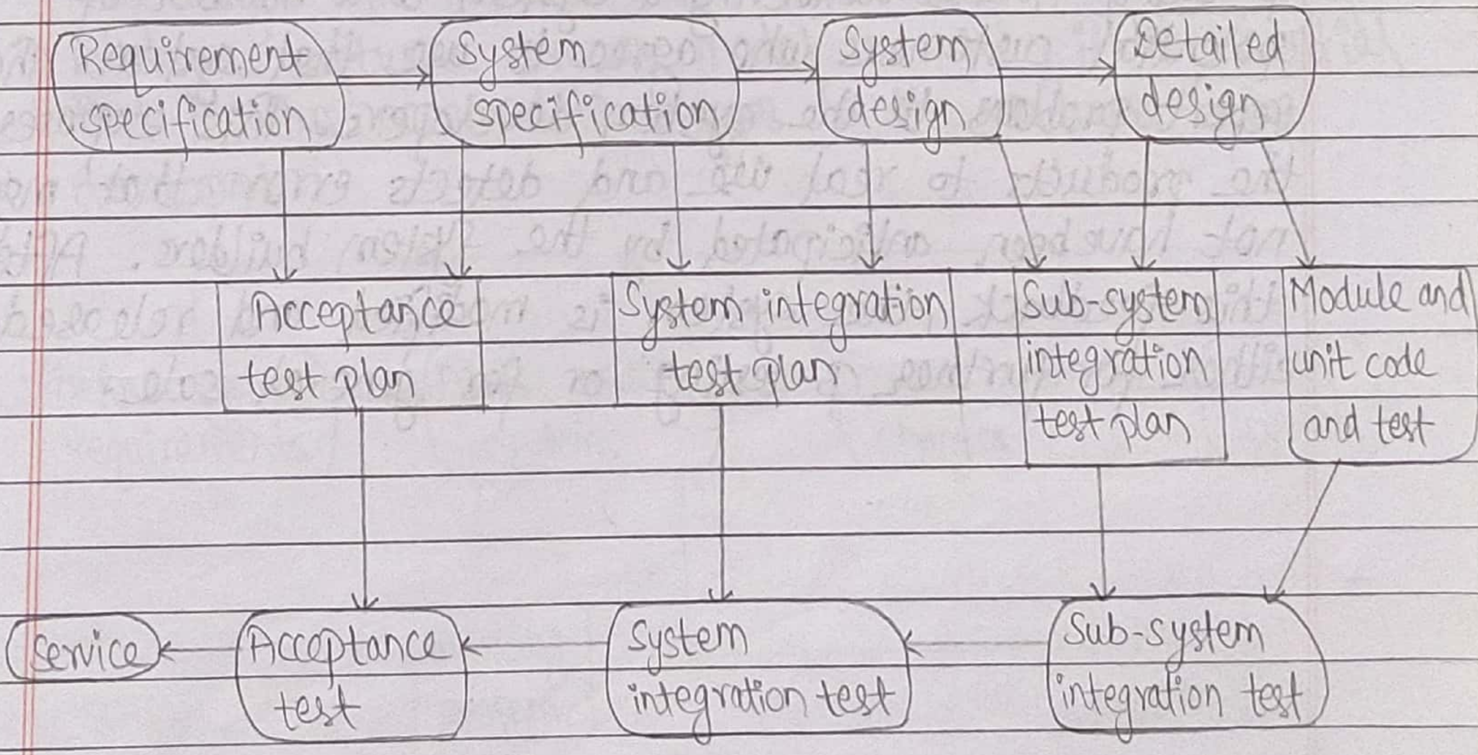


Fig:- Testing phases in software process

Normally, component development and testing process are interleaved. Programmers make up their own test data and incrementally test the code as it is develope

Alpha testing :-

Acceptance testing is sometimes called alpha testing Custom systems are developed for single client. The alpha testing process continues until the system developer and client agree that the delivered system is an acceptable implementation of system requirement.

β - testing :-

When a system is to be marketed as a software product, a testing process called β-testing is often used. β-testing involves delivering a system to a number of potential customers who agree to use that system. They report problems to the system developers. This exposes the products to real use and detects error that may not have been anticipated by the system builders. After this feedback, the system is modified and released either for further β-testing or for general sale.

4. Software Evolution

Software evolution is a process of developing s/w initially, then repeatedly updating it for various reason. The flexibility of software system is one of the main reason why more and more software is being incorporated in large, complex systems. As compared to the hardware systems, the changes and updates are more cheaper to be made in software, at any time after the system development. Even extensive changes are still much cheaper then corresponding changes to system hardware.

In all large projects, change is inevitable. The system requirement changes as the business procuring the system response to external pressure and management priorities change. As new technologies become available, new design and implementation possibilities emerge. Therefore, whatever software process model is used, it is essential that it can accomodate changes to the software being developed.

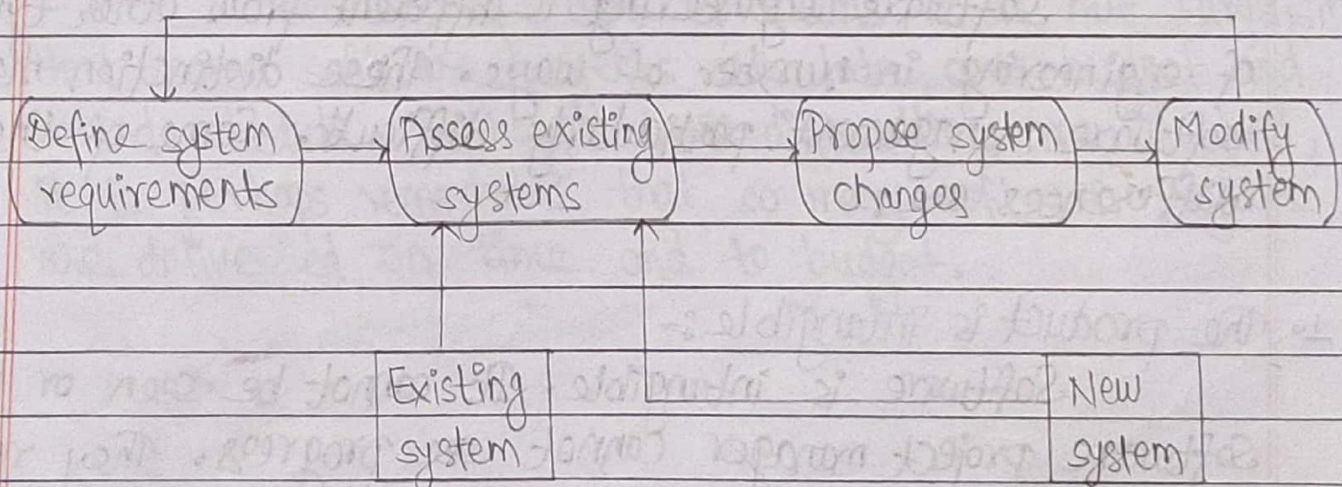


Fig :- System evolution

1.4) # Project Management

Introduction:-

Software project management is an essential part of software engineering. Good management cannot guarantee project success. However, bad management usually results in project failure: the software is delivered late, costs more than originally estimated and fails to meet its requirements.

Software manager are responsible for planning and scheduling project development. They supervise the work to ensure that it is carried out to the required standards and monitor progress to check that the development is on time and within budget. We need software project management because professional software engineering is always subject to organization budget and schedule constraints. The software project manager's job is to ensure that the software projects meet these constraints and delivers software that contributes to the goals of the company developing software.

Software engineering is different from other types of engineering in number of ways. These distinction makes software management particularly difficult. Some of these differences are:-

1. The product is intangible:-

Software is intangible. It cannot be seen or touched. Software project manager cannot see progress. They rely on other to produce the documentation needed to review progress.

2. There are no standard software process :-

In engineering disciplines with long history, the process is tried and tested. These have standard, well understood engineering process : incase of bridges and buildings. However, software processes vary dramatically from one organization to other. Although, understanding of software process has increased in recent years, we still cannot reliably predict when a particular software process is likely to cause development problems. This is specially true when the s/w project is part of wider system engineering project.

3. Large software projects are often one-off projects:-

Large software projects are usually different in some ways from previous projects. Therefore even the manager who have a large body of previous experience may find it difficult to anticipate problems. Furthermore, rapid change in technology makes manager experience obsolete.

Because of these problems, it is not surprising that some software projects are late, over budget and behind schedule. Although, these difficulties are involved, it is perhaps remarkable that so many software projects are delivered on time and to budget.

Scanned with CamScanner

Management activities:-

The job of software manager varies tremendously depending on the organization and software product being developed. However, most manager take responsibility at some stage for some or all of the following activities:

1. Proposal writing :-

The first stage in the software project may involve writing a proposal to win a contract to carry out the work. The proposal describes the objectives of the project and how it will be carried out. It usually includes cost and schedule estimates, and justifies why the project contract should be awarded to a particular organization or team. Proposal writing is a critical task, as the existence of many software organization depends on having enough proposals accepted and contracts awarded.

2. Project planning and scheduling :-

Project planning is concerned with identifying activities, milestone and deliverables produce by a project. A plan is drawn up to guide the development towards the project goals.

3. Project cost :-

Cost estimation is related activity that is concerned with estimating the resources required to accomplish the project plan. Project costs are calculated during the planning phase of a project and must be approved before work begins.

4. Project Monitoring and reviews:-

Project monitoring is a continuing project activity. The manager must keep track of the progress of the project and compare actual and planned progress and costs. Although most organization have formal mechanism for monitoring, a skilled manager can often form a clear picture of what is going on through informal discussion with project staff.

During a project it is normal to have a number of formal project management reviews. They are concerned with reviewing overall progress and technical development of the project and checking whether the project and goals of the organization paying for the software are still aligned.

5. Personnel selection and evaluation:-

Project manager usually have to select people to work on their project. Ideally, skilled staff with appropriate experience will be available to work on the project. However, in most cases, manager have to settle for less-than-ideal project team. The reason for these are

a) The project budget may not cover the use of highly paid staff. Less-experienced, less-well paid staff may have to be used.

b) Staff with the appropriate experience may not be available either within on organization or externally. It may be impossible to recruit new staff to the project. Within the organization, the best people may already be allocated to other projects.

c) The organization may wish to develop the skill of its employees. Inexperienced staff may be assigned to project to learn and gain experience.

6. Report writing and presentation :-
Project manager are usually responsible for reporting on the project to both for client and contractor organization. They have to write concise, clear (coherent) document that abstract critical information from the detail project report. They must be able to present this information during progress reviews.

Project Planning :-
Effective management of software project depends on thoroughly planning the progress of the project. Manager must anticipate problems that might arise and prepare the tentative solutions to those problems. A plan drawn up at the start of project, should be used as the driver for the project. This initial plan should be the best possible plan, given the available information. It evolves as the project progress and better information become available. Planning is an iterative process; which is only complete when the project itself is complete. As project information becomes available during the project, the plan should be regularly revised.

At the beginning of planning process, we should assess the constraints (required delivery date, staff available, overall budget, etc.) affecting the project. In conjunction with this, we should estimate project parameters such as its structure, size and distribution of functions. We next define the progress milestone and deliverables. The process then enters a loop. An estimated schedule for the project and the activities defined in the schedule are started or given permission to continue. After sometime, the progress should be reviewed and discrepancies should be noted from planned schedule. Because initial estimates of project parameters are tentative, we always have to modify the original plan.

As more information become available, we need to revise our original assumption about the project and project schedule. If it is seen that the project cannot be completed in given schedule, the review of the project is done to find an alternative approach that falls within the project constraints and meet the schedule.

The pseudo code below sets out a project planning process for software development. It shows that the planning is an iterative process, which is only completed when the project itself is completed.

```
↳ establish the project constraints
↳ Make the initial assesments of the project parameters
↳ Define the project milestones and deliverables.
↳ While project has not been completed or cancelled loop
      Draw up the project schedule
      Initiate activities according to schedule
      wait (for a while)
      Review project progress
      Revise estimates of project parameter
      Update the project schedule
      Renegotiate project constraint and deliverables
      if (problem arises) then
            initiate technical review and possible revision
      end if
end loop
```

Fig:- Project planning

Project plan :-

The project plan sets out the resources available to the project, the work breakdown and a schedule for carrying out the work. In some organizations, the project plan is a single document that includes the different types of plan. The details of the project plan vary depending on the type of project and organization. However, most plans should include the following section.

1. Introduction :
This briefly describes the objectives of the project and sets out the constraints (e.g. budget, time) that affect the project management.

2. Project organization :
This describes the way in which the development team is organized, the people involved and their role in the team.

3. Risk analysis :
This describes possible project risks, the likelihood of these risk arising and risk reduction strategies that are proposed.

4. Hardware and software resource requirements :
This specifies the hardware and support software required to carry out the development. If hardware has to be bought, estimate of the prices and delivery schedule should be included.

5. Work breakdown:

This sets out the breakdown of the project into activities and identify the milestone and deliverables associated with each activity.

6. Project schedule:

This shows the dependencies between activities, the estimated time required to reach each milestone and allocation of people to activities.

7. Monitoring the reporting mechanism:

This section defines the management report that should be produced, when those should be produced and project monitoring mechanism used.

~~Ma~~ The project plan should be regularly revised during the project. Some part such as project schedule will change frequently other part will be more stable. To simplify revisions, the document should be organized into seperate section that can individually replaced as plan ~~#~~ evolves.

Milestones and deliverables:

Information is required for managers to do their jobs. Because software is intangible, this information can only be provided as reports and documents that describe the state of the software being developed. Without this information, it is impossible to assess how well the work is progressing and cost estimate and schedules cannot be updated.

When planning a project, we should establish a series of milestones, where a milestone is a recognizable end-point of a software activity. At each milestone, there should be formal output, such as a report, that can be presented to management. The milestone may be short containing report of what has been completed. Milestone should represent the end of a, distinct, logical stage in the project.

- A deliverables is a project result that is delivered to the customer. It is usually delivered at the end of major project phase such as specification or design. Deliverables are usually milestones, but milestones need not be deliverables.

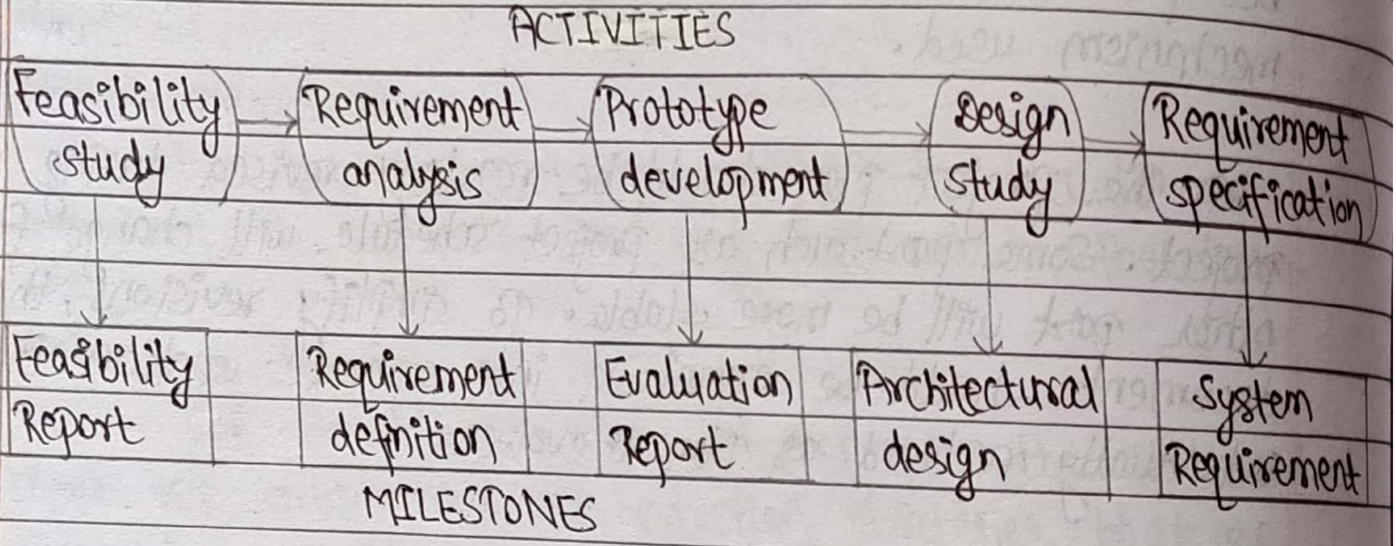


Fig:- Milestones in the requirements process

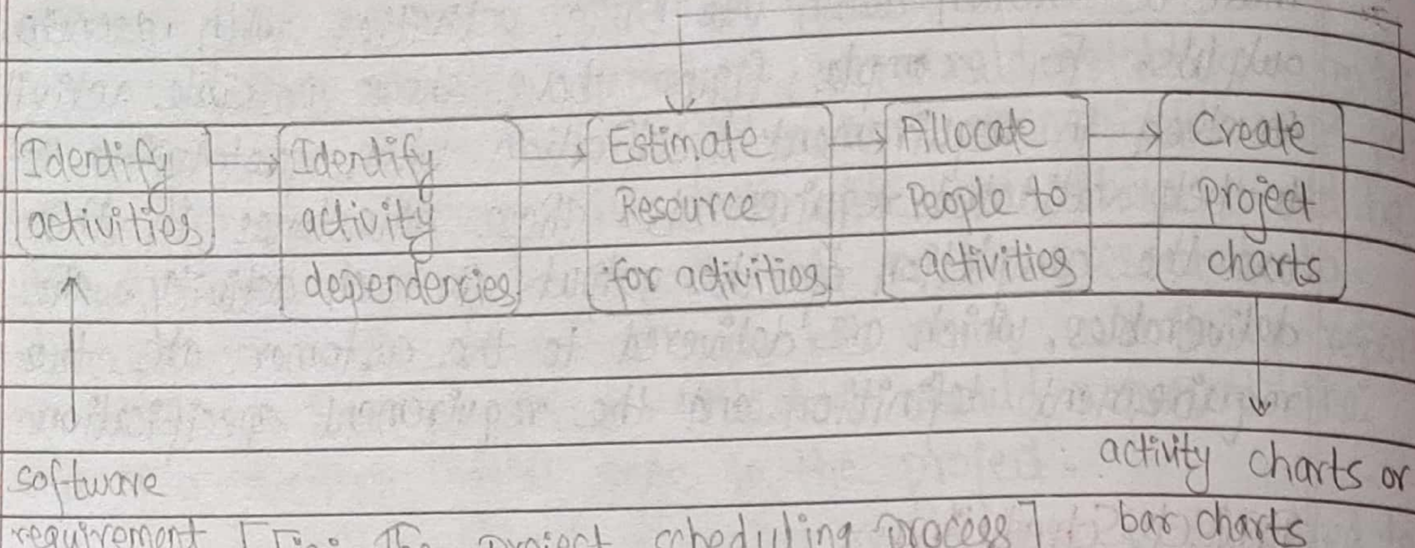
Milestones may be internal project result that are used by the project manager to check project progress but which are not delivered to customer.

To establish milestones, the software process must be broken down into basic activities with associated outputs. For example; figure above shows possible activities involved in requirement specification when prototyping is used to help validated requirements. These milestones in this case are the completion of the outputs for each activity. The project deliverables, which are delivered to the customer, are the requirement definition and the requirement specification

Project Scheduling:

Project scheduling is one of the most difficult jobs for the project manager. Managers estimate the time and resources required to complete activities and organize them into coherent sequence. Unless the project being scheduled is similar to previous project, previous estimates are an uncertain basis for new project scheduling. Schedule estimation is further complicated by the fact that different projects may use different design methods and implementation language.

Project scheduling (as shown in figure) involves separating the total work involved in a project into seperate activities and judging the time required to complete these activities. Usually, some of these activities are carried out in parallel. For optimal use of the work force, these parallel activities need to be coordinated. It is important to avoid the situation where the whole project is delayed because of critical task is unfinished.



[Fig:- The project scheduling process]

When we are estimating schedules, we should not assume that every stage of the project will be project free.

People working on project may fall ill or may leave, hardware may breakdown, an essential support s/w or h/w may be deliver late. If the project is new and technically advanced, certain parts of it may turn out to be more difficult and take longer than originally anticipated.

We also have to estimate the resources needed to complete each task. The principle resource is the human effort required. Other resources may be the disk space required on a server, the time required on a specialized hardware such as simulator, and the travel budget required for project staff.

A good thumb rule is to estimate as if nothing will go wrong, then increase our estimate to cover anticipated problems. A further contingency factor to cover unanticipated problems may be added to the estimate. This tra contingency factor depends on the type of project, the process parameter (deadline, standards, etc.) and quality and experience of the software engineers working on the project.

Project schedules are usually represented as a set of charts showing the work breakdown, activities dependencies and staff allocations.

Risk Management :-

Risk management is increasingly seen as one of the main jobs of ~~the~~ project manager. It involves anticipating risks that might affect the project schedule or quality of the software being developed and taking the action to avoid these risk. The result of risk analysis should be documented in the project plan along with an analysis of the consequences of a risk occuring. Effective risk management makes it easier to cope with problems and to ensure that these do not lead to unacceptable budget or schedule slipage.

A risk may threaten the project, the software that is being developed or the organization. There are, therefore, three related categories of risk:

1. Project risk :

Project risks are risks that affect the project schedule or resources. An example may be the loss of an experienced designer.

2. Product risk :

Product risks are risks that affect the quality or performance of the software being developed. An example might be failure of purchased component to perform as expected.

3. Business Risk:

Business risks are the risks that affect the organization developing or procuring the software. For example, a competitor introducing a new product is a business risk.

The above given risk are interrelated and most often overlap with each other. For instance, if an experienced programmer leaves a project, this can be a project risk because the delivery of the system may be delayed. It can also be a product risk because a replacement may not be as experienced so may make programming errors. Finally, it can be a business risk because the programmer's experience is not available for future business.

Risk management is particularly important for s/w projects because of the inherent uncertainties that must project face. These stem from loosely defined requirements, difficulties in estimating the time and resources required for software development, dependence on individual skills and requirement changes due to changes in customer needs.

Possible software risks:

Risk Type : Project

Risk	Description
Staff turnover	Experienced staff will leave the project before it is finished.
Management change	There will be a change of organizational management with different priorities
Hardware unavailability	Hardware which is essential for the project will not be delivered on schedule.

Risk Type : Project and Product

Risk	Description
Requirement change	There will be a larger number of changes to the requirements than anticipated.
Specification delay	Specifications of essential interfaces are not available on schedule.
Size underestimate	The size of the system has been under estimated.

Risk Type : Product

Risk	Description
CASE tool under performance	CASE tool which support the project do not perform as anticipated.

Risk Type : Business

Risk	Description
Technology change	The underlying technology on which the system is built suspended by new technology.
Product competition	A competitive product is marketed before the system is completed.

Risk Management Process :-

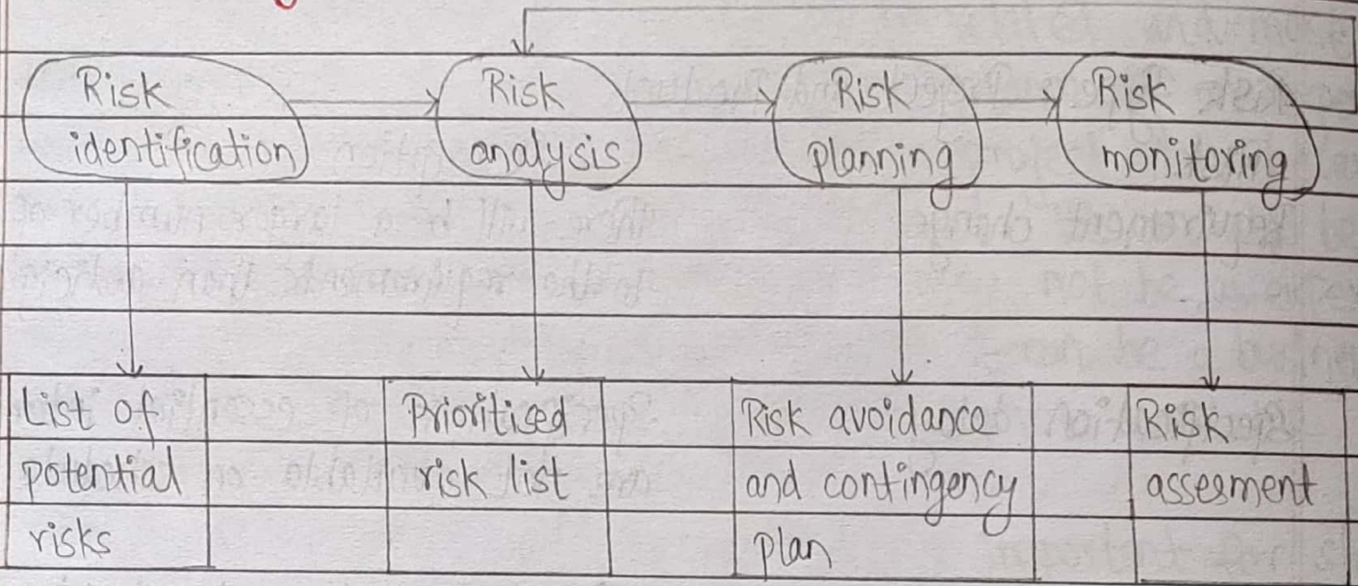


Fig:- The risk management process

The process of risk management is illustrated in above figure. It involves several stages:

1. Risk identification :-

Risk identification is the first stage of risk management. It is concerned with discovering and identifying possible risk to the project. Generally, possible project, product and business risks are identified.

There are at least 6 types of risks that can arise:

a) Technology risk :- Risk that derive from the software or h/w technologies that are used to develop the system.

b) People risk :- Risk that are associated with the people in the development team.

c) Organizational risk :- Risk that derive from the organizational environment where the s/w is being developed.

d) Tools risk :- Risk that derive from the CASE tools and other support s/w used to develop the system.

e) Requirement risk :- Risk that derive from changes to the customer requirements and the process of managing the requirement change.

f) Estimation risk :- Risk that derive from the management estimates of the system characteristics and the resources required to build the system.

2. Risk analysis :-

Risk analysis is the process of analyzing the risk that has been identified and making a judgement about the probability and the seriousness of it. This process is based on our own judgement and experience, which is why experienced project managers are generally the best people to help with risk management. These risk estimates should not generally be precised numeric assessments but should be based around a number of bands.

- The probability of the risk might be assesed as very low (< 10%), low (10-25%), moderate (25-50%), high (50-75%) or very high (> 75%).
- The effects of the risk might be assessed as catastrophic, serious, tolerable or insignificant.

3. Risk planning :-

The risk planning process considers each of the risks that have been identified and identifies strategies to mange the risk. There is no simple process that can follow to establish risk management plans. It relies on the judgement and experience of the project manager.

The following three strategies can be used for risk planning:

a) Avoidance strategies: Following these strategies means that the probability that the risk will arise will be reduced. For example: replace potentially defective components with components of non-reliability.

b) Minimization strategies: Following these strategies means that the impact of the risk will be reduced. Example: a risk minimization strategies is that for staff illness; re-organize team so that there is more overlap of work and people therefore they understand each other's job.

c) Contingency plan : Following these strategies means that we are prepared for the worst and have a strategy in place to deal with it. An example of a contingency strategy is the strategy for organizational financial problems. Example : Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.

We can see here the analogy with the strategies used in critical system to ensure reliability, security and safety. Essentially, it is best to use a strategy that avoids the risks. If this is not possible, use one that reduces the chances that the risk will have serious effects. Finally, have strategies in place that reduce the overall impact of a risk on the project or product.

4. Risk monitoring:-

Risk monitoring involves regularly assessing each of the identified risk to decide whether or not that risk is becoming more or less probable and whether effects of the risk have changed. Of course, this cannot usually be observed directly, so we have to look at other factors that give us clues about the risk probability and its effects. This factors are obviously dependent on the types of risk.

Risk monitoring should be a continuous process, and at every management progress review, we should consider and discuss each of the key risk seperately.

UNIT 2:

2.1) Software Requirements

The requirements for a system are the descriptions of the services provided by the system and its operational constraints. These requirements reflect the needs of customer for a system that helps to solve some problem such as controlling a device, placing an order or finding information. The process of finding out, analyzing, documenting and checking services and constraints is called requirement engineering.

In some cases, a requirement is simply a high level, abstract statement of a service that the system should provide or a constraint on the system. At the other extreme, it is a detailed, formal definition of system function. To make a clear seperation between different level of description, here we will use term "user requirement" to mean the high level abstract requirement and the term "system requirements" to mean the detailed description of what the system should do.

1. User requirements:

These are statements in natural language plus diagram, of what services the system is expected to provide and the constraints under which it must operate.

2. System requirements:

These set out the system functions, services and operational constraints in detail. The system requirements document (sometime called a functional specification) should be precised.

It should define exactly what is to be implemented. It may be part of the contract between the system buyer and the software developers.

Types of requirement :

Software system requirement are often classified as functional requirement, non-functional requirement or domain requirement.

1. Functional requirements :-

These are statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations. In some cases, the functional requirements may also explicitly state what the system should not do.

The functional requirements for a system describe what the system should do. These requirement depend on the type of s/w being developed, the expected users of the software and the general approach taken by the organization when writing requirements. When expressed as user requirements, the requirement are usually described in a fairly abstract way. However, functional system requirement describe the system function in details, its input and outputs, exceptions and so on.

In principle, the functional requirement specification of the system should be both complete and consistent. Completeness means that all services required by user should be defined. Consistency means that requirements should not have contradictory definition. Example :- features of software which clients demand, business rules of the particular organization for which we are developing software.

2. Non-functional requirements :-

Non-functional requirements are requirements that are not directly concerned with the specific functions delivered by the system. They may relate to emergent system properties such as reliability, response time and storage occupancy. Alternatively, they may define constraints on the system such as the capabilities of input/output devices and the data representation used in system interfaces.

Non-functional requirements are rarely associated with individual system features. Rather, these requirements specify or constraint the emergent properties of the system. Therefore, they may specify system performance, security, availability, and other emergent properties.

Non-functional requirement arise through user needs, because of budget constraints, organizational policies, the need for interoperability with other software or hardware systems, or external factors such as safety regulation or privacy legislation.

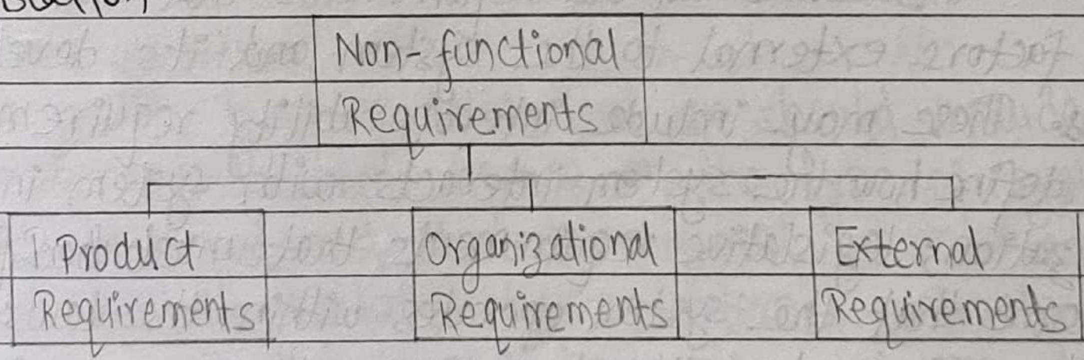


Fig:- Types of non-functional requirement

Types of non-functional requirements:

I) Product requirement :-

The requirements specify or constraint the behaviour of a software. Example include performance requirements on how fast the system must execute and how much memory it requires, reliability requirements that sets out the acceptable failure rate, security requirements, probability requirements and usability requirements.

II) Organizational requirement:-

These requirement are broad system requirements derived from policies and procedures in customer's and developer's organization. Examples include operational process requirements that define how the system will be used, development process requirements that specify the programming language, the development environment or process standard to be used, and environmental requirements that specify the operating environment of the system.

III) External requirement :-

This covers all the requirements that are derived from factors external to the system and its development process. These may include interoperability requirements that define how the system interacts with system in other organization; legislative requirements that must be followed to ensure that the system operates within the law: and ethical requirements. Ethical requirements are requirements placed on a system to ensure that it will be acceptable to its user and general public.

3. **Domain requirements:-**

These are requirement that come from the application domain of the system and that reflects characteristics and constraints of that domain. They may be functional or non-functional requirement.

Software requirement documentation:-

Software requirement documentation is the official statement of what system developer should implement. It should include both the user requirement for system and detail specification of the system requirement. The requirement has a diverse set of users ranging from senior management of the organization that is paying for the system to the engineer responsible for developing the software. The diversity of possible users means that the requirement document has to be compromise between communicating the requirement to customer, defining the requirement in precise detail for developer and tester and including information about possible system evolution.

The level of detail that we should include in a requirement document depends on the type of system that is being developed and the development process used. When the system will be developed by an external contractor, critical system specification need to be precise and very detail. When there is more flexibility in the requirements and where an in-house, iterative development process is used, the requirement document can be much less detailed and any ambiguities result during development of the system.

The IEEE standard suggest the following structure for the requirement documents.

1. Introduction:
   1.1 Purpose of the requirement document
   1.2 Scope of the product
   1.3 Definitions, acronyms and abbreviations
   1.4 References
   1.5 Overview of the remainder of the document.

2. General description:
   2.1 Product perspective
   2.2 Product functions
   2.3 User characteristics
   2.4 General constraints
   2.5 Assumption and dependencies

3. Specific requirements:

It covers functional, non-functional and interface requirement. This is obviously the most critical part of the document but because of the wide variability in organizational practice, it is not appropriate to define a standard structure for this section. The requirements may document external interfaces, describe system functionality and performance, specify logical database requirements, design constraints, emergent system properties and quality characteristics.

4. Appendices

5. Index :

Although the IEEE standard is not ideal, it contains a great deal of good advices on how to write requirements and how to avoid problems. It is too general to be an organizational standard.

Users of Requirement Document:

System customer →	Specify the requirement and read them to check that they meet their needs. Customer specifies changes to the requirements.
Manager →	Use the requirement document to plan bid for the system and to plan the system development process.
System engineers →	Use the requirement to understand what system is to be developed.
System test engineers →	Use the requirement document to develop validation plan for the system.

System maintenance engineers	→	Use the requirement document to understand the system and the relationship between its components.

Requirement engineering process :-

The goal of requirement engineering process is to create and maintain a system requirement document. The overall process includes four high level requirement engineering sub-processes. These are concern with accessing whether the system is useful to the business (feasibility study), discovering requirement and analyzing them (elicitation and analysis), converting this requirement into some standard form (specialization), and checking that the requirement actually defines the system that the customer wants (validation).

Figure below shows the relationship between these activities. It also shows the document produced at each stage of the requirement engineering process.

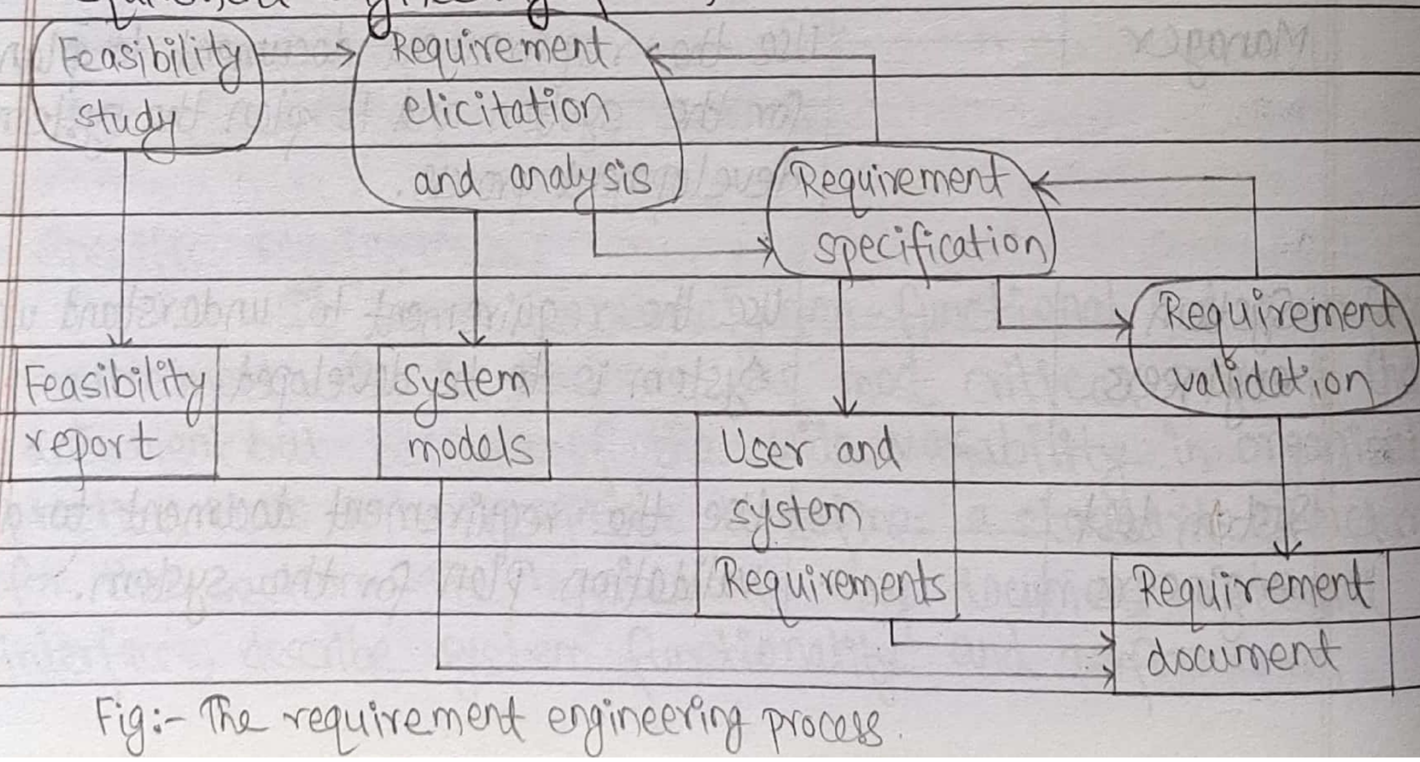


Fig:- The requirement engineering process

1. Feasibility study:

For all new systems, the requirement engineering process should start with a feasibility study. The input to the feasibility study is a set of preliminary business requirements, an outline description of the system and how the system is intended to support business process. The results of feasibility study should be a report that recommends whether or not it is worth carrying on with requirement engineering and system development process.

A feasibility study is a short, focused study that aims to answer a number of question.

a) Does the system contribute to the overall objectives of the organization?
b) Can the system be implemented using current technology and within given cost schedule constraint?
c) Can the system be integrated with other systems which are already in place?

In feasibility study, we may consult information sources such as the managers of the departments where the system will be used, software engineers who are familiar with the type of system i.e. purposed, technology expert and end users of the system.

Once we have the information, we write feasibility study report. We should make recommendation about whether or not the system development should continue. In the report, we may purpose changes to the scope, budget and schedule of the system and suggest further high level requirement for the system.

2. Requirement elicitation and analysis :

In this activity, software engineers work with customers and system end-users to find out about application domain, what services the system should provide, the required performance of the system, hardware constraints and so on.

Requirement elicitation and analysis may involve a variety of people in an organization. The term stakeholder is used to refer any person or group of persons who will be affected by the system directly or indirectly. Stakeholder may be engineers who are developing or maintaining related systems, business managers, domain experts and trade union representatives.

A very general process model of elicitation and analysis process is shown in figure below. Each organization will have its own version of instanstiation of these general model, depending on local factor such as expertise of the staff, the type of the system being developed and the standards used.

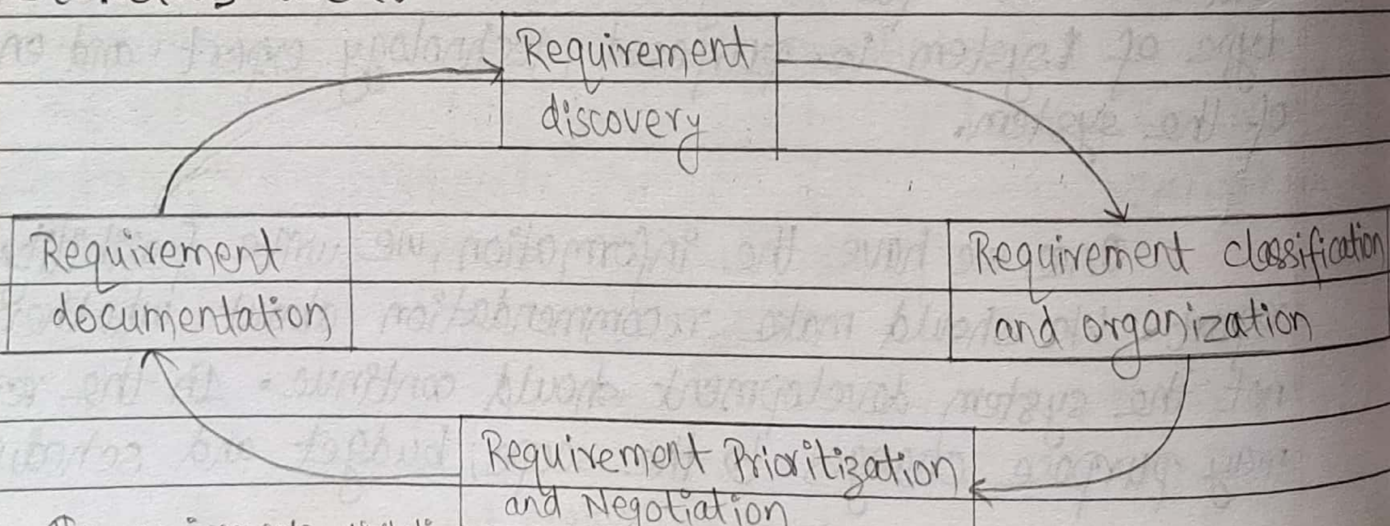


Fig:- The requirements elicitation and analysis process.

a) Requirement Discovery :- This is the process of interacting with the stakeholder in the system to collect their requirements. Domain requirements from stakeholders and documentation are also discovered during this activitity.

b) Requirement classification and organization :- This activity takes the unstructured collection of requirements, group of related requirement and organizes them into coherent clusters.

c) Requirement Prioritization and Negotiation :- Inevitably, where multiple stakeholders are involved, requirements will conflict. This activity is concerned with prioritizing requirement, finding and resolving requirement conflict through negotiation.

d) Requirement documentation :- The requirements are documented and input into the next round of the spiral. formal or informal requirements documents may be produced.

Figure above shows the requirement elicitation and analysis in an iterative process with continual feedback from each activity to other activities. The process cycle starts with requirement discovery and ends with requirement documentation. The analyst understanding of the requirements improves with each round of the cycle.

3. Requirement specification:

It is the process of writing down the user and system requirements into a document. The requirement should be clear, easy to understand, complete and consistent. The requirement specification must specify merits and demerits along with recommendation requirement. In practice, this is difficult to achieve as stakeholders interprete the requirements in different ways and there are often inherent conflicts and inconsistencies in the requirements.

4. Requirement validation:

Requirement validation is the process of checking that requirements actually defines the system that the customer really wants. It overlaps with the analysis as it is concerned with finding problems with the requirements. Requirement validation is important because errors in requirements document can lead to extensive rework cost when those problems are discovered during development or after the system is in service.

During the requirements validation process, different types of check should be carried out on the requirements in the requirements document. These checks include:

a) Validity check: A user may think that a system is needed to perform certain functions. However, further thought and analysis may identify additional or different functions that are required.

b) Consistency check: Requirements in the document should not conflict. That is, there should not be contradictory constraints a different descriptions of same system function.

c) Completeness check: The requirement document should include requirements, that define all functions and constraints intended by the system users.

d) Realism check: Using the knowledge of existing technology, the requirement should be checked to ensure that they could actually be implemented. These checks should also take account of budget and schedule for the system development.

e) Verifiability: To reduce the potential for dispute between customer and contractor, system requirements should always be written so that they are verifiable.

A number of requirement validation techniques can be used.

i) Requirement review: The requirements are analyzed systematically by a team of reviewers who checks for error and inconsistencies.

ii) Prototyping: In this approach to validation, an executable model of the system is demonstrated to end users and customers. They can experiment with this model to see if it meets their real needs.

iii) Test case generation: Requirements should be testable. If the tests for the requirement are devised as a part of validation process, these often reveals requirement problems.

Requirement Management:-

Requirement management can be defined as a process of eliciting, documenting, organizing and controlling changes to the requirements. Generally, the process of requirements management begins as soon as the requirement document is available, but 'planning' for managing the changing requirements should start during the requirements elicitation process.

The essential activities performed in requirement management are listed below:

1. Recognizing the need for change in the requirements
2. Establishing a relationship amongst stakeholders and involving them in the requirement engineering process.
3. Identifying and tracking requirement attributes.

Requirements management enables the development team to identify, control and track requirements and changes that occur as the software development process progress. Other advantages associated with the requirement management are listed below:

1. Better control of complex projects
2. Improve software quality
3. Reduce projects cost and delays
4. Improve team communication.

5. Easing compliance with standards and regulation.

2.2) Software Prototyping :-

The goal of prototyping based development process is to counter the limitation of the waterfall model. The basic idea is that instead of freezing the requirement before any designed or coding can proceed, a prototype is built to understand the requirements. The prototype is developed based on the currently known requirements. By using the prototype, the client or customer can get an actual feel of the system, because the interaction with the prototype can enable the customer to better understand the requirements of the desired system. Prototyping is an attractive idea for complicated and large system for which there is no manual process or existing system to help determine the requirements of the desired system.

In prototyping, a statement of the system requirement is completed and is used by the development team as the basis for the software. A prototype is an initial version of a software system that is used to demonstrate concept, try out design option and to find out more about the problem and its possible solutions.

A software prototype can be used in an s/w development process in several ways.

1. In the requirement engineering process, a prototype can help with the elicitation and validation of the system requirement.

2. In the system design process, a prototype can be used to explore particular s/w solution and to support user interface design.
3. In the testing process, a prototype can be used to run back to back tests with the system that will be delivered to the customer.

System prototypes allow users to see how the system will support their work. They may get new ideas for requirements and find areas of strength and weakness in the software. A software prototype may be used while the s/w being designed to carry out design requirement experiment to check the feasibility of the proposed design.

Benefits of using prototype:-
1. Improved system usability
2. A closer match of system to users need
3. Improved design quality
4. Reduce development effort.

Rapid Software Development:-

Software is part of all business operation so it is essential that new software is developed quickly to take advantage of new opportunities and to respond to competitive pressure. Rapid development is therefore often the most critical requirements for them. Software system, businesses are operating in a changing environment, it is often impossible to derive a complete set of stable software requirements.

The requirement that are proposed inevitably change because customers find it impossible to predict how a system will affect working practices, how it will interact with other system and what users operation should be automated. It may be only after a system has been delivered and user gain experience with it that the real requirements become clear.

Rapid software development processes are designed to produce useful software quickly. Generally, they are iterative processes where specification, design, development and testing are interleaved. The software is not developed and deployed in its entirety but in a series of increment, with each increment including new system functionality.

There are many approaches to rapid software development, they share some fundamental characteristics:

1. The process of specification, design and implementation are concurrent. There is no detailed system specification and design documentation is minimized or generated automatically by the programming environment used to implement the system. The user requirement document defines only the most important characteristics of the system.

2. The system is developed in a series of incremements. End users and other system stakeholders are involved in specifying and evaluating each increments. They may propose changes to the software and new requirements that should be implemented in later increments of the system.

3. System user interface are often developed using an interactive development system that allow the interface design to be quickly created by drawing and placing icon on the interface. The system may then generate a web-based interface for a browser or interface for a specific platform such as Microsoft Windows.

User Interface Prototyping :-

User interface (UI) prototyping is an iterative analysis technique in which user actively involved in the mocking-up of the UI for a system. UI prototype has several purposes:

- As an analysis artifact that enables us to explore the problem space with our stakeholders.
- As a requirements artifacts to initially envision the system.
- As a design artifacts that enables us to explore the solution space of our system.
- A vehicle for us to communicate the possible UI designs of our system.
- A potential foundation from which we can continue developing the system.

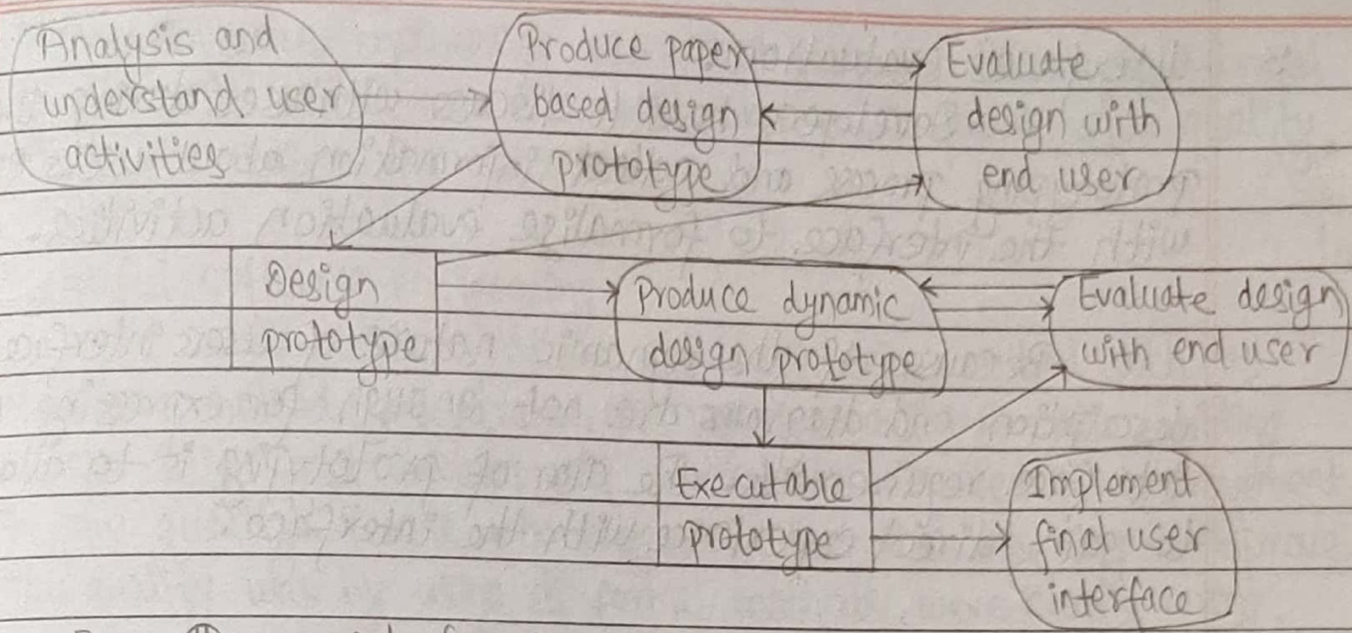


Fig:- The user interface design process

The user interface design process is described into three phases:

1. User Analysis:
The user analysis process developed an understanding of the task that user do their working environment, the other system that they use; how they interact with other people in their work and so on.

2. System Prototyping:
User interface design and development is an iterative process. Although, users may talk about the facilities they need from an interface, it is very difficult for them to specify facilities until they see something tangible. Therefore, the developer has to develop system prototype and expose them to users.

3 Interface evaluation :
Developers have to discuss with users during the prototyping process and collect information about users experience with the interface to formalize evaluation activities.

Because of the dynamic nature of users interface, textual description and diagrams are not enough for expressing user interface requirements. The aim of prototyping is to allow users to gain direct experience with the interface.

2.3) Formal Specification.

Introduction :

Formal methods of s/w development are not widely used in industrial software development. Most s/w development companies do not consider it cost effective to apply them in their s/w development process. The term formal method is used to refer to any activities that rely on mathematical representation of software including formal system specification, specification analysis and proof, transformational development and program verification.

All these activities are dependent on a formal specification expressed in a language whose vocabulary, syntax, and semantics are formally defined. A formal definition means that the specification language must be based on mathematical concept whose properties are well understand.

Many software engineering researches proposed that using formal development method was the best way to improve s/w quality, the main reason for this is:

1. Successful software engineering:
The user of other s/w engineering method such as structure method, configuration management and information hiding in s/w design and development process have resulted in improvement in s/w quality. People who suggested that the only way to improve s/w quality was by using ~~or~~ of formal methods were clearly wrong.

2. Market changes:
S/w must be developed quickly and customer are sometimes willing to accept s/w with some faults, if rapid delivery can be achieved.

3. Limited scope of formal method:
Formal methods are not well suited to specifying user interface and user interaction. The user interface component has become a greater and greater part of most system. So we can only really use formal methods when parts of the system

4. Limited scalability of formal method:
Successful project that have used formal methods have been concerned with relatively small system. As system increase in size, the time and effort require to develop a formal specification grows disproportionately.

These factor means that most s/w development companies have been unwilling to risk using formal method in their development process. However formal specification is an excellent way of discovering specification errors and presenting the system specification in precise way.

Formal Specification in Software Process:

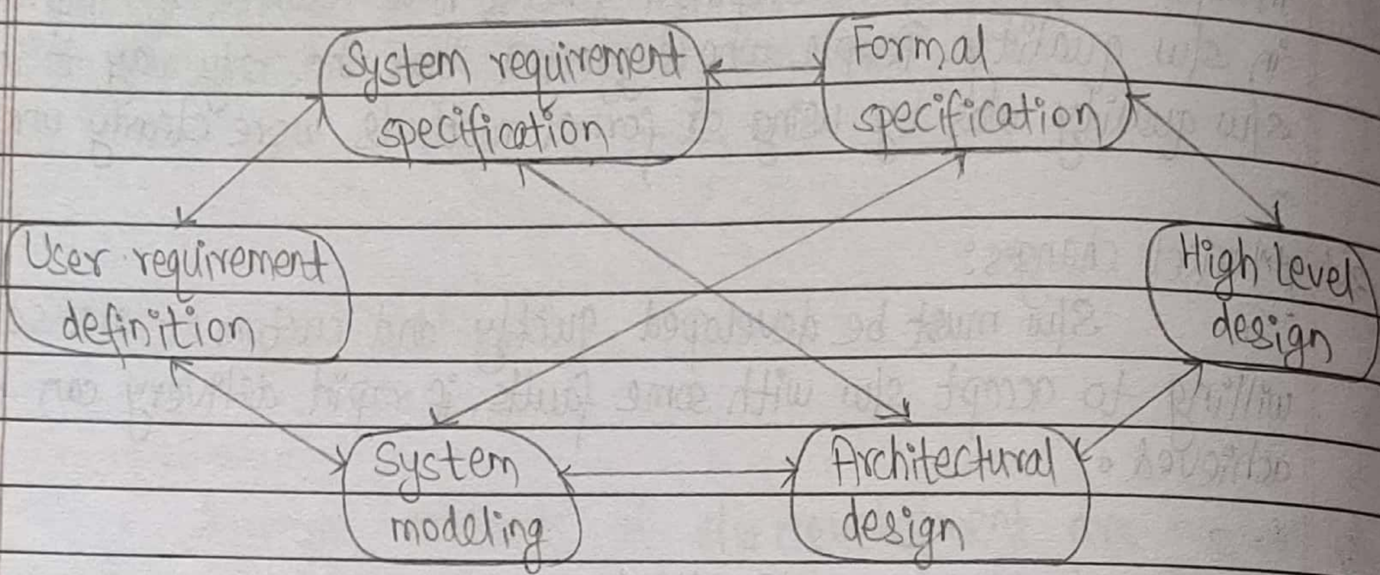


Fig:- Formal specification in the software process

The system requirement and system designs are expressed in details and carefully analyzed and checked before implementation begins. A formal specification of software is developed after the system requirement have been specified but before the detailed system designed. The main benefit of formal specification is its ability to uncover problems and ambiguities in the requirements. Creating a formal specification force to make a detailed system analysis that usually removes errors and inconsistencies in the requirement specification.

Two fundamental approaches for formal specification have been used to write detail specification for s/w system.

1. Algebraic Approach:
In algebraic approach, system is described in terms of operation and their relationship.

2. Model Based Approach:
In model based approach, the system is built using mathematical construct such as sets and sequence and the system operation are defined by how they modify the system state.

Interface Specification:-

Large systems are usually decomposed into subsystems that are developed independently. Subsystem make use of other subsystem, so an essential part of specification is to define subsystem. Once the interface are agreed and defined, the subsystem can then be designed and implemented independently. Subsystem interface are often defined as a set of object or components. These describe the data and operation that can be accessed by the ~~sy~~ subsystem interface. The process of developing a formal specification of a subsystem interface includes the following activities:

1. Specification structure:-
Organize the formal interface specification into set of abstract data type or object class. We should define the operation associated with each class.

2. Specification Name:
Establish a name for each abstract type specification. Decide whether they require generic parameters and decide name for the each object identifies.

3. Operation Selection:
Choose a set of operation based on identified interface functionality. We may have to add functions to initially identify definition.

4. Informal operation Specification:
Write an informal specification of each operation.

5. Syntax Definition:
Define the syntax of operations and the parameter to each.

6. Semantic Definition:
Define the semantics of the operation by describing what condition is usually true for different operation combination.

Behaviour Specification:-
The simple algebraic technique can be used to describe interfaces where the object state changing depending on the previous operation result. Where this condition holds we say it the behaviour properties of system. The specification which is used to specify such type of system property is called behavioural specification. As their size of system is increased, the description of system behaviour becomes increasingly difficult to understand.

Unit 3.1 Architecture Design

Introduction

For building the specified software system, designing the software architecture is a key step. Any complex software is composed of sub a system that interacts under the control of system design such that the system provides the expected behavior. While designing a software system, the logical approach is to identify the sub system that should compose the system, the interface of these sub system and the rules for interaction between the subsystem.

The initial design process of identifying these sub system and establishing a framework for sub system is called architectural design. The output of this design process is the description of the software architecture. Architectural design is the first stage in the design process and represents the critical link between the design requirement engineering processes. The architectural design process is concerned with establishing a basic structural framework that identifies the measure component of the system and the communication between these components.

Architecture is a design of system which gives a very high level view of the parts of system and how they are related to form the whole system.

Three advantages of explicitly designing and documenting software architecture are

1. Stake- holder communication

The architecture is a high level presentation of the system that may be used as a reference for discussion with different range of stakeholders.

2. System Analysis

Making the system architecture explicit at an easy stage in the software development requires some analysis. Architectural design decision has a profound effect on whether the system can meet critical requirements such performance, reliability and maintainability, security etc.

3. Reuse

A system architectural model is a compact and manageable description of how a system is organized and how the component interoperates. The component of the architectural design may be used in another software development process.

The system architecture affects the performance robustness; distribute ability and maintainability of the system.

Architectural Design Decisions

Architectural design is a creative process to establish a system organization or structure will satisfy the functional and non functional requirements. It is a creative process so the activities within the process differ radically depending on the type of system being developed, the background and experience of the system architect and the specific requirements for the system.

During the architectural design process, system architect have to make a number of fundamental decisions that affects the system and its development process. Based on their knowledge and experience, they have to answer the following fundamental questions-

- How will the system be distributed across a number of processors?
- What architectural styles are appropriate for the system?
- What will be the fundamental approach used to structure the system?
- How will the structural unit in the system be decomposed into modules?
- How will the architectural design be evaluated?
- How should the architecture of the system be documented?

The final product of architectural design process in an architectural design document. This may include a number of graphical representations of the system along with associated descriptive text. It should describe how the system is structured into sub system, the approach adopted and how each subsystem is structured into modules.

System Organization or System Structure

The organization of a system reflects the basic strategy that is used to structure a system. Software designer have to make a decision on the overall organizational model of a system only in the architectural design process. The structure of the software system depends on type of software being developed, knowledge and experience of designer, type of model used in the development process and types of customers or organization for which the software is being developed. Some system structure models are as follows-

1. Repository Model

Subsystems making of a system must exchange information So that they can work together effectively. There are two fundamental ways in which this can be done.

a. All shared data is held in a central database that can be accessed by all subsystems.
b. Each subsystem maintains its own database. Data is interchanged with other subsystem by passing massage to them.
   This model is suitable to the application when data is gathered by one subsystem and used by another subsystem.

Advantages and Disadvantages of Shared Repository Model

i. It is an efficient way to share large amount of data. There is no need to transmit data explicitly form one subsystem to another.
f. However, subsystem must agree on the repository data model. Inevitably, this is a compromise between specific need of each too. Performance may be adversely affected by this compromise. It is difficult or impossible to generate new data model if their data models do not fit the agreed schema.
g. Subsystems that produce data need not be concerned with how that data is used by other system.

   However, evolution may be difficult as a large volume of information is generated according to an agreed data model. Translating this into new model will certainly be expensive; it may difficult or even impossible.

h. Activity such as backup, security, access control and recovery from error are centralized. They are the responsibility of the repository manager. Tools can focus on their principal function rather than concerned with these issues.

   However, different sub-systems may have different requirements for security, recovery, & backup policies. The repository forces on all sub-system.

2. Client/ Server Model

The client/server architecture model is a system model where the system is organized as a set of servers and associated servers and clients that access and use the services. The major components of this model are:

a. Set of servers that offer services to other subsystem.
b. A set of clients that call the services offered by server.
c. A network that allows the client to access these services. This is not strictly necessary as the both client and server could run on a single machine.

94

Client may have to know the name of the available servers and the service that they provide. Client access the services provided by a server through the remote procedure call using a request reply protocol. A client makes a request to a server and waits until it receive a reply.

Advantages of Client-Server Model

The important advantages of client server model are that it is a distributed architecture. Effective use can be made of networked system with many distributed processors. It is easy to add new server and integrate it with the rest of the system or to upgrade servers transparently without affecting other parts of system.

3. Layered Model
   The layered model of architecture organizes a system into layers, each of which provide a set of services. Each layer can be thought of as an abstract machine whose language is defined by the services provided by the layer. This language is used to implement the next level of abstract machine. The layered approach supports the incremental development of system. As a layer is developed, some of the services provided by that layer may be made available to the users. This architecture is changeable; a layer can be replaced by another equivalent layer.

Modular Decomposition Style

After overall system organization has been chosen, we need to make a decision on the approach to be used in decomposing subsystems into modules. The components in modules are usually smaller than subsystem, which allows alternative decomposition style to be used. A subsystem is a system in its own right whose operation does not depends on the services provided by other subsystem.

Subsystems are composed of modules and have defined interface, which are used for communication with other subsystem. Module is normally a system component that provides one or more services to other modules.There are two main strategies that can be used when decomposing a subsystem into modules.

1. Object-Oriented Decomposition
   Decompose a system into set of communicating objects.

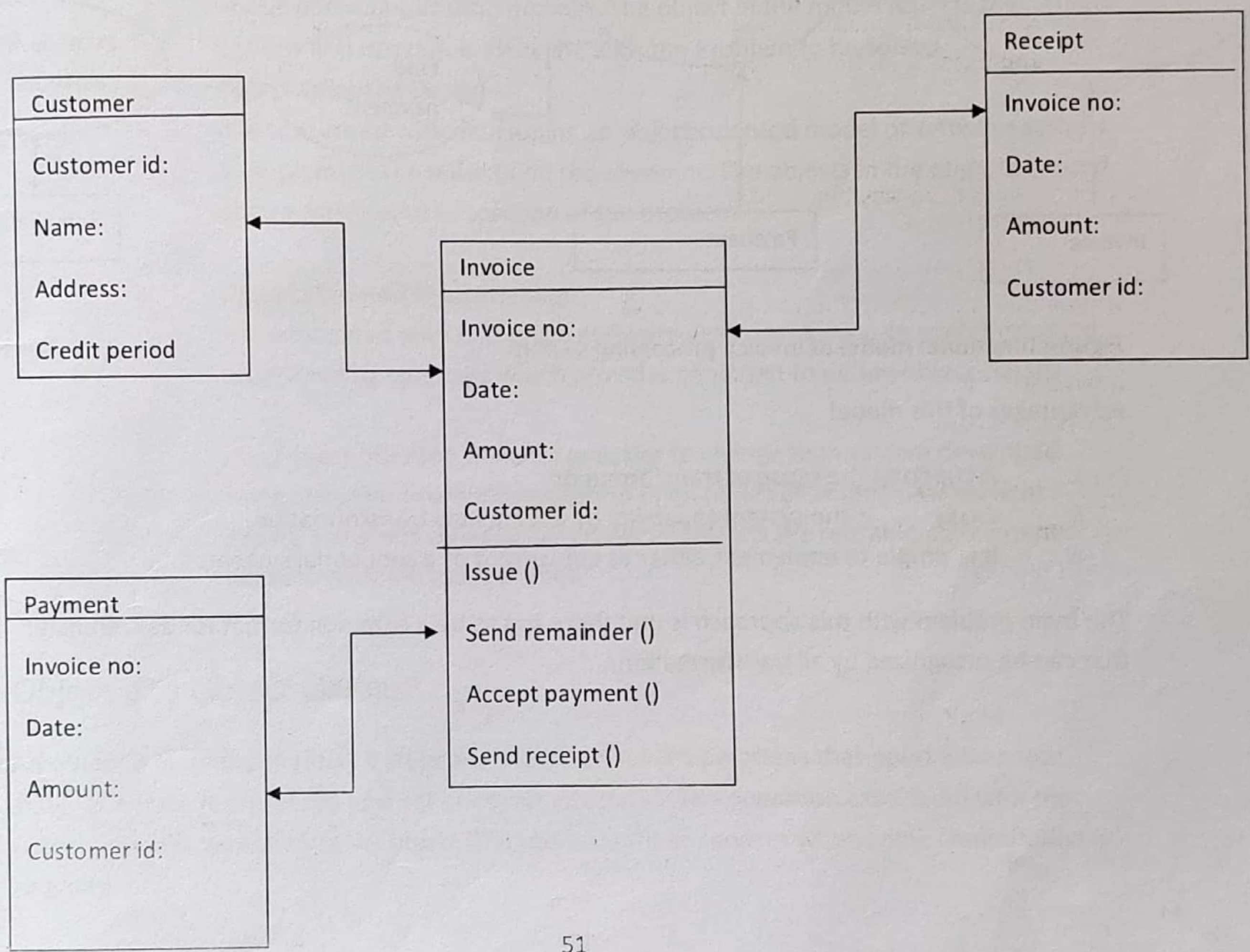
2Function Oriented Decomposition:Decompose a system into functional modules that accept input data and transform it into output data.

95

In the object –oriented approach modules are object with private state and define operation on that state. In function oriented approach, modules are functional transformation.

1. Object-Oriented Decomposition

An object oriented architecture model structures the system into the set of loosely coupled objects with well defined interface. Objects call the services offered by other objects. Object oriented decomposition is concerned with object class, their attributes and their operations. An object class is an abstraction over a set of objects that identifies common attributes and services or operation that are provided by each object. Objects are executable entities with the attributes and the services of the object-class. Object oriented approach involves identifying the classes of object that are important for the software system. A decomposition scheme shows how an object class is related to other class through common attributes and services. The advantages of object oriented approach are that objects are loosely coupled so the implementation of objects can be modified without affecting other objects.



96

Figure: an object oriented model of an invoice processing

2. Function Oriented Decomposition

Function oriented decomposition systems are organized as functional components. Functional components are called functional transformation. Functional transformation processes their inputs and produce outputs. Input data flows through these transformation until it connected to output. The transformation may execute sequentially or in parallel.

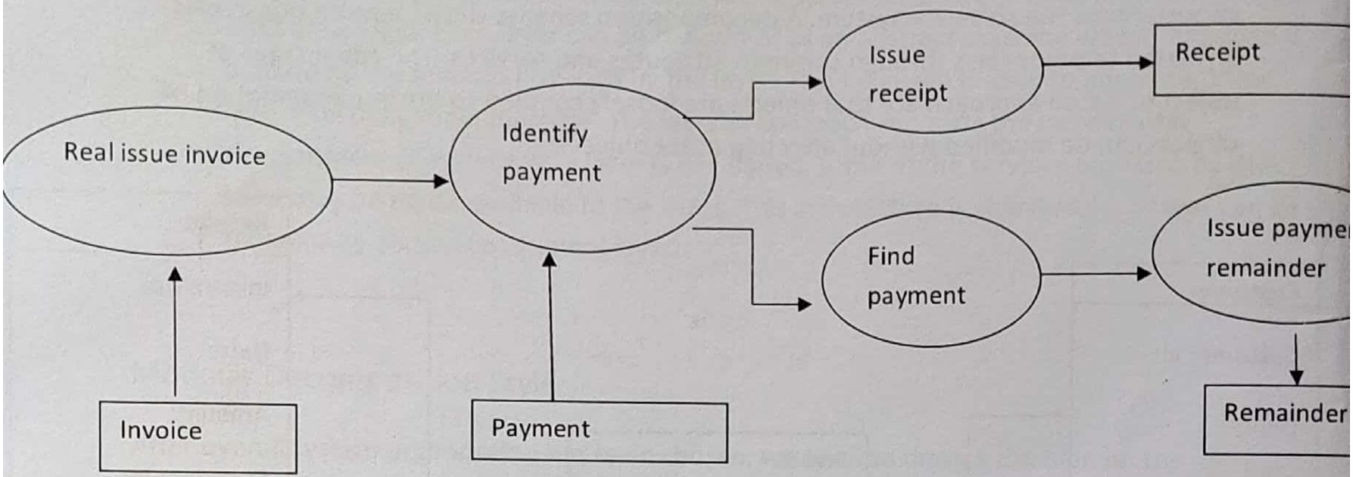


Figure: functional model of invoice processing system

Advantages of this model

i. It supports the reuse of transformation.
ii. Easily the system capability by adding new transformation.
iii. It is simple to implement either as concurrent or a sequential system.

The main problem with this approach is that there has to be a common format for data transfer that can be recognized by all transformation.

(94)

3.2 Object Oriented Design

Introduction

Object –oriented approach for software development have become extremely popular in recent years. An object oriented system is made up of interacting objects that maintain their own local state and private operation on that state. The representation of the state is private and cannot be accessed directly from outside the object. Object oriented design processes involve designing object class and the relationship between these classes. Object classes define the object in the system and this interaction. Object oriented design is a part of object oriented design is a part of object oriented development where an object oriented strategy is used throughout the development process. The object oriented strategies for software development are:

i. Object-oriented Analysis
   Object oriented analysis is concerned with developing an object oriented model of the application domain. The object in the model reflects the entities and operation associate with the Problem to be solved.
ii. Object Oriented Design
   It is concerned with developing an object oriented model of software system to implement the indentified requirement. The objects in the object oriented design are related to solution of the problem.

iii. Object Oriented Programming
   It is concerned with realizing a software design and using an object oriented programming language which provides construct to define object classes.

Object oriented system are easier to change than system developed using changing the implementation of an object or adding new services should not affect other system objects. Objects are reusable components because they are independent.

Object and Object classes

An object is an entity that has a state and a defined set of operations that operates on that state. The state represented as a set of object attributes. The operation associated with the object provides service to other object that requests these services when some computation is required.

Objects are created according to an object class definition. Object class includes declaration of all the attributes and operations that should be associated with object of that class. In an UML (unified modeling language), an object class is represented as a named rectangle with two section. The object attribute are listed in the top section. The operations that are associated with the objects are set out in the bottom section.

For eg

Employee
Name: string Address: string Date of birth: date Employee no: date Social security no: string Salary: integer Status:{current, left, retired } string
Join() Leave() Retire() Change details()

Fig: An employee object

Features of Object Oriented Design

Object oriented approach may have several features that make the software development process easier and software product more efficient.

Some common features of object oriented are as follows:

1. An object oriented model closely represents the problem domain which makes it easier to produce and understand design.
2. Changing requirements can be adopted easily by modifying existing object class or adding new object class.
3. Increase reusability of code.
4. Data and operations are encapsulated so the external object cannot access. Operations which maintains the software security.
5. The encapsulated data is limited to the operations defined on that data. Hence it becomes much easier to ensure that the integrity of data is preserved.
6. Inheritance is concept unique to object oriented. Inheritance promotes reuse by defining common operation of the subclasses is a super class.
7. Polymorphism comes in the form that a reference in object oriented program can refer to object of the different type of different times.

99

Object Oriented Design Process

Object oriented design has several stages.

I. System context and modes of use.
II. Define the system architecture.
III. Object identification
IV. Design model
V. Object interface identification.

i. System Context and Modes of Use

The first stage in any software design process is to develop an understanding of the relationship between the software that is being developed and its external environment. The understanding of such relationship is necessary to provide the required system functionality and specify system structure to communicate with its environments.

The system context and modes of system use represent two complementary model of the relationship between a system and its environment.

a. The system context is static models that describe the other system in that environment.
b. The model of the system use is dynamic models that describe how the systems actually interact with its environment.

ii. System Architectural

Once the interaction between the software system that is being designed and system environment have been defined. We use this information as a basis for designing the system architecture we should try to decompose a system so that architecture is as simple as possible.

iii. Object Identification

At this stage objects that are related to problem domain and solution domains are identified. The design is associated in term of these classes. There have been various proposals made about how to identify object classes:-

a. Use a grammatical analysis of natural language description of a system. Objects and attributes are noun and operations and services are verb.
b. Use entities in the application domain.
c. Use of behavioral approach where the first understand the overall behavior of the system.

Object classes, attributes, and operations that are initially identified from the informal system description can be a starting point for the design.

iv. Design Models

Design model show the objects or object classes in a system and appropriate relationship between these entities. Design models are the bridge between the requirements for the system and the system implementation. An important step in the design process is to be decided when design models that we need and level of details of these models. The type of models and level of details depends on the types of system that is being developed. A sequential data processing system will be designed in different way and different design models will be used.

There are two types of design models that should normally be produced to describe an object orient6ed design.

a. Static Model

Static model describe the static structure of the system using object classes and their relationship. Important relationship may be documented at this stage.

b. Dynamic Model

Dynamic model describe the dynamic structure of the system and so the interaction between the systems objects. Interaction that may be documented includes the sequence of service request made by object and the way in which the state of the system is related to these object interaction.

v. Object Interface Specification

An important part of any design process is specification of the interface between the components in the design. We need to specify interface so that the objects and component can be designed in parallel. Once an interface has been specified, the developer of other object may assume that interface will be implemented

The representation should be hidden and object operation provides access and updates the data. If the presentation is hidden, it can be changed without affecting the object that uses these attributes. Object interface design is concerned with specified the details of the interface that are used to communicate between the objects.

3.2.3 Control models

A control model deals with control flow between the subsystems. In order to make a system work effectively, its constituent sub systems must be controlled to ensure that the services are delivered in a precise manner. There are two types of control models. They are:

1. Centralized Control Model

   In this model, one of the subsystems is designated as system controller with responsibility for managing the execution of other subsystems. Depending on whether the controller system executes sequentially or in parallel, the centralized controlled models are of two types:

   i. Call- return Model

      This is a top- down sub-routine model where the control passes from a higher level subroutine in a hierarchy to the lower level routine. This model is only applicable to sequential system. In this model, currently executing subroutine has the responsibility for control. It can either call other subroutines or returns control to its parent.

      Eg: the call-return control model of ATM is as follows:

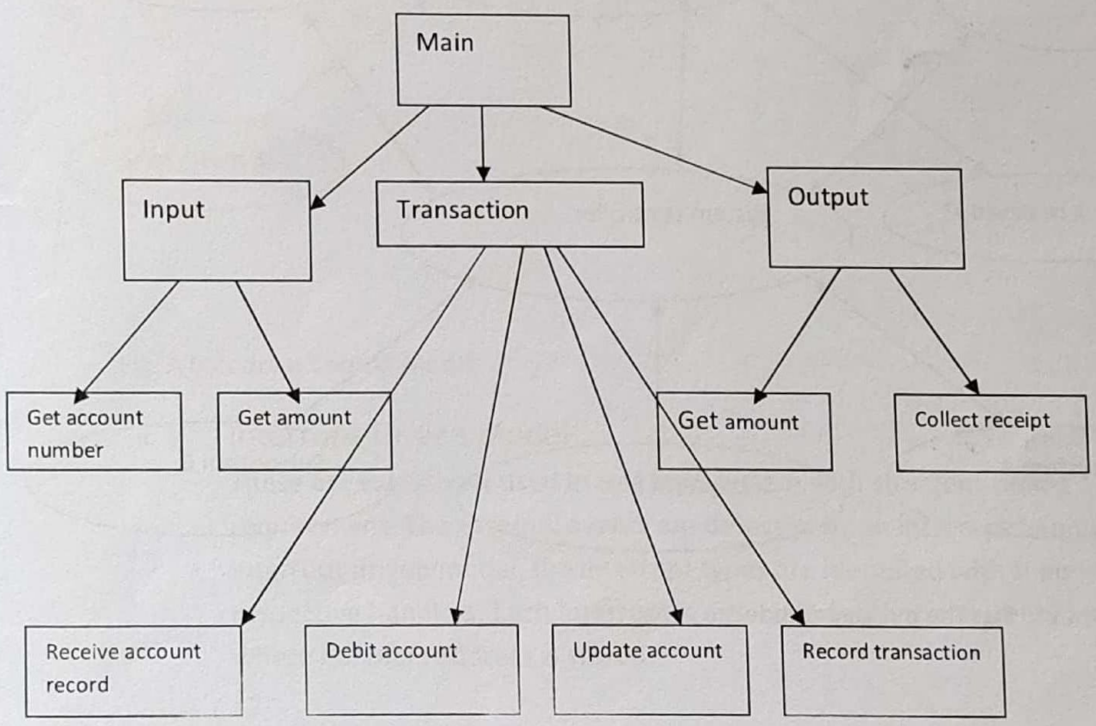


Fig: Call-return model of ATM

ii. Manager Model

This is applicable for concurrent systems. One system component is designed as a system manager that manages the starting, stopping, and coordination of other system processes. A process is a subsystem that can execute in parallel with other processes.

It is also applied in sequential systems where a management routine calls a particular subsystem depending on the values of same state variables through case statement. It is useful in soft-real time systems where time is not a tight constraint. It requires very powerful algorithm to control concurrent processes.

Eg: The generic manager model is as follows:

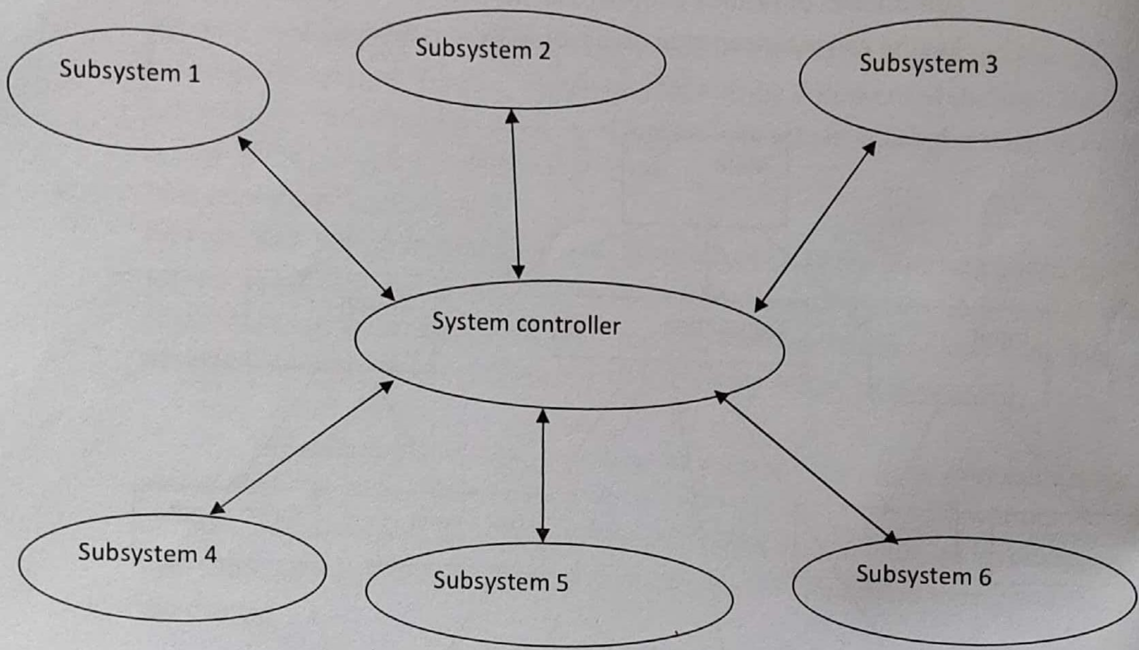


Fig: The manager model

103

vent -Driven Model

These models are driven by externally generated events. There are two types of such models which are:

i. Broadcast Model

These are appropriate for integrating subsystems distributed across different computers on a network. In this model, an event is broadcast to all the subsystems. Any subsystems which can handle that event respond to it. The event and message handler maintain a register of subsystems and the events of interest to them. The event handler detects the event to those subsystems that have registered an interest in the event. A subsystem can send a message to another subsystem.

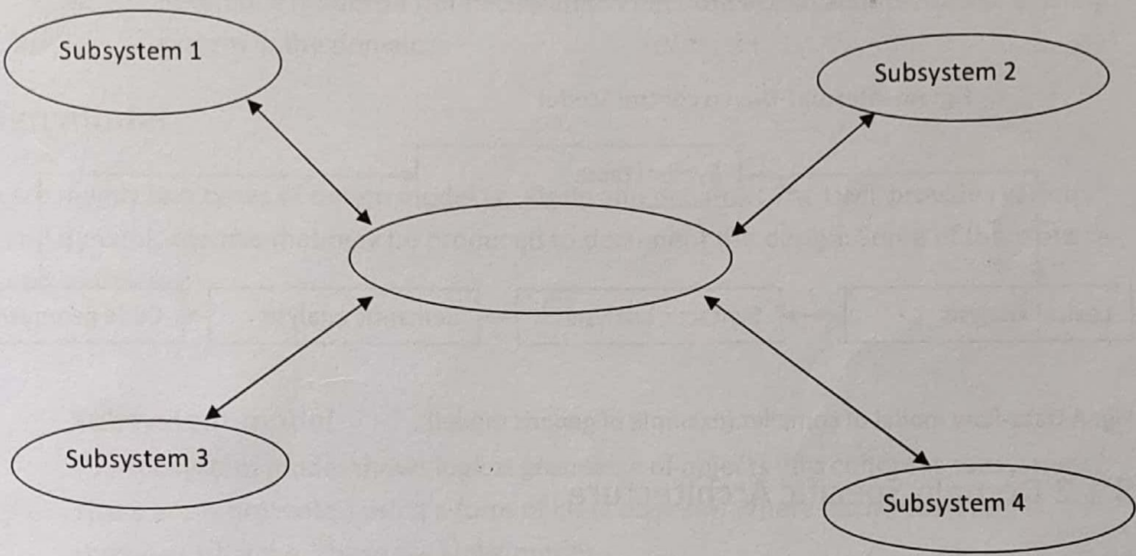


Fig: A broadcast Control Model

ii. Interrupt-Driven Model

These are exclusively used in real time system with stringent timing requirement. The external events are detected by an interrupt handler. In an interrupt driven model, the interrupt types are identified with their respective handlers. Each interrupt is associated with the memory location where handlers address is stored.

Eg:

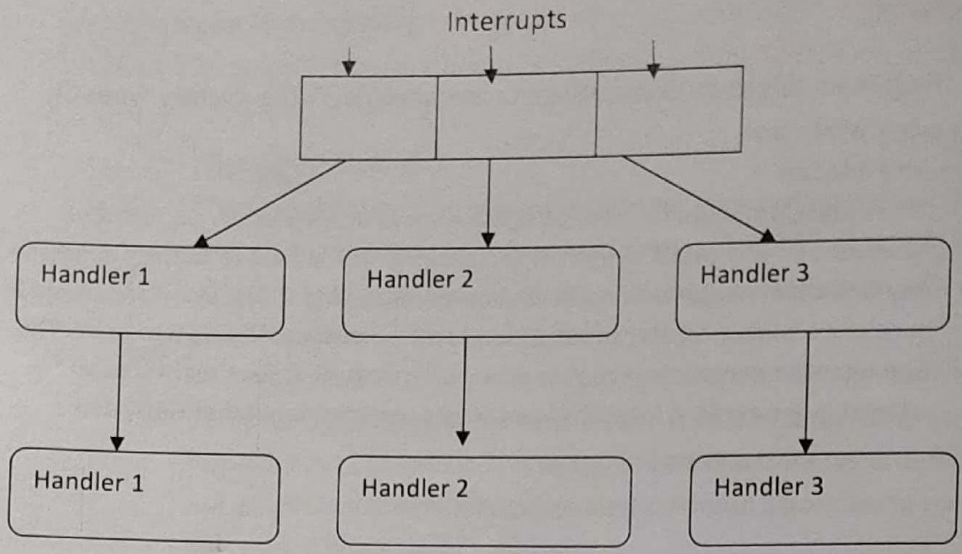


Fig: An interrupt-Driven control Model

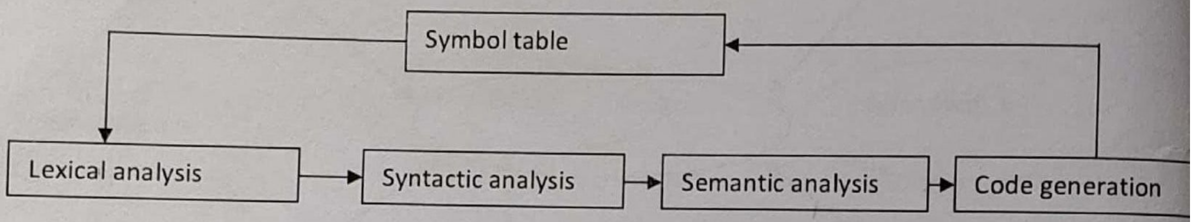


Fig: A Data-flow model of compiler (example of generic model)

3.1.2 Domain Specific Architecture

The above architecture models are general models. They can be applies to many classes of application. As well as those general models, architectural models that are specific to particular application domain are called domain specific architecture.

There are two types of domain specific architecture models:

i. General Models

These are abstractions from a number of real systems. They encapsulate principal characteristics of these real time systems. These may be reused directly in a design. Eg: compiler model.

ii. Reference Model

These are more abstract and describe a larger class of systems. These include all the features that system might incorporate. These are used to communication domain concepts and compare or evaluate possible architectures. It may be used as a basis for system implementation. Eg OSI model.

Difference between Generic and Reference Model

i. Generic model may reuse directly in a design.
ii. Reference model are normally used to communicate domain concept and compare possible architectures.
iii. Generic models are usually derived "bottom-up" from existing system.
iv. Reference modes are derived from "top-down".
v. Generic models are abstract system representations.
vi. Reference model do not necessarily reflect the actual architecture of existing system in the domain.

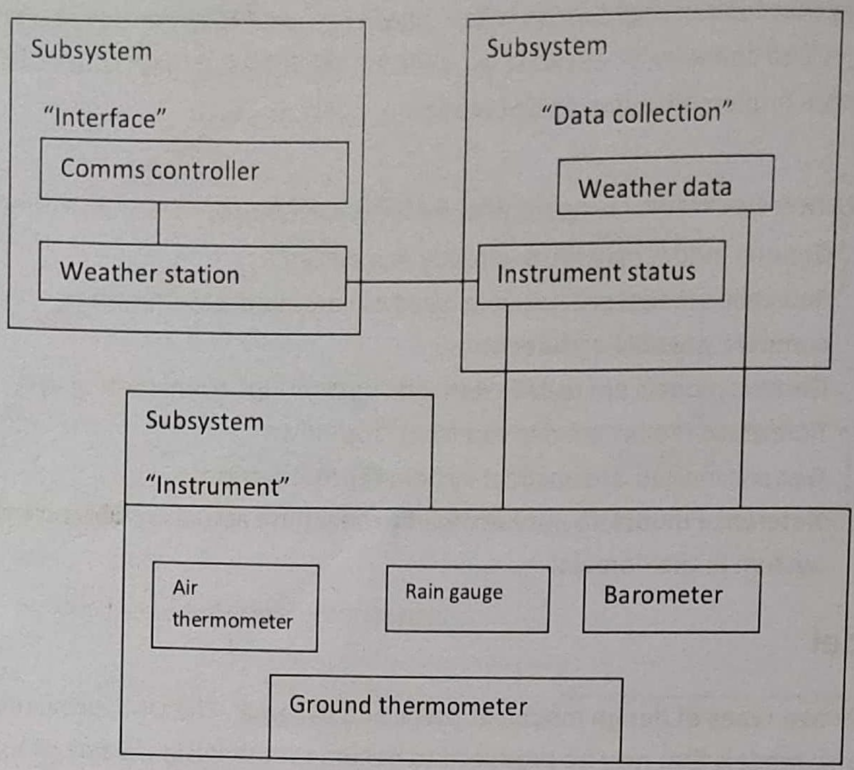
Design model

There are mainly two types of design model i.e. static and dynamic. The UML provides various static and dynamic models that may be produced to document the design. Some of them are described as follows:

i. **Subsystem model**

The subsystem model shows logical groupings of objects into coherent subsystems. These are represented using a form of class diagram. Where each subsystem is shown as package. These are static model.

Eg: The class diagram or sub-system model for weather station is as follows:



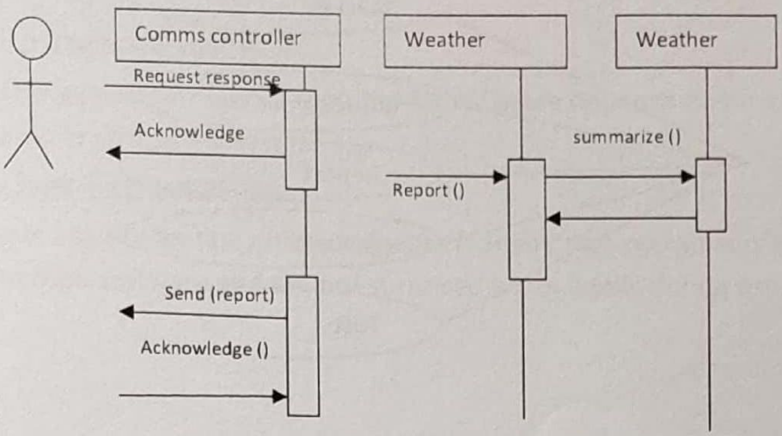
ii. **Sequence Model**

These are the dynamic model that shows the sequence of object interactions. These are represented using a UML sequence or a collaboration diagram. In a sequence model,

a. Objects involved in the interaction are arranged horizontally with a vertical line linked to each object.
b. Time is represented vertically.
c. Labeled arrows linking the vertical lines represent the interaction between objects.
d. The thin rectangle on the object lifeline represents the time when the object is "controlling object" in the system.

107

Eg: the sequence diagram for weather system is as follows:



Sequence diagrams are used to model the combined behavior of a group of objects but to summarize the behavior of a single object in response to the message it can process, state machine model is used. State machine model shows how individual objects change their state in response to events and are dynamic models and are represented using state chart diagram.

iii. **Use-Case model (Diagram)**(Imp)

This represents an object interaction with the system. In this model, each possible interaction is named in an ellipse and external entity involved in the interaction is represented by a stick figure. Each use-case description helps designers to identify objects and operations in the system.

Eg: The use-case model for weather station system is as shown below. In this example, the external entity is not a human but is data processing system for the weather data. This example shows that weather station interacts with external entities fro startup, shutdown, reporting weather data that has been controlled, instrument testing and calibration.

108

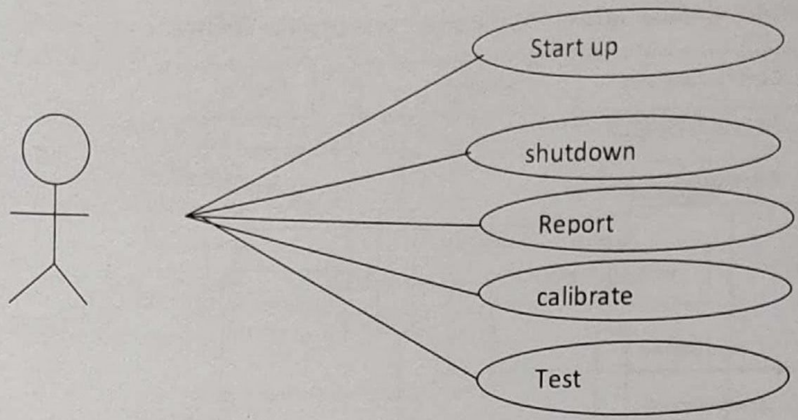


Fig: Use-Case for weather station

iv. Activity Model

Unit 4.1 Verification and Validation

Introduction:

During and after the implementation process, the program being developed must be checked to ensure that it needs its specification and the functionality expected by the user paying for software. Verification and validation is the name to these checking and analysis. Verification and validation starts with requirement review and continue to design, review and code inspection to product testing. Verification and validation is not the same thing.

Validation

Are we building the right product?

Verification

Are we building the product right?

These definitions tell us that the role of verification involves checking that the software confirms to its specification. We should check that it meet its specified functional and non functional requirements.

The aim of validation is to insure that the software system meets the customer's expectation. It goes beyond checking that the system confirms to its specification to showing that the software does what the customer expects it to do. The expectation of the system user and the current marketing environment for the software system are:

i. **Software Function**

The level of confidence required for software depends on how it can handle critical data or critical situations.

ii. **User Expectation**

It is a sad reflection on the software industry that many users have low expectation of their software and are not surprised when it fails during use.

iii. **Environment Marketing**

When a system is marketed, the seller of the system must take into account Competitions programs, the price those customers are willing to pay for the system and required schedule for delivering that system.

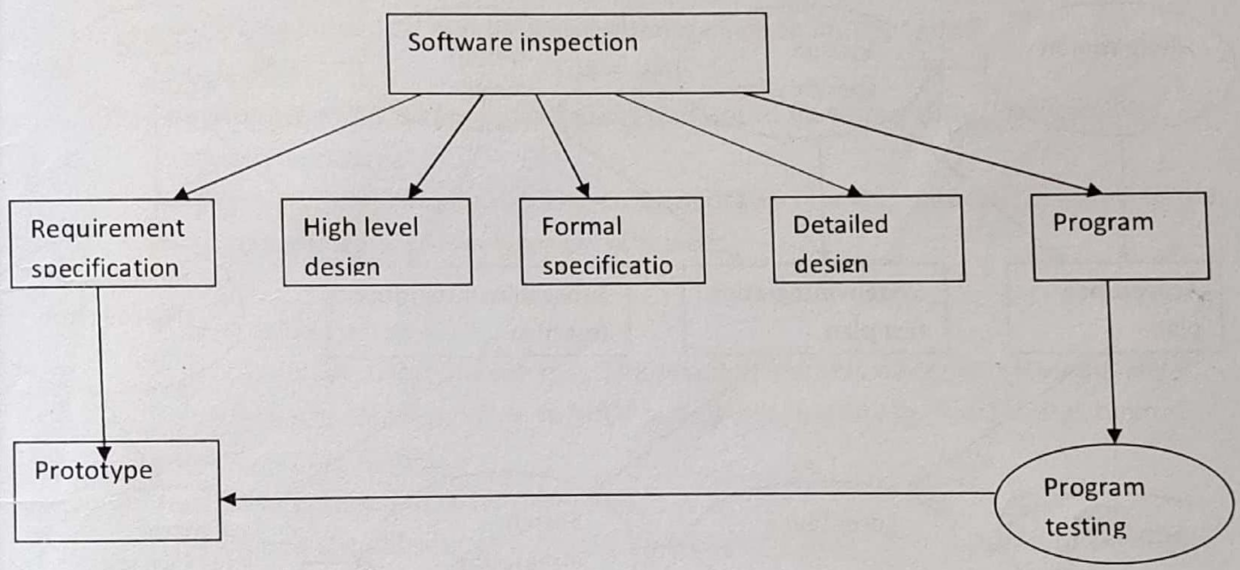


Fig: verification and validation

Within the verification and validation process, there are two complementary methods used for software system checking and analysis.

a. Software Inspection

110

Software inspection analyzes and checks system representation such as the Requirement document, design program diagrams, and the program source code. We can use inspection at all the system software development process.

b. Software Testing

Software testing involves running and implementation of the software with test data. We examine the output of the software, its operational behavior to check out that it performing as the user requirement.

Verification and Validation Planning

Verification and validation is an expensive process and more than half of the system development budget may be spend on verification and validation process. Careful planning is needed to get most out of inspection and testing and to control the cost of the verification and validation process.

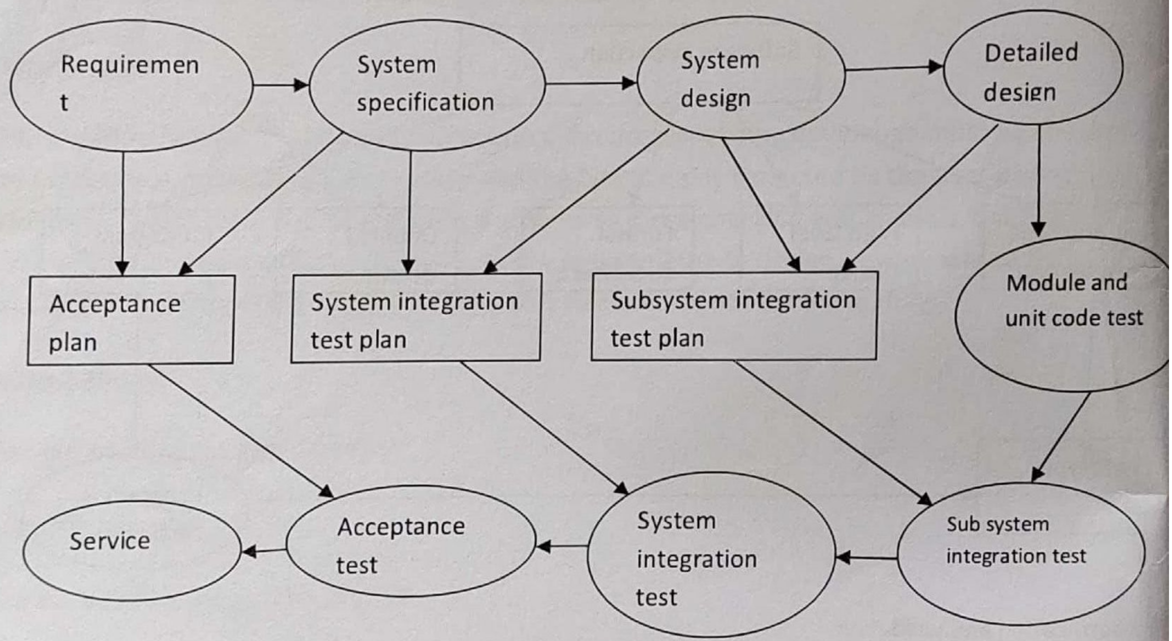


Fig: test plan as a link between development and testing

System verification and validation planning start early in the development process. The models shows that test plan should be derived from the system specification and design. This model also breaks down system verification and validation into number of stage. Each stage is driven by tests that have been defined to check the conformance of the program with its design and specification. As part of the verification and validation planning process, we should decide on balance between static and dynamic approach to verification and validation, draw up standards and procedures for software inspection and testing, establish check list to drive program inspection and define software test plan. The effort devoted to inspection and testing depends on types of system being developed and organizational expertise with program inspection. As a general rule, the more critical system, the more effect should be developed to verification and validation.

The test plan is concerned with establishing standards for the testing process and helping managers to allocate resources and estimate testing schedules. As well as setting out testing schedule and procedures, the test plans define the hardware and software resources that are required for testing. This is useful for manager who is responsible for ensuring that these resources are available to the testing team. The structure component of software test plan is:

i. The testing process
   A description of major phases of testing process
ii. Requirement Traceability
   Users are most interested in the system meeting its requirements and the testing should be planned so that all requirements are individually tested
iii. Tested items
   The product of the software processes that are to be tested should be specified.
iv. Testing schedule
   An overall testing schedule and resource allocation for this schedule in linked to the more general project development schedule
v. Test recording procedure
   It is not enough simply to run test. The result of the test must be systematically recorded. It must be possible to audit testing process to check that it has been carried out correctly.
vi. Hardware and Software Requirement
   This section should set out the hardware and software tools required or estimated for testing process.
vii. Constraint
   Constraint affecting the testing process such as staff shortage should be anticipated in this section

Software inspection

Software inspection is a static verification and validation process in which a software system is reviewed to find errors, omissions and anomalies. Generally inspection focus on source code but any readable representation of the software such as its requirements specification or design model can be inspected. When we inspect a system, we use knowledge of system, its application domain, design model and the programming to discover error. There are three major advantages of inspection over testing:

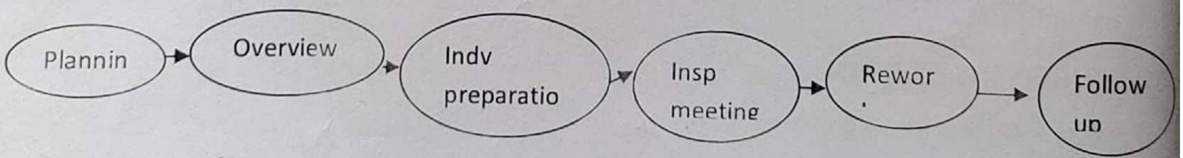
i. During testing errors can hide other errors once one error is discovered, we can never be sure if other output anomalies are due to new errors or side errors of original errors. But inspection is static process we do not have to be connected with interaction between errors. Consequently, a single inspection can discover many errors in the system.
ii. In complete version of system can be inspected without additional cost. If the program is incomplete, then we need to develop specialized test harness to test the part of that program.
iii. As well as searching for program defect and inspection can consider broader quality attribute of a program such as compliance with standards, portability, and maintainability.

Several studies and experiments that have demonstrate that inspections are more effective for defect discovering than program testing.

Program Inspection Process

Program inspections are reviews whose objective is to detect program defect. The notation of a formalized inspection process was developed by IBM in 1970s. The key difference between program inspection and other types of quality review is that the specific goal of inspection is to find program defects rather than consider design issue. Defects may be logical errors anomalies in the code that might indicate an error condition. By contrast, other types of review may be more concerned with schedule, cost, progress against define milestone or assessing whether the software is likely to meet organizational goal. The program inspection is a formal process.

That is carried out by a team. The team members systematically analyze the code and point out the possible defect.



113

Fig: the inspection process

During an inspection a check list of common programming errors are often used to focus the discussion. The inspection team is modulator responsible for inspection panning. This involves selecting an inspection team, organizing a meeting room and ensuring that the material to be inspected and its specification are complete.

The program is to be inspected is presented to be inspected team during the review stage when another code describes what program is intended to do. This is followed by a period of individual preparation. Each inspection team member studies the specification and program and looks for defect in the code.

During the inspection, the program author should make change to the code to correct identified defect. The time needed for an inspection and amount of code that can be covered depends on the experience of inspection team, programming language and application domain.

Inspection checks

Fault classes — inspection check

i. Data fault → Are all program variable initialized before their use?
→ Should the upper bound of array be equal to the size of the array? Or size-1?
→ Is there any possibility of buffer flow?

ii. Control Fault → For each conditional statement is the condition correct?
→ Is each loop contains terminating value?
→ Are compound statement correctly bracketed?
→ Is case statement, are all possible cases accounted?
→ Is a break is required after each case in case statement has it been Included?

iii. Input/ output fault → Are all input variable used?

→ Are all output variable assigned a value before they are out-putted

iv. Interface fault → Do all function and method call have the correct no of Parameters?

→ Do formal and actual parameters types match?

→ Are all parameter in the high order?

v. Storage mgt fault → If linked structure is modified, have all links been correctly Reassigned

→ If dynamic storage is used has space been allocated?

Clean room Software Development

Clean room software development is a software development philosophy that uses formal method to support software inspection. The clean room approach to software development is based on five key strategies:

i. Formal specification
   The software to be developed is formally specified. A state transition model that shows system response to the input/ output is used to express the specification.

ii. Incremental Development
   The software is partition into increments that are developed and validated separately using clean room process. These increments are specified with customer requirement at an early stage in the process.

iii. Structured Programming
   Only a limited number of control and data abstraction construct are used. The program development process of step wise refinement of the specification. The aim is to systematically transform the specification to create the program code.

iv. Static Verification
   The developed software is statically verified by using various software inspection techniques.

v. Statistical Testing of the system

115

The integrated software increments are tested statistically determines its reliability. These statistical tests are based on operational profile which is developed in parallel with the system specification.

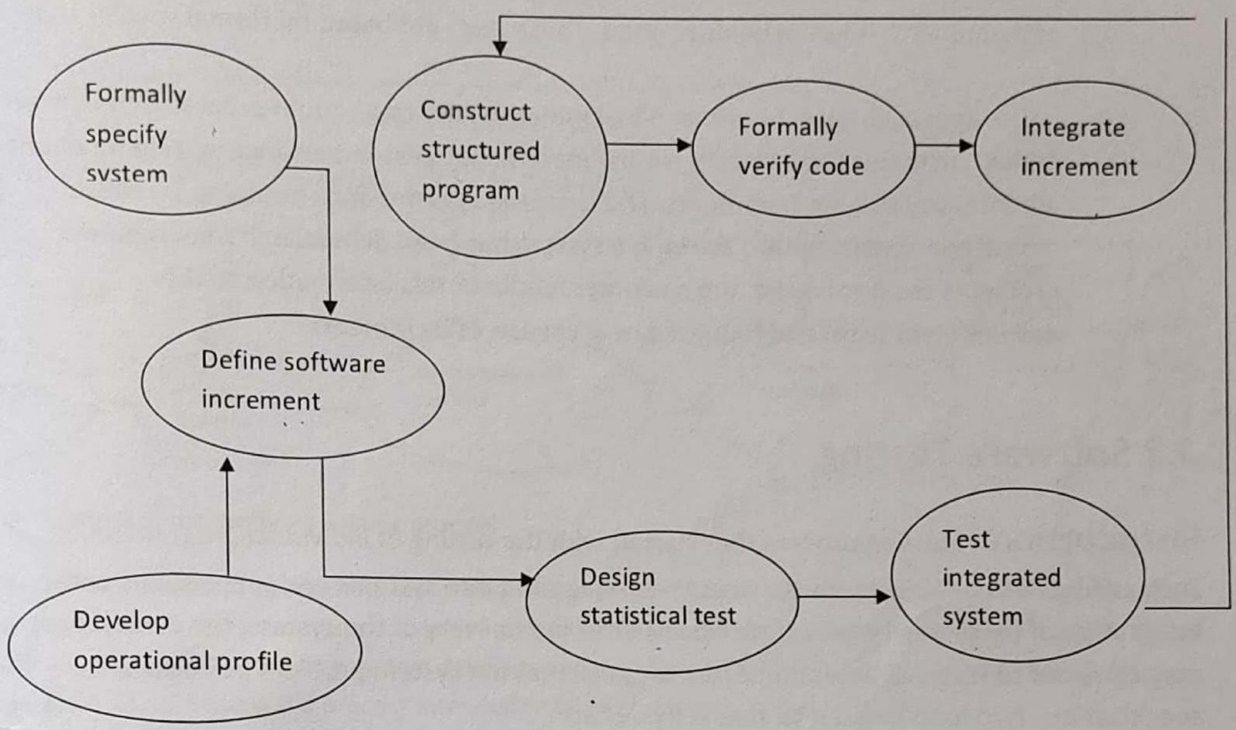


Fig: clean room development process

There are three teams involved when the clean room process is used for the large software development.

i. The significance team
The significance team is responsible for developing and maintaining the system specification. This team produce customer oriented specification and the mathematical specification for verification. In some cases, when the specification team also takes responsibility for development.

ii. The development Team

This team has the responsibility of developing and verifying the software. The software is not executed during the development process. A structured formal approach to verification based on inspection of code supplemented with correctness argument is used.

iii. The certification team
This team is responsible for developing a set of statistical test to exercise the software after it has been developed. These tests are based on formal specification.

The approach to incremental development in the clean room process is to deliver critical customer functionality is early increments. Less importance system functions are included in later increments. The customer has the opportunity to try these critical increments before the whole system has been delivered. If requirements problems are discovered, the customer feedback this information to the development team and request a new version of increments.

4.2 Software Testing

Introduction: A testing process that started with the testing of individual program units such as functions or objects. These were then integrated into systems and sub systems and Integration of these units was tested. Finally, after the delivery of the system, the customer may carry out of series of acceptance test to check that the system problem perform as its specification. Two fundamental testing activities are:

i. Component Testing
In component testing, the part of the system is tested individually. The aim of testing stage is to discover defect by testing individual program components. These components may be functions, objects, or reusable components.

ii. System testing
In system testing, the whole system is tested to verify that the system performs as its specification. During system testing, the components are integrated to form a subsystem or complete system. At this stage, system testing should focus on establishing that the system needs its functional and non functional requirements.

117

The software testing process has two goals:

i. To demonstrate to the developer and customer that the software are needs its requirement.

ii. To discover faults or defect in the software where the behavior of the software is incorrect, undesirable, or does not confirm to its specification.

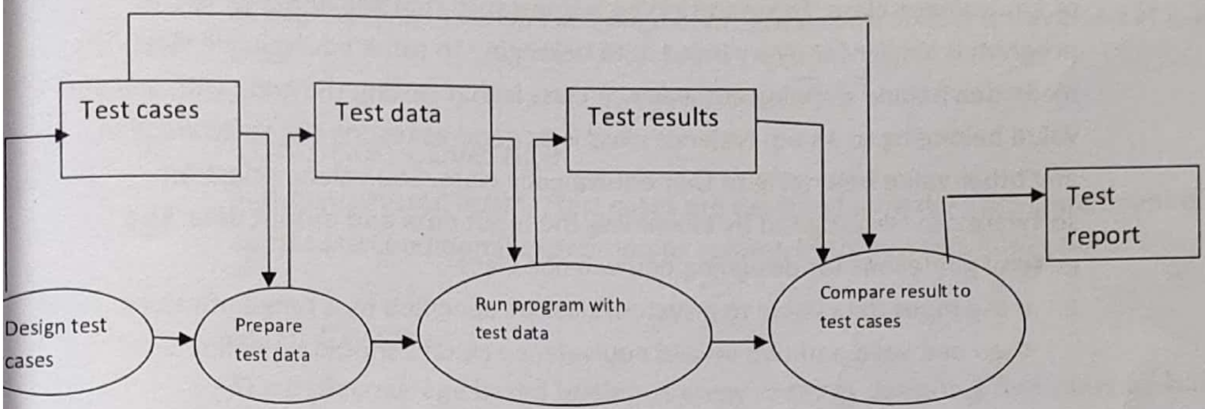


Fig: A model of the software testing process

For validation testing, a successful test is one where the system performs correctly. For defect testing, a successful test is one that exposes a defect that causes the system to perform incorrectly. The goal of software testing is to convince system developer and customer that the software is good enough for operational use. Testing is a process intended to build confidence in the software.

Test cases are specification of the input to the test and the expected output from the system. Test data are input for the system testing.

Types of testing

1. Unit Testing

Unit testing is undertaken after a module has been coded and successfully reviewed. Unit testing is the testing of different units of system. In order to test a single module, a complete environment is needed to provide all that is necessary for execution of the module. The following steps are needed in order to test the module.

a. The procedures belonging to other module that the modules under test call.
b. Non local data structure that the module access.

118

2. Black box Testing

In the black box testing, test cases are designed from an examination of input/ output values only and no knowledge of design or code is required. The following are two main approaches to designing black box test case:

i. Equivalence class Partitioning

In this approach, the domain of input values to a program is partitioned into set of equivalence class. This partitioning is done such that the behavior of the program is similar for every input data belonging to same equivalence class. The main idea behind defining equivalence class is that testing the code with any one value belonging to an equivalence class is as good as testing the software with any other value belonging to that equivalence class. Equivalence class for software can be designed by examining the input data and output data. The general guidelines for designing equivalence class-

a. If the input data value to a system class be specified by a range of values, then one valid and two invalid equivalence classes should be defined.

b. If the input data assumes values from a set of discrete number of some domain then one equivalence class for valid input values and another equivalence class for invalid input values should be defined.

ii. Boundary value analysis

A type of programming errors frequently occurs at boundaries of different equivalent classes of inputs. The reason behind such error might purely be due to the psychological factor. Programmers often fail to see the special processing required by input values that lie at the boundary of different equivalence classes.

3. White box testing

White box testing strategy is said to be stronger than another strategy. If all types of errors detected by the first testing strategy is also detected by the second testing strategy and the second testing strategy additionally detects some more types of errors. The following approaches are used for white box testing.

i. Statement coverage

The statement coverage strategy aims to design test cases. So that, every statement in the program is executed at least once. The principal idea governing the statement coverage strategy is that unless a statement is executed, it is very hard to determine if an error exists in that statement. Unless a statement is executed, it is very difficult to observe whether it cause failure due to some illegal memory access, wrong result computation, etc.

ii. Branch Coverage

In the branch coverage testing strategy, test cases are designed to make each branch condition to assume true and false values. Branch testing is also known as edge testing in which each edge of program control flow is traversed at least once.

iii. Condition Coverage

In this structured testing, test cases are designed to make each component of a composite conditional expression to assume both true and false values.

iv. Path Coverage

The path coverage based testing strategy requires designing test cases such that all linearly independent paths in the program are executed at least once. A linearly independent path can be defined in term of control flow graph of a program

Control Flow Graph (CFG)

A control flow graph describes the sequence in which the different instructions of a program get executed. In other words, a control flow graph describes how the control flows through the program. In order to draw the control flow graph of program, all the statement of the program must be numbered first. The different number statement serve as nodes of the control flow graph. An edge from one node to another node exists. If the execution of the statement representing the first node can result in the transfer of control to another node.

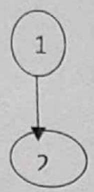
The control flow graph for any program can be easily drawn by knowing how to represent the sequence, selection and iteration type of statement in control flow graph.

Sequence	selection	iteration
I. a =5	i. if (a>b)	i. While (a>b)
Ii. b= a*2	ii.c= 3	{

120

iii. Else c=5

iv. C= c*c

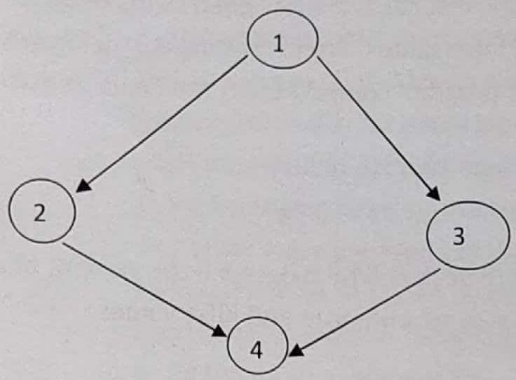


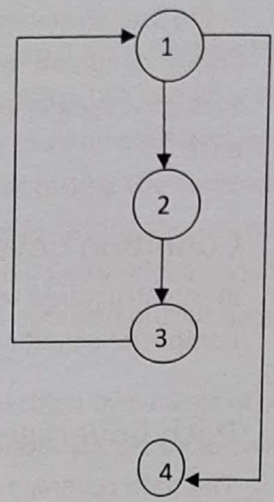
ii. b=b-1

iii. b=b*a

}

Iv .C=a+b





4. Mutation Testing

In mutation testing, the software is first tested by using any initial test suite built up from the different white box testing strategy. After the initial testing is complete, mutation testing is taken up. The idea behind mutation testing is to make a few arbitrary changes to a program at a time. Each time the program is changed, it is called mutated program and the affected is called mutant.

A mutated program is tested against the full test suite of the program. If there exists at least one test case in the test suite for which a mutant gives an incorrect result, then the mutant is said to be dead. If a mutant remains alive, even after all the test cases have been exhausted, the test data is enhanced to kill the mutant. The process of generation and killing of mutant can be automated by defining a set of primitive changes that can be applied to the program. These primitive changes can be alteration such as changing an arithmetic operator, changing the value of a constant, changing a data type, etc.

5. Integration Testing

The primary objectives of integration testing is to test the module interface i.e. there are too errors in the parameters passing when one module invokes another module. During integration testing, different modules of system are integrated in a planned manner using an integration plan. This integration plan specifies the steps and orders in which modules are combined to realize the full version of system. After each integration step, the partially integrated system is tested. An important factor that guides the integration plan is the module dependency graph. The module dependency graph denotes the order in which the different module calls each other. There are various types of integration testing approaches. Any one of the approaches can be used to develop the integration test plan.

6. Validation Testing (Requirements- based testing)

A general principle of requirement engineering is that requirement should be written in such a way that a test can be designed so that an observer can check that the requirement has been satisfied. Requirement based testing is a systematic approach to test case design when we consider each requirement and derive a set of tests for it. Testing the requirement does not mean just writing a single test. We normally have to write several tests to ensure that we have coverage of the requirement.

7. Validation Testing(statistical testing)

This is used for testing performance and reliability of system. Testing is performed using test data that reflect operational profile of the system. The operational profile of the system reflects how system will be used. Reliability of system is estimated by determining frequency of system failure. Performance is evaluated by measuring execution and response time.

4.3 Critical System validation

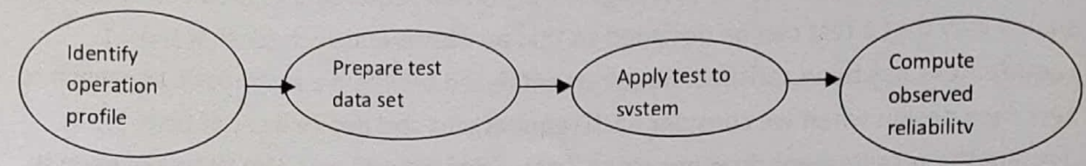
Introduction:

The verification and validation of a critical system has much in common with the validation of any other system. The verification and validation processes should demonstrate that the system meets its specification and that the system services and behavior supports the customer's requirements. However, for critical system, where a high level of dependability is required, additional testing and requirements are required to produce evidence that the system is trustworthy. The cost of verification and validation for critical systems are usually much higher than for other classes of systems. Although, the critical system validation process mostly focuses on validating the system, related activity should verify that defined system development processes have been followed.

Reliability Validation (Statistical Testing)

Reliability is a complex concept that should always be considered at the system rather than the individual concept level. To validate the system requirements, we have to measure the reliability of the system as seen by a typical system user. Reliability validation is sometimes called statistical testing. The aim of statistical testing is to assess system reliability.

In reliability, measurement process is as shown below.



4.4 Software Cost Estimation

Introduction:

Software cost estimation is the part of project planning process. To estimate the cost of software project, we have to estimate how many software engineers are required to accomplish the project.

To obtain the total cost of software project, the project manager must prepare a detailed estimate of the following:

i. Efforts required to complete an activity
ii. Duration of the activity
iii. Total cost of the activity

The initial costs are tentative, hence are revised regularly as project progresses. There are three important components involve in estimating the software project cost:

i. Effort cost (cost of paying software engineers)
ii. Hardware and software costs including maintenance
iii. Travel and training cost

123

Software Productivity

This is an attribute of the software production process. Software engineering is the most determinative factor of productivity. Abilities of engineers may significantly vary. When engineers of different capabilities work as a team, the productivity of the team is a function of productivity of individual engineers.

Measurement of Productivity

To estimate the project resource requirements, it is necessary for the project manager to determine the productivity of engineers involved in software development process. Productivity is estimated by dividing some attribute of software by total efforts.

There are two types of productivity measures:

1. Size- Related Measures

   This approach is first developed when most programming was done in FORTAN, assembly language, or COBOL. It is expressed as line of codes counting the total number of lines of source code delivered, divided by total time in programmer-months.

   Productivity= LOC/person-months

   Quality=errors/LOC

   Eg suppose 5000 is the LOC and 24 person-months

   Productivity=LOC/person-months

   =5000/24 ⟹ 208.33 lines/month

2. Function – Related Measures

   These are indirect measures of software and process by which it is developed. In this, we concentrate on functionality of software rather than LOC. There are two types of function-related measures which are:

   i. Function Point

      It is the best known measure of software productivity. These techniques express the productivity as function points produced per-month. To obtain the function point, we must count the following data:

      - No of inputs refers to input data

124

- No of user output — refers to report, screen error message, etc
- No of user inquiries — refers to online input and immediate online Output response

* No of files — refers to database, file, etc

* No of external interface — refers to data file on type or disk

Once these data are collected, function point is computed as,

F_{p}= total-count*[0.65+0.01*SUM (F_i)]

Where f_i are complexity adjustment value noted from the table (i=1 to 14)

Productivity= FP/person-months

Quality = defects/FP

ii. Object Points

This technique is used as an alternative to function point method for software that is developed using 4GL or equivalent programming languages. Objects point method considers only screen, reports, and 3GL modules that must be developed to supplement the 4GL code. These are computed on the basis of different weighting points.

Estimation Techniques

Cost estimation is difficult task due to several reasons such as

- Initially all system requirements may not be clear.
- Development technology may be new.
- Skills of the personal involved in project development may not be clear.

There are several techniques available to estimate the cost of software development, some of them are:

1. Algorithm Cost Modeling

In this technique, cost is estimated as a mathematical function of product, project, and process attributes whose values are estimated by project managers. Most commonly used product attribute for cost estimation is LOC (code size).

i. The COCOMO Model

125

Constructive cost model (COCOMO) is one of the best documented algorithmic cost estimation models. This is an empirical model derived after analyzing the data related to 63 completed software projects.

Basic COCOMO Model

It is the first and simplest version of COCOMO model, also known as COCOMO 81 model. This gives an approximate estimate of the project parameters. This model distinguishes three classes of software project.

- Organic or Simple: in this model, relatively small software teams develop software in a highly familiar environment. Most people connected with the project have extensive experience in working with related systems within the organization.
- Embedded: In this mode, software projects needs to operate within tight constraints. The product must operate within (or is embedded in) a strongly coupled complex of hardware, software, regulations, and operational procedures such as air traffic control system or electronic fund transfer system.
- Semidetached or Moderate: This represents the intermediate stage between organic and embedded modes. This is an intermediate level of project characteristic or a mixture of organic and embedded mode characteristics.

The basic COCOMO model is given by following expression:

$Effort = a_1*(KLOC)^{a}{}_2*PM$

$Tdev = b_1*(effort)^{b}{}_2*months$

Where,

- KLOC (or size) is estimated size of software product expressed in kilo lines of code.
- A_1, a2, b1, b2 are constants for each category of software products.
- Tdev is the estimated time to develop the software, expressed in months.
- Effort is total effort required to develop software product, expressed in person-month (PM) or labor-months (LM)

For different classes, the above parameters are assigned different values as:

126

Class	complexity	a1	a2	b1	b2
Organic	simple	2.4	1.05	2.5	0.38
Semidetached	moderate	3.0	1.12	2.5	0.35
Embedded	complex	3.6	1.20	2.5	0.32

This model is very simple to use and gives a quick approximate cost estimate and this model assumes that development process would adopt waterfall model.

a. Intermediate COCOMO Model

There have been radical changes in software development approaches since basic COCOMO model was proposed, such as use of prototyping techniques, incremental approach in software development, application of 4GLs, etc. To reflect these changes, the original COCOMO model was updated using COCOMO II. This model supports spiral model and contains several sub models that produce increasingly detailed estimate. The sub-models of COCOMO II are:

i. An Application Composite Model

This model estimates the efforts required to develop a prototyping project. It supports the cost estimation of the projects that are developed using existing component, scripting or database programming.

The efforts can be estimated using following expression.

PM= (NAP*(1-% reuse/100))/PROD

Where,

PM is effort in person-month

NP is no of object points.

% reuse is expected percentage of reuse

PROD is productivity.

ii. The Early Design Stage

At this stage, the process moved little forward. The developer may try an alternative architecture. There is still not enough information to arrive at an accurate estimate.

The effort is estimated as,

Effort = a*[KLOC] ^{b}*M

Where a, and b are constants that ranges from 2.5 and 1.1 to 2.4 respective.

KLOC (size) is kilo lines of code

M is multiplier given as

M=PERS * RCPX * RUSE * POIF * PREX * FCIL * SCED

Where, PERS= personal capability

RCPX= reliability and complexity

Ruse= reuse required

POIF= platform difficulty

PREX= personal experience

FCIL = support facilities

SCED= schedule

iii. The Reuse Model

This model is used to estimate the effort required to integrate reusable or automatically generated code. Reuse code are of two types. I.e. Black box code and white box code. Black box code is a code that can be reused without understanding and modifying it and hence its development cost is zero.

But white box code is understood and integrated with new code, thus cost is involved. The effort is computed as,

PM=a*(KLOC*(AT/100))/ATPROD

Where, AKLOC= no of automatically generated KLOC

AT= percentage of total software code that is automatically generated

ATPROD= productivity level for this types of code production

iv. The Post-Architecture Model

In this stage, when system architecture has been designed, size of the system can be estimated more accurately. The estimate process at this level uses a more extensive set of multipliers. The effort is given as,

Effort = $a*(KLOC)^{b}*M$

B may be related to different level of project complexity.

Testing work Benches

A software testing workbench is an integrated set of tools to support the testing process. It may also include tools to simulate other parts of the system and to generate system test data. Some of the tools included in the testing workbench are as follow:

i. Test Manager: Manages the running of program test. They keep the track of test data, expected results ad program facilities tested.

ii. Test Data Generator: Generates test data for the program to be tested. This may be accomplished by setting data from a database or by using patterns to generate random data of correct form.

iii. Oracle: generate prediction of expected test results. It may be either previous program version or prototype systems.

iv. File Comparator: compares the result of program test with previous test results and reports differences between them. These are used in regression testing where the results of executing different versions are compared.

v. Report Generator: Provides report definition and generation facilities for test results.

vi. Dynamic Analyzer: adds code to a program to count the number of times each statement has been executed

vii. Simulator: Target simulators simulate the machine on which the program is to execute. User interface simulators simulate multiple simultaneous user interactions.

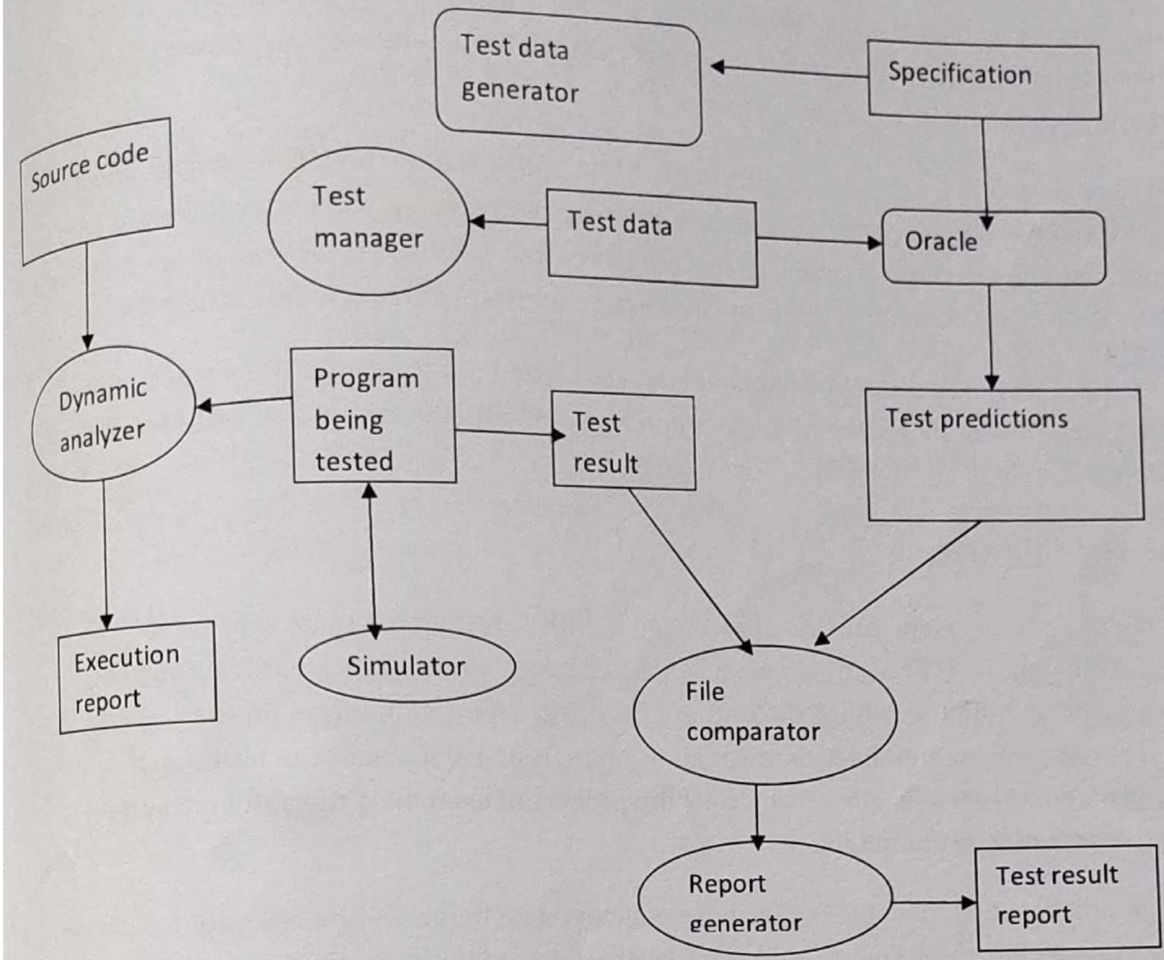


Fig: a Testing workbench

When used for large system testing, tools have to be configured and adopted for the specific system that is being tested eg:

i. New tools may have to be added, and some existing tools may not be required.
ii. Scripts may have to be written for user interface simulator and patterns defined for test data generators.
iii. Special purpose file comparator may have to be written that include knowledge of the structure of test result in file.

A significant amount of effort and time is usually needed to create a comprehensive testing workbench.

4.5 Software Reengineering

Introduction:

The reengineering of software is described by Chikofsky and Cross as "the examination and alternation of a system to reconstitute in a new form". Less formally reengineering is the modification of software system that takes place after it has been reverse engineered, generally to add new functionality to correct errors.

Reengineering is the subsequent modification of the system. Reengineering is mostly used in the context where a legacy system is involved. Software systems are evolving in high rate because there is more research to make them better. Software system in most cases nee to operate in a new computing platform. In these circumstances reengineering helpful.

Reengineering is the set of activities that are carried out to restructure a legacy system to a new system with better functionalities and confirm to the hardware and software quality constraint.

Reverse Engineering

"Reverse engineering is the process of analyzing a subject system to create representations of a system at a higher level of abstraction". It can also be seen as going backwards through the development cycle. In this model, the output of the implementation phase (in source code form) is reverse engineered back to the analysis phase is an inversion of the traditional waterfall model. Reverse engineering is only the process of examining the software system under consideration is not modified.

In practice, two main types of reverse engineering emerge. In first case, source code is already available for the software but higher level aspects of the program are poorly documented or documented but no longer valid are discovered.

In the second case, there is no source code available for the software and any efforts towards discovering one possible source code for the software are regarded as reverse engineering. Reverse engineering of the software can make use of clean room design to avoid copy right infringement.

Black box testing in software engineering is a lot similar with reverse engineering where the goal is to find bugs and undocumented features by basing (hit) the software from outside.

Other purpose of reverse engineering includes security auditing, removal of copy protection (cracking) circumvention of access, restriction often present in consumer electronics, customization of embedded systems (such as engine management system), in house repair, or retrofits, enabling of additional feature on low cost "crippled" hardware (such as graphic card chipsets), or even more satisfaction of curiosity.

Safety Assurance

The purpose of safety assurance and reliability validation has different objectives. Safety cannot be meaningfully specified in a quantities way and so cannot be measured when a system is tested. Safety assurance is concerned with establishing confidence interval in the system that might vary from 'very low' to "very high". This is matter for professional judgment based on the body of evidence about the system, its environment and the development process. In many cases this confidence is partly based on the experience of the organization developing the system. Safety assurance must be backed up by tangible evidence from the system design, the result of verification and validation and the system development process that might be used.

The verification and validation process for safety critical system have much in common with the comparable processes of any other system with high reliability requirement. Safety assurance concentrates on faults of the system with hazard potential. If it can be assured that these fault cannot occur or if they do, the associated hazard will not result in an accident, then the system is safe.

Safeties proof are an effective safety assurance technique. They show that an identified hazardous condition can never occur. They are usually simpler than proving that a program meets its specification.

Security Assessment

The assessment of system security is becoming increasingly important as more and more critical systems are Internet-enabled and can be accessed by anyone with a network connection. This means verification and validation processes for web-enabled system must focus on security assessment. Where the ability of the system to resist different types of attack is tested. However, these types of security assessment are very difficult to carry out. Fundamentally, the reason why security is so difficult to assess is that security requirement, like some safety requirements are shall not requirements. That is they specify what should not happen rather than system functionality or required behavior. It is not usually possible to define this unwanted behavior as simple constraint that may be checked by the system.

There are four complementary approaches to security checking.

1. Experience-Based Validation
   In this system is analyzed against types of attack that are known to the validation team. This type of validation is usually carried out in conjunction with tool-based validation. This approach may use all system documentation and could be part of other system reviews that checks for errors and omission.

2. Tool-Based Validation
   In this case, various security tools such as password checkers are used to analyze the system. Password checks detect insecure password such as common names or strings of consecutive letters. This is an extension of experience based validation

3. Tiger Team
   In this case, a team is set up and given the objectives of breaching the system security. The simulate attacks on the system and their ingenuity to discover new ways to compromise the system security. This approach can be very effective if team members have previous experience with breaking into system.

4. Formal Specification
   A system can be verified against a formal security specification. However, as in other areas, formal verification for security is not widely used.

Software Productivity

A project manager generally performs the productivity of software engineers. These productivity estimates are needed to help, define the project schedule or cost, to inform investment decision or to assess whether process or technology improvements are effective.

Productivity estimates are generally based on measuring attribute of the software and dividing this by total effort required for development. There are two types of metrics that have been used:

i. Size Oriented
ii. Function Oriented

Estimation Techniques

Estimation of various project parameters is a basic project planning activity. The important project parameters that are estimated include project size, effort required to develop the software project duration and cost. These estimates not only help in quoting the project cost to the customer, but are also useful in resource planning and scheduling.

There are various techniques for estimation; some of them are given below:

1. Expert Judgment
2. COCOMO IInd
3. Function Point
4. LOC

Project Scheduling

WBS

A work breakdown structure (WBS) is deliverable oriented decomposition of a project into small components. It defines and groups a project's discrete work element in a way that helps organize and define the total work space of the project.

A work breakdown structure element may be a product, data, a service or any combination. A WBS also provides the necessary framework for detailed costs estimating and control along with providing guidance for schedule development and control.

The WBS is tree structure, which shows a subdivision of effort required to achieve an objective; for example a program, project, and contract. In a project or contract, the WBS is developed by starting with the objectives and successively subdividing it into manageable component in term of size, duration, and responsibility. (Eg. Systems, subsystems, component, tasks, subtask and work package) which includes all necessary steps to achieve the objectives.

The WBS provides a common framework for the natural development of overall planning and control of contract and is the basis for dividing work into definable increments from which the statement of work can be developed and technical, schedule, cost, and labor hour reporting can be established.

CPM (Critical Path Model)

The CPM is a step by step methodology, technique, or algorithm for planning projects with numerous activities that involve complex, independent interactions. CPM is an important tool for project management because it identifies critical and non- critical tasks to prevent conflicts and bottlenecks. CPM is often applied to the analysis of a project network logic diagram to produce maximum practice efficiency. The basic steps involved in CPM are:

1. Determining required tasks
2. List required tasks in sequence
3. Create flowchart including each required task
4. Identify all critical and non critical relationship (path) among required task
5. Assign an expected completion/ execution time for each required task
6. Study all critical relationships to determine all possible alternatives or back up for as many as possible.

Often a major objective in CPM is to complete the project in shortest time possible. One way to do this is called fast tracking, which involves performing activities in parallel and adding resources to shorten critical path durations (called crashing) the critical path. This may results in expressions, which leads to increasing project complexity, durations or both.

PERT Chart

A PERT chart is a project management tool used to schedule, organize, and coordinate tasks within a project. PERT stands for Program Evaluation Review Technique.

A PERT chart presents a graphic illustration of a project as a network diagram consisting of numbered node (either circles or rectangles) representing events, or milestones in the project linked by labeled vectors representing tasks in the project.

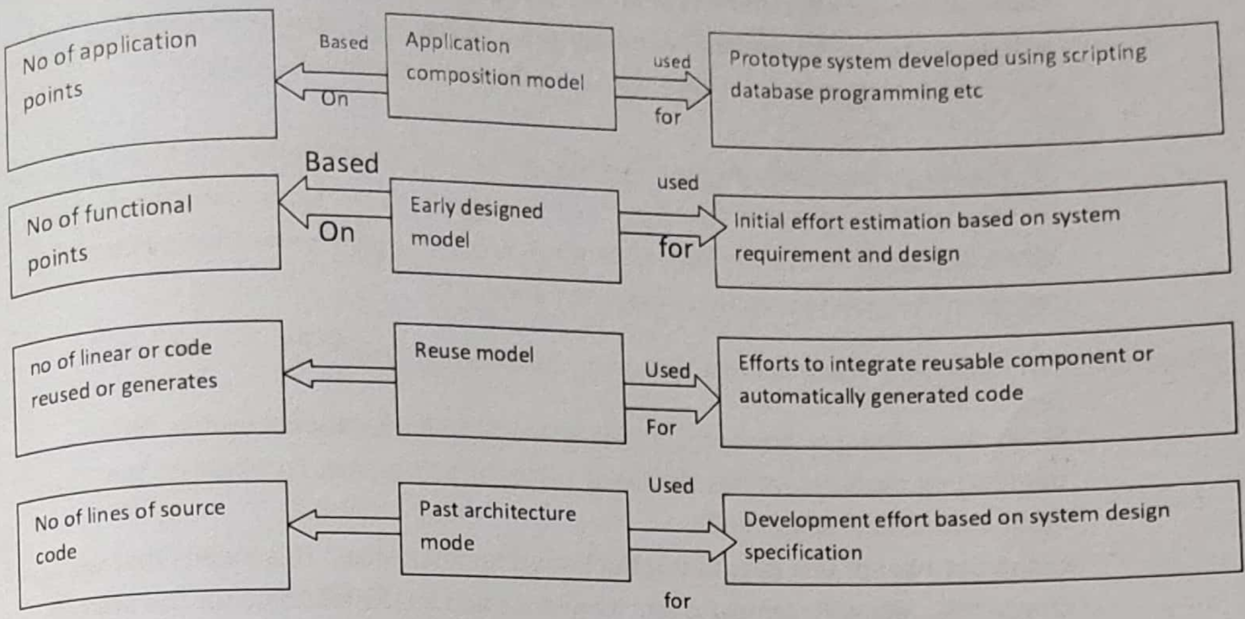
PERT vs. CPM

Pert is event oriented while CPM is activity oriented. In PERT time is related to costs while in CPM, time is related to costs.

COCOMO II

COCOMO II model recognizes different approaches to software development such as prototyping, development by component composition, and use of database programming. COCOMO II supports spiral model of development and embeds several sub model that produce increasingly detailed estimates. These can be used in successive rounds of development spiral. Fig below shows the COCOMO II sub model and where they are used:

135



i. An application Composition Model

This assumes that the system is created from reusable component, DB programming or scripting. It is designed to make estimates of prototype development. Software size estimates are based on application point, and a single size (productivity formula) is used to estimate the effort required. Application points are same as object points which are alternative to function point.

ii. An Early Designed Model

This model is used during early stages of the system design after the requirements have been established. Estimates are based on function points which are then converted to no of line of source code. This method uses standard formula of cost estimation with a simplified of 7 multipliers.

iii. The Reuse Model

This model is used to compute effort required to integrate the reusable component and or program code that is automatically generated by design or program translation tool. It is generally used in conjunction with post- architectural model.

iv. A Post- Architectural Model

Once system architecture has been designed, a more accurate estimation of software size can be made. This model also uses the standard formula of cost estimation. However, it includes a more extensive set of LT multipliers.

In large system different part may be developed using different technologies and we may not have to estimate all parts of the system to the same level of accuracy. In such cases, we can use the appropriate sub model for each part of the system and combine the result to create composite estimates.

The process of measuring reliability of a system involves four stages:

i. Study the existing systems of same type to establish operational profile. An operational profile identifies classes of system inputs and the probability that these input will occur in normal use.
ii. Construct a set of test data that reflect operational profiles. This means that we create test data with same probability distribution as the test data for the system that have been studied.
iii. Test the system using test data and count the number and types of failures that occurs. The time of these failures are also logged.
iv. After observing a statistical significant no. of failures, compute the software reliability and workout the appropriate reliability metric.