Unit:-1 Introduction to Software Engineering

Definition of software engineering

Software is an organized collection of relevant programs and documentation. A software system consist of a number of several programs, configuration files which are used to set of these programs and system documentations which describe the structure of system and user documentation which explain how to use the software.

Definition of software engineering

Software engineering is an engineering approach for software development. It is an engineering discipline which is concerned with all aspect of software production. Software engineering is a discipline whose focus is the cost effective development of high quality software.

We can view it as a systematic collection of past experience. The experience is arranged in the form of methodology and guidelines. A small program can be written without using software engineering principles. But, if one wants to develop a large software product, then software engineering are necessary to achieve good quality software. Without using software engineering principals, it would be difficult to develop large programs. In industry, it is usually needed to develop large programs to accommodate multiple functions. A problem with developing such large program is that the complexity and difficulty levels of the programs increase exponentially with their size.

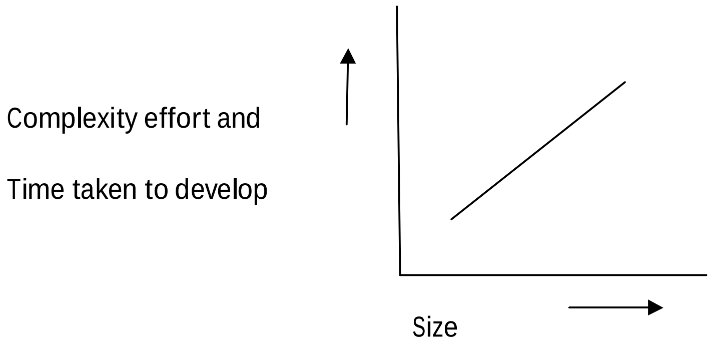


Fig:-increase the complexity according to size

Software engineering helps to reduce the programming complexity. Software engineering principles use two important techniques to reduce programming complexity abstraction and decomposition.

The principal of abstraction implies that a problem can be simplified by omitting irrelevant details. The main purpose of abstracting is to consider only those aspects of problem that are relevant for certain purpose, and do not consider other aspects that are not relevant for the given purpose. Abstraction is a powerful way of reduce the complexity of problem.

The other approach to tackle problem complexity is decomposition. In this technique a complex problem is divided into several smaller problems, and then the smaller problems are solved one by one. However, in this technique is any random decomposition of problem into smaller part will not help. The problem has to be decomposed such that each component of the problem can be solved independently and then the solution of different components can be combined to get the full solution. If the different sub component can be solved separately. In this way, the decomposition process minimizes the complexity of problems.

In general, software engineers adopts a systematic and organized approach to their work as this is often the effective way to produce high quality software.

Characteristics of software

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

i. Software does not wear out

Hardware exhibits relatively high failure rate in its life cycle. Defects are corrected for sometime as the time passes, the failure rate rise again as hardware component buffer from the commutative effect of dust, temperature and other many environmental factor. So we say that hardware begins t

Software is not susceptible to environmental factor that cause hardware wears out. Undiscovered defect will cause high failure rate early in the life of program. However, these are corrected during its lifecycle and software will undergo changes and maintenance. Software can be changed easily during its lifecycle.

Although some similarities exists 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 attribute that reflect the quality of that software. These attributes are not directly concerned with what the software does. They reflect its behavior while it is executing, the structure and organization of source program and associated documentation. The essential attributes of good software are:

a. **Maintainability**

b. Software should be written in such a way that it may evolve to meet the changing requirement of the customers. This is a critical attribute because software changes continuously to satisfy the changing business environment and customer requirement.

c. **Efficiency**
Software should not make wasteful use of system resources such as memory and processor. Efficiency includes responsiveness, processing time, memory utilization, etc.

d. **Usability**
Software must be useable without wasting effort of user for whom it is designed. This means that it should have appropriate user interface and adequate documentation.

e. **Dependability**
Software dependability has a range of characteristics including reliability, security and safety. Dependable software should not cause physical and economical damage in the event of system failure.

Challenges Facing Software Engineering

Software engineering focus on the following challenges:

i. Heterogeneity Challenge
Every organization is required to operate as a distributed system across the network that includes different types of computer and with different types of supporting system. The heterogeneity challenge is the challenge of developing software that is flexible enough to cope with heterogeneous system.

ii. Delivery Challenge
Traditional software engineering techniques are time consuming. The time they take is required to achieve software quality. However, business organization today must be responsive and change very rapidly. Their supporting software must change equally. The delivery challenge is the challenge to provide the software within short time without compromising software quality.

iii. Trust Challenge
Trust challenge is to develop a technique that demonstrates that software can be trusted by its user.

Difference between software engineering and computer science

Computer science is concerned with the theory and methods that underlies computer and software systems. Whereas, software engineering is concerned with the practical problem of producing software. Some knowledge of computer science is essential for software engineers as the same knowledge of physics is essential for electrical engineers.

Difference between software engineering and system engineering

Software engineering is concerned with all aspect of the development and evolution of the computer system or other complex system. Whereas software plays major roles in system engineering. System engineering is there for, concerned with hardware development process design.

Software process

A software process is the set of activities and associated results that produce a software product. There are four fundamental process activities that are common to all software process. They are

1. **Software specification**
   Software specification specify what software is to be produced, requirements of users feasibility of software and other constraints on its operation.
2. **Software development**
   In this activity, the software is designed and programmed according to the software specification.
3. **Software validation**
   In this activity, the software is checked to ensure that it performs its task according to software specification and satisfies user requirements.
4. **Software evolution**
   In this activity, software is modified to adopt users changing requirements and market requirement.

Software Lifecycle Model/Software Process Model

Software lifecycle model is a descriptive and diagrammatic representation of the software lifecycle. A lifecycle model represents all the activities required to make a software product through its lifecycle phase. It also captures the order in which these activities are to be undertaken.

A lifecycle model maps the different activities performed on software productions from its inception to retirement. Different lifecycle models may map the basic development activities to phases in different ways. Thus, no matter which lifecycle model followed, the basic activities are included in all lifecycle models.

Software characteristics

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

1. Although some similarities exist between software development and hardware Manufacture, the two activities are fundamentally different in both activities, high quality is achieved through design.

2. Software does not wear out (expire). Hardware exhibits relatively high failure rate early in its lifecycle. Defects are corrected and the failure to steady state level for some time. As time

passes, the failure rate rise again as hardware component suffer from the cumulative effects of dust vibration, temperature and many other environmental factor stated simply the hardware begins to wear out.

Software is not susceptible to environmental maladies that cause hardware wear out. Undiscovered defects will cause high failure rate early in the life of programs. However there are corrected during its life, software will undergo changes or maintenance. As the changes are made it is likely that some new defects will be introduced.

System

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

The successful functioning of each system depends on the functioning of some other component. For example software can only operate if the processor can only carry out computation if software system defining these computations has been successfully installed.

System Properties

1) Volume
2) Usability
3) Repair ability
4) Security
5) Reliability

1). Volume The volume of system (total space occupied) varies depending on how the component assemblies are arranged and connected.

2). Usability: This property reflects how easily the system is to be used. It depends on the system component, its operator and its operating environment.

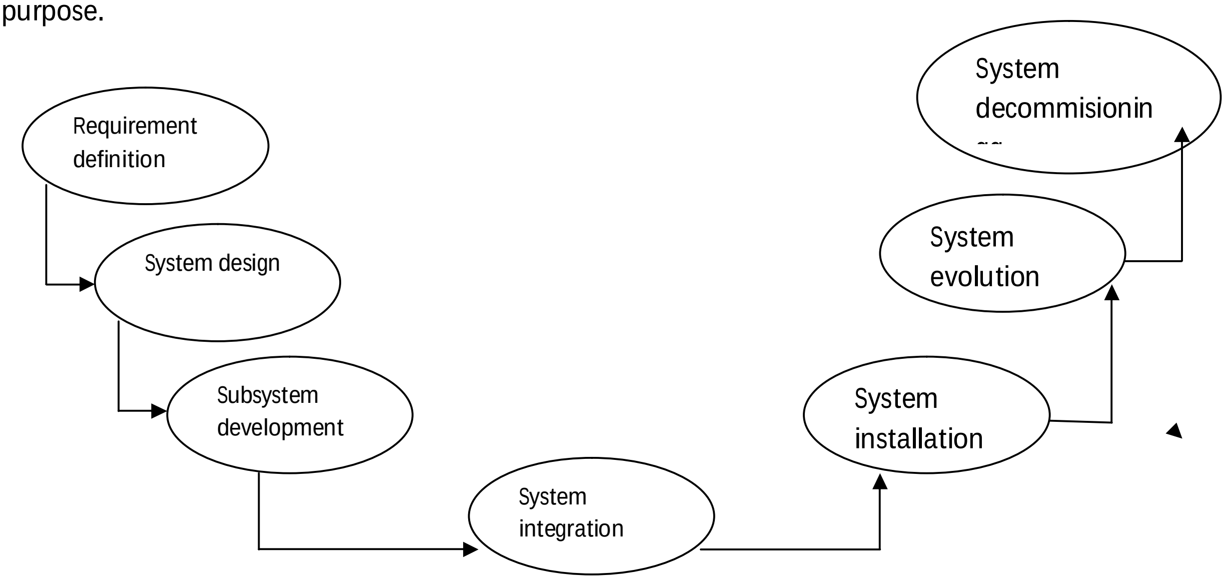
3). Repair ability: This property of system reflects how easily the problem in a system is to be fixed and removed. It depends on ability to diagnosis the problem and modifies or replaces components that cause problem.

4). Security: The security of the system is complex property that cannot be easily measured. The security property of system depends on the design issue.

5). Reliability: System reliability depends on component reliability unexpected interaction between Component can cause new type of failure and affect the reliability of system.

System Engineering

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



1. Requirement Definition
   System requirement definition specifies what the system should do and essential and desirable system property. The system requirement definition phase involves the consultation with the system customer and end user. The requirement definition phase usually concentrated on driving three types of requirements.
   i. Abstract functional requirement
      The basic functions that the system must provide are defined at an abstract level. More detailed functional requirement specifications are specified at the subsystem level.
   ii. System properties
      There are non functional system properties such as availability, usability, performance, and safety. These non functional system properties affect the requirement for all subsystem.
   iii. Characteristics
      It is some time 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 definition phase is to establish a set of overall objective that the system should meet.
2. System Design
   System design is concerned with how the system functionality is to be provided by the system component. The system design phase involves the following activities:

i. Partitioning requirement
Analyze the requirement and organize them into related groups.

ii. Identify Subsystem
In this activity, the subsystem should identify that can individually and collectively meet the requirement. Groups of requirements are usually related to subsystem.

iii. Assign Requirement to Subsystem
The related groups of requirements are assigned to the subsystem.

iv. Specify Subsystem Functionality
The specific function provided by each subsystem is specified. The relationship between subsystems is also identified at this activity.

v. Define Subsystem Interface
Define the interface that are provided and required by each subsystem. Once these interface have been agreed, then it become possible to develop subsystem in parallel.

There is a lot of feedback and iteration from one activity to another activity in this phase. This process ends when the review and evaluation show that the requirement and high level design are sufficiently detailed to allow the next phase of system design.

3. System Modeling
During the system requirement and design activities, the system may be modeled as a set of components and relationship between these components. These are normally illustrated graphically in a system architecture model that gives the overview of the system organization or structure.

The system architecture may be presented as block of diagrams showing the major sub system for drawing a block diagram, we should represent each sub system using a rectangle and relationship between these sub systems.

4. System Development
During the subsystem development, the subsystem identify during system design are implemented. This may involve starting another engineering process for individual subsystem. All subsystems are developed from sketch design during development phase. Subsystems are usually developed in parallel.

Software Process

A software process is a set of activities that leads to the production of software product. These activities may involve the development of software from scratch in a standard programming language. Computer added software engineering tools can support some processing activities. There is no ideal process and many organizations have developed their own approach to software development. There are many software processes.

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.

A: Water fall model

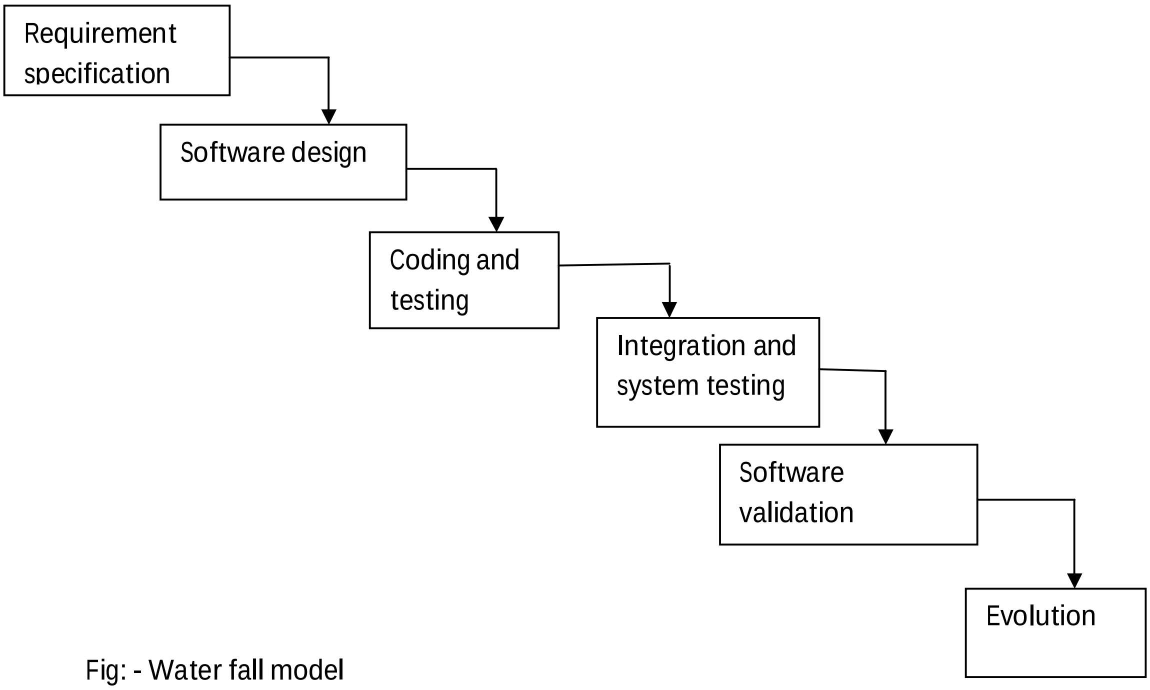


Fig: - Water fall model

In the waterfall model, the fundamental process activity: specification, development, validation, and evolution are represented as a separate process phases such as requirement specification, software design, coding ,and testing, integration and system 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 requirements are identified and analyzed. At the end of this phase, a written document is generated called requirement specification. This phase consist two activities: requirement gathering and analysis and requirement specification.

The goal of the 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 contradictions since each user typically has only a partial and incomplete view of the system. Therefore it is necessary to identify contradiction and ambiguities in the requirements and resolve them through further discussion with the customer.

After that the requirement specification activity can start. During this activity, the user's requirements are systematically organized into a 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 flow chart for each component of software are designed.

3. 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 the design is implemented as a program module. The end product of this phase is s set of program that has been tested individually. During this phase, end module is tested to determine the current functioning of the individual module and remove error is any. Unit testing is performed to verify that each unit needs its specification.

4. Integration and System Testing

   Interpretation 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 an almost never integrated into one shot. Integration is normally carried out inclemently over a number of steps. During each integration step, 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 plan document identifies all testing related activities that must be performed, specify the schedule of testing and allocated required resources.

5. Implementation

   After testing the system, the software is implemented for operational use. During the software implementation phase, the software is installed in a suitable machine environment according to the implementation phase. During implementation phase, the file conversion activities are performed.

6. Evaluation
   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

The advantage of waterfall model is that documentation is produced at each phase and that is fit with other engineering process models. Its major problem is it's inflexible partitioning of the software development process into distinct stage. Commitments must be made at an early stage in the process which makes it difficult to respond to changing customer requirement

Therefore the waterfall model should only be used when the requirements are well understood.

B. 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 versions until an adequate system has been developed. Specification, development, and validation activities are inter-leaved than separate with rapid feedback across activities. There are two fundamental evolutionary development models:

a. Prototyping Model
   The prototype concentrates on experimenting with the user requirement that are poorly understood.

b. Exploratory Model
   The objectives of this model are to work with the customers to explore their requirement and deliver a final system. The development starts with the part of the system that is understood. The systems are expanded by adding new features purposed by customers.

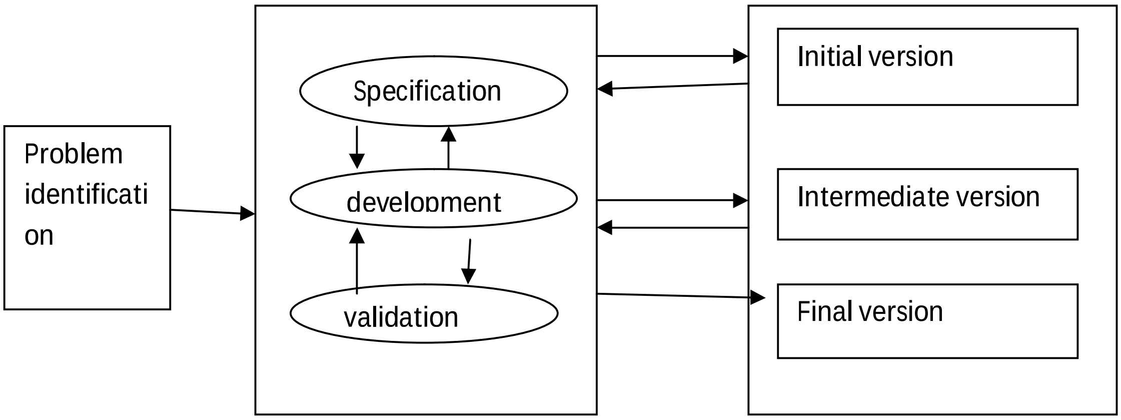


Fig: - Evolutionary Development Model

Necessity of prototype

There are several users of prototype and important purpose of prototype is to illustrate input data format, message, reports, data flow etc to the user. This is a valuable mechanism for gaining better understanding of costumers. Another reason for developing prototype is that it is impossible to get a perfect product in the first attempt. Many researchers 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 final product.

C. Component Based Software Engineering Model

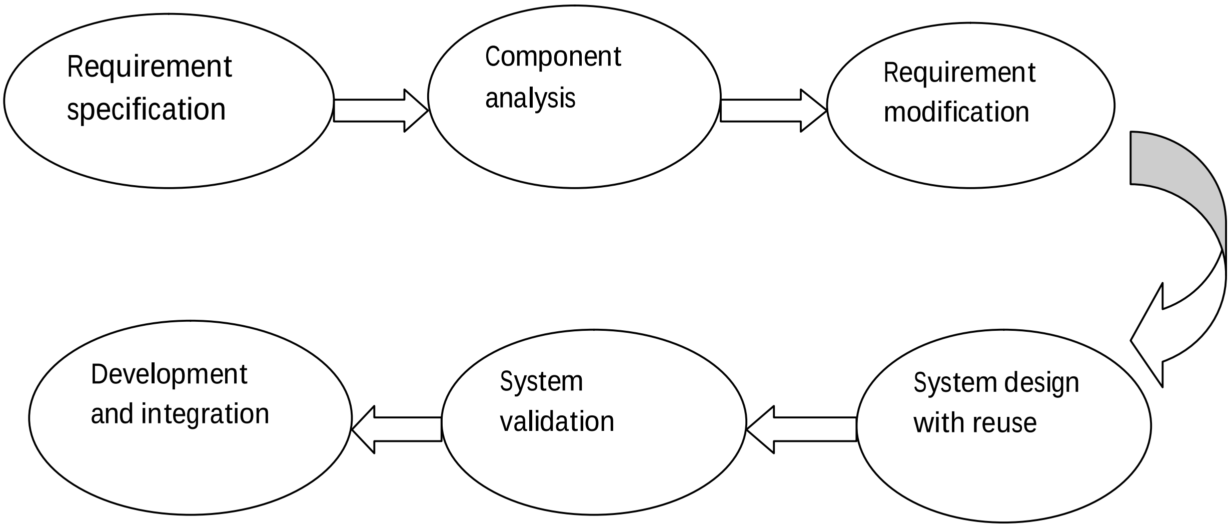


Fig: - component based software engineering model

This method relies on a large base of re-useable software components and some integrating frameworks for these components. Sometimes these components are system that may provide specific functionality such as text formatting, numerical calculation etc.

The requirement specification stage and validation stage are comparable with other processes. The intermediate stages of components based model are different.

i. **Requirement specification**
   In this stage the system service, constraints and goals are established by consultation the system. The system user requirements are identified and analyzed. After that a document is generated called requirement specification.
ii. **Component analysis**
   In this stage, a search is made for component to implement the specification. Usually there is no exact match and the component that may be used only provides some functionality required for software.
iii. **Requirement modification**
   During this stage requirements are analyzed using information about the components that have been discovered. They are then modified to reflect the available components. Where modifications are impossible, the components analysis activity may re-enter to search for alternative solution.

iv. **System design with reuse**
   During this phase, the framework for system is designed or an existing framework is reused. The designer takes into account. The components that are reused and organized the frame work to cater the requirement specification. If reusable components are not available design the frameworks for each component.
v. **Development and integration**
   The components that cannot be exists externally is developed and components are integrated to create the new system.
vi. **System validation**
   In this phase, the development system is verified and checked whether it satisfy requirement specification property or not. If the system do not specify user requirements, the new version of system should be redesigned.

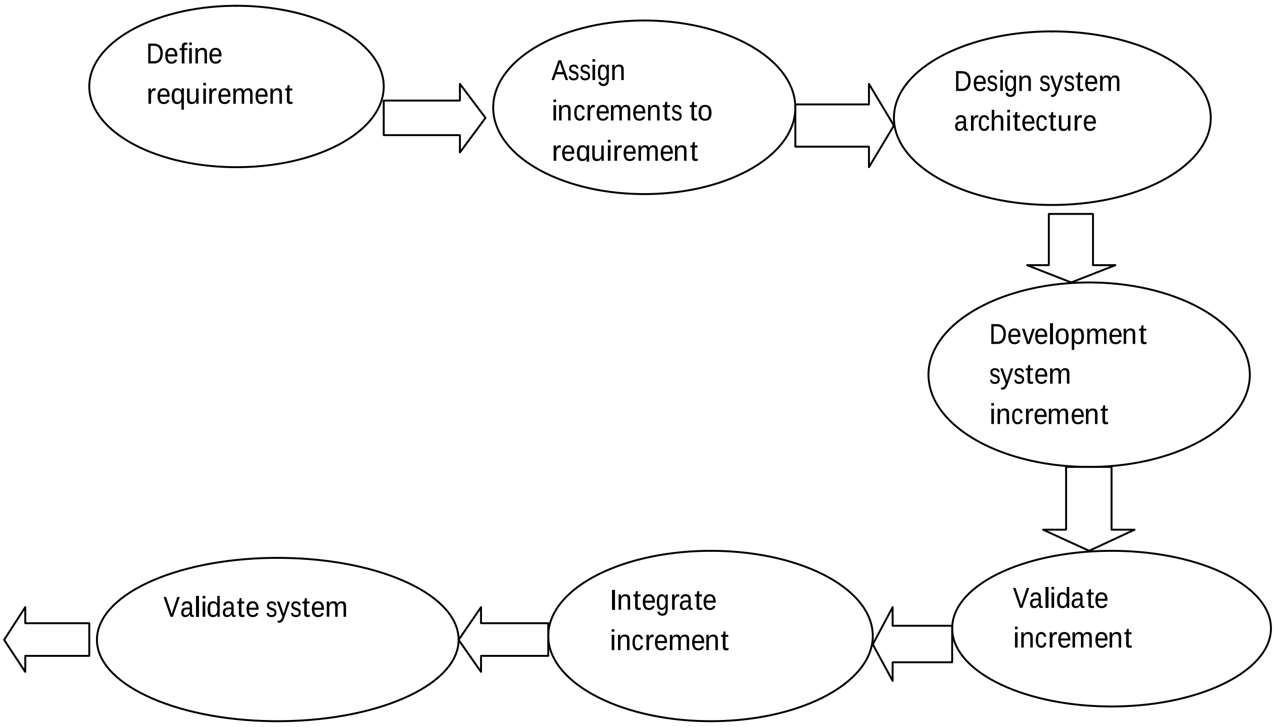
The component based software engineering model has the advantage of reusing the software components that have to be developed and reducing cost of software development. It usually leads to faster delivery of the software.

Process iteration

Change is inheritable in all large software projects. The system requirement changes as the business procuring the system respond to external pressure. Management properties are also changed. As a new technology become available, design and implementation change. This means that the software process activities are regularly repeated in response to change request. Two process models that had been explicitly designed to support process iteration.

1. **Increment delivery**

   The software specification, design and implementation are broken down into a series of increments that are developed.


Define requirement
Assign increments to requirement
Design system architecture
Development system increment
Validate increment
Integrate increment
Validate system

The water fall model of software development requirement requires costumer for a system to commit to a set of requirements before design begin and the designer commit to a particular design strategy before implementation to the requirements require rework of the requirement, design and implementation. However, the separation of design and implementation should lead to well documented system.

Incremental delivery is an approach that combines the advantage of these models. In incremental development process costumers identify the services to be provided by the system. They identify which of the services are most important to them. A number of increments are defined with each increment providing a subset of the system functionally. The allocation of the services priority service delivered first.

One the system increments have been identified the requirements for the service to be delivered to the first increment that increment is developed. During development further requirement analysis for later increment can take place, but requirement changes for the current increments are not accepted. Once an increment is completed and delivered costumers can put it in to service. This means that they take early part of the system that helps them clear their requirements for the later functionality expands with each in increments.

Advantage of incremental delivery

i. Costumers do not have to wait until the entire system is delivered. Costumer gains valuable information from the increments satisfies their most critical requirements so they can use the software immediately.

ii. Costumer can use the early increments as a prototypes and gain experience that inform their requirements for later increments.

iii. There is lower risk of overall project failure.

iv. As the high priority service are delivered first and the later increments are integrated with them.

2. Spiral Model

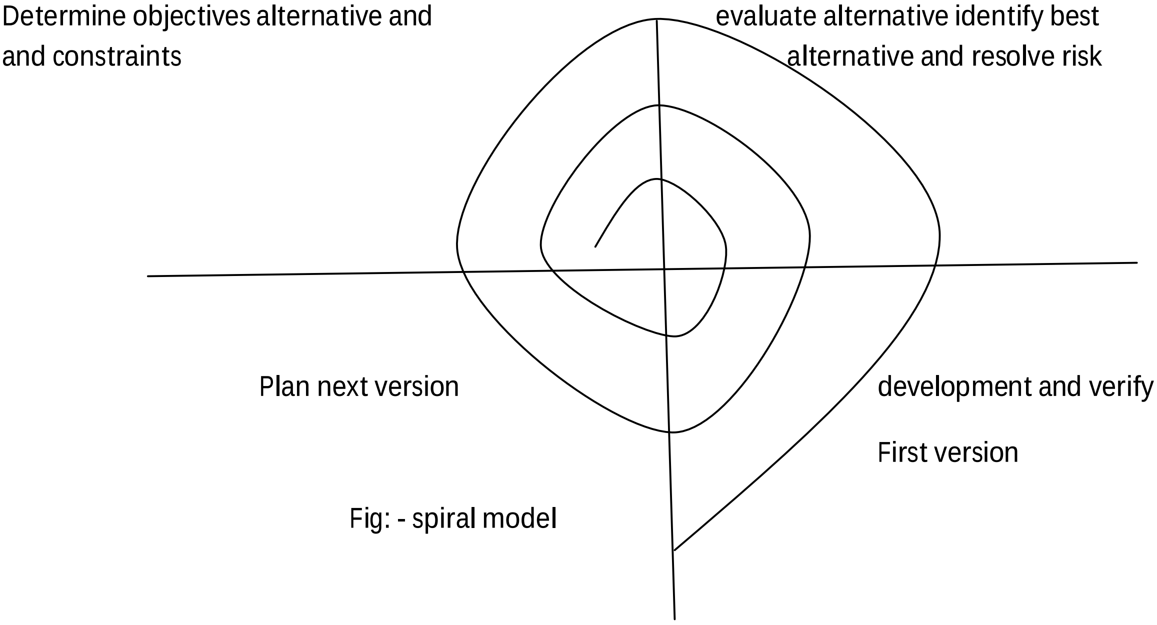

Determine objectives alternative and
and constraints
evaluate alternative identify best
alternative and resolve risk
Plan next version
development and verify
First version

Fig: - spiral model

The spiral model represents the software development process as a spiral rather than a sequence of activities. Each loop in the spiral represents the phase of software development process. In spiral model, total software development process activities are divided into four group and each quadrant represents specific group. The first quadrant of the spiral model represents the requirement determination and object testing, the second quadrant represent requirement evaluation and risk analysis, third quadrant represent development and validation and fourth quadrant represents planning. After combination of the each iteration, a new version of software is developed. This iterative process is continued until the fully satisfied systems have been designed.

- Requirement determination and object setting
  In this quadrant, user requirements are identified and specific objective of the system are identified. Constraint on the process and products are identified and a detailed management plan is drawn up. During this, the alternative solutions are specified and analyzed.

- Evaluate Requirement and Risk Analysis
  Feasibility study should be performed to verify the requirements and check whether the project is feasible or not. Project risks are identified and analyzed. After analyzing risk, proper actions are carried out to reduce the risk.

- Development and Validation
  After the risk evaluation, a development model for the software is designed. The software are developed and verified according to the requirement specification.

- Planning
  The project is received and a decision is made whether to continue the project or not. If it is decided to continue, plans are drawn out for the next phase of the project.

  The main difference between spiral and other software process model is that the explicit re-organization of the risk is in the spiral model.

Software Activities

Four basic software activities are requirement specification, software design and implementation, software validation and software evolution are organized differently in different development model. In water fall model, they are organized in sequence, where as in evolutionary model, they are carried out depending upon the type of software, development personnel and organizational structure. There is no right or wrong ways to organize these activities.

1. **Software specification**
   Software specification is the process of understanding and defines what services are required from the system and identifying the constraints on the system operation and designing.

Requirement engineering is a particularly critical stage of the software development process, as errors at this stage inscrutably lead to problems in the system design and implementation. The requirement engineering leads to the production of the requirement documents that is the specification for the software requirements are usually presented at two levels user requirements and system requirements.

There are four main phase in the requirement specification activity.

i. **Feasibility Study**
   An estimate is made of whether the identified user requirements may be satisfied by using current software and hardware technology. The study considers whether the proposed system will be cost effective from a business point of view and whether it can be developed within existing budget.

ii. Requirement Elicitation and Analysis

This is the process of deriving the system requirements through observation of an existing system discussion with potential user and task analysis. This may involve the development of core system model and prototype. This helps the system analyst to understand the system requirement and operation.

iii. Requirement Specification

The activity of translating the information gathered during the requirement determination and analyze into the documents that defines the set of requirements. In requirement specification, two types of requirement may be included in the document: user requirement and system requirement.

iv. Requirement Validation
   The activity checks the requirement for realism, consistency and completeness. During this phase, errors in the requirement document are discovered and modified to correct those errors.

2. Software Design and Implementation:
   The implementation stage of the software development is the process of converting system specification into an executable system. Software design is description of the structure of the software to be implemented the data which is part of the software, the interface between software component and algorithm used. Designers do not arrive at a finished design immediately but develop the design iteratively through a number of versions.

The design process may involve developing several models of the software at different level of abstraction. As the design is decomposed, errors in earlier stage are discovered. This feedback allows earlier design model to be improved.

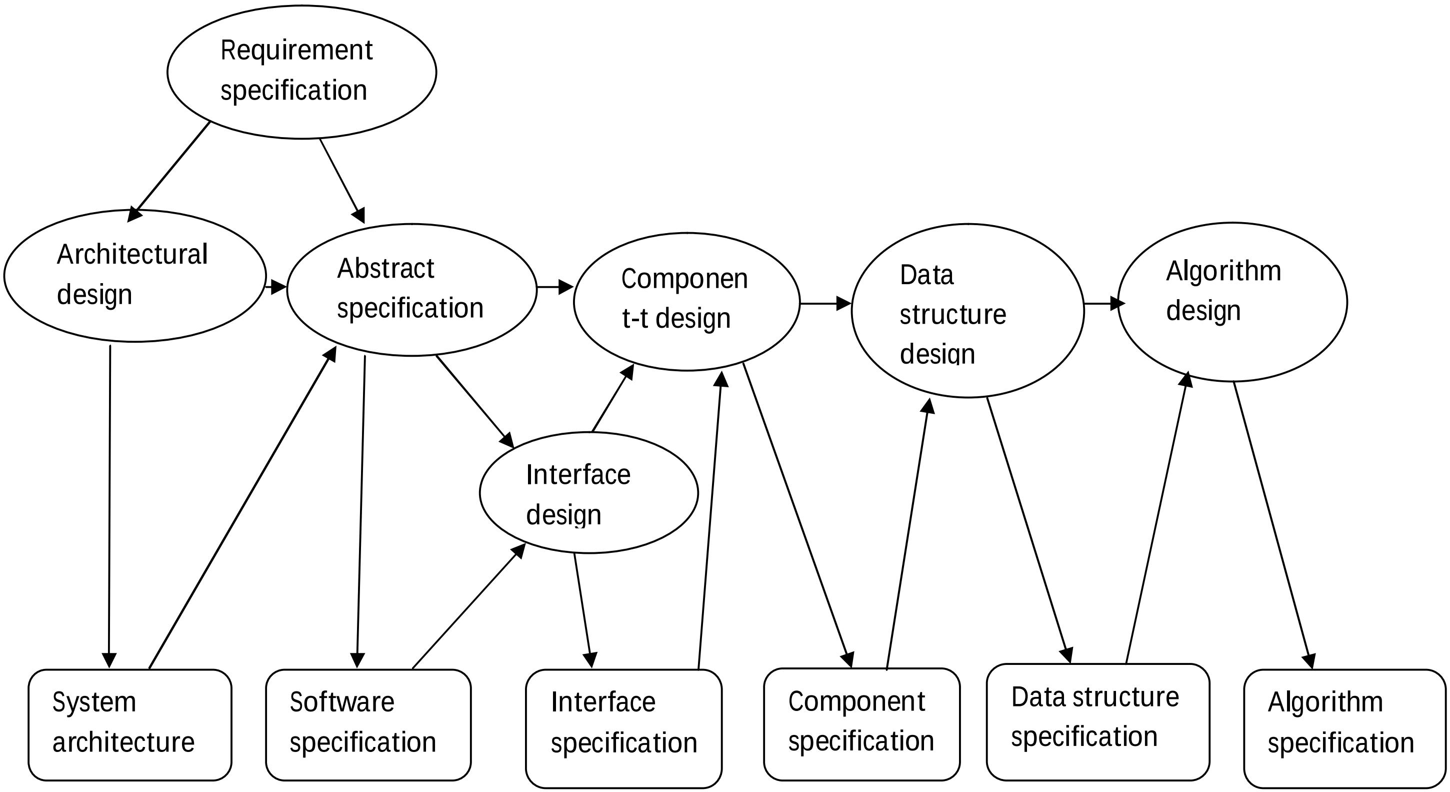


Fig: - general model of design process

The diagram suggests that stages of the design process are sequential. In fact, design process activities are interleaved. The specification for the stage is the output of each design activity. The specification may be an abstract and formal specification. That is produced to clarify the requirement or it may be a specification of software part is realized. As the design processes continue this specification become more detail. The specification design process activities are:

1). Architectural design

The system making of the system and their relationship are identified and documented.

2). Abstract specification

For each sub system and abstract specification of its services and constraint under which it must operate are identified and specified.

3). Interface design

For each sub system its interface with other sub system is design and documented. The interface specification must allow the sub system to be used without knowledge of the sub system operation.

4). Component design

Services are allocated to components and the interfaces of these components are designed.

5). Data structure design

The data structure used in the system implementation is designed in detail and specify.

6). Algorithm design

The algorithm used to provide services are designed in detail and specify.

This is a general model of design process. The real and practical process may adopt it in different ways. Possible adoptions are

a) Last two stages of design, data structure and algorithm design may be delayed until the implementation process.
b) If exploratory approach to design is used, the system interface may be designed after the data Structure has been specified.
c) The abstract specification stage may skipped, although it is usually an essential part of critical system design.

System validation

System validation is the process of checking the software to ensure that it works exactly according to the requirement specification. During software validation process, the software is tested with different types of users on actual working data. If software works properly with specified environment and actual operating data, then it is said to that the software confirms user requirement specification. It involves checking process such as inspection and reviews at the stage of the software process from the user requirement definition to program development. The majority of validation costs are incurred after implementation, when the operational system is tested. System should not be tested as a single unit. It should be performed in different stage.

1). Component or unit test

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

2). System testing

The components are integrated to make up the system. This process is concerned with finding errors between two components and interface. It also concerned with validated that system meet its functional and non functional requirement.

3). 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 costumer rather than with simulated.

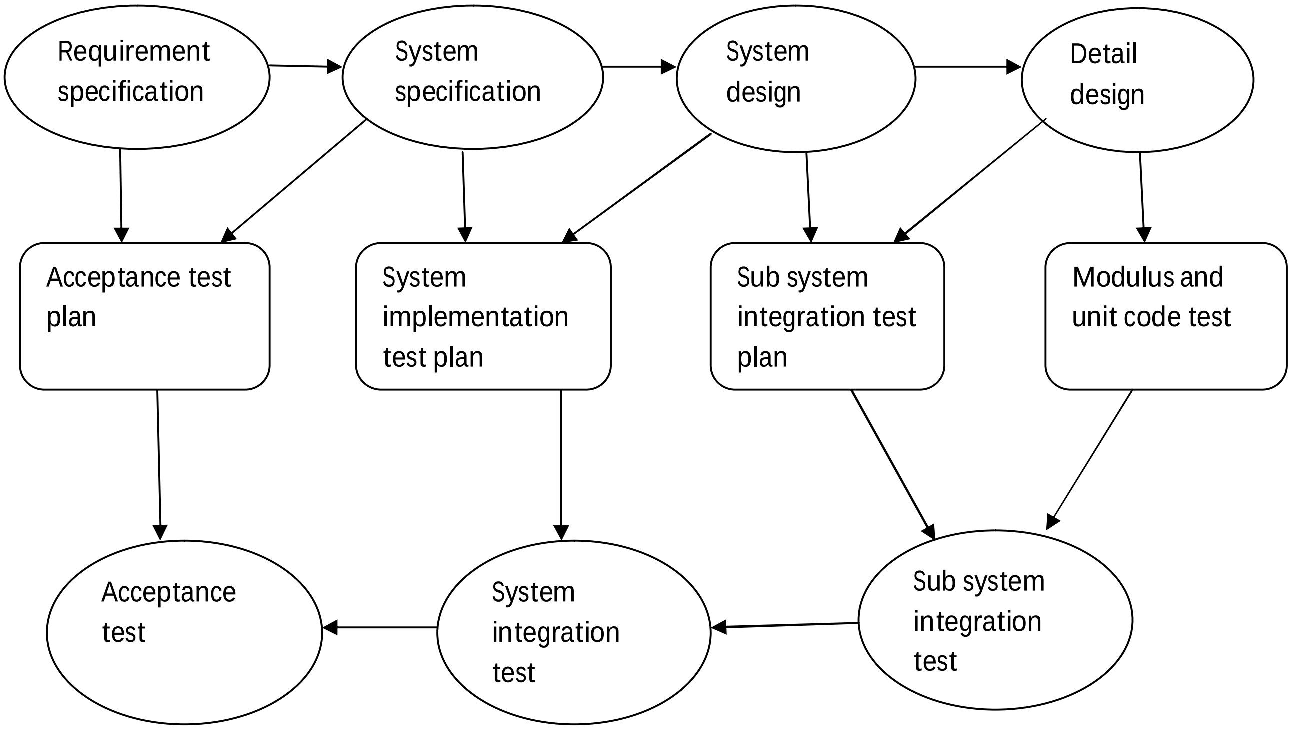

Requirement specification
System specification
System design
Detail design
Acceptance test plan
System implementation test plan
Sub system integration test plan
Modulus and unit code test
Acceptance test
System integration test
Sub system integration test

Fig: - testing phases in software testing

Normally component development and testing are inter leaved. Programmer make up their own test data and test the code as it is developed.

Software Evolution

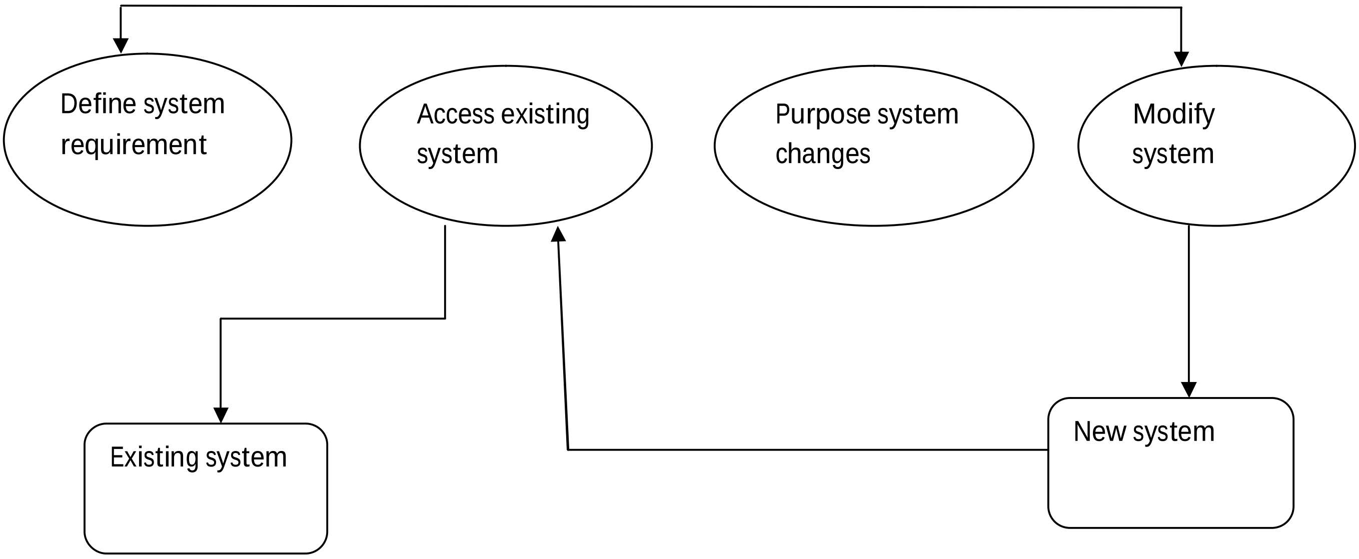


Fig: - software evolution

The feasibility of software system is one. For example the main reason whiles more and more software is being incorporated in large and complex system. Once a decision has been made to change hardware, it is very compensate to change the hardware at any time during or after the software development. There has always been a split between the process of software development and process of software evolution. Software systems was a system while software evolution process starts from existing system and make some changes to existing system or replace existing system.

Project Management

Software project management is an essential part of software engineering project management evolves the planning, monitoring and control of the person, process and events that occur as software evolves from a primary concept to an operational implementation; software managers are responsible for planning scheduling project development. They supervise the work to ensure that it is carried out to the required standards and monitor program to check that development is on time within budget.

Software managers do the same kind of jobs as other engineering project manager. However software engineering is different from other type of engineering of a number of ways. Their destination makes software project management particularly difficult. Some of the difficulties are:

1). The Project is Intangible

The manager ship building project or civil engineer project can see the product being developed. If a selling series, an import on he product is visible. It is easy to handle the project software is intangible. Software project manager cannot see the progress, so it is difficult to schedule and manage activities

2). There is no standard software process

In engineering disciplines with a long history, the process is tried and tested. Software processes vary dramatically from one organization to other.

3). Large Software Projects are often one of Project

Large software projects are usually different to some ways from previous project.

Project management involves the planning, monitoring, and control of the people, process, and event that occur as software evolves from a preliminary concept to an operational implementation.

Software project management begins with a set of activities that are collectively called project planning. Before the project can begin, the manager and the software team must estimate the work to be done, the resources that will be required and the time that will taken from start to finish. It would seem responsible to develop an estimate before start creating the software. Estimation begins with a description of the scope of the product. Until the scope is bounded, it is not possible to develop a meaningful estimate. The problem is then decomposed into a set of smaller problem and each of these is estimated using historical data and experience.

Management activities

It is impossible to write a standard job description for software manager. The job varies depending on the organization and the software product being developed. However, most managers take responsibilities at some stage for some all of the following activities.

1). Personal writing

The first stage in software project may involve writing a proposal to carry out the work and win the contract. The personal describes the objectives of the project and how it will be carried out. It usually includes cost and schedule estimation and justify why the project contract should be awarded to a particular organization. Proposal writing is a critical task as the existence of many software organizations.

2). Project planning and scheduling

Project planning is concerned with identifying activities, objective and deliverable produced by a project. The software manager and software team must estimate the work to be done. The resource that will be spend from start to finish. A plan is drawn up to guide the development towards the project goal

3). Project cost

Cost estimation is related activity that is concerned with estimating resources required to accomplish the project. Estimation begins with a description of the scope of the project. Until the scope is bounded, it is not possible to develop meaningful estimation.

4). Project monitoring and reviews

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

5). Personal selection and evaluation

Project manager usually have to select personal to work on their project. Skilled staff will appropriate experience will be available to work on the project. However, in most cases manager have to settle for a less experience projecting. The reasons for these are

a) The project budget may not cover the use of highly paid staff. Less experienced, less paid staff may have to be used.
b) Staff with the appropriate experience may not be available either within organization or externally.
c) The organization may wish to develop the skill of its employee.

6). Report writing and presentation

Project manager are usually responsible for writing on the project to both for client and contractor organization. They have to write concise, clear document that abstract critical information from the detail project report. They must be able to present this information during project review.

Project Planning

Effective management of software depends on the planning of the project. Manager must anticipate problem that may arise and prepare temptation solution to these problems. A plan draw up at the start of project should be used as the driver for the project. This initial plan should be the best possible plan that gives the available information. Planning is an iterative process which is only complete when the project itself is complete. As project information available during the project the plan should be regularly revised.

At the beginning of planning process, manager should estimate the constraints such as delivery date, staff available, budget etc, that effect the project. Manager should also estimate project parameter such as its structure, size and distribution of function. Then manager define the progress then enter a loop. A estimated schedule for the project and the activities defined in the schedule are started. After some time, manager should review progress and discrepancies from the project parameters are tentative, manager will always have to modify the original plan. As more information becomes available, manager revises his original assumption about the project and the project schedule.

Steps of project planning

Establish the project constraints make initial estimation of project parameters. Define project milestones and deliverables while project has not been completed or cancelled loop.

Drawn up project plan, initial activities according to schedule, review project progress, revise estimates of project parameters, renegotiate project constraints and deliverables, if (problem arise) then initiate technical review and possible revision.

```
Establish the project constraint
Make initial assessment of the project parameters
Define project milestones and deliverables
While project has not been completed or controlled
Loop
Draw a project schedule
Initiate activities according to schedule
Review project progress
Revise estimate of the project parameter
Update the project schedule
Re- negotiate project constraint and deliverables
If(problem arise) then
Initiate technical review and possible revision
End loop
```

Project plan

The project plan sets out the resources required for the project, resource available to the project, the work brisk down and schedule for carrying out the work. The detail of the project plan depending on the type of project and organization. Most plans should include the following section.

1). Introduction

It describes the objective the project and sets out the constraint such as budget, time, resource etc that affect the project management.

2). Project organization

It describes the way in which the development team is organized, the person involved and their role in the team.

3). Risk analysis

It describe possible project risk, the likelihood of this risk arise and the risk reduction strategy that are proposed.

4). Hardware and software resources requirement

It specifies the hardware and the supported software required to carry out the development. If hardware has to be bought estimate its price.

5). Work break down

It sets out the breakdown of activities of the project in to activities and identifies milestone and deliverables associated with each activity.

6). Project schedule

It shows the dependencies between activities, the estimated time required to reach milestone and the allocation of person to activities.

7). Monitoring and reporting mechanism

It defines the management reports that should be produced and the project monitoring mechanism used to control the development activities and report generation.

Manager should regularly revise the project plan during the project development. Some part of the project plan will changes frequently as the project progress towards and more information is available.

Milestone and deliverables

Managers need information to do their jobs. Because software is intangible, the information can only be provided as report and document that describe the state of software being developed. When planning a project, manager should establish a series of a milestone where milestone is a recognizable end point of software process activity. At each milestone there should be a formal out put such as report that can be presented to the management. Milestone report need not be large document. Millstone should represent the end of a distinctly logical stage in the project.

A deliverable is a project result that is delivered to the costumer. It is usually delivered at the end of some major project stage such as specification, design, and implementation. Deliverables are usually milestone, but milestone need not be deliverable. To establish milestone, the software process must be break down on to activities with associate output.

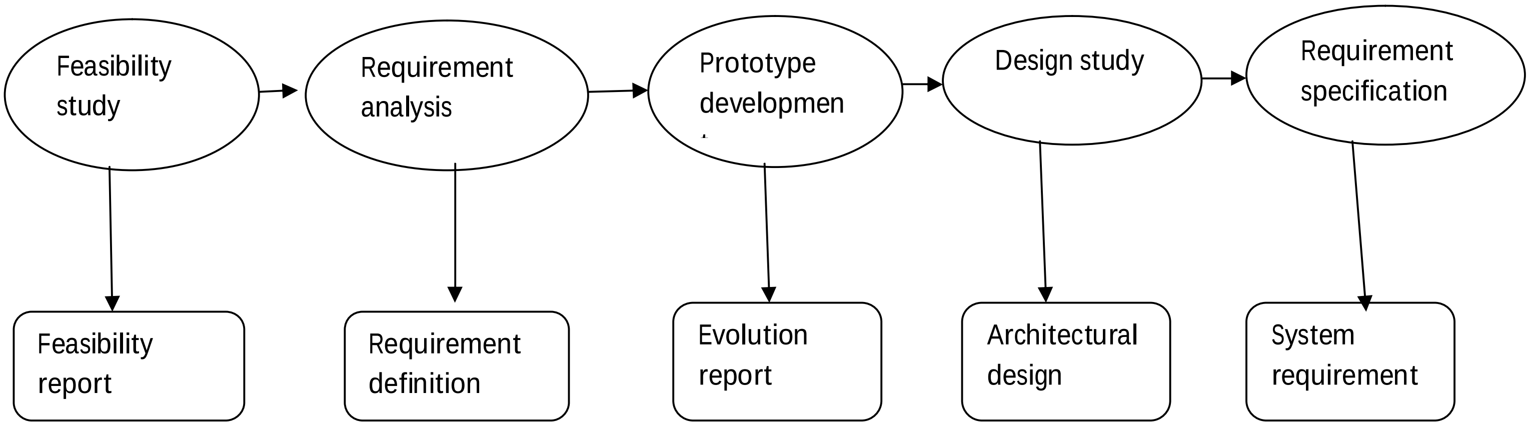
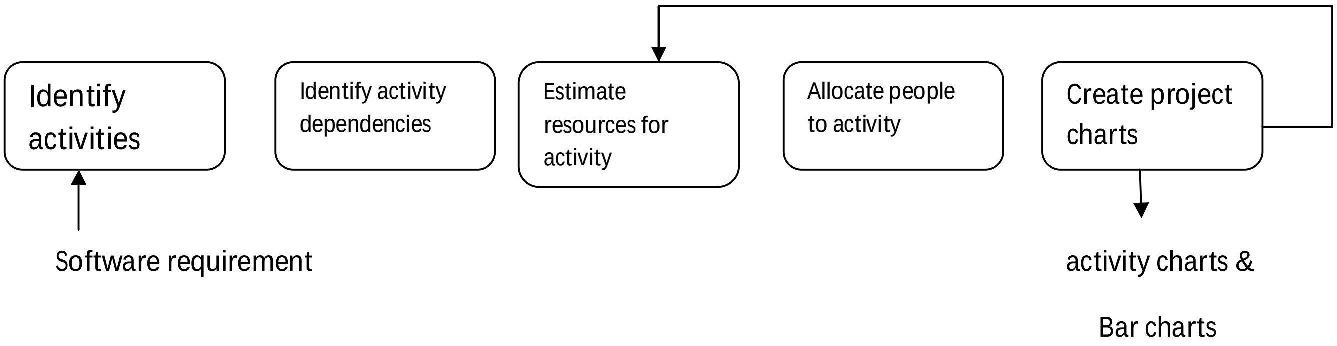


Fig: - milestone in the requirement process

Project scheduling

Project scheduling is the one of the most different job for a project a manager. Project manager estimate the time and resources require to complete activities and organize them in to coherent sequence. Schedule estimation is further complicated by the fact that different project may use different design methods and implementation language.

If the project is technically advanced, initial estimation will almost certainly optimism even when manager try to consider all eventualities. In this respect software scheduling is no different from scheduling any other type of large advanced project. Project scheduling involves separating the total work involved in a project in to a separate activities and judging the time require to complete these activities. Usually some of these activities are carried out in parallel. Manager has to co-ordinate these parallel activities and organizes the work, so that the work force is used optimally.


Identify activities
Identify activity dependencies
Estimate resources for activity
Allocate people to activity
Create project charts
Software requirement
activity charts &
Bar charts

It is suggested that where manager is estimating scheduling he should not assume that every stage of the project will be problem free. If the project is new and technically advanced, certain parts of it may turn out to be more difficult and take longer time than original assumption. As well as time the manager have to estimate resources to complete each task.

Risk Management

Risk management is one of the main jobs of project manager. It involves anticipating risks that might affect the project schedule or the quality of software being developed and taking appropriate action to avoid the risk.

The result of risk analysis should be documented in the project plan along with an analysis of consequence of a risk occurring. Effective risk management makes it easier to cope with problem and to ensure that these do not load to unacceptable budget or schedule slippage. A risk is some things that manager prefer not to have happen. There are three related categories of risk.

1. Project risk: Project risk is risk that affects the project schedule.
2. Product risk: Product risk are risk that affects quality or performance of the software being developed
3. Business risk: Business risk is a risk that affects the organizations developing or procuring the software.
4. These risk type over loop if an experienced programmer leaves a perfect, 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 and may make programming errors. Finally it can be a business risk because the programmer experiences are not available for building for future business.

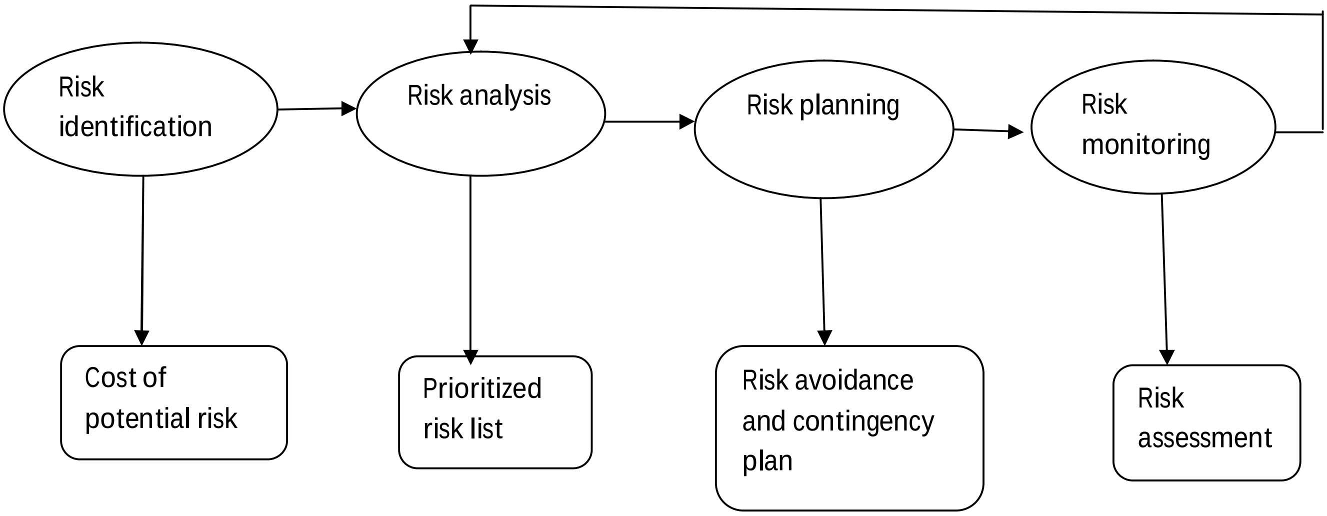
Possible software risk

Risk	Type	Description
Staff turn over	Project	Experienced staff will leave the project before it is finished
Management change	Project	There will be a change of organizational managements with different priorities
Hardware unavailability	Project	Hardware which is essential for the project will not be delivered in schedule
Requirement change	Project/product	There will be larger number of changes to the required then anticipated
Specification Daley	Project/product	Specification of essential interfaces are not available on schedule
Size underestimate	Project/product	The size of the system has been under estimated
Case tool under performance	Product	CASE tools which supports the project do not perform on

		anticipated
Technology change	Business	The underlying technology on which the system is built is super send by new technology
Product competition	Business	A competitive product is marketed before the system is completed

The risk that may effects project depends on the project on the organizational environments where the software being developed risk management is particularly importance for software project because of the inherent uncertainties that must project face. These stem from loosely define requirements difficulties in estimating time and resource require for software development, dependence on individual skill and requirements changes due to change in costumer need. Manager has to anticipate risk, understand the impact of this risk on the project, product and business and take steps to avoid this risk.

Risk management process



The risk that may affect the project depends on project and organizational environment where the software is being developed. Risk management is particularly important for software project because of the inherent uncertainties that must project face. These stem from loosely defined requirements, difficulties in estimating the time and resources required for the software development; dependency on individual skills and requirements changes due to changes in customers' needs. Manager has to anticipate risk, understand the impact of these risks on the project, product, and business and take necessary steps to avoid these risks.

1). Risk identification

Possible project, product, and business risks are identified.

2). Risk analysis

The like hood and consequence of these risks are assessed.

3). Risk planning

Plans to address the risk either by avoiding it or minimizing its effect on the project are drawn up

4) **Risk monitoring:**

Risk is constantly assessed and plans for risk avoidance.

The risk management process is an iterative process which continues throughout the project. Once an initial set of plans are drawn up, the situation is monitored. As more information about the risk becomes available, the risk has to be analyzed and new priorities are established. The risk avoidance and contingency plans may be modified as new risk information emerges. Manager should document the outcomes of the risk management process in a risk management plan. This should include a discussion of the risk faced by the project, analysis of these risks and plans that are required to manage these risks.

Risk identification

Risk identification is the first stage of risk management. It is concerned with discovering possible risk to the project. Risk identification can be carried out as a team process using a brain storming or may simply based on experience. To help the process, a checklist of different types of risk may be used. There are at least six types of risk that are:

i. Technology risk
Technology risks are risks that derive from the software or hardware technologies that are used to develop the software.

ii. Organizational Risk
Organizational risks are risk that derive from the organizational environment where the software being developed.

iii. People Risk
People risks are risks that are associated with peoples involved in software development.

iv. Tools Risk
Tools risks are risks that derive from the CASE tool and other supported software used to develop the software.

v. Requirement Risk
Requirement risks are risks derived from changes to the customer requirements and the process of managing requirement change.

vi. Estimation Risk

Estimation risks are risks that derive from the management estimates of the system characteristics and the resources required to develop the software.

Risk planning

The risk planning processes consider each of the key risk that have been identified and identify strategies to manage the risk. There is no simple process that can be followed to establish risk management plan. It relies on the judgement that has been identified for the risk management plans are as follows:

i. Avoidance Strategy

Avoidance strategy means that the probability that the risk will arise will be reduced.

ii. Minimization Strategy

This strategy means that the impact of the risk will be reduced.

iii. Contingency plan

Contingency plan means that you are prepared for the worst case and have a strategy to deal with it.

Risk Monitoring

Risk monitoring involves regularly accessing each of the identified risk to decide whether or not that risk is becoming more or less probable and weather the effort of the risk have changed. Risk monitoring should be continuous process and at every management process review, manager should consider and discuss each of the risk repeatedly.

UNIT 2.1 SOFTWARE REQUIREMENT

Introduction:-

The requirements for a system are the description of the services provided by the system and its operational constraints. These requirements reflect the needs of the customer for the system that helps to solve some problems. The process of finding out, analyzing and documenting and checking these services and constraints is called requirement engineering.

A requirement is simply high level abstract statement of a service that the system should provide. Software requirements are classified in to user requirements and system requirements. User requirements are statement of what services the system is expected to provide and the constraints under which it must operate. System requirement set out the systems function, services and operational constraints in detail. Different levels of system specifications are useful because they communicate information about the system to different type of readers.

Types of requirements:-

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

1. Functional requirements:-
   These are statements of services of the system should provide how the system should react to particular inputs and how the system should behave in particular situations. The functional requirement for a system should describe what the system do or perform. These requirements depend on the type of software being developed, the expected user of the s/w and the general approach taken by the organization when writing requirements. Functional requirement describe system function in detail it s input, output, and so on.
   The functional requirements discuss the functionality required from the system. The system is considered to perform a set of high level function. Each function of the system can be considered as a transformation of a set of input to the corresponding set of output. Functional requirement for a software system may be expressed in a number of ways.

The functional requirements define specific facilities to be provided by the system. These have been taken from the user requirement document. It is necessary for a software developer to interpret an ambiguous requirement to simplify its implementation. The functional requirement specification of a system should be complete and consistent.

Completeness means that all services required by the user should be defined. Consistency means that requirement should not have any contradiction.

2 Non functional requirements

Non-functional requirements are the requirements that are not directly concern with the specific function of the system. They may related to emergent system properties such as reliability, response time and storage occupancy. They may define constraint on a system such as the capabilities of input/output devices and data representation used in the system interface. Non-functional requirement specify the emergent property of the system. They may specify the performance, security, availability and other property. This means that they are often more critical than functional requirement. Non-functional requirements are not just concern with a software system to be developed. Some non-functional requirements may constraint the process that should be used to develop the system.

Types of non-functional requirements:-

I) Product requirement:-

Product requirement specify product behavior. For example; performance requirement on how fast the system must execute and how much memory it requires reliability requirements that sets out the acceptable failure rate, probability requirement and usability requirement.

II) Organizational requirement:-

Organizational requirements are delivered from policies, procedure and structure of the customer and developers organization.

III) External requirements:-

External requirements may include interoperability requirements that define how the system interacts with systems in other organization.

3. **Domain requirements:-**

   Domain requirements are derived from the application domain of the system rather than from the specific needs of the system requirements. Domain requirements are important because they often reflect fundamental concept of the application.

Software requirement documentation:-

Software requirement documentation is the official statement of what the 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 user ranging from the senior management of the organization to the engineers of responsible for developing the software. The diversity of the possible user 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 should include in a requirement document depends on the type of software that is being developed and the development process used. When the software will be developed by an external controller, critical system specification need to be precise and detail.

IEEE standard suggest the following structure for the requirement document.

1 Introduction:-

1.1 Purpose of the requirement document
1.2 Scope of the document
1.3 Definition, abbreviation
1.4 References
1.5 Overview of the reminder of document

2 General descriptions:-

2.1 Product perspective

2.2 Product functions

2.3 User characteristics

2.4 General constraints

2.5 Assumption and dependencies

Specific requirements cover functional, non-functional, domain and interface requirement.

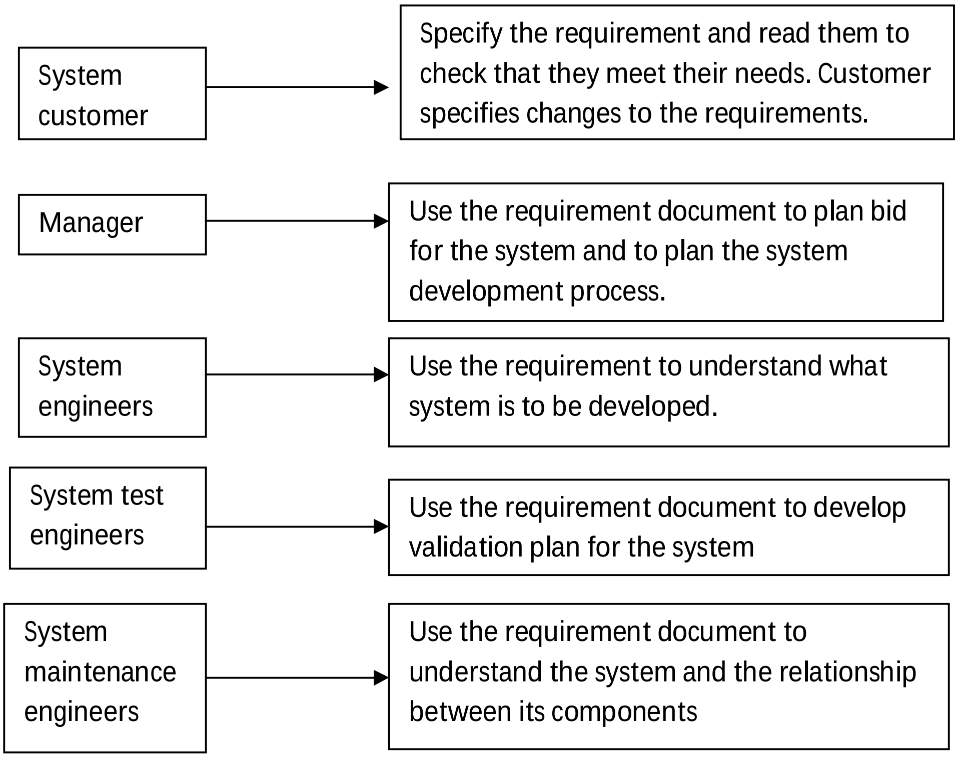
This is obviously the most critical part of the document. The requirement may document external interface, describe system functionality and performance, specify logical database requirement, design constraints, emergent system properties and quality. It is not appropriate to define a standard structure for this section.

4. Appendices

5. Index

Although the IEEE standard is not details. It contains great deal of good advices on how to write requirements and how to avoid problems.

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 the 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 in to some standard form (specialization), and checking that the requirement actually defines the system that the customer wants (validation).

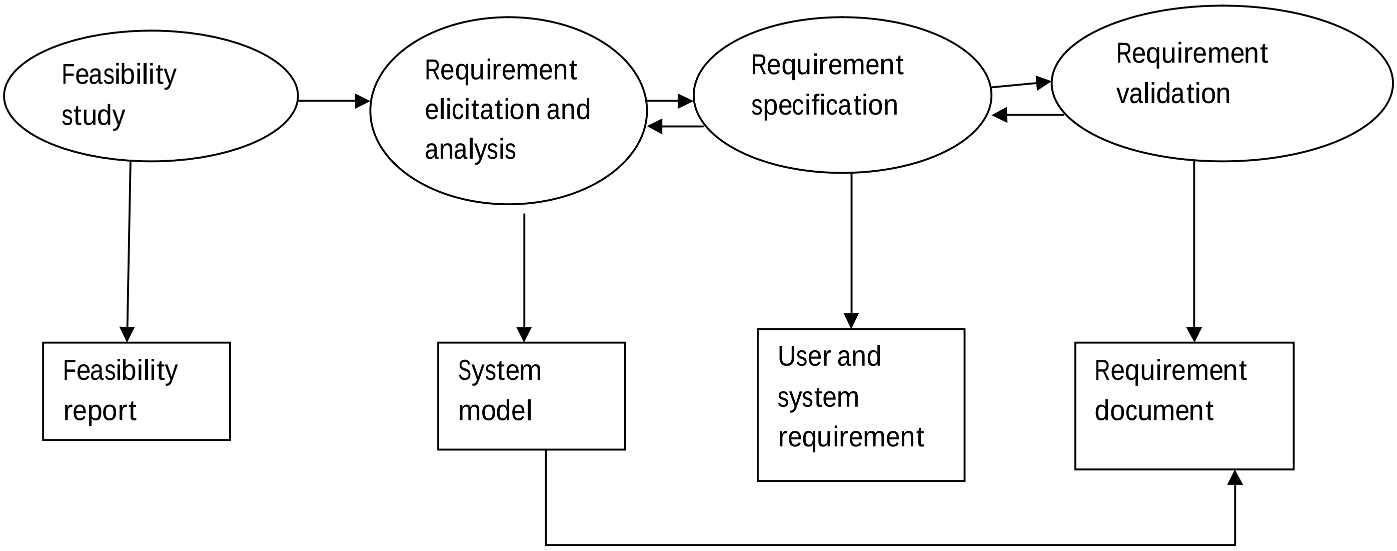


Fig: - the requirement engineering process.

These activities are concern with the discovery documentation and checking of requirements. The people involved in development gain better understanding of what they want to do the software, modification made to the software and organizational environment. The process of managing these changing requirements is called requirement management.

1. Feasibility Study:

The requirement engineering process should start with a feasibility study. The input to the feasibility study is a set of preliminary requirements of the system and how the system is intended to support business process. The result of the feasibility study should be a feasibility report that recommends whether or not it is carrying out system development process.

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 is integrated with other systems which are already in existence?

If the system does not support objectives of organization, it has no real values to the organization .After collecting and analyzing the information, the feasibility report should be written/w manager should make a recommendation about whether or not the system development should continue.

2. Requirement elicitation and analysis:

The next stage of requirement engineering process is requirement elicitation and analysis. In this activity, s/w engineering work with customer and system end user to find out information about application domain, what services the system should provide, the required performance of the system, hardware constraint and so on.

Requirement elicitation and analysis may have 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 individually. Stakeholder includes end users who interact with the system and everyone in an organization that may be affected by its installation.

A very general process model of elicitation and analysis process is as follow:

Each organization will have its own version of the requirement elicitation and analysis process, depending on local factor such as expertise of the staff, the type of system should be developed and the standards used in system development.

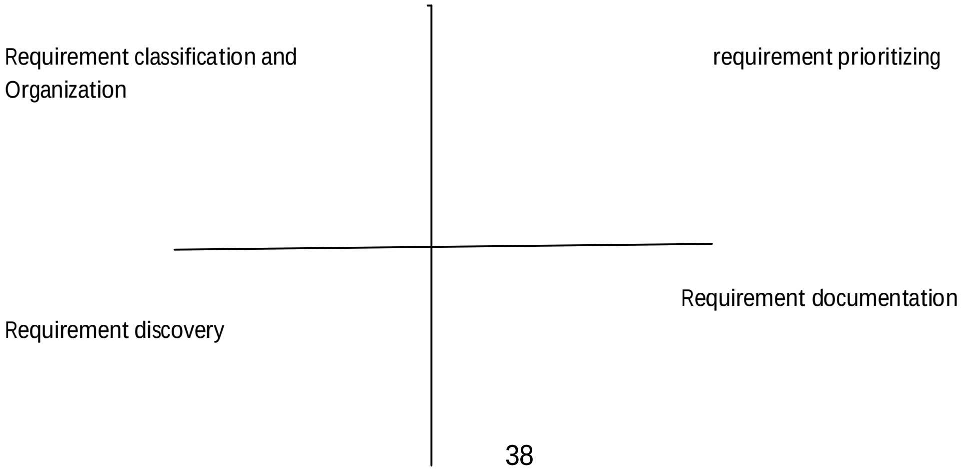
Activities in Requirement Elicitation and Analysis:

i. Requirement Discovery
This is the process of interacting with the stakeholder in the system to collect their requirements.

ii. Requirement Classification and Organization:
This activity takes the unstructured collection of requirements, group of related requirement and organizes them into a proper structured manner.

iii. Requirement Prioritizing:
Where multiple users are involved, requirement will be conflict .This activity is concerned with prioritizing requirement, finding and resolving requirement conflicting with discussing stakeholders.

iv. Requirement Documentation:
The requirement is documented in a proper sequence which is input to the next stage of requirement engineering process.


Requirement classification and Organization
requirement prioritizing
Requirement documentation
Requirement discovery

Requirement elicitation and analysis is an iterative process with continual feedback from each activity to other activities.

4. Requirement Specification

   Requirement specification is the process of generating requirement document that is useful for all types of system user. Requirement specification should be précised and complete. The requirement specification must specify merits and demerits along with recommendation of requirement.

5. Requirement validation:

   Requirement validation is concerned with showing that the requirement actually defines the system that the customer wants .Requirement validation overlaps analysis .it is concerned with finding the problems in the requirement.

Requirement validation is important because errors in a requirement document can lead to extensive rework and cost, when they are discovered during development or after the system is in service. During the requirement validation process; checks should be carried on the requirements in requirement 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

Requirement should not be conflict .there should no contradiction and differences between descriptions of same system function.

c. Completeness check:

   The requirement document should include requirements, which define all functions intended by the system user.

5. Realism check

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

6. Verifiability

To reduce the potential contradiction between customer and constrictor. The system requirement should verify. This enable both agree upon the system constraint

A number of requirement validation techniques can be used.

i. Requirement review

The requirements are analyzed systematically by a team of reviewers.

ii. Prototyping

In this approach an executable model of system is demonstrate to the end users and customers .they can experiment with this model to see if it meets their needs.

iii. Test case generation

Requirement should be testable .if the tests for the requirement are devised as a part of validation process, this often relief requirement.

6. Requirement Management

The requirement for large software system is always changing .one reason for this is that these systems are usually developed to provide services of different kind of users. During the software process, stakeholder understanding of the problem constantly changing .requirement management is the process of understanding and controlling changes to system requirement.

Software manager need to keep track of individual requirement and maintain link between dependent requirements so that manager can assess the input of requirement changes .software manager need to establish a formal process for making change proposal and linking these to the system requirement.

2.2 Software Prototyping:

Introduction:

The goal of prototyping based development process is to counter the two 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 help 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.

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 propose design.

Benefit of Using Prototype

1. Improved system usability

2. A closer match of system to users need.

3. Improved design quality

4. Reduce development effort.

2.1.1 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 changing to predict how a system will affect working ,how it will interact with other system and what users operation should be automated.

S/w development processes that are based on completely specifying requirements then designing, building and testing the system .to rapid s/w development, as the requirement changes or as requirement problems are discovered the system design and implementation has to be reworked and related.

Rapid s/w development processes are designed to produce useful s/w quickly .it is iterative process where specification, design, development and testing are interleaved. There are many approaches to rapid s/w 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 by the programming environment used to implement the system. The user requirement document defines only the most important characteristic of the system.

2. The system is developed in a series of increments .each users and other system stakeholders are involved in specifying and evaluating each increments .they may propose changes to the s/w 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.

2.1.2 User Interface Prototyping:

User interface design is an interactive process where users interact with designers and interface prototype to be decided on the features, organization and look and feel of the system user interface. Sometimes the interface is separately prototyped in parallel with other s/w engineering activities.

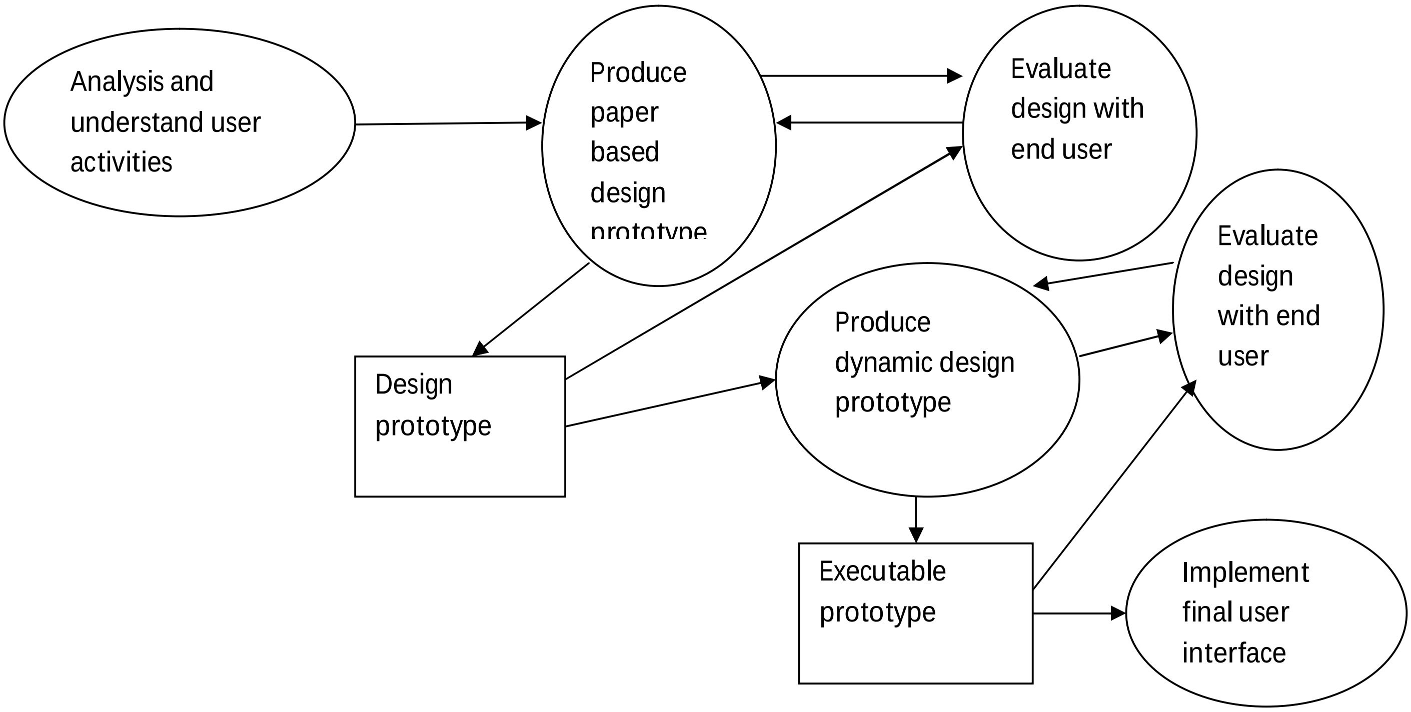

Analysis and understand user activities
Produce paper based design prototype
Evaluate design with end user
Design prototype
Produce dynamic design prototype
Evaluate design with end user
Executable prototype
Implement final user interface

Figure: 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:

2.3.1 Introduction:

Formal methods of s/w development are not widely used in industrial s/w 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 s/w 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 s/w 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 s/w 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 ins/w quality. People who suggested that the only way to improve s/w quality was by using or formal methods were clearly wrong.

2. Market changes:

In the 1980's s/w quality was seen as the key s/w engineering problem .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

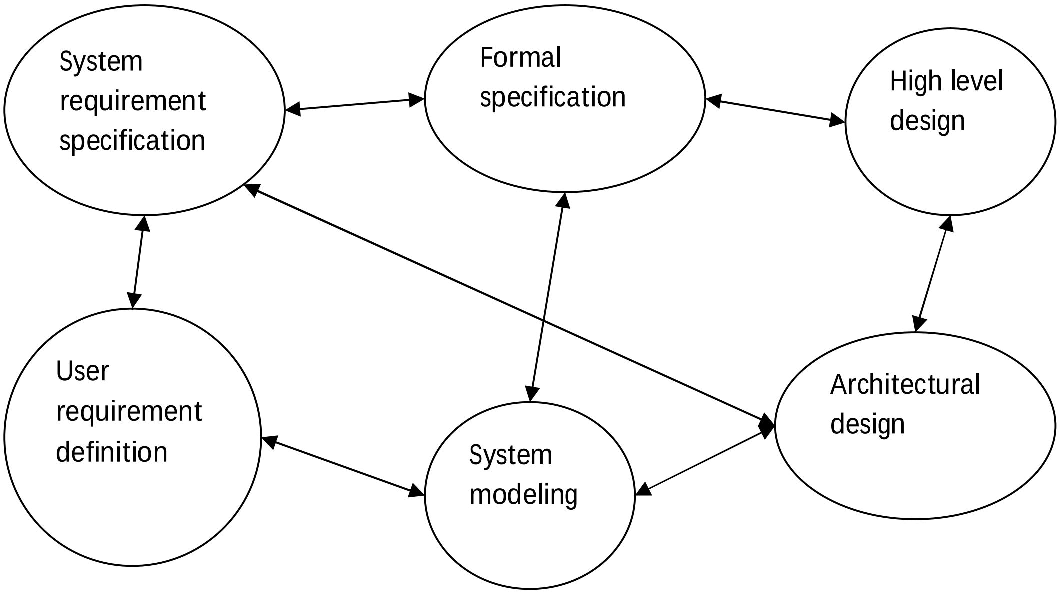
Formal methods are not well suited to specifying user interface and user interaction .The user interface component has become a greater and greater part of most system. So we can only really use formal methods when parts of the system

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.

2.3.2 Formal Specification in Software Process



Fig; formal specification in the s/w process

The system requirement and system designs are expressed in details and carefully analyzed and checked before implementation begins. A formal specification of s/w 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.

2.3.3 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 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.

2.3.4 Behavior 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 behavior properties of system .the specification which is used to specify such type of system property is called behavioral specification. As their size of system is increased, the description of system behavior 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.

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.

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.

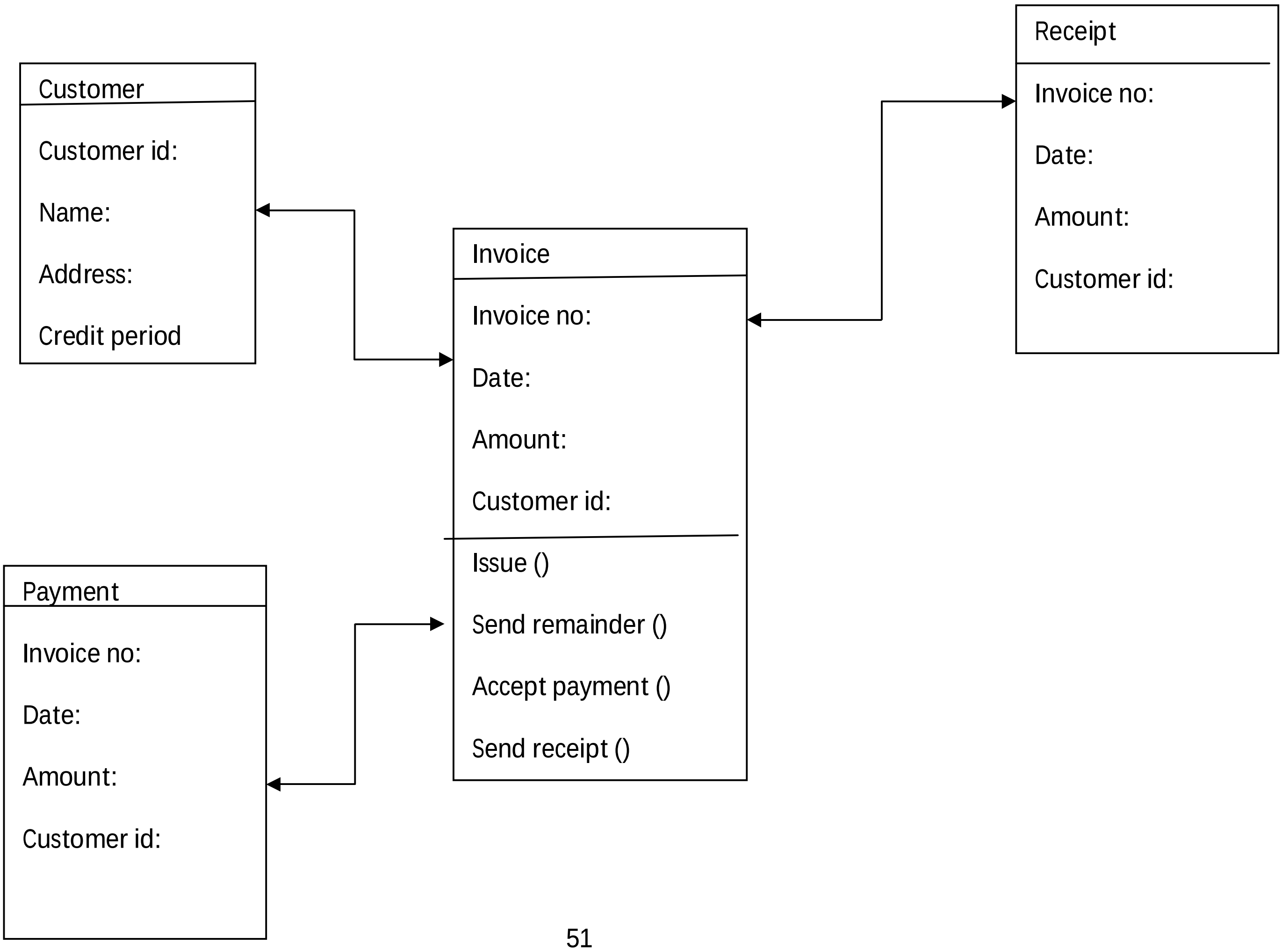


Figure: an object oriented model of an invoice processing

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.

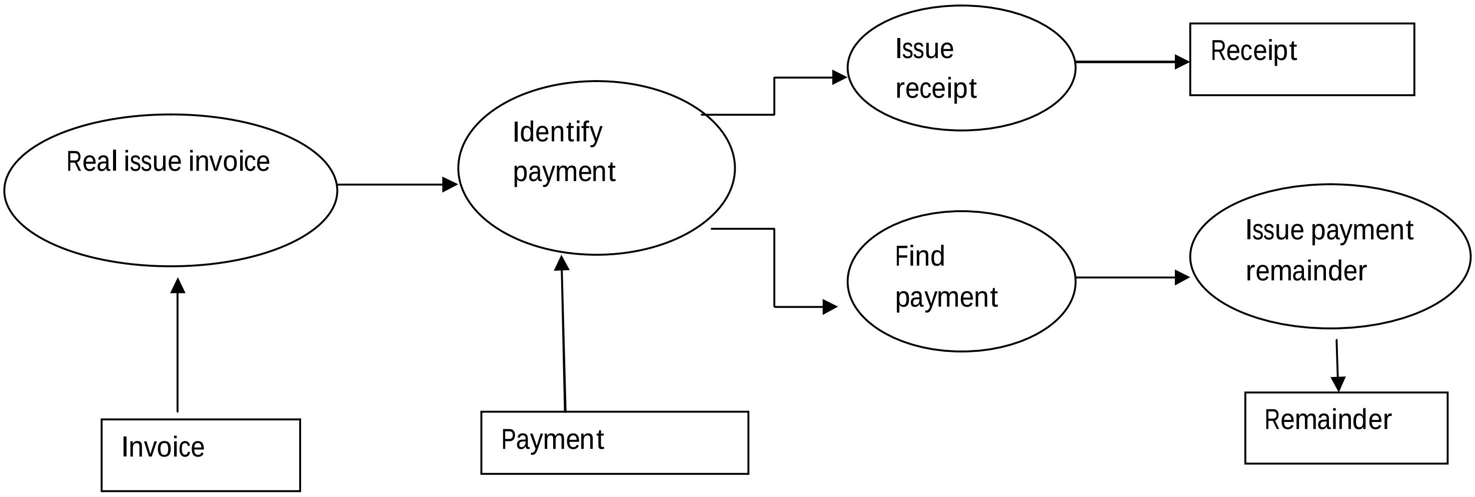


Figure: functional model of invoice processing system

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.

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.

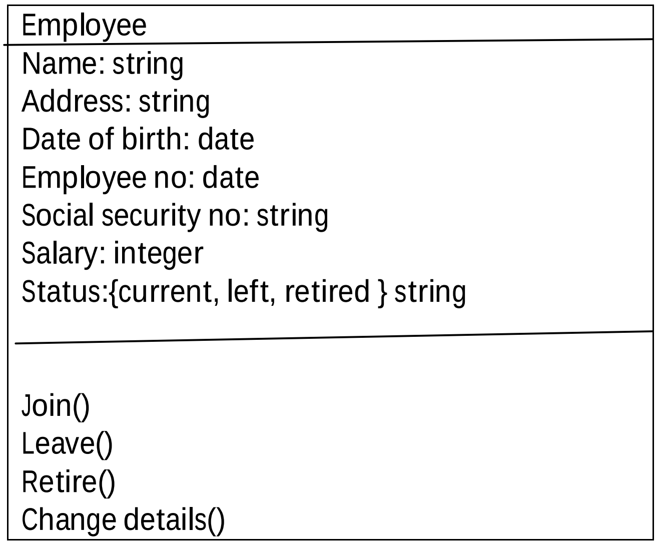
Object oriented system are easier to change than system developed using changing the implementation of an object or adding new services should not affect other system objects. Objects are reusable components because they are independent.

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.

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:

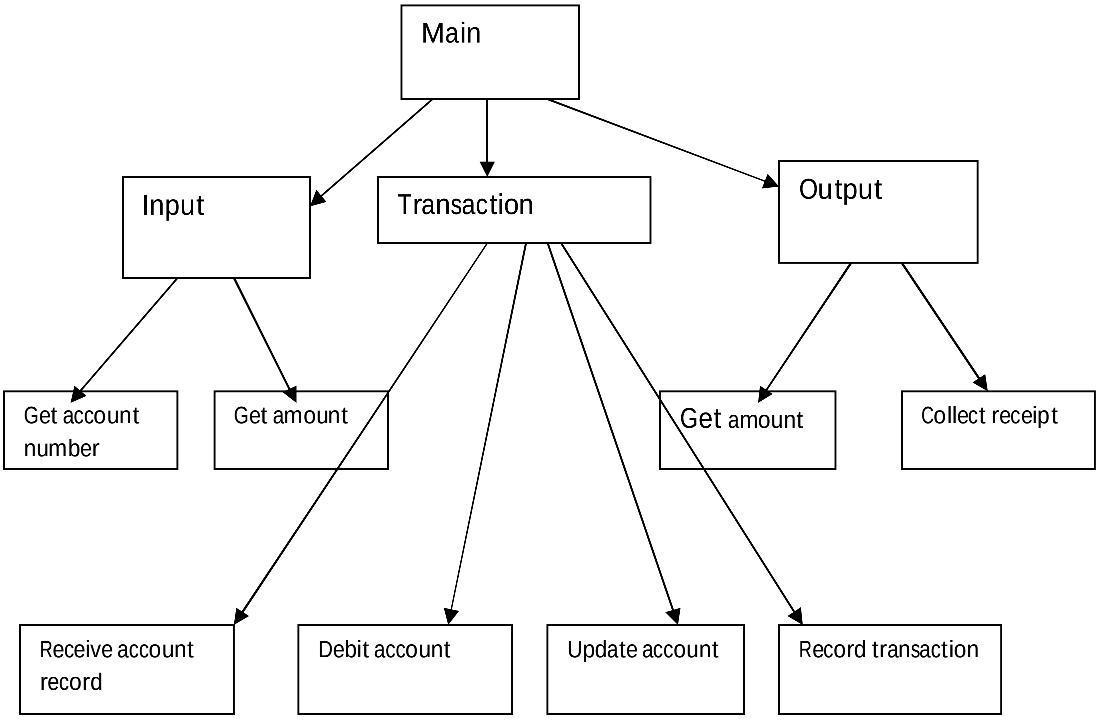


Fig: Call-return model of ATM

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:

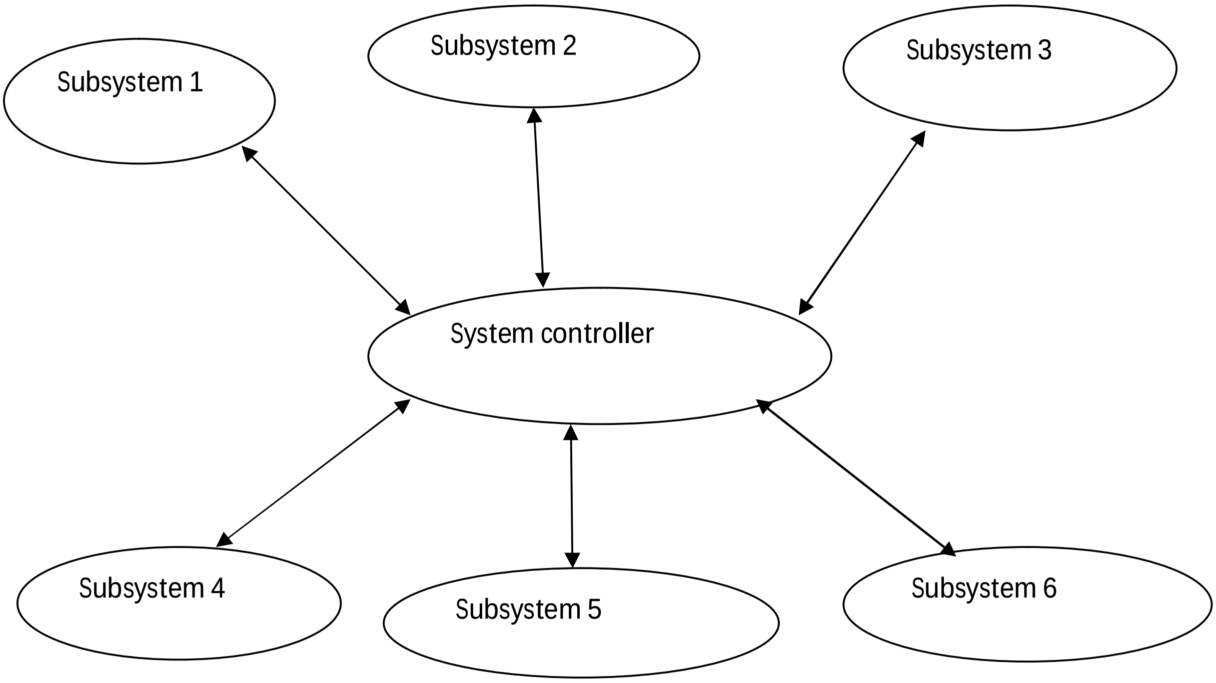


Fig: The manager model

Event -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.

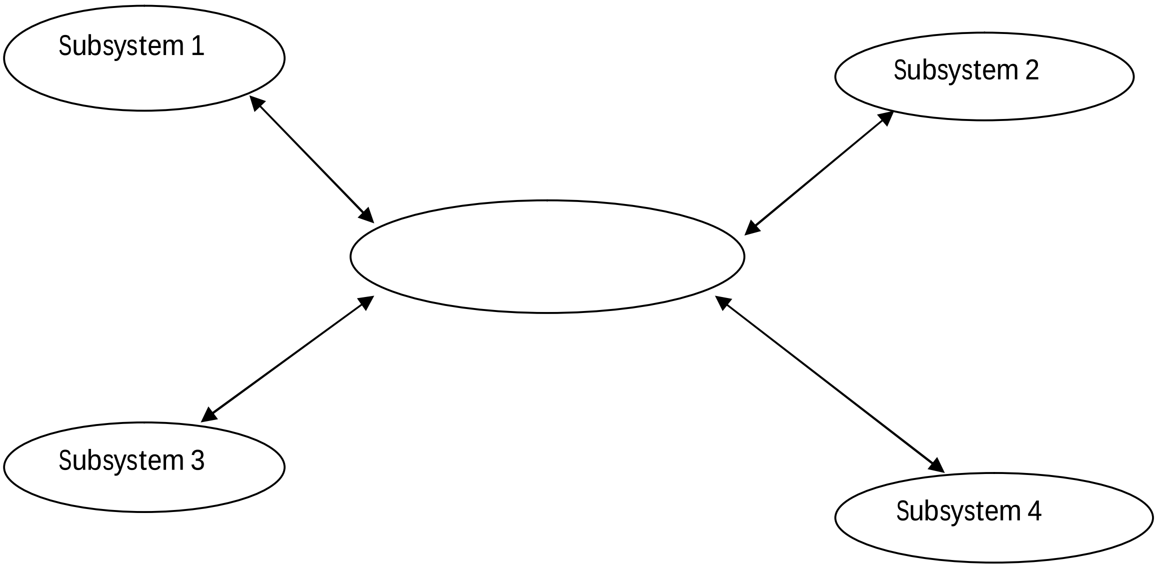


Fig: A broadcast Control Model

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:

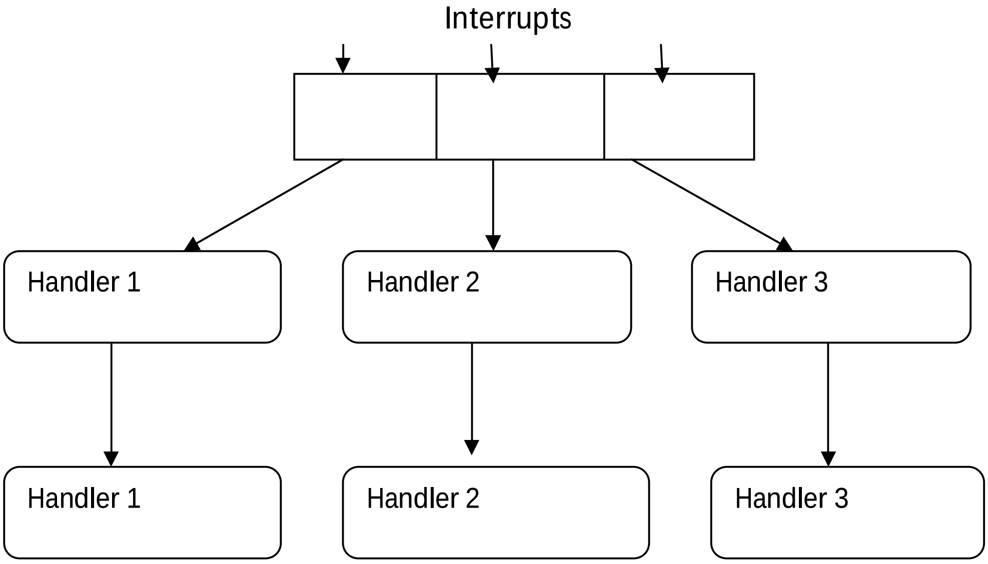


Fig: An interrupt-Driven control Model

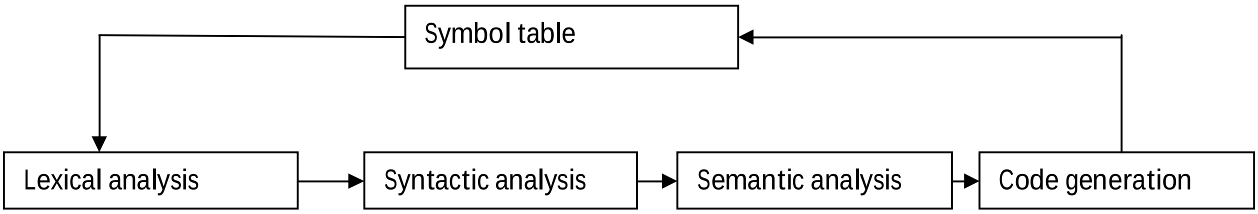


Fig: A Data-flow model of compiler (example of generic model)

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.

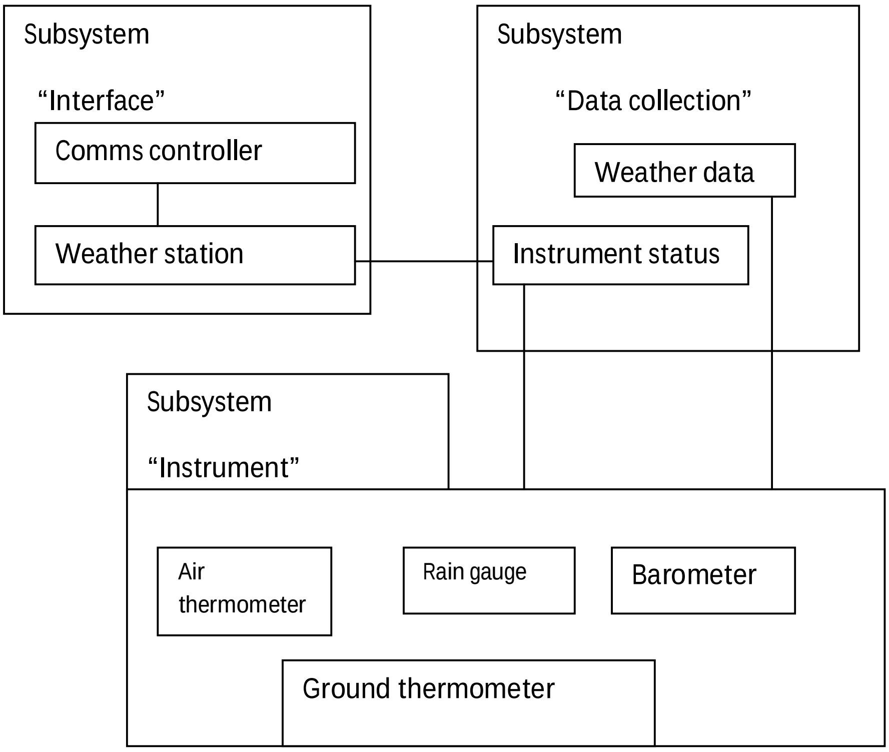
Design model

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:

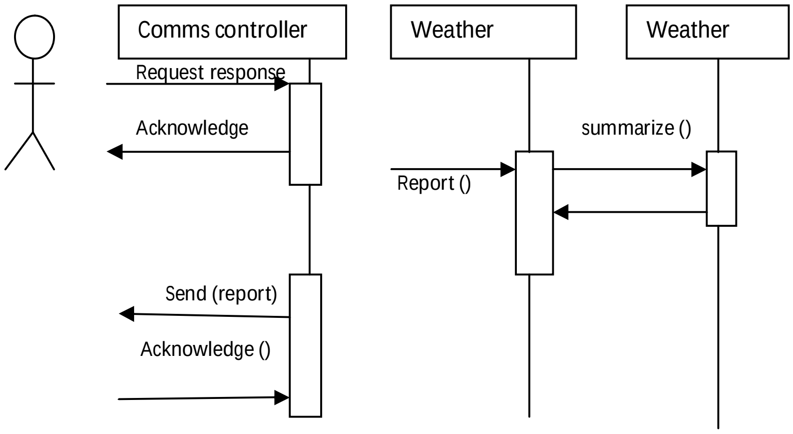


ii. Sequence Model

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.

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)

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.

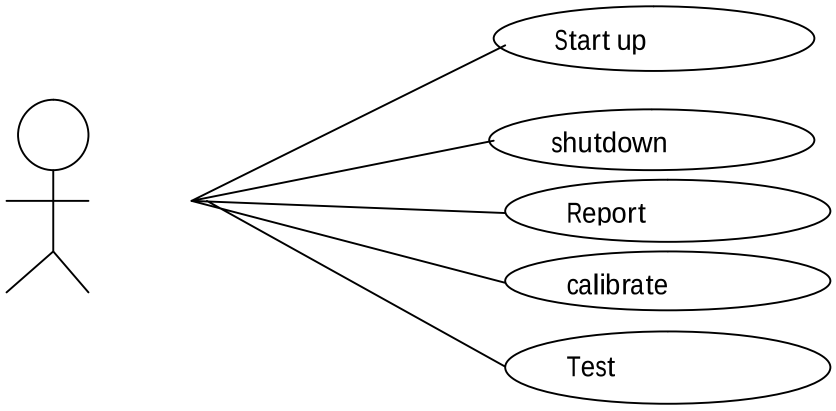


Fig: Use-Case for weather station

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.

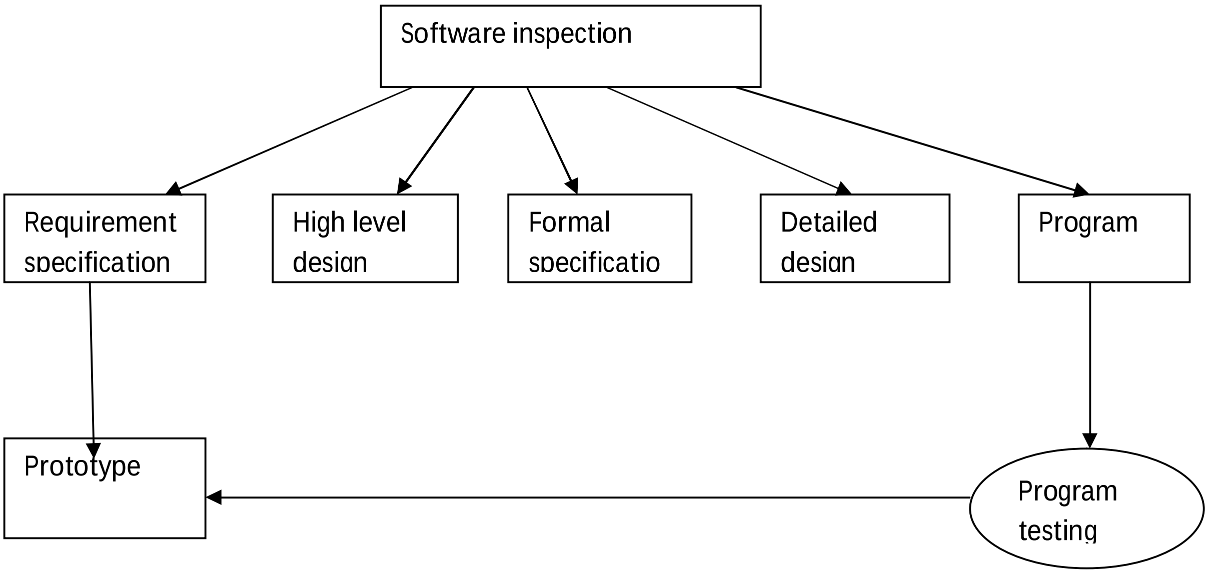


Fig: verification and validation

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

a. Software Inspection

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.

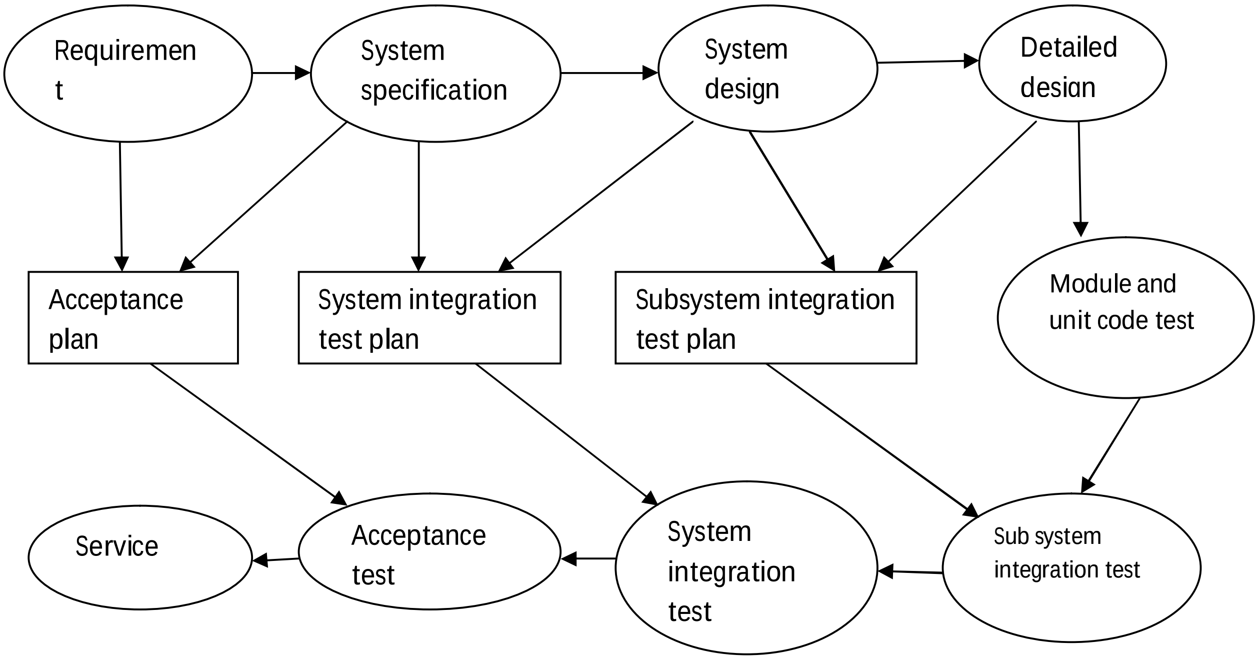


Fig: test plan as a link between development and testing

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.

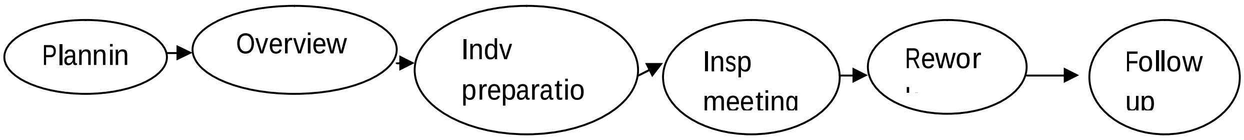


Fig: the inspection process

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

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.

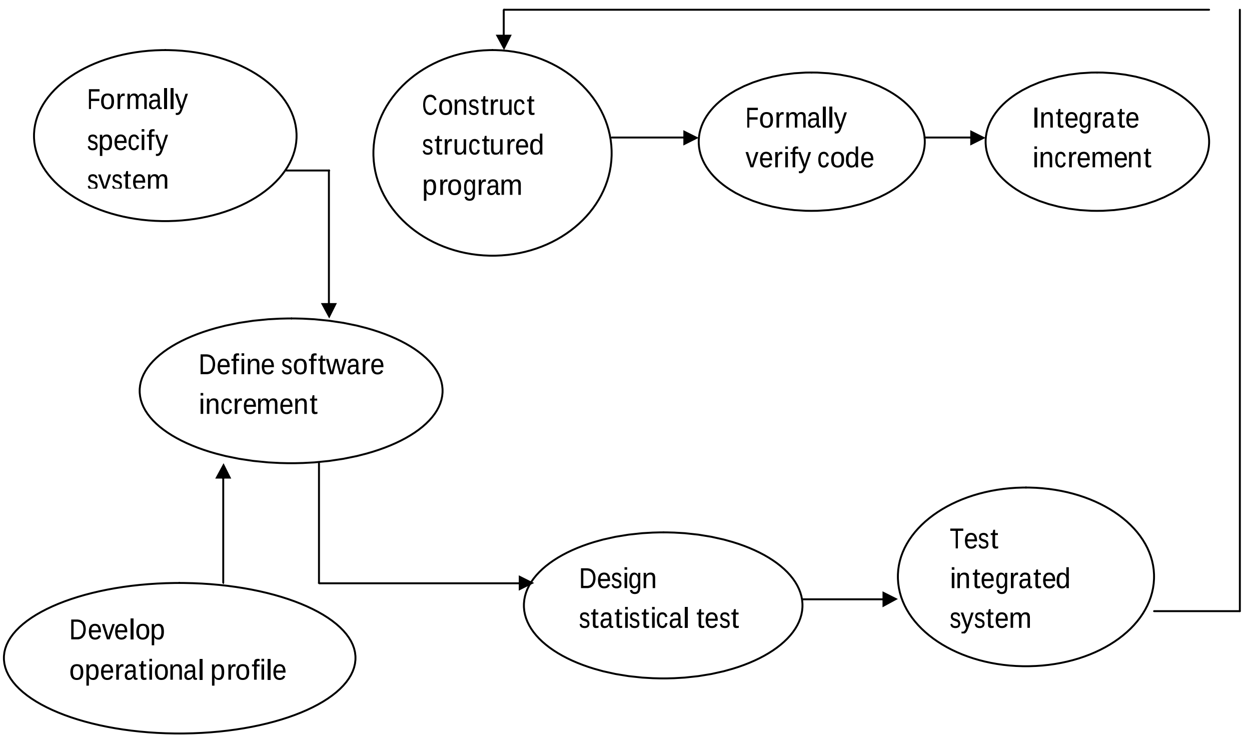


Fig: clean room development process

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.

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.

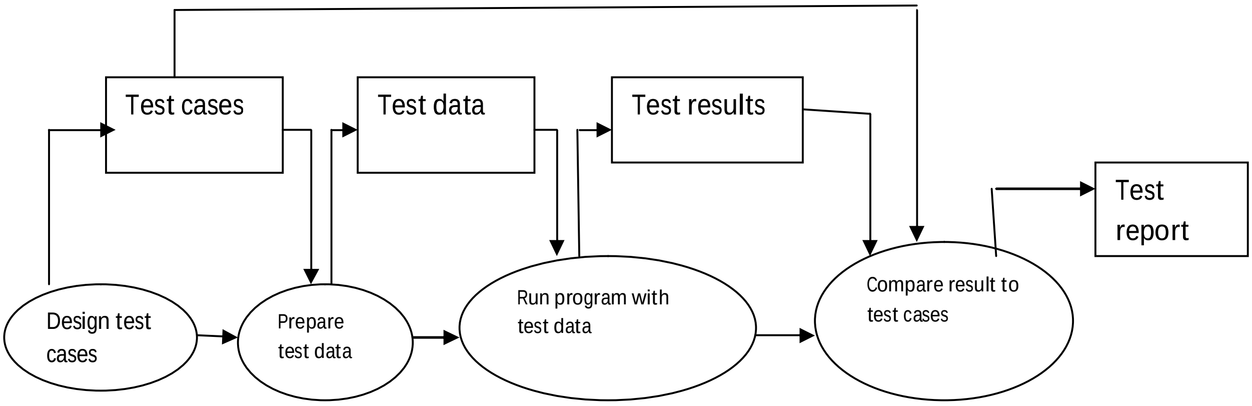


Fig: A model of the software testing process

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.

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

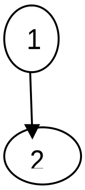
The path coverage based testing strategy requires designing test cases such that all linearly independent paths in the program are executed at least once. A linearly independent path can be defined in term of control flow graph of a program

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.

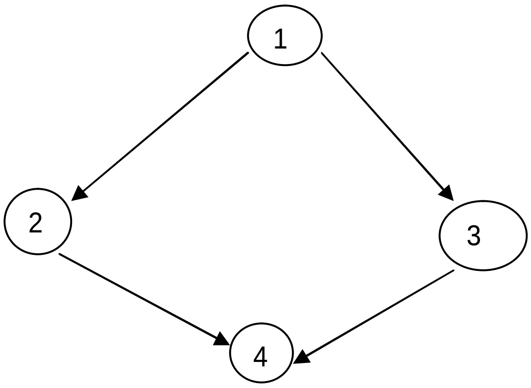
The control flow graph for any program can be easily drawn by knowing how to represent the sequence, selection and iteration type of statement in control flow graph.

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



iii. Else c=5

iv. c= c*c

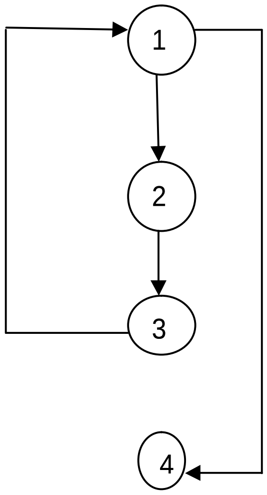


ii. b=b-1

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

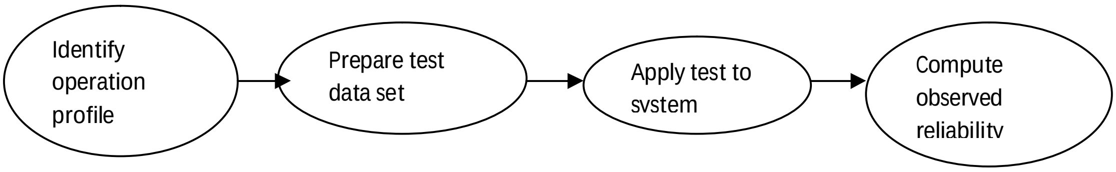
Introduction:

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

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

- 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

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:

$\text{Effort} = a_1*(\text{KLOC})^{a}{}_{2}*\text{PM}$

$\text{Tdev} = b_1*(\text{effort})^{b}{}_{2}*\text{months}$

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:

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.

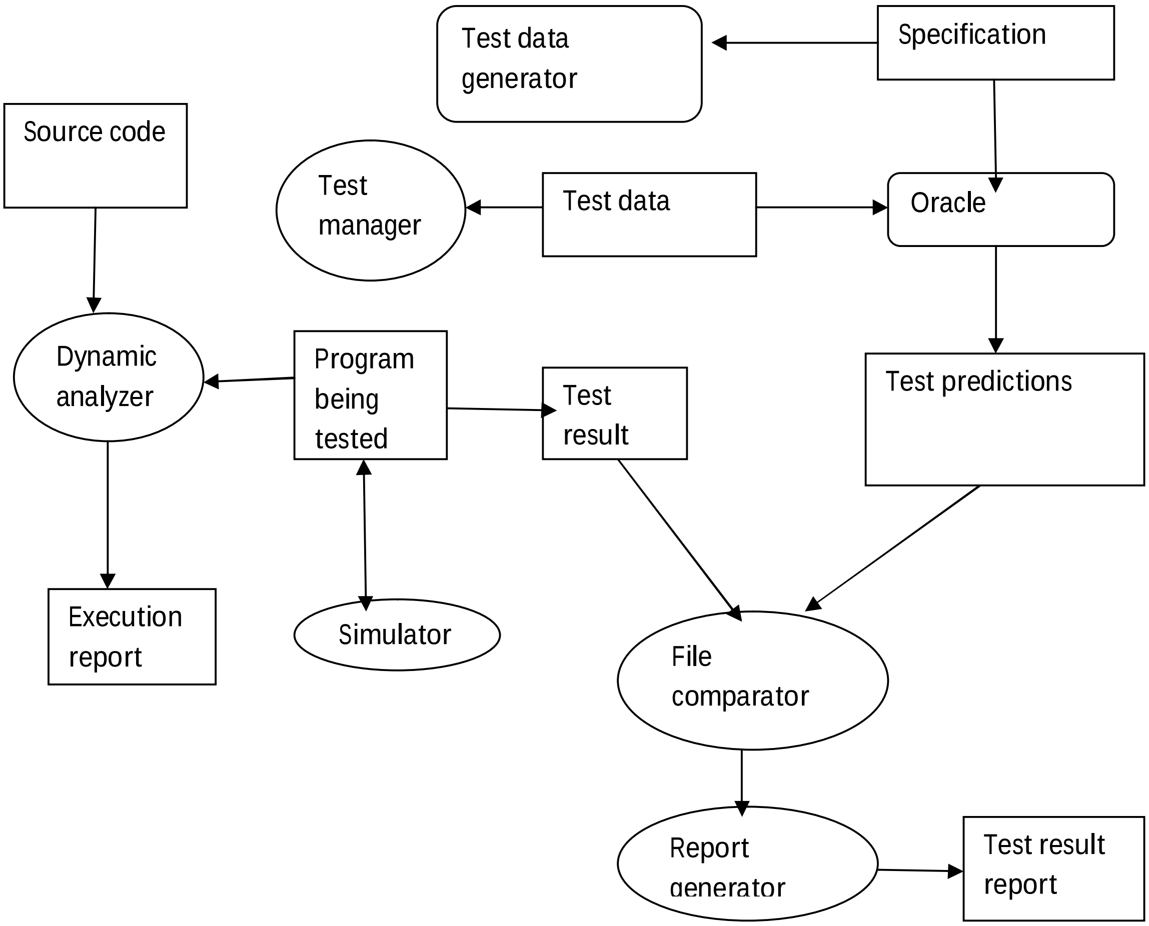


Fig: a Testing workbench

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.

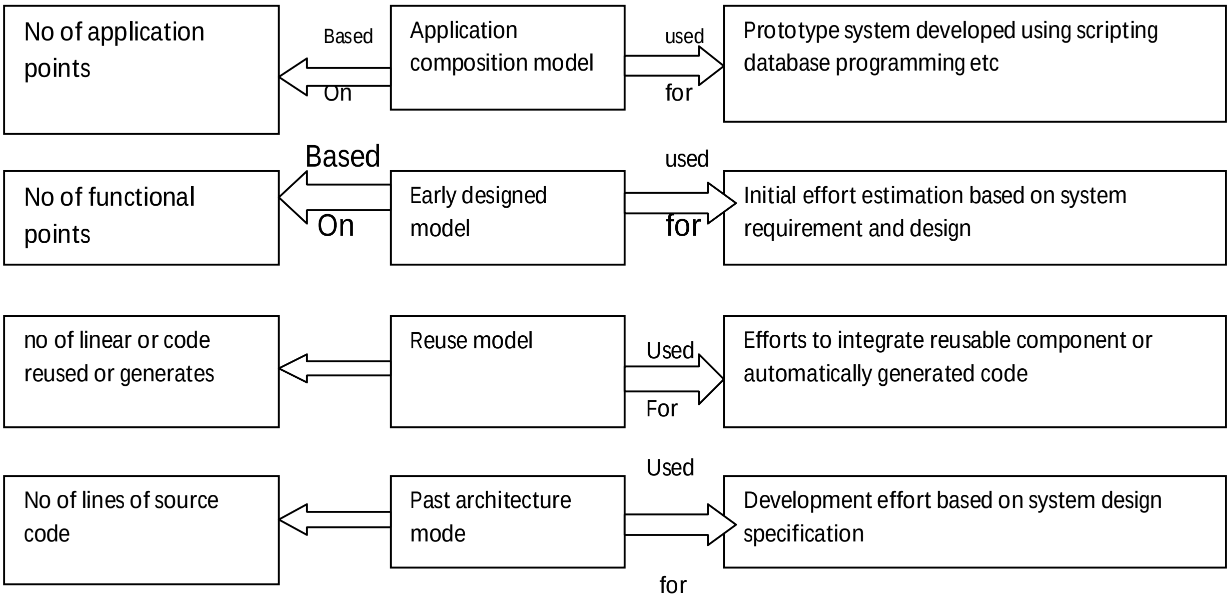
A PERT chart presents a graphic illustration of a project as a network diagram consisting of numbered node (either circles or rectangles) representing events, or milestones in the project linked by labeled vectors representing tasks in the project.

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:


No of application points
Based
On
Application composition model
used
for
Prototype system developed using scripting database programming etc
Based
On
No of functional points
Early designed model
used
for
Initial effort estimation based on system requirement and design
no of linear or code reused or generates
Reuse model
Used
For
Efforts to integrate reusable component or automatically generated code
Used
No of lines of source code
Past architecture mode
Development effort based on system design specification
for

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.