Subject: Software Engineering and Project Management

Semester: VI

Unit 1

Topic: Introduction to Software Engineering, Characteristics

1) Explain various software characteristics in details. [6 M] Summer -15

OR

2) Explain the concept of software Engineering, also explain its characteristic in brief. [7M] S17

OR

3) Define software engineering. Explain software characteristics in detail. [6M] W-18 S-16

Fig Software Engineering

Definition: Software is (1) instructions (computer programs) that when executed provide desired function and performance,(2) data structures that enable the programs to adequately manipulate information, and (3) documents that describe the operation and use of the programs.

Software engineering is the process of analyzing user needs and designing, constructing, and testing end user applications that will satisfy these needs through the use of software programming languages. It is the application of engineering principles to software development. In contrast to simple programming, software engineering is used for larger and more complex software systems, which are used as critical systems for businesses and organizations.

Software Characteristics:

To gain an understanding of software (and ultimately an understanding of software engineering), it is important to examine the characteristics of software that make it different from other things that human beings build. When hardware is built, the human creative process (analysis, design, construction, testing) is ultimately translated into a physical form. If we build a new computer, our initial sketches, formal design drawings, and bread boarded prototype evolve into a physical product (chips, circuit boards, power supplies, etc.). Software is a logical rather than a physical

system element. Therefore, software has characteristics that are considerably different than those of hardware:

1. Software is developed or engineered; it is not manufactured in the classical sense.
Although some similarities exist between software development and hardware manufacture,

the two activities are fundamentally different high quality is achieved through good design, but the manufacturing phase for hardware can introduce quality problems that are nonexistent (or easily corrected) for software.

Both activities are dependent on people, but the relationship between people applied and work accomplished is entirely different .Both activities require the construction of a "product" but the approaches are different.

Software costs are concentrated in engineering. This means that software projects cannot be managed as if they were manufacturing projects.

2. Software doesn't "wear out."

Figure 1.1 depicts failure rate as a function of time for hardware. The relationship, often called the "bathtub curve," indicates that hardware exhibits relatively high failure rates early in its life (these failures are often attributable to design or manufacturing defects); defects are corrected and the failure rate drops to a steady-state level (ideally, quite low) for some period of time.

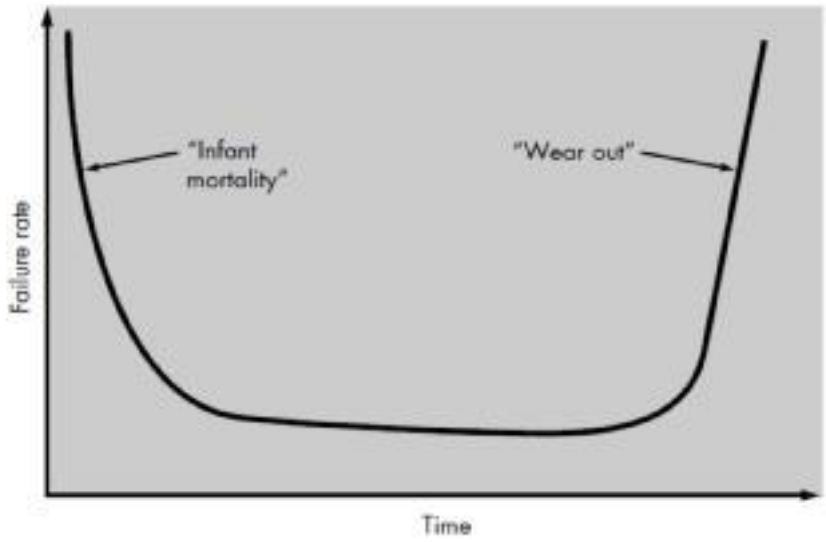


Fig.1 Failure curve for hardware

As time passes, however, the failure rate rises again as hardware components suffer from the cumulative effects of dust, vibration, abuse, temperature extremes, and many other environmental maladies. Stated simply, the hardware begins to wear out.

Software is not susceptible to the environmental maladies that cause hardware to wear out. In theory, therefore, the failure rate curve for software should take the form of the “idealized curve” shown in Fig.2 Undiscovered defects will cause high failure rates early in the life of a program. However, these are corrected (ideally, without introducing other errors) and the curve flattens as

shown. The idealized curve is a gross oversimplification of actual failure models for software. However, the implication is clear—software doesn't wear out. But it does deteriorate!

uring its life, software will undergo change (maintenance).Changes are made, it is likely that some new defects will be introduced, causing the failure rate curve to spike as shown in Fig.2 Before the curve can return to the original steady-state failure rate, another change is requested, causing the curve to spike again. Slowly, the minimum failure rate level begins to rise—the software is deteriorating due to change.

Another aspect of wear illustrates the difference between hardware and software.

When a hardware component wears out, it is replaced by a spare part. There are no software spare parts. Every software failure indicates an error in design or in the process through which design was translated into machine executable code. Therefore, software maintenance involves considerably more complexity than hardware maintenance.

3. Although the industry is moving toward component-based assembly, most software continues to be custom built.

Consider the manner in which the control hardware for a computer-based product is designed and built. The design engineer draws a simple schematic of the digital circuitry, does some fundamental analysis to assure that proper function will be achieved, and then goes to the shelf where catalogs of digital components exist. Each integrated circuit (called an *IC* or a *chip*) has a part number, a defined and validated function, a well-defined interface, and a standard set of integration guidelines. After each component is selected, it can be ordered off the shelf. A view of reuse to encompass not only algorithms but also data structure. Modern reusable components encapsulate both data and the processing applied to the data, enabling the software engineer to create new applications from reusable parts. For example, today's graphical user interfaces are built using reusable components that enable the creation of graphics windows, pull-down menus, and a wide variety of interaction mechanisms. The data structure and processing detail required to build the interface are contained with a library of reusable components for interface construction.

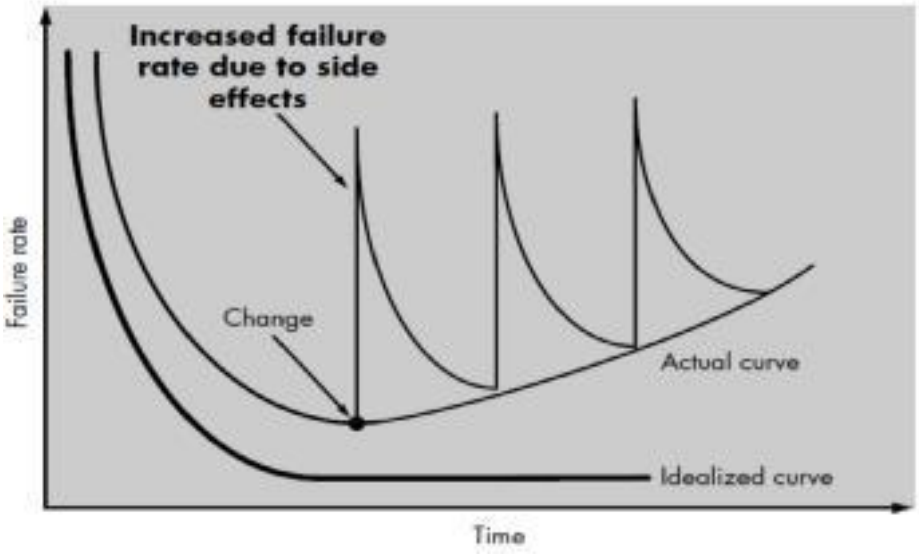


Fig. 2 Idealized and actual failure curves for software

1) Explain generic view of software Engineering in detail. [5M] W-16

Engineering is the analysis, design, construction, verification, and management of technical (or social) entities. Regardless of the entity to be engineered, the following questions must be asked and answered:

• What is the problem to be solved?

• What characteristics of the entity are used to solve the problem?

• How will the entity be constructed?

• What approach will be used to uncover errors that were made in the design and construction of the entity?

• How will the entity be supported over the long term, when corrections, adaptations, and enhancements are requested by users of the entity.

Throughout this book, we focus on a single entity—computer software. To engineer software adequately, a software engineering process must be defined. In this section, the generic characteristics of the software process are considered. Later in this chapter, specific process models are addressed. The work associated with software engineering can be categorized into three generic phases, regardless of application area, project size, or complexity. Each phase addresses one or more of the questions noted previously. The definition phase focuses on what. That is, during definition, the software engineer attempts to identify what information is to be processed, what function and performance are desired, what system behavior can be expected, what interfaces are to be established, what design constraints exist, and what validation criteria are required to define a successful system. The key requirements of the system and the software are identified. Although the methods applied during the definition phase will vary depending on the software engineering paradigm (or combination of paradigms) that is applied, three major tasks will occur in some form: system or information engineering software project planning, and requirements analysis . The development phase focuses on how. That is, during development a software engineer attempts to define how data are to be structured, how function is to be implemented within a software architecture, how procedural details are to be implemented, how interfaces are to be characterized, how the design will be translated into a programming language (or nonprocedural language), and how testing will be performed. The methods applied during the development phase will vary, but three specific technical tasks should always occur: software design, code generation, and software testing .The support phase focuses on change associated with error correction, adaptations required as the software's environment evolves, and changes due to enhancements brought about by changing customer requirements. The support phase reapplies the steps of the definition and development phases but does so in the context of existing software. Four types of change are encountered during the support phase:

Correction. Even with the best quality assurance activities, it is likely that the customer will uncover defects in the software. Corrective maintenance changes the software to correct defects.
Adaptation. Over time, the original environment (e.g., CPU, operating system, business rules, external product characteristics) for which the software was developed is likely to change. Adaptive maintenance results in modification to the software to accommodate changes to its external environment.

Enhancement. As software is used, the customer/user will recognize additional functions that will provide benefit. Perfective maintenance extends the software beyond its original functional requirements.

Prevention. Computer software deteriorates due to change, and because of this, preventive maintenance, often called software reengineering, must be conducted to enable the software to serve the needs of its end users. In essence, preventive maintenance makes changes to computer programs so that they can be more easily corrected, adapted, and enhanced.

The phases and related steps described in our generic view of software engineering are complemented by a number of umbrella activities.

Typical activities in this category include:

- Software project tracking and control
- Formal technical reviews
- Software quality assurance
- Software configuration management
- Document preparation and production
- Reusability management
- Measurement
- Risk management

Software Myths

1) State and explain different types of software Myths. [7M] W-15

OR

2) What are the practioner myths? Explain.[4M] W-18 , W-16

Management myths.

Managers with software responsibility, like managers in most disciplines, are often under pressure to maintain budgets, keep schedules from slipping, and improve quality. Like a drowning person who grasps at a straw, a software manager often grasps at belief in a software myth, if that belief will lessen the pressure (even temporarily).

Myth: We already have a book that's full of standards and procedures for building software, won't that provide my people with everything they need to know?

Reality: The book of standards may very well exist, but is it used? Are software practitioners aware of its existence? Does it reflect modern software engineering practice? Is it complete? Is it streamlined to improve time to delivery while still maintaining a focus on quality? In many cases, the answer to all of these questions is "no."

Myth: My people have state-of-the-art software development tools, after all, we buy them the newest computers.

Reality: It takes much more than the latest model mainframe, workstation, or PC to do high-quality software development. Computer-aided software engineering (CASE) tools are more important than hardware for achieving good quality and productivity, yet the majority of software developers still do not use them effectively.

Myth: If we get behind schedule, we can add more programmers and catch up (sometimes called the Mongolian horde concept).

Reality: Software development is not a mechanistic process like manufacturing. In the words of Brooks [BRO75]: "adding people to a late software project makes it later." At first, this statement may seem counterintuitive. However, as new people are added, people who were working must spend time educating the newcomers, thereby reducing the amount of time spent on productive development effort. People can be added but only in a planned and well-coordinated manner. **Myth:** If I decide to outsource3 the software project to a third party, I can just relax and let that firm build it.

Reality: If an organization does not understand how to manage and control software projects internally, it will invariably struggle when it outsources software projects.

Customer myths. A customer who requests computer software may be a person at the next desk, a technical group down the hall, the marketing/sales department, or an outside company that has requested software under contract. In many cases, the customer believes myths about software because software managers and practitioners do little to correct misinformation. Myths lead to false expectations (by the customer) and ultimately, dissatisfaction with the developer.

Myth: A general statement of objectives is sufficient to begin writing programs— we can fill in the details later.

Reality: A poor up-front definition is the major cause of failed software efforts. A formal and detailed description of the information domain, function, behavior, performance, interfaces, design constraints, and validation criteria is essential. These characteristics can be determined only after thorough communication between customer and developer.

Myth: Project requirements continually change, but change can be easily accommodated because software is flexible.

Reality: It is true that software requirements change, but the impact of change varies with the time at which it is introduced. Figure 1.3 illustrates the impact of change. If serious attention is given to up-front definition, early requests for change can be accommodated easily. The customer can review requirements and recommend modifications with relatively little impact on cost. When changes are requested during software design, the cost impact grows rapidly. Resources have been committed and a design framework has been established. Change can cause upheaval that requires additional resources and major design modification, that is, additional cost. Changes in function, performance, interface, or other characteristics during implementation (code and test) have a severe impact on cost. Change, when requested after software is in production, can be over an order of magnitude more expensive than the same change requested earlier.

Practitioner's myths. Myths that are still believed by software practitioners have been fostered by 50 years of programming culture. During the early days of software, programming was viewed as an art form. Old ways and attitudes die hard.

Myth: Once we write the program and get it to work, our job is done.

Reality: Someone once said that "the sooner you begin 'writing code', the longer it'll take you to get done." Industry data ([LIE80], [JON91], [PUT97]) indicate that between 60 and 80 percent of all effort expended on software will be expended after it is delivered to the customer for the first time.

Myth: Until I get the program "running" I have no way of assessing its quality.

Reality: One of the most effective software quality assurance mechanisms can be applied from the inception of a project—the formal technical review. Software reviews (described in Chapter 8) are

a "quality filter" that have been found to be more effective than testing for finding certain classes of software defects.

Myth: The only deliverable work product for a successful project is the working program. **Reality:** A working program is only one part of a software configuration that includes many elements. Documentation provides a foundation for successful engineering and, more important, guidance for software support.

Myth: Software engineering will make us create voluminous and unnecessary documentation and will invariably slow us down.

Reality: Software engineering is not about creating documents. It is about creating quality. Better quality leads to reduced rework. And reduced rework results in faster delivery times. Many software professionals recognize the fallacy of the myths just described. Regrettably, habitual attitudes and methods foster poor management and technical practices, even when reality dictates a better approach. Recognition of software realities is the first step toward formulation of practical solutions for software engineering.

A Layered Technology

1) Explain software engineering- a layered Technology [7 M] W-17, S-16 S-17

or

2) "software engineering- a layered Technology", comment.[6M] W-15

Software development is totally a layered technology. That means, to develop software one will have to go from one layer to another. The layers are related and each layer demands the fulfillment of the previous layer. Figure below is the upward flowchart of the layers of software development.

Fig. Software Engineering layers

A Quality Focus: Software engineering must rest on an organizational commitment to quality. Total quality management and similar philosophies foster a continuous process improvement culture, and this culture ultimately leads to the development of increasingly more mature approaches to software engineering. The bedrock that supports software engineering is a quality focus.

Process: The foundation for software engineering is the process layer. Process defines a framework for a set of Key Process Areas (KPAs) that must be established for effective delivery of software engineering technology. This establishes the context in which technical methods are applied, work products such as models, documents, data, reports, forms, etc. are produced, milestones are established, quality is ensured, and change is properly managed.

Methods: Software engineering *methods* provide the technical how-to's for building software. Methods will include requirements analysis, design, program construction, testing, and support. This relies on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

Tools: Software engineering *tools* provide automated or semi-automated support for the process and the methods. When tools are integrated so that information created by one tool can be used by another, a system for the support of software development,

called computer-aided software engineering, is established.

Software process Framework

1) **What is software process framework.[7M] W-15**

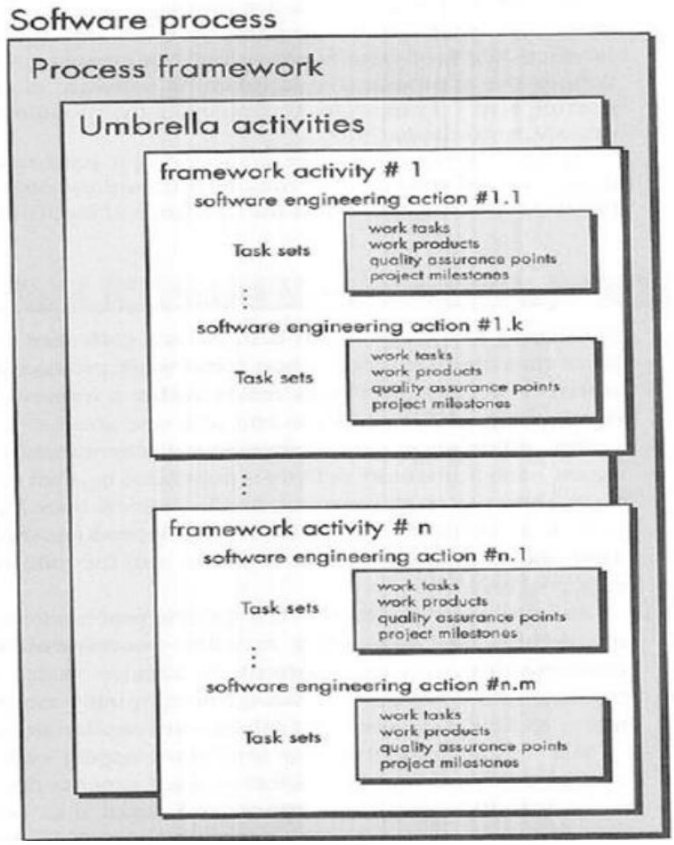
or

2) Explain common process framework for software engineering in details.[7M]S-15 W-17

or

3) Explain the software process framework. [6M] S-17

A *common process framework* is established by defining a small number of framework activities that are applicable to all software projects, regardless of their size or complexity. A number of *task sets* each a collection of software engineering work tasks, project milestones, work products, and quality assurance points enable the framework activities to be adapted to the characteristics of the software project and the requirements of the project team. Finally, umbrella activities such as software quality assurance, software configuration management, and measurement overlay the process model. Umbrella activities are independent of any one framework activity and occur throughout the process.


Software process
Process framework
Umbrella activities
framework activity # 1
software engineering action #1.1
Task sets
work tasks
work products
quality assurance points
project milestones
software engineering action #1.k
Task sets
work tasks
work products
quality assurance points
project milestones
framework activity # n
software engineering action #n.1
Task sets
work tasks
work products
quality assurance points
project milestones
software engineering action #n.m
Task sets
work tasks
work products
quality assurance points
project milestones

Framework activities:

Communication: Project requirements are collected in this activity. This framework activity is the main focus of the project managers and stakeholders. This framework activity includes communication and coordination with the clients.

Planning: This framework action incorporates data about the technical work to be planned, risks to be faced, resources needed for task completion, the decision of the milestone deadline to release the product in production.

Modeling: This framework phase includes the creation of product models by analyzing requirements and designing flowcharts and use cases which make developers and customers get a better understanding of the software functionalities that will achieve all requirements.

Construction: In this framework activity, the actual development of the product begins with code construction and then testing to fix errors and issues.

Deployment: In this framework phase, product prototypes are delivered or the completed software is delivered to the users. Feedback is obtained for getting delivered software's evaluation to add new features or future needs of the software.

Umbrella Activities:

Software project tracking and control

- In this activity, the developing team accesses project plan and compares it with the predefined schedule.
- If these project plans do not match with the predefined schedule, then the required actions are taken to maintain the schedule.

Risk management

- Risk is an event that may or may not occur.
- If the event occurs, then it causes some unwanted outcome. Hence, proper risk management is required.

Software Quality Assurance (SQA)

- SQA is the planned and systematic pattern of activities which are required to give a guarantee of software quality.

For example, during the software development meetings are conducted at every stage of development to find out the defects and suggest improvements to produce good quality software.

Formal Technical Reviews (FTR)

- FTR is a meeting conducted by the technical staff.
- The motive of the meeting is to detect quality problems and suggest improvements.
- The technical person focuses on the quality of the software from the customer point of view.

Measurement

- Measurement consists of the effort required to measure the software.
- The software cannot be measured directly. It is measured by direct and indirect measures.
- Direct measures like cost, lines of code, size of software etc.
- Indirect measures such as quality of software which is measured by some other factor. Hence, it is an indirect measure of software.

Software Configuration Management (SCM)

- It manages the effect of change throughout the software process.

Reusability management

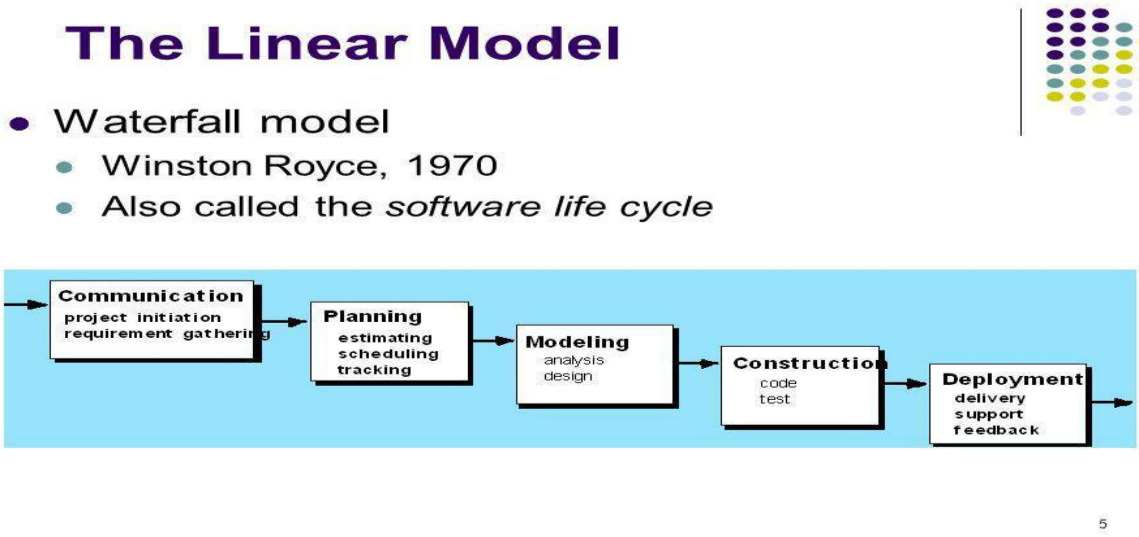
- It defines the criteria for reuse the product.
- The quality of software is good when the components of the software are developed for certain application and are useful for developing other applications.

Work product preparation and production

- It consists of the activities that are needed to create the documents, forms, lists, logs and user manuals for developing a software.

Waterfall Model

1) Define waterfall model?[3M] S-17



- The waterfall model, sometimes called the classic life cycle, suggests a systematic, sequential approach6 to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in ongoing support of the completed software.

- The waterfall model is the oldest paradigm for software engineering. However, over the past three decades, criticism of this process model has caused even ardent supporters to question its efficacy

Among the problems that are sometimes encountered when the waterfall model is applied are

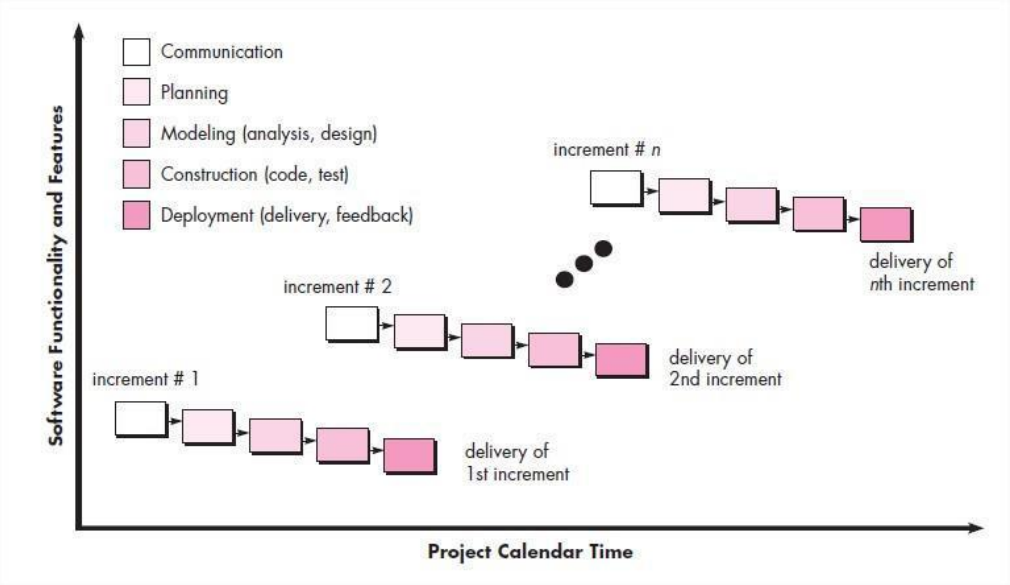
1. Real projects rarely follow the sequential flow that the model proposes. Although the linear model can accommodate iteration, it does so indirectly. As a result, changes can cause confusion as the project team proceeds.

2. It is often difficult for the customer to state all requirements explicitly. The waterfall model requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects.

3. The customer must have patience. A working version of the program(s) will not be available until late in the project time span. A major blunder, if undetected until the working program is reviewed, can be disastrous.

Incremental Process Models

Incremental model



- The incremental model combines elements of the linear sequential model (applied repetitively) with the iterative philosophy of prototyping.

- the incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces a deliverable "increment" of the software.

- For example, word-processing software developed using the incremental paradigm might deliver basic file management, editing, and document production functions in the first increment; more sophisticated editing and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advanced page layout capability in the fourth increment.

- It should be noted that the process flow for any increment can incorporate the prototyping paradigm. When an incremental model is used, the first increment is often a core product. That is, basic requirements are addressed, but many supplementary features (some known, others unknown) remain undelivered. The core product is used by the customer (or undergoes detailed review). As a result of use and/or evaluation, a plan is developed for the next increment.

- The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality. This process is repeated following the delivery of each increment, until the complete product is produced.

- The incremental process model, like prototyping and other evolutionary approaches, is iterative in nature. But unlike prototyping, the incremental model focuses on the delivery of an operational product with each increment. Early increments are stripped down versions of the final product, but they do provide capability that serves the user and also provide a platform for evaluation by the user.

- Incremental development is particularly useful when staffing is unavailable for a complete implementation by the business deadline that has been established for the project. Early increments can be implemented with fewer people. If the core product is well received, then additional staff can

be added to implement the next increment. In addition, increments can be planned to manage technical risks.

- For example, a major system might require the availability of new hardware that is under development and whose delivery date is uncertain. It might be possible to plan early increments in a way that avoids the use of this hardware, thereby enabling partial functionality to be delivered to end-users without inordinate delay.

Advantages :

- Generates working software quickly and early during the software life cycle.
- This model is more flexible – less costly to change scope and requirements.
- It is easier to test and debug during a smaller iteration.
- In this model customer can respond to each built.
- Lowers initial delivery cost.
- Easier to manage risk because risky pieces are identified and handled during it'd iteration.

Disadvantages:

- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall.

THE RAD MODEL

1) Write a short note on RAD model. [3M] W-15

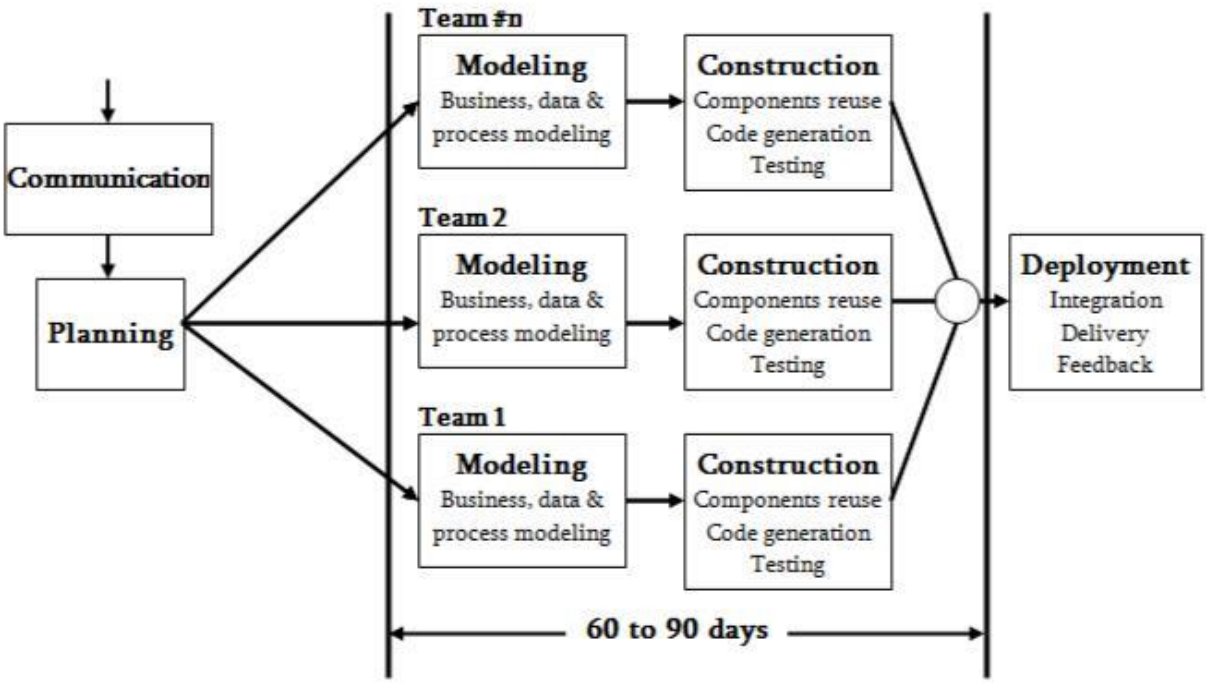


Figure : Flowchart of RAD model

Rapid application development (RAD) is an incremental software development process model that emphasizes an extremely short development cycle. The RAD model is a "high-speed" adaptation of the linear sequential model in which rapid development is achieved by using component-based construction. If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a "fully functional system" within very short time periods

Business modeling. The information flow among business functions is model in a way that answers the following questions: What information drives the business process? What information is generated? Who generates it? Where does the information go? Who processes it?

Data modeling. The information flow defined as part of the business modeling phase is refined into a set of data objects that are needed to support the business.

Process modeling. The data objects defined in the data modeling phase are transformed to achieve the information flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.

Application generation. RAD assumes the use of fourth generation techniques (Section 2.10). Rather than creating software using conventional third generation programming languages the RAD process works to reuse existing program components (when possible) or create reusable components (when necessary). In all cases, automated tools are used to facilitate construction of the software.

Testing and turnover., many of the program components have already been tested. This reduces overall testing time. However, new components must be tested and all interfaces must be fully exercised.

Advantages	Disadvantages
• Flexible and adaptable to changes	• It can't be used for smaller projects
• It is useful when you have to reduce the overall project risk	• Not all application is compatible with RAD
• It is adaptable and flexible to changes	• When technical risk is high, it is not suitable
• It is easier to transfer deliverables as scripts, high-level abstractions and intermediate codes are used	• If developers are not committed to delivering software on time, RAD projects can fail
• Due to code generators and code reuse, there is a reduction of manual coding	• Reduced features due to time boxing, where features are pushed to a later version to finish a release in short period
• Due to prototyping in nature, there is a possibility of lesser defects	• Reduced scalability occurs because a RAD developed application begins as a prototype and evolves into a finished application • Each phase in RAD delivers highest priority functionality to client

	• Progress and problems accustomed are hard to track as such there is no documentation to demonstrate what has been done
• With less people, productivity can	• Requires highly skilled designers or

be increased in short time	developers

EVOLUTIONARY SOFTWARE PROCESS MODELS

There are 3 types:

1. Prototype model
2. Spiral model

- Concurrent development model Evolutionary models are iterative type models.
- They allow to develop more complete versions of the software.

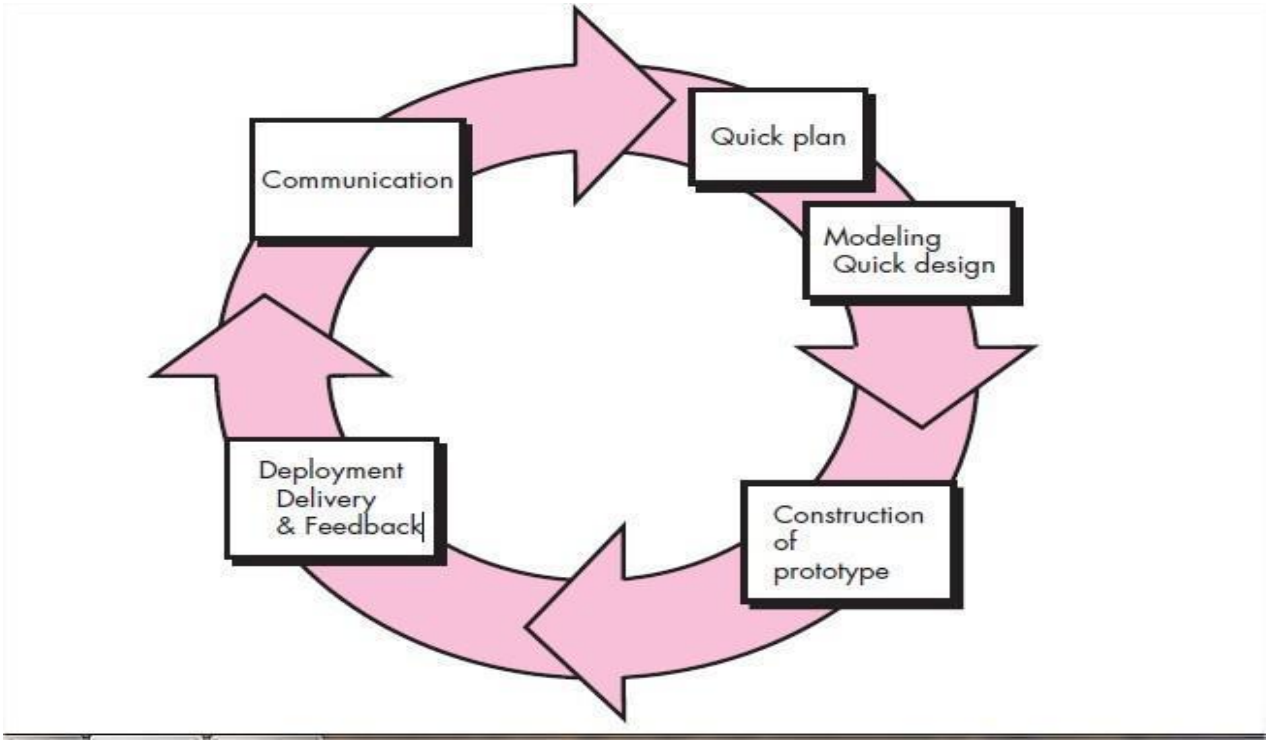
3. **Following are the evolutionary process models.**

1. The prototyping model
2. The spiral model
3. Concurrent development model

1) Define evolutionary process model.[3M] S-17

Prototype model

- Prototype is defined as first or preliminary form using which other forms are copied or derived.
- Prototype model is a set of general objectives for software.
- It does not identify the requirements like detailed input, output.
- It is software working model of limited functionality.
- In this model, working programs are quickly produced.



The different phases of Prototyping model are:

1. Communication

In this phase, developer and customer meet and discuss the overall objectives of the software.

2. Quick design

- Quick design is implemented when requirements are known.
- It includes only the important aspects like input and output format of the software.
- It focuses on those aspects which are visible to the user rather than the detailed plan.
- It helps to construct a prototype.

3. Modeling quick design

- This phase gives the clear idea about the development of software because the software is now built.
- It allows the developer to better understand the exact requirements.

4. Construction of prototype

The prototype is evaluated by the customer itself.

5. Deployment, delivery, feedback

- If the user is not satisfied with current prototype then it refines according to the requirements of the user.
- The process of refining the prototype is repeated until all the requirements of users are met.

- When the users are satisfied with the developed prototype then the system is developed on the basis of final prototype.

Advantages of Prototyping Model

- Prototype model need not know the detailed input, output, processes, adaptability of operating system and full machine interaction.
- In the development process of this model users are actively involved.
- The development process is the best platform to understand the system by the user.
- Errors are detected much earlier.
- Gives quick user feedback for better solutions.
- It identifies the missing functionality easily. It also identifies the confusing or difficult functions.

Disadvantages of Prototyping Model:

- The client involvement is more and it is not always considered by the developer.
- It is a slow process because it takes more time for development.
- Many changes can disturb the rhythm of the development team.

- It is a thrown away prototype when the users are confused with it.

Spiral Model:

1) Write short note on spiral Model. [3M] W-15

2) Explain spiral model of software development. State its advantages and disadvantages. [7M] S-15 S-18 S-16

3) Explain Boehm model of software development with neat sketch along with its advantages and disadvantages. [7M] W-16

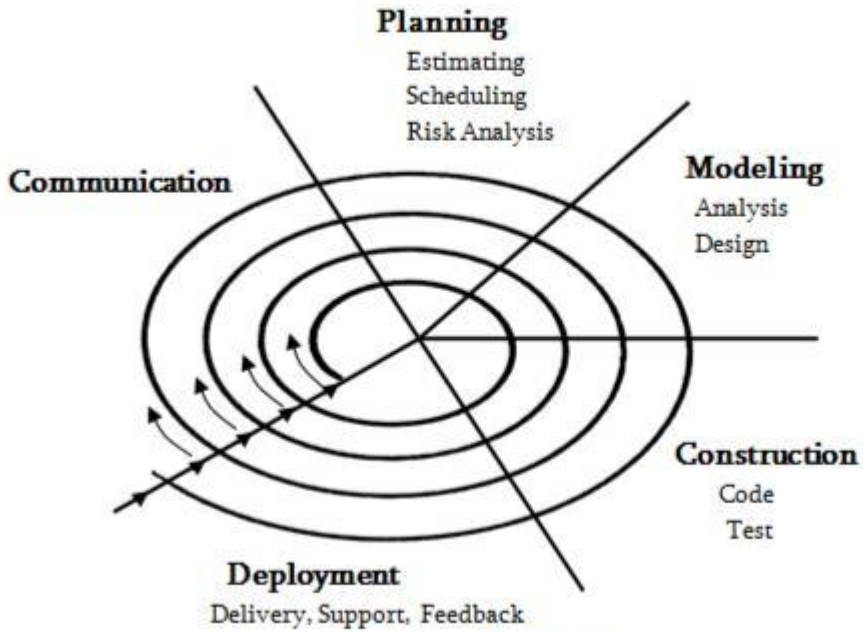


Figure : Spiral Model

- spiral Model is a combination of a waterfall model and iterative model. Each phase in spiral model begins with a design goal and ends with the client reviewing the progress. The spiral model was first mentioned by Barry Boehm in his 1986 paper.

- The development team in Spiral-SDLC model starts with a small set of requirement and goes through each development phase for those set of requirements. The software engineering team adds functionality for the additional requirement in every-increasing spirals until the application is ready for the production phase.

Planning	• It includes estimating the cost, schedule and resources for the iteration. It also involves understanding the system requirements for continuous communication between the system analyst and the customer

Risk Analysis	• Identification of potential risk is done while risk mitigation strategy is planned and finalized
Engineering	• It includes testing, coding and deploying software at the customer site
Evaluation	• Evaluation of software by the customer. Also, includes identifying and monitoring risks such as schedule slippage and cost over run

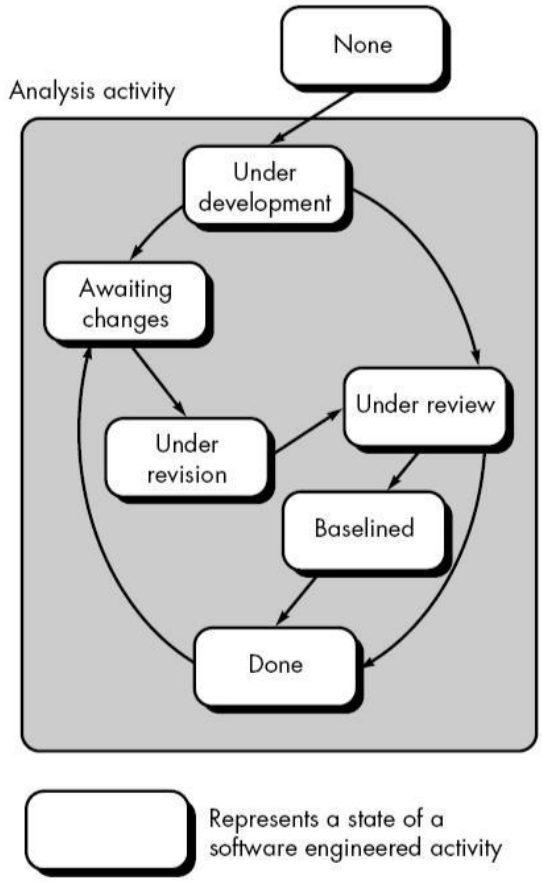
Advantages and Disadvantages of Spiral Model

Advantages	Disadvantages
• Additional functionality or changes can be done at a later stage	• Risk of not meeting the schedule or budget
• Cost estimation becomes easy as the prototype building is done in small fragments	• It works best for large projects only also demands risk assessment expertise
• Continuous or repeated development helps in risk management	• For its smooth operation spiral model protocol needs to be followed strictly

- Development is fast and features are added in a systematic way
- Documentation is more as it has intermediate phases

- There is always a space for customer feedback
- It is not advisable for smaller project, it might cost them a lot

The concurrent development model



- The concurrent development model is called as concurrent model.
- The communication activity has completed in the first iteration and exits in the awaiting changes state.
- The modeling activity completed its initial communication and then go to the underdevelopment state.
- If the customer specifies the change in the requirement, then the modeling activity moves from the under development state into the awaiting change state.
- The concurrent process model activities moving from one state to another state.

Advantages of the concurrent development model

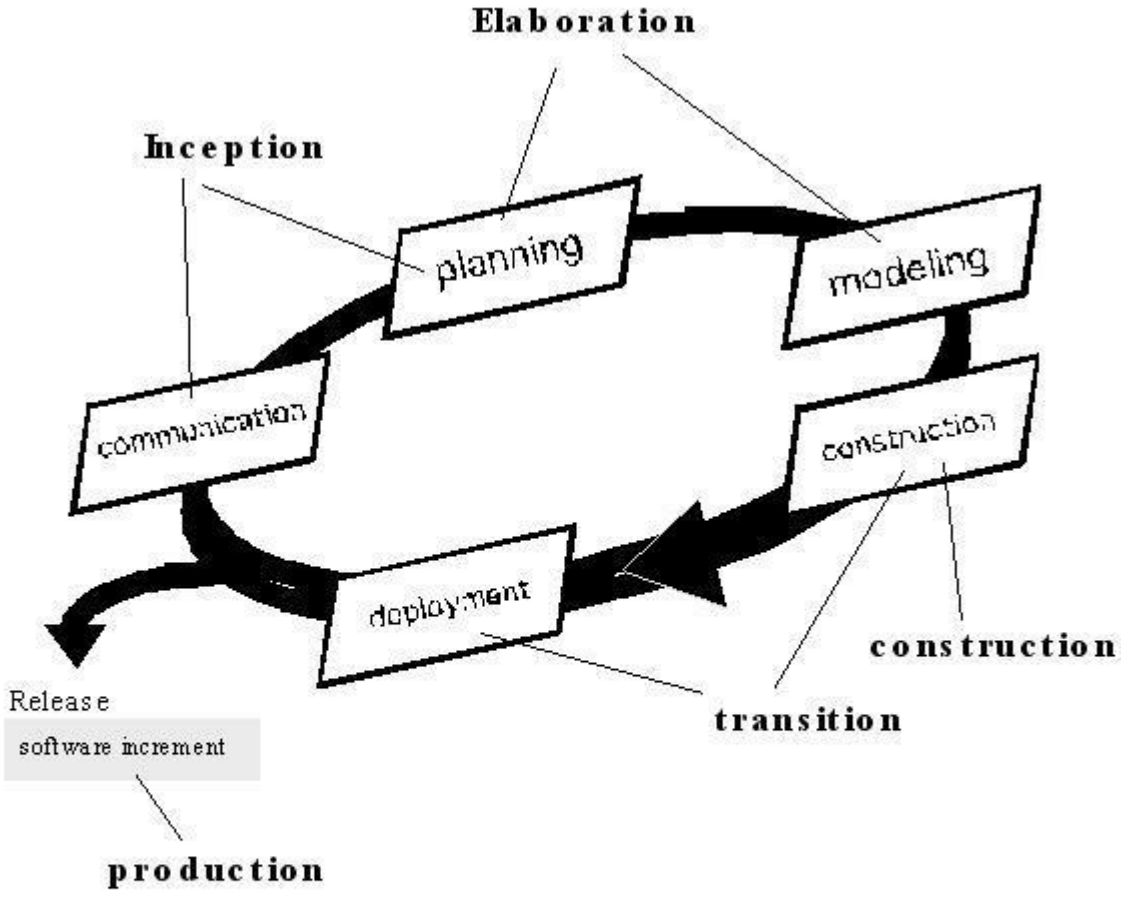
- This model is applicable to all types of software development processes.
- It is easy for understanding and use.
- It gives immediate feedback from testing.
- It provides an accurate picture of the current state of a project.

Disadvantages of the concurrent development model

- It needs better communication between the team members. This may not be achieved all the time.
- It requires to remember the status of the different activities

Unified process model:

1) Explain unified process model for developing the software.[6M] S-15 W-17
2) Explain the phases of unified process modelling.[7M] S-18 W-16



- **Unified process** is an attempt to draw on the best features and characteristics of conventional software process models.
- It recognizes the importance of customer communication and makes more efficient methods for describing the customer's view of a system.

Phases of the unified process:

The phases of the unified process are

1. Inception
2. Elaboration
3. construction
4. Transition
5. production

Inception:

- The inception phase of the unified process includes both customer communication and planning activities.
- By working together with the customer and end users, the business requirements for the software are identified, a rough architecture for the system is proposed

Elaboration

- Elaboration encompasses the customer communication and modeling activities.
- This will refines and expands the preliminary use cases that were developed as part of the inception phase and expands the architectural representation to include five different views of the software they are:

  Use case model, analysis model, design model, implementation model, and deployment model

Construction

- The construction phase develops the software components,
- As components are being implemented, unit tests are designed and executed for each component.

Transition

- Transition phase encompasses the latter stages of construction activity and the first part of the deployment activity.
- At the conclusion of the transition phase the software increment becomes a usable software release.

Production:

- It coincides with the deployment activity. In this phase the on going use of the software is monitored,
- Support for the infrastructure is provided, and defect reports and request for changes are submitted and evaluated.

Unit 2

1)Explain System Engineering hierarchy ? in details. W-17 , S-16

Ans: Regardless of its domain of focus, system engineering encompasses a collection of top-down and bottom-up methods to navigate the hierarchy.

The system engineering process usually begins with a "world view."

The world view is refined to focus more fully on

specific domain of interest. Within a specific domain, the need for targeted system elements (e.g., data, software, hardware, people) is analyzed.

The world view (WV) is composed of a set of domains (Di), which can each be a system or system of systems in its own right.

WV = {D1, D2, D3, . . . , Dn}

Each domain is composed of specific elements (Ej) each of which serves some role in accomplishing the objective and goals of the domain or component:

Di= {E1, E2, E3, . . . , Em}

Finally, each element is implemented by specifying the technical components (Ck) that achieve the necessary function for an element:

Ej= {C1, C2, C3, . . . , Ck}

- *Assumptions: Reduce the number of possible permutations and variations.*
- *Simplifications: enable the model to be created in a timely manner.*

To understand the flow of information.

Internal Demand

External Request

- *Limitations: Help to bound the system.*
- *Constraints: Will guide the manner in which the model is created and the approach taken when the model is implemented.*
- *Preferences: Indicate the preferred architecture for all data, functions, and technology.*

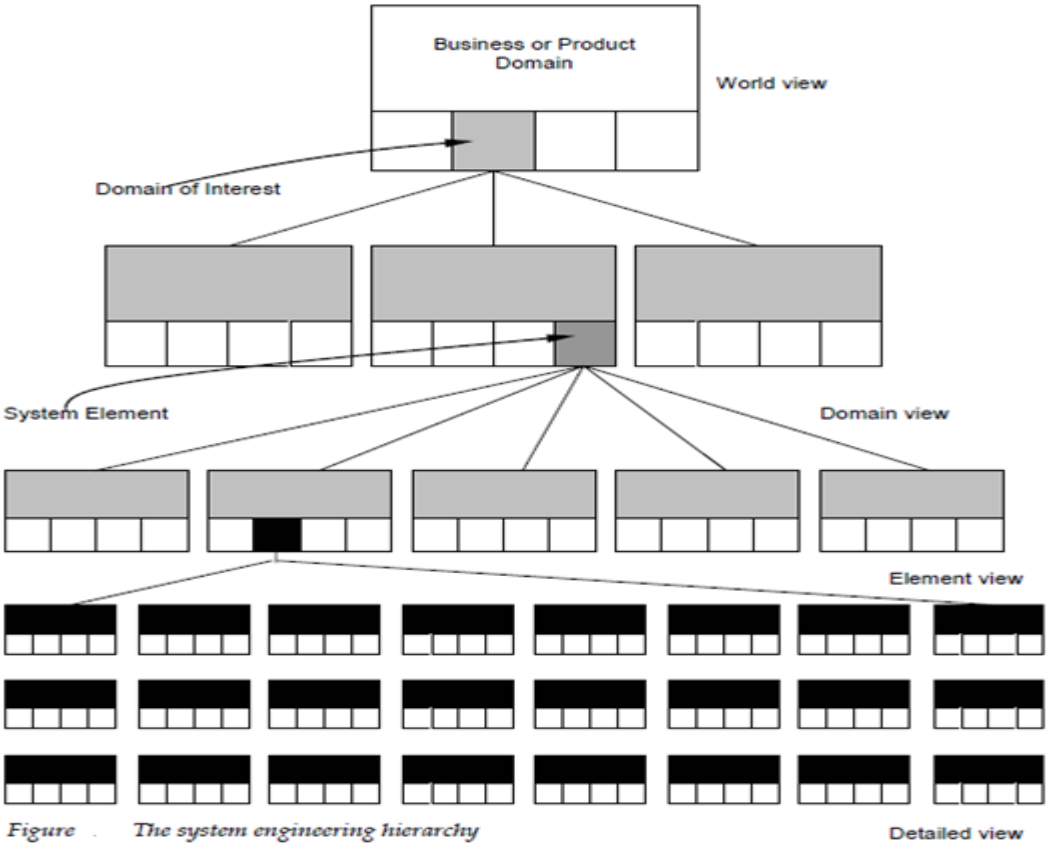


Figure . The system engineering hierarchy

2) **What is FAST? Explain in detail. (7) W-17, W-15, W-16 , S-16**

Ans: **Facilitated Application Specification Technique** ("FAST") is a technique for requirements elicitation for software development. The objective is to close the gap between what the developers intend and what users expect. It is a team-oriented approach for gathering requirements.

- Meetings are conducted at a neutral site attended by both developers and users.
- The group establishes rules for preparation and participation.
- An agenda is suggested that with enough formality to cover all important points but informal enough to encourage the free flow of ideas.
- A facilitator controls the meeting.
- A definition mechanism is used.

The main goal is to identify the problem, propose solutions, negotiate different approaches, and specify a preliminary set of software requirements in an atmosphere that is conductive to accomplish the goal.

after initial meeting, user and developer should write a one or two product request form. Before the next meeting it is distributed to all other attendees. Each attendee is asked to make the following lists:

1. List of objects
2. List of services
3. List of constraints
4. performance criteria

Representatives of FAST

1. Marketing person
2. Software and hardware engineer
3. Representative from manufacturing
4. An outside faciitator

3)Explain the components of computer based system? W-17 , W-15, S-16

Ans: **:** Defined: A set or arrangement of elements that are organized to accomplish some predefined goal by processing information

- The goal may be to support some business function or to develop a product that can be sold to generate business revenue
- A computer-based system makes use of system elements
- Elements constituting one system may represent one macro element of a still larger system
- Example :
  - A factory automation system may consist of a numerical control machine, robots, and data entry devices; each can be its own system
  - At the next lower hierarchical level, a manufacturing cell is its own computer-based system that may integrate other macro elements
- The role of the system engineer is to define the elements of a specific computer-based system in the context of the overall hierarchy of systems
- A computer-based system makes use of the following four system elements that combine in a variety of ways to transform information
  - Software: computer programs, data structures, and related work products that serve to effect the logical method, procedure, or control that is required
  - Hardware: electronic devices that provide computing capability, interconnectivity devices that enable flow of data, and electromechanical devices that provide external functions
  - People: Users and operators of hardware and software
  - Database: A large, organized collection of information that is accessed via software and persists over time
- The uses of these elements are described in the following:
  - Documentation: Descriptive information that portrays the use and operation of the system
  - Procedures: The that define the specific use of each system element or the procedural context in which the system resides

4) What is SRS? Explain in brief. W-17 ,W-15, S-16

A software requirements specification describes the essential behaviour of a software product from a user's point of view.

The purpose of the SRS is to:

a. **Establish the basis for agreement between the customers and the suppliers on what the software product is to do.** The complete description of the functions to be performed by the software specified in the SRS will assist the potential user to determine if the software specified meets their needs or how the software must be modified to meet their needs
b. **Provide a basis for developing the software design.** The SRS is the most important document of reference in developing a design
c. **Reduce the development effort.** The preparation of the SRS forces the various concerned groups in the customer's organisation to thoroughly consider all of the requirements before

design work begins. A complete and correct SRS reduces effort waisted on redesign, recoding and retesting. Careful review of the requirements in the SRS can reveal omissions, misunderstandings and inconsistencies early in the development cycle when these problems are easier to correct

d. **Provide a basis for estimating costs and schedules.** The description of the product to be developed as given in the SRS is a realistic basis for estimating project costs and can be used to obtain approval for bids or price estimates
e. **Provide a baseline for validation and verification.** Organisations can develop their test documentation much more productively from a good SRS. As a part of the development contract, the SRS provides a baseline against which compliance can be measured
f. **Facilitate transfer.** The SRS makes it easier to transfer the software product to new users or new machines. Customers thus find it easier to transfer the software to other parts of their organisation and suppliers find it easier to transfer it to new customers
g. **Serve as a basis for enhancement.** Because the SRS discusses the product but not the project that developed it, the SRS serves as a basis for later enhancement of the finished product. The SRS may need to be altered, but it does provide a foundation for continued product evaluation.

Contents of SRS:

1)Category: What kind of your software is

E.g.: Desktop application, web application, Android application

2)Purpose: Describe what is the purpose of making this system (as nothing is without reason)

3)Scope: What is the area it covering, what is its range, to what limits it help you.

4)Introduction: Define the existing system and

Proposed system

5)Advantages: Define the advantages of the system.

6)Functional Requirement of the system

7)Non Functional Requirement of the system.

8)Software Tools: Mention the software tools

which will involve in all this development process.

9) Deployment: What kind of environment will be

needed to deploy the software.

10)Hardware Specifications: The hardware required

To develop this system.

11)Gantt Chart: Chart use to show the schedule of project .

5) What is requirement engineering? Explain steps in Requirement Engineering (7)

Ans: We have trouble understanding the requirements that we do acquire from the customer

- We often record requirements in a disorganized manner
- We spend far too little time verifying what we do record
- We allow change to control us, rather than establishing mechanisms to control change
- Most importantly, we fail to establish a solid foundation for the system or software that the user wants built
- Many software developers argue that
- Building software is so compelling that we want to jump right in (before having a clear understanding of what is needed)
- Things will become clear as we build the software
- Project stakeholders will be able to better understand what they need only after examining early iterations of the software
- Things change so rapidly that requirements engineering is a waste of time
- The bottom line is producing a working program and that all else is secondary
- All of these arguments contain some truth, especially for small projects that take less than one month to complete
- However, as software grows in size and complexity, these arguments begin to break down and can lead to a failed software project

A Solution: Requirements Engineering

- Begins during the communication activity and continues into the modeling activity
- Builds a bridge from the system requirements into software design and construction
- Allows the requirements engineer to examine
  - the context of the software work to be performed
  - the specific needs that design and construction must address
  - the priorities that guide the order in which work is to be completed
  - the information, function, and behavior that will have a profound impact on the resultant design

Requirements Engineering Tasks:

- Seven distinct tasks
  - Inception
  - Elicitation
  - Elaboration
  - Negotiation
  - Specification
  - Validation
  - Requirements Management
- **Inception:** During inception, the requirements engineer asks a set of questions to establish.
  - A basic understanding of the problem
  - The people who want a solution
  - The nature of the solution that is desired
  - The effectiveness of preliminary communication and collaboration between the customer and the developer
- Through these questions, the requirements engineer needs to…
  - Identify the stakeholders
  - Recognize multiple viewpoints
  - Work toward collaboration
  - Break the ice and initiate the communication
- **Elicitation:** Eliciting requirements is difficult because of
  - Problems of scope in identifying the boundaries of the system or specifying too much technical detail rather than overall system objectives

- Problems of understanding what is wanted, what the problem domain is, and what the computing environment can handle (Information that is believed to be "obvious" is often omitted)
- Problems of volatility because the requirements change over time

- Elicitation may be accomplished through two activities
  - Collaborative requirements gathering
  - Quality function deployment

Elaboration:

- During elaboration, the software engineer takes the information obtained during inception and elicitation and begins to expand and refine it
- Elaboration focuses on developing a refined technical model of software functions, features, and constraints
- It is an analysis modeling task
  - Use cases are developed
  - Domain classes are identified along with their attributes and relationships
  - State machine diagrams are used to capture the life on an object
- The end result is an analysis model that defines the functional, informational, and behavioral domains of the problem

Negotiation:

- During negotiation, the software engineer reconciles the conflicts between what the customer wants and what can be achieved given limited business resources
- Requirements are ranked (i.e., prioritized) by the customers, users, and other stakeholders
- Risks associated with each requirement are identified and analyzed
- Rough guesses of development effort are made and used to assess the impact of each requirement on project cost and delivery time
- Using an iterative approach, requirements are eliminated, combined and/or modified so that each party achieves some measure of satisfaction

Specification: A specification is the final work product produced by the requirements engineer

- It is normally in the form of a software requirements specification
- It serves as the foundation for subsequent software engineering activities
- It describes the function and performance of a computer-based system and the constraints that will govern its development
- It formalizes the informational, functional, and behavioral requirements of the proposed software in both a graphical and textual format

Validation:

- During validation, the work products produced as a result of requirements engineering are assessed for quality
- The specification is examined to ensure that
  - all software requirements have been stated unambiguously
  - inconsistencies, omissions, and errors have been detected and corrected
  - the work products conform to the standards established for the process, the project, and the product
- The formal technical review serves as the primary requirements validation mechanism
  - Members include software engineers, customers, users, and other stakeholders

Requirements Management:

- During requirements management, the project team performs a set of activities to identify, control, and track requirements and changes to the requirements at any time as the project proceeds
- Each requirement is assigned a unique identifier
- The requirements are then placed into one or more traceability tables

- These tables may be stored in a database that relate features, sources, dependencies, subsystems, and interfaces to the requirements
- A requirements traceability table is also placed at the end of the software requirements specification

6) Explain Business Process Engineering in detail. W-15

Ans: The final BPE step—*construction and integration* focuses on implementation detail. The architecture and infrastructure are implemented by constructing an appropriate database and internal data structures, by building applications using software components, and by selecting appropriate elements of a technology infrastructure to support the design created during BSD. Each of these system components
must then be integrated to form a complete information system or application. The integration activity also places the new information system into the business area context, performing all user training and logistics support to achieve a smooth transition.

Three different architectures must be analyzed and designed within the context of business objectives and goals:

- data architecture
- applications architecture
- technology infrastructure
- ***The data architecture*** provides a framework for the information needs of a business or business function.
- ***The application architecture*** encompasses those elements of a system that transform objects within the data architecture for some business purpose.
- **The technology infrastructure** provides the foundation for the data and application architectures. The infrastructure encompasses the hardware and software that are used to support the application and data.

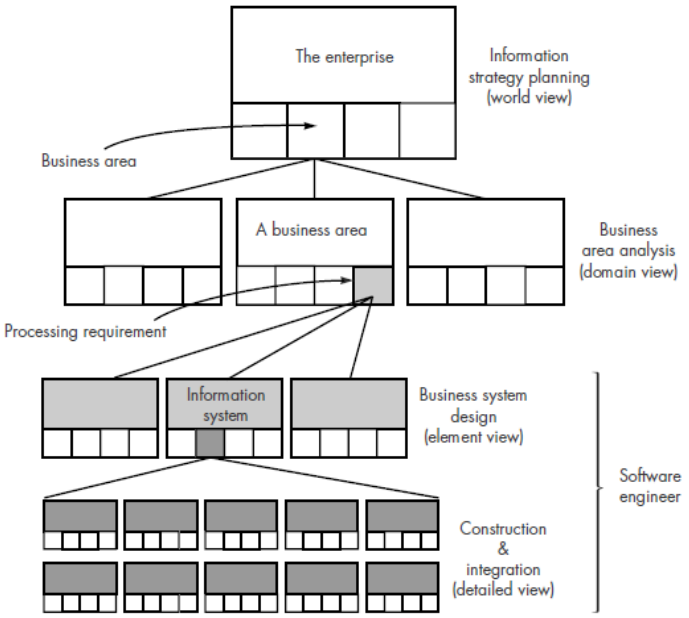
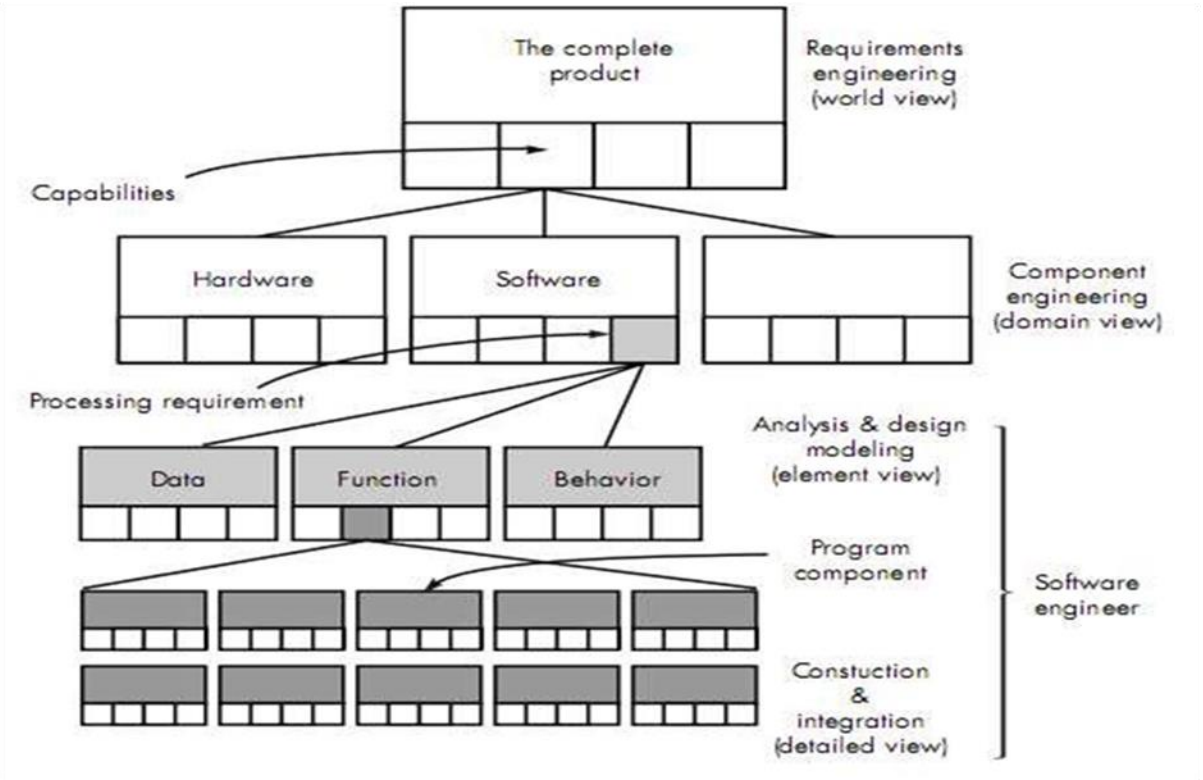


Fig. The business process engineering hierarchy

7) Describe product engineering concept in detail. **S-18**



The goal of product engineering is to translate the customer's desire for a set of defined capabilities into a working product.

To achieve this goal, product engineering—like business process engineering—must derive architecture and infrastructure.

- **he architecture encompasses four distinct system components: software, hardware, data (and databases), and people.**
- **The infrastructure includes the technology required to tie the components together and the information that is used to support the components.**

Unit 3

1) Explain Behaviroural Model in brief. **(6)**

Ans: All behavioural models really do is describe the control structure of a system.
This can be things like:

- Sequence of operations
- Object states
- and Object interactions

Furthermore, this modelling layer can also be called Dynamic Modelling. The activity of creating a behavioural model is commonly known as behavioural modelling. As well as this, a system should also only have one behavioural model – much like functional modelling.

How do we Represent them?
So, how do we represent behavioural models? Well, we represent them with dynamic diagrams. For example:

- (Design) Sequence Diagrams
- Communication Diagrams *or* collaboration diagram
- State Diagrams *or* state machine diagram *or* state chart

If we have both a sequence diagram AND and communication diagram, then together these are known as interaction diagrams, this is because they both represent how objects interact with one another using messages.

What do they Describe?
A behavioural model describes when the system is changing.
The key feature (subject) of a behavioural model is – Objects.

2) Explain following terms :
i) Abstraction. ii) Modularity
iii) Information Hiding iv) Refactoring.

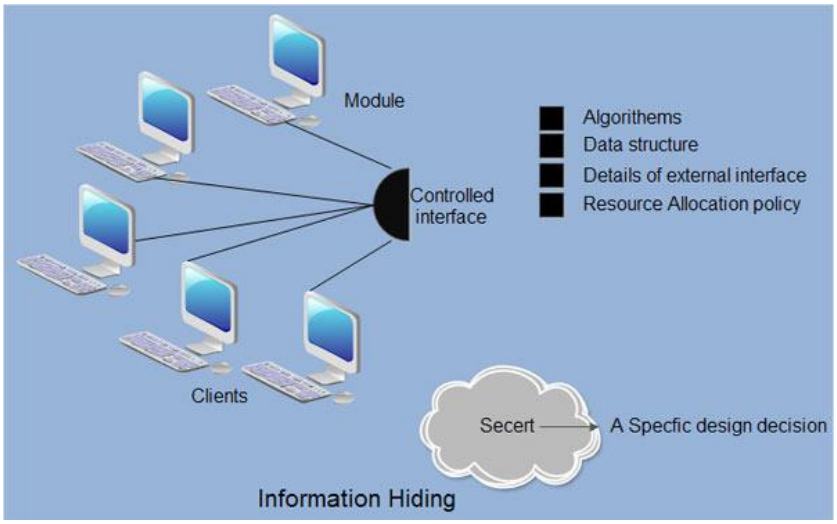
Ans: (i) Abstraction: Abstraction refers to a powerful design tool, which allows software designers to consider components at an abstract level, while neglecting the implementation details of the components. The concept of abstraction can be used in two ways: as a process and as an entity. As a process, it refers to a mechanism of hiding irrelevant details and representing only the essential features of an item so that one can focus on important things at a time. As an entity, it refers to a model or view of an item.
Each step in the software process is accomplished through various levels of abstraction. At the highest level, an outline of the solution to the problem is presented whereas at the lower levels, the solution to the problem is presented in detail. For example, in the requirements analysis phase, a solution to the problem is presented using the language of problem environment and as we proceed through the software process, the abstraction level reduces and at the lowest level, source code of the software is produced.
There are three commonly used abstraction mechanisms in software design, namely, functional abstraction, data abstraction and control abstraction. All these mechanisms allow us to control the complexity of the design process by proceeding from the abstract design model to concrete design model in a systematic manner.

1. **Functional abstraction:** This involves the use of parameterized subprograms. Functional abstraction can be generalized as collections of subprograms referred to as 'groups'. Within these groups there exist routines which may be visible or hidden. Visible routines can be used within the containing groups as well as within other groups, whereas hidden routines are hidden from other groups and can be used within the containing group only.
2. **Data abstraction:** This involves specifying data that describes a data object. For example, the data object *window* encompasses a set of attributes (window type, window dimension) that describe the window object clearly. In this abstraction mechanism, representation and manipulation details are ignored.
3. **Control abstraction:** This states the desired effect, without stating the exact mechanism of control. For example, if and while statements in programming languages (like C and C++) are abstractions of machine code implementations, which involve conditional instructions. In the architectural design level, this abstraction mechanism permits specifications of sequential subprogram and exception handlers without the concern for exact details of implementation.

(ii) **Information Hiding:** Modules should be specified and designed in such a way that the data structures and processing details of one module are not accessible to other modules. They pass only that

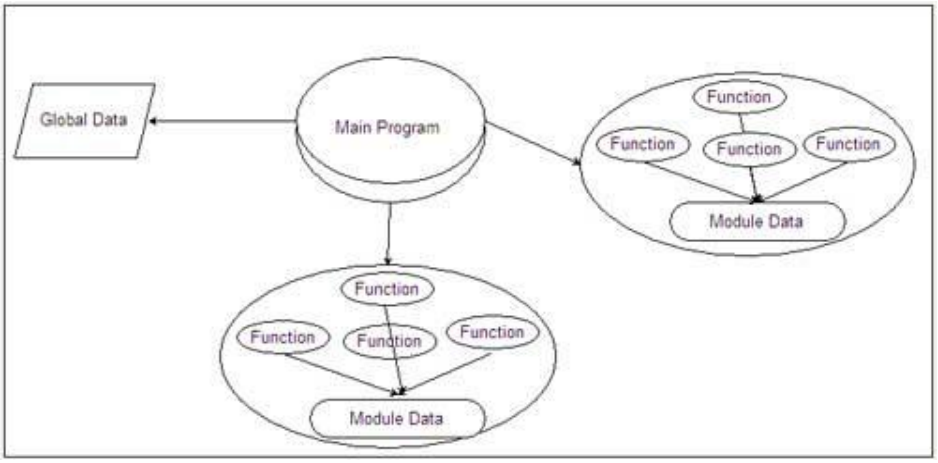
much information to each other, which is required to accomplish the software functions. The way of hiding unnecessary details is referred to as **information hiding. IEEE** defines information hiding as 'the technique of encapsulating software design decisions in modules in such a way that the module's interfaces reveal as little as possible about the module's inner workings; thus each module is a 'black box' to the other modules in the system. Information hiding is of immense use when modifications are required during the testing and maintenance phase. Some of the advantages associated with information hiding are listed below.

1. Leads to low coupling
2. Emphasizes communication through controlled interfaces
3. Decreases the probability of adverse effects
4. Restricts the effects of changes in one component on others
5. Results in higher quality software.



Refactoring: Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure. The concept of refactoring covers practically any revision or cleaning up of source code. It *Improves the Design of Existing Code*. One approach to refactoring is to improve the structure of source code at one point and then extend the same changes systematically to all applicable references throughout the program. The result is to make the code more efficient, scalable, maintainable or reusable, without actually changing any functions of the program itself.

Modularity is achieved by dividing the software into uniquely named and addressable components,which are also known as **modules.** A complex system (large program) is partitioned into a set of discrete modules in such a way that each module can be developed independent of other modules. After developing the modules, they are integrated together to meet the software requirements. Note that larger the number of modules a system is divided into, greater will be the effort required to integrate the modules.



Modularizing a design helps to plan the development in a more effective manner, accommodate changes easily, conduct testing and debugging effectively and efficiently, and conduct maintenance work without adversely affecting the functioning of the software.

3) Explain the components of analysis modelling.. (5)

Ans: 1. Scenario based element

- This type of element represents the system user point of view.
- Scenario based elements are use case diagram, user stories.

2. Class based elements

- The object of this type of element manipulated by the system.
- It defines the object,attributes and relationship.
- The collaboration is occurring between the classes.
- Class based elements are the class diagram, collaboration diagram.

3. Behavioral elements

- Behavioral elements represent state of the system and how it is changed by the external events.
- The behavioral elements are sequenced diagram, state diagram.

4. Flow oriented elements

- An information flows through a computer-based system it gets transformed.
- It shows how the data objects are transformed while they flow between the various system functions.
- The flow elements are data flow diagram, control flow diagram.

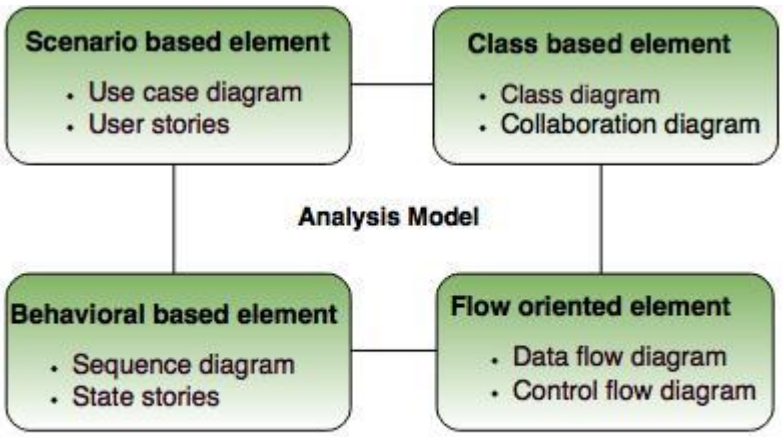


Fig. - Elements of analysis model

4) Explain different design principles in detail. (10)

Ans:

Principle 1. ***Design should be traceable to the requirements model.***

The requirements model describes the information domain of the problem, user-visible functions, system behavior, and a set of requirements classes that package business objects with the methods that service them. The design model translates this information into an architecture, a set of subsystems that implement major functions, and a set of components that are the realization of requirements classes. The elements of the design model should be traceable to the requirements model.

Principle 2. ***Always consider the architecture of the system to be built.***

Software architecture (Chapter 9) is the skeleton of the system to be built. It affects interfaces, data structures, program control flow and behavior, the manner in which testing can be conducted, the maintainability of the resultant system, and much more. For all of these reasons, design should start with architectural considerations. Only after the architecture has been established should component-level issues be considered.

Principle 3. ***Design of data is as important as design of processing functions.***

Data design is an essential element of architectural design. The manner in which data objects are realized within the design cannot be left to chance. A well-structured data design helps to simplify program flow, makes the design and implementation of software components easier, and makes overall processing more efficient.

Principle 4. ***Interfaces (both internal and external) must be designed with care.***

The manner in which data flows between the components of a system has much to do with processing efficiency, error propagation, and design simplicity. A well-designed interface makes integration easier and assists the tester in validating component functions.

Principle 5. ***User interface design should be tuned to the needs of the end user. However, in every case, it should stress ease of use.***

The user interface is the visible manifestation of the software. No matter how sophisticated its internal functions, no matter how comprehensive its data structures, no matter how well designed its architecture, a poor interface design often leads to the perception that the software is "bad."

Principle 6. ***Component-level design should be functionally independent.***

Functional independence is a measure of the "single-mindedness" of a software component. The functionality that is delivered by a component should be cohesive—that is, it should focus on one and only one function or subfunction.5

Principle 7. ***Components should be loosely coupled to one another and to the external environment.*** Coupling is achieved in many ways—via a component interface, by messaging, through global data. As the level of coupling increases, the likelihood of error propagation also increases and the overall maintainability of the software decreases. Therefore, component coupling should be kept as low as is reasonable.

Principle 8. ***Design representations (models) should be easily understandable.***

The purpose of design is to communicate information to practitioners who will generate code, to those who will test the software, and to others who may maintain the software in the future. If the design is difficult to understand, it will not serve as an effective communication medium.

Principle 9. ***The design should be developed iteratively. With each iteration, the designer should strive for greater simplicity.***

Like almost all creative activities, design occurs iteratively. The first iterations work to refine the design and correct errors, but later iterations should strive to make the design as simple as is possible.

Unit 4

1) Write notes on any two.
i) White Box Testing.
ii) Black Box Testing.
iii) Art of debugging.. **(14)**

Ans: White Box Testing :

White Box Testing (also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing) is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential. White box testing is testing beyond the user interface and into the nitty-gritty of a system.

This method is named so because the software program, in the eyes of the tester, is like a white/ transparent box; inside which one clearly sees.

Definition by ISTQB

- **white-box testing:** Testing based on an analysis of the internal structure of the component or system.
- **white-box test design technique:** Procedure to derive and/or select test cases based on an analysis of the internal structure of a component or system.

EXAMPLE

A tester, usually a developer as well, studies the implementation code of a certain field on a webpage, determines all legal (valid and invalid) AND illegal inputs and verifies the outputs against the expected outcomes, which is also determined by studying the implementation code.

White Box Testing is like the work of a mechanic who examines the engine to see why the car is not moving.

LEVELS APPLICABLE TO

White Box Testing method is applicable to the following levels of software testing:

- Unit Testing: For testing paths within a unit.
- Integration Testing: For testing paths between units.
- System Testing: For testing paths between subsystems.

However, it is mainly applied to Unit Testing.

WHITE BOX TESTING ADVANTAGES

- Testing can be commenced at an earlier stage. One need not wait for the GUI to be available.

- Testing is more thorough, with the possibility of covering most paths.

WHITE BOX TESTING DISADVANTAGES

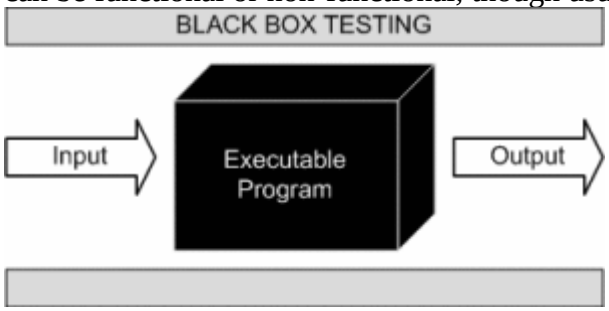
- Since tests can be very complex, highly skilled resources are required, with thorough knowledge of programming and implementation.
- Test script maintenance can be a burden if the implementation changes too frequently.
- Since this method of testing is closely tied with the application being testing, tools to cater to every kind of implementation/platform may not be readily available.

White Box Testing is contrasted with Black Box Testing. Read the Differences between Black Box Testing and White Box Testing.

Black Box Testing

DEFINITION

Black Box Testing, also known as Behavioral Testing, is a software testing method in which the internal structure/ design/ implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.



This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see. This method attempts to find errors in the following categories:

- Incorrect or missing functions
- Interface errors
- Errors in data structures or external database access
- Behavior or performance errors
- Initialization and termination errors

Definition by ISTQB

- **black box testing:** Testing, either functional or non-functional, without reference to the internal structure of the component or system.
- **black box test design technique:** Procedure to derive and/or select test cases based on an analysis of the specification, either functional or non-functional, of a component or system without reference to its internal structure.

EXAMPLE

A tester, without knowledge of the internal structures of a website, tests the web pages by using a browser; providing inputs (clicks, keystrokes) and verifying the outputs against the expected outcome.

LEVELS APPLICABLE TO

Black Box Testing method is applicable to the following levels of software testing:

- Integration Testing
- System Testing
- Acceptance Testing

The higher the level, and hence the bigger and more complex the box, the more black box testing method comes into use.

BLACK BOX TESTING TECHNIQUES

Following are some techniques that can be used for designing black box tests.

- *Equivalence partitioning:* It is a software test design technique that involves dividing input values into valid and invalid partitions and selecting representative values from each partition as test data.
- *Boundary Value Analysis:* It is a software test design technique that involves determination of boundaries for input values and selecting values that are at the boundaries and just inside/ outside of the boundaries as test data.

- *Cause Effect Graphing:* It is a software test design technique that involves identifying the cases (input conditions) and effects (output conditions), producing a Cause-Effect Graph, and generating test cases accordingly.

BLACK BOX TESTING ADVANTAGES

- Tests are done from a user's point of view and will help in exposing discrepancies in the specifications.
- Tester need not know programming languages or how the software has been implemented.
- Tests can be conducted by a body independent from the developers, allowing for an objective perspective and the avoidance of developer-bias.
- Test cases can be designed as soon as the specifications are complete.

BLACK BOX TESTING DISADVANTAGES

- Only a small number of possible inputs can be tested and many program paths will be left untested.
- Without clear specifications, which is the situation in many projects, test cases will be difficult to design.
- Tests can be redundant if the software designer/ developer has already run a test case.
- Ever wondered why a soothsayer closes the eyes when foretelling events? So is almost the case in Black Box Testing.

iii) **(iii) Art of Debugging:** Debugging happens as a result of testing. When a test case uncovers an error, debugging is the process that causes the removal of that error.

1. The Debugging Process

Debugging is not testing, but always happens as a response of testing. The debugging process will have one of two outcomes: (1) The cause will be found, corrected and removed, or (2) the cause will not be found. Why is debugging difficult?

- The symptom and the cause are geographically remote.
- The symptom may disappear when another error is corrected.
- The symptom may actually be the result of nonerrors (eg round off in accuracies).
- The symptom may be caused by a human error that is not easy to find.
- The symptom may be intermittent.
- The symptom might be due to the causes that are distributed across various tasks on diverse processes.

2) Explain how unit testing is performed for software. (7)

Ans: Unit testing focuses verification effort on the smallest unit of software design—the software component or module. Using the component-level design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and uncovered errors is limited by the constrained scope established for unit testing. The unit test is white-box oriented, and the step can be conducted in parallel for multiple components.

Unit Test Considerations

The tests that occur as part of unit tests are illustrated schematically in Figure. The module interface is tested to ensure that information properly flows into and out of the program unit under test. The local data structure is examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution.

Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing. All independent paths (basis paths) through the control structure are exercised to ensure that all statements in a module have been executed at least once. And finally, all error handling paths are tested.

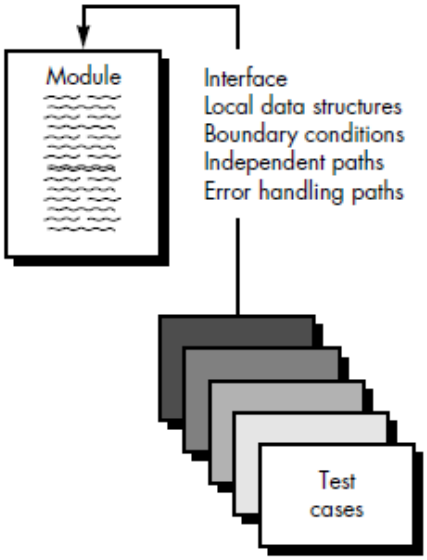


Fig. Unit Test

Tests of data flow across a module interface are required before any other test is initiated. If data do not enter and exit properly, all other tests are moot. In addition, local data structures should be exercised and the local impact on global data should be ascertained (if possible) during unit testing.

Selective testing of execution paths is an essential task during the unit test. Test cases should be designed to uncover errors due to erroneous computations, incorrect comparisons, or improper control flow. Basis path and loop testing are effective techniques for uncovering a broad array of path errors.

Among the more common errors in computation are

(1) misunderstood or incorrect arithmetic precedence, (2) mixed mode operations, (3) incorrect initialization,(4) precision inaccuracy, (5) incorrect symbolic representation of an expression.

Comparison and control flow are closely coupled to one another (i.e., change of flow frequently occurs after a comparison).

Test cases should uncover errors such as

(1) comparison of different data types, (2) incorrect logical operators or precedence,

(3) expectation of equality when precision error makes equality unlikely, (4) incorrect comparison of variables, (5) improper or nonexistent loop termination, (6) failure to exit when divergent iteration is encountered, and (7) improperly modified loop variables.

Good design dictates that error conditions be anticipated and error-handling paths set up to reroute or cleanly terminate processing when an error does occur.

3**) What do you mean by System testing. (3)**

Ans System testing is the type of testing to check the behaviour of a complete and fully integrated software product based on the software requirements specification (SRS) document. The main focus of this testing is to evaluate Business / Functional / End-user requirements.

This is black box type of testing where external working of the software is evaluated with the help of requirement documents & it is totally based on Users point of view. For this type of testing do not required knowledge of internal design or structure or code.

This testing is to be carried out only after *System Integration Testing* is completed where both Functional & Non-Functional requirements are verified.

In the integration testing testers are concentrated on finding bugs/defects on integrated modules. But in the *Software System Testing* testers are concentrated on finding bugs/defects based on software application behavior, software design and expectation of end user.

Why system testing is important:

a) In Software Development Life Cycle the System Testing is perform as the first level of testing where the System is tested as a whole.

b) In this step of testing check if system meets functional requirement or not.

c) *System Testing* enables you to test, validate and verify both the Application Architecture and Business requirements.

d) The application/System is tested in an environment that particularly resembles the effective production environment where the application/software will be lastly deployed.

Generally, a separate and dedicated team is responsible for system testing. And, System Testing is performed on staging server which is similar to production server. So this means you are testing software application as good as production environment.

4)Write a short note on:

(iii) Art of Debugging

Ans: Refer Q7 S-16

(iii) Art of Debugging: Debugging happens as a result of testing. When a test case uncovers an error, debugging is the process that causes the removal of that error.

1. The Debugging Process

Debugging is not testing, but always happens as a response of testing. The debugging process will have one of two outcomes: (1) The cause will be found, corrected and removed, or (2) the cause will not be found. Why is debugging difficult?

- The symptom and the cause are geographically remote.
- The symptom may disappear when another error is corrected.
- The symptom may actually be the result of nonerrors (eg round off in accuracies).
- The symptom may be caused by a human error that is not easy to find.
- The symptom may be intermittent.
- The symptom might be due to the causes that are distributed across various tasks on diverse processes.

2. Psychological Considerations

There is evidence that debugging is an innate human trait. Some people are good at it and others not. Although experimental evidence on debugging can be considered in many ways large variations in debugging ability has been identified in software engineering of the same experience.

3. Debugging Approaches

Regardless of the approach that is used, debugging has one main aim: to determine and correct errors. The aim is achieved by using systematic evaluation, intuition, and good fortune. In general three kinds of debugging approaches have been put forward: Brute force, Back tracking and Cause elimination.

Brute force is probably the most popular despite being the least successful. We apply brute force debugging methods when all else fails. Using a “let the computer find the error” technique, memory dumps are taken, run-time traces are invoked, and the program is loaded with WRITE

statements. Backtracking is a common debugging method that can be used successfully in small programs. Beginning at the site where a symptom has been uncovered, the source code is traced backwards till the error is found. In cause elimination a list of possible causes of an error are identified and tests are conducted until each one is eliminated.

5. Write a short note on: (Any Three) (14)

(i) System Testing (ii) Integration Testing (iii) Unit Testing (iv) Alpha and Beta Testing

Ans: (i) System Testing:

System testing is the type of testing to check the behaviour of a complete and fully integrated software product based on the software requirements specification (SRS) document. The main focus of this testing is to evaluate Business / Functional / End-user requirements.

This is black box type of testing where external working of the software is evaluated with the help of requirement documents & it is totally based on Users point of view. For this type of testing do not required knowledge of internal design or structure or code.

This testing is to be carried out only after *System Integration Testing* is completed where both Functional & Non-Functional requirements are verified.

In the integration testing testers are concentrated on finding bugs/defects on integrated modules. But in the *Software System Testing* testers are concentrated on finding bugs/defects based on software application behavior, software design and expectation of end user.

Why system testing is important:

a) In Software Development Life Cycle the System Testing is perform as the first level of testing where the System is tested as a whole.

b) In this step of testing check if system meets functional requirement or not.

c) *System Testing* enables you to test, validate and verify both the Application Architecture and Business requirements.

d) The application/System is tested in an environment that particularly resembles the effective production environment where the application/software will be lastly deployed.

Generally, a separate and dedicated team is responsible for system testing. And, System Testing is performed on staging server which is similar to production server. So this means you are testing software application as good as production environment.

(ii) Integration Testing:

Normally Integration testing is done after "Unit testing".

Once all the individual units are created and tested, we start combining those "Unit Tested" modules and start doing the integrated testing. So the meaning of Integration testing is quite straight forward-

Integrate/combine the unit tested module one by one and test the behavior as a combined unit.

The main function or goal of Integration testing is to test the interfaces between the units/modules.

The individual modules are first tested in isolation. Once the modules are unit tested, they are integrated one by one, till all the modules are integrated, to check the combinational behavior, and validate whether the requirements are implemented correctly or not.

Here we should understand that, Integration testing does not happens at the end of the cycle, rather it is conducted simultaneously with the development. So in most of the times all the modules are not actually available to test and here is what the challenge comes to test something which does not exists!

There are two major ways of carrying out an integration test:

1.Bottom-up method : Bottom-up integration testing begins with unit testing, followed by tests of of progressively higher-level combinations of units called modules or builds.

2.Top-down method: In top-down integration testing, the highest-level modules are tested first and progressively lower-level modules are tested after that.

In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application, preferably in scenarios designed to mimic those it will encounter in customers' computers, systems and networks.

.6) Explain in detail White-Box Testing and Black-Box Testing. (14)

Ans: White Box Testing :

White Box Testing (also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing) is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential. White box testing is testing beyond the user interface and into the nitty-gritty of a system.

This method is named so because the software program, in the eyes of the tester, is like a white/ transparent box; inside which one clearly sees.

Definition by ISTQB

- **white-box testing:** Testing based on an analysis of the internal structure of the component or system.
- **white-box test design technique:** Procedure to derive and/or select test cases based on an analysis of the internal structure of a component or system.

EXAMPLE

A tester, usually a developer as well, studies the implementation code of a certain field on a webpage, determines all legal (valid and invalid) AND illegal inputs and verifies the outputs against the expected outcomes, which is also determined by studying the implementation code.

White Box Testing is like the work of a mechanic who examines the engine to see why the car is not moving.

LEVELS APPLICABLE TO

White Box Testing method is applicable to the following levels of software testing:

- Unit Testing: For testing paths within a unit.
- Integration Testing: For testing paths between units.
- System Testing: For testing paths between subsystems.

However, it is mainly applied to Unit Testing.

WHITE BOX TESTING ADVANTAGES

- Testing can be commenced at an earlier stage. One need not wait for the GUI to be available.
- Testing is more thorough, with the possibility of covering most paths.

WHITE BOX TESTING DISADVANTAGES

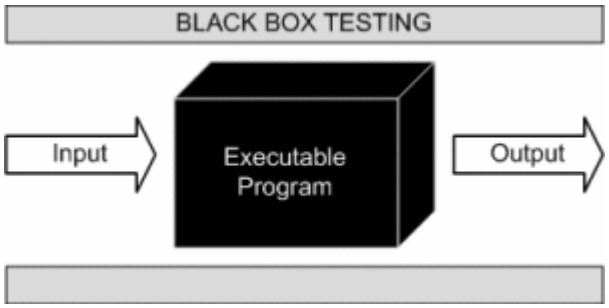
- Since tests can be very complex, highly skilled resources are required, with thorough knowledge of programming and implementation.
- Test script maintenance can be a burden if the implementation changes too frequently.
- Since this method of testing is closely tied with the application being testing, tools to cater to every kind of implementation/platform may not be readily available.

White Box Testing is contrasted with Black Box Testing. Read the Differences between Black Box Testing and White Box Testing.

Black Box Testing

DEFINITION

Black Box Testing, also known as Behavioral Testing, is a software testing method in which the internal structure/ design/ implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.



This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see. This method attempts to find errors in the following categories:

- Incorrect or missing functions
- Interface errors
- Errors in data structures or external database access
- Behavior or performance errors
- Initialization and termination errors

Definition by ISTQB

- **black box testing:** Testing, either functional or non-functional, without reference to the internal structure of the component or system.
- **black box test design technique:** Procedure to derive and/or select test cases based on an analysis of the specification, either functional or non-functional, of a component or system without reference to its internal structure.

EXAMPLE

A tester, without knowledge of the internal structures of a website, tests the web pages by using a browser; providing inputs (clicks, keystrokes) and verifying the outputs against the expected outcome.

LEVELS APPLICABLE TO

Black Box Testing method is applicable to the following levels of software testing:

- Integration Testing
- System Testing
- Acceptance Testing

The higher the level, and hence the bigger and more complex the box, the more black box testing method comes into use.

BLACK BOX TESTING TECHNIQUES

Following are some techniques that can be used for designing black box tests.

- *Equivalence partitioning:* It is a software test design technique that involves dividing input values into valid and invalid partitions and selecting representative values from each partition as test data.
- *Boundary Value Analysis:* It is a software test design technique that involves determination of boundaries for input values and selecting values that are at the boundaries and just inside/ outside of the boundaries as test data.
- *Cause Effect Graphing:* It is a software test design technique that involves identifying the cases (input conditions) and effects (output conditions), producing a Cause-Effect Graph, and generating test cases accordingly.

BLACK BOX TESTING ADVANTAGES

- Tests are done from a user's point of view and will help in exposing discrepancies in the specifications.
- Tester need not know programming languages or how the software has been implemented.
- Tests can be conducted by a body independent from the developers, allowing for an objective perspective and the avoidance of developer-bias.
- Test cases can be designed as soon as the specifications are complete.

BLACK BOX TESTING DISADVANTAGES

- Only a small number of possible inputs can be tested and many program paths will be left untested.
- Without clear specifications, which is the situation in many projects, test cases will be difficult to design.

- Tests can be redundant if the software designer/ developer has already run a test case.
- Ever wondered why a soothsayer closes the eyes when foretelling events? So is almost the case in Black Box Testing.

7) Explain how unit testing is performed for software. (7)

Ans: Unit testing focuses verification effort on the smallest unit of software design—the software component or module. Using the component-level design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and uncovered errors is limited by the constrained scope established for unit testing. The unit test is white-box oriented, and the step can be conducted in parallel for multiple components.

Unit Test Considerations

The tests that occur as part of unit tests are illustrated schematically in Figure. The module interface is tested to ensure that information properly flows into and out of the program unit under test. The local data structure is examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution.

Boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing. All independent paths (basis paths) through the control structure are exercised to ensure that all statements in a module have been executed at least once. And finally, all error handling paths are tested.

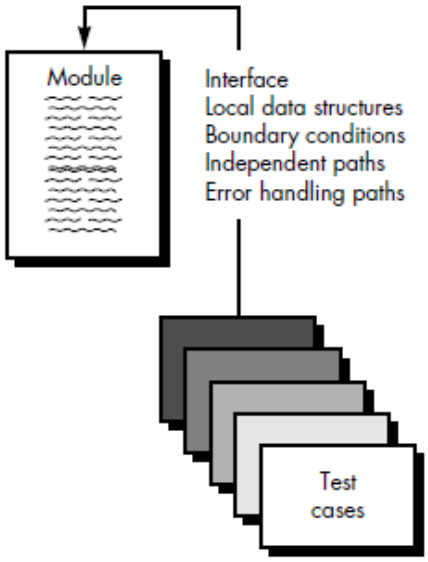


Fig. Unit Test

Tests of data flow across a module interface are required before any other test is initiated. If data do not enter and exit properly, all other tests are moot. In addition, local data structures should be exercised and the local impact on global data should be ascertained (if possible) during unit testing.

Selective testing of execution paths is an essential task during the unit test. Test cases should be designed to uncover errors due to erroneous computations, incorrect comparisons, or improper control flow. Basis path and loop testing are effective techniques for uncovering a broad array of path errors.

Among the more common errors in computation are

(1) misunderstood or incorrect arithmetic precedence, (2) mixed mode operations, (3) incorrect initialization,(4) precision inaccuracy, (5) incorrect symbolic representation of an expression.

Comparison and control flow are closely coupled to one another (i.e., change of flow frequently occurs after a comparison).

Test cases should uncover errors such as
(1) comparison of different data types, (2) incorrect logical operators or precedence,
(3) expectation of equality when precision error makes equality unlikely, (4) incorrect comparison of variables, (5) improper or nonexistent loop termination, (6) failure to exit when divergent iteration is encountered, and (7) improperly modified loop variables.
Good design dictates that error conditions be anticipated and error-handling paths set up to reroute or cleanly terminate processing when an error does occur.

8) What is difference between software testing and debugging. Explain debugging process in detail. (7)

Ans:

Testing	Debugging
1. Testing always starts with known conditions, uses predefined methods, and has predictable outcomes too.	1. Debugging starts from possibly un-known initial conditions and its end cannot be predicted, apart from statistically.
2. Testing can and should definitely be planned, designed, and scheduled.	2. The procedures for, and period of, debugging cannot be so constrained.
3. It proves a programmers failure.	3. It is the programmer's vindication.
4. It is a demonstration of error or apparent correctness.	4. It is always treated as a deductive process.
5. Testing as executed should strive to be predictable, dull, constrained, rigid, and inhuman.	5. Debugging demands intuitive leaps, conjectures, experimentation, and some freedom also.
6. Much of the testing can be done without design knowledge.	6. Debugging is impossible without detailed design knowledge.
7. It can often be done by an outsider.	7. It must be done by an insider.
8. Much of test execution and design can be automated.	8. Automated debugging is still a dream for programmers.
9. Testing purpose is to find bug.	9. Debugging purpose is to find cause of bug.

Debugging is considered to be a complex and time-consuming process since it attempts to remove errors at all the levels of testing. To perform debugging, debugger (debugging tool) is used to reproduce the conditions in which failure occurred, examine the program state, and locate the cause. With the help of debugger, programmers trace the program execution step by step (evaluating the value of variables) and halt the execution wherever required to reset the program variables. Note that some programming language packages include a debugger for checking the code for errors while it is being written.

The Debugging Process

During debugging, errors are encountered that range from less damaging (like input of an incorrect function) to catastrophic (like system failure, which lead to economic or physical damage). Note that with the increase in number of errors, the amount of effort to find their causes also increases.

Once errors are identified in a software system, to debug the problem, a number of steps are followed, which are listed below.

1. **Defect confirmation/identification:** A problem is identified in a system and a defect report is created. A software engineer maintains and analyzes this error report and finds solutions to the following questions.
    1. Does a .defect exist in the system?
    2. Can the defect be reproduced?
    3. What is the expected/desired behavior of the system?
    4. What is the actual behavior?
2. **Defect analysis:** If the defect is genuine, the next step is to understand the root cause of the problem. Generally, engineers debug by starting a debugging tool (debugger) and they try to understand the root cause of the problem by following a step-by-step execution of the program.
3. **Defect resolution:** Once the root cause of a problem is identified, the error can be resolved by making an appropriate change to the system by fixing the problem.

When the debugging process ends, the software is retested to ensure that no errors are left undetected. Moreover, it checks that no new errors are introduced in the software while making some changes to it during the debugging process.

Unit 5

1) Explain McCall's Quality factors. (7)

Ans: 1.Correctness:

- A software system is expected to meets the explicitly specified functional requirements and the implicitly expected non-functional requirements.
- If a software system satisfies all the functional requirements, the system is said to be correct.

2.Reliability

- Customers may still consider an incorrect system to be reliable if the failure rate is very small and it does not adversely affect their mission objectives.
- Reliability is a customer perception, and an incorrect software can still be considered to be reliable.

3.Efficiency:

- Efficiency concerns to what extent a software system utilizes resources, such as computing power, memory, disk space, communication bandwidth, and energy.
- A software system must utilize as little resources as possible to perform its functionalities.

4.Integrity:

- A system’s integrity refers to its ability to withstand attacks to its security.
- In other words, integrity refers to the extent to which access to software or data by unauthorized persons or programs can be controlled.

5.Usability:

- A software is considered to be usable if human users find it easy to use.
- Without a good user interface a software system may fizzle out even if it possesses many desired qualities.

6.Maintainability:

- Maintenance refers to the upkeep of products in response to deterioration of their components due to continuous use of the products.
- Maintenance refers to how easily and inexpensively the maintenance tasks can be performed.
- For software products, there are three categories of maintenance activities : corrective, adaptive and perfective maintenance.

7.Testability:

- Testability means the ability to verify requirements. At every stage of software development, it is necessary to consider the testability aspect of a product.
- To make a product testable, designers may have to instrument a design with functionalities not available to the customer.

8.Flexibility:

- Flexibility is reflected in the cost of modifying an operational system.
- In order to measure the flexibility of a system, one has to find an answer to the question: How easily can one add a new feature to a system.

9.Portability

- Portability of a software system refers to how easily it can be adapted to run in a different execution environment.
- Portability gives customers an option to easily move from one execution environment to another to best utilize emerging technologies in furthering their business.

10.Reusability

- Reusability means if a significant portion of one product can be reused, maybe with minor modifications, in another product.
- Reusability saves the cost and time to develop and test the component being reused.

11.Interoperability :

- Interoperability means whether or not the output of one system is acceptable as input to another system, it is likely that the two systems run on different computers interconnected by a network.
- An example of interoperability is the ability to roam from one cellular phone network in one country to another cellular network in another country.

2) Explain Quality function deployment. **(6)**

Ans: Quality function deployment (QFD) is the translation of user requirements and requests into product designs. The goal of QFD is to build a product that does exactly what the customer wants instead of delivering a product that emphasizes expertise the builder already has
In QFD, quality is a measure of customer satisfaction with a product or a service. QFD is a structured method that uses the seven management and planning tools to identify and prioritize customers' expectations quickly and effectively.
Beginning with the initial matrix, commonly termed the **house of quality**, depicted in Figure 1, the QFD methodology focuses on the most important product or service attributes or qualities. These are composed of customer *wows, wants,* and *musts.*

Once you have prioritized the attributes and qualities, QFD deploys them to the appropriate organizational function for action, as shown in Figure 2. Thus, QFD is the deployment of customer-driven qualities to the responsible functions of an organization.
Many QFD practitioners claim that using QFD has enabled them to reduce their product and service development cycle times by as much as 75 percent with equally impressive improvements in measured customer satisfaction.

QFD use: Measures, sometimes considered in connection with their direction of improvement, are evaluated in each cell in terms of "how important would that response to this (row's) requirement be?" This can help a software development team in a number of ways:

- It checks a team's shared understanding about what the requirements and measures really mean. This may sound simple, but the act of having a short discussion about the ways in which each prospective response could relate to each requirement uncovers differences of opinion and perspective across the team. Better for a team to struggle at this early stage to reach some common view of what's meant by each requirement and measure.
- While the requirements should have already been characterized (e.g., Kano classifications) and prioritized before a QFD, the prioritization of measures during this kind of QFD can focus work on solution generation (stimulating extra creativity on the solution aspects connected with the most important measures).
- Prioritized measures suggest focus for measurement systems analysis and test. The most important measures should be done most carefully – using a measurement system that's been most rigorously built and checked.

3) List and explain activities involved in Software Quality Assurance. (7)

Ans: SQA is the process of evaluating the quality of a product and enforcing adherence to software product standards and procedures. It is an umbrella activity that ensures conformance to standards and procedures throughout the SDLC of a software product. There are a large number of tasks involved in SQA activities.
These include:
i. Formulating a quality management plan
ii. Applying software engineering techniques
iii. Conducting formal technical reviews
iv. Applying a multi-tiered testing strategy
v. Enforcing process adherence
vi. Controlling change
vii. Measuring impact of change
viii. Performing SQA audits
ix. Keeping records and reporting

i. Formulating a Quality Management Plan

One of the tasks of SQA is the formulation of a quality management plan. The quality management plan identifies the quality aspects of the software product to be developed. It helps in planning checkpoints for work products and the development process. It also tracks changes made to the development process based on the results of the checks. The quality management plan is tracked as a live plan throughout the SDLC.

ii. Applying Software Engineering

Application of software engineering techniques helps the software designer to achieve high quality specification. The designer gathers information using techniques such as interviews and FAST. Using the information gathered, the designer prepares project estimation with the help of techniques such as WBS, SLOC estimation, or FP estimation.

iii. Conducting Formal Technical Reviews

Formal technical review (FTR) in conducted to assess the quality and design of the prototype. It is a meeting with the technical staff to discuss the quality requirements of a software product and its design quality. FTRs help in detecting errors at an early phase of development. This prevents errors from percolating down to the latter phases and resulting in rework.

iv. Applying a Multi-tiered Testing Strategy

Software testing is a critical task of SQA activity, which aims at error detection. Unit testing is the first level of testing. The subsequence levels of testing are integration testing and system level testing. There are various testing strategies followed by organization. At times, developers perform unit testing and integration testing with independence testing support. There are also occasions where testers perform functional testing and system level testing with developer support. Sometimes beta testing with selected clients is also conducted to test the product before it is finally released.

v. Enforcing Process Adherence

This task of SQA emphasizes the need for process adherence during product development. In addition, the development process should also adhere to procedures defined for product development. Therefore, this is a combination of two tasks, product evaluation and process monitoring

vi. Product Evaluation

Product evaluation ensures that the standards laid down for a project are followed. During product evaluation, the compliance of the software product to the existing standards is verified. Initially, SQA activities are conducted to monitor the standards and procedures of the project. Product evaluation ensures that the software product reflects the requirements identified in the project management plan.

vii. Process Monitoring

Process monitoring ensures that appropriate steps to follow the product development procedures are carried out. SQA monitors processes by comparing the actual steps carried out with the steps in the documented procedures. Product evaluation and process monitoring ensure that the development and control processes described in the project management plan are correctly carried out. These tasks ensure that the project-re1ated procedures and standards are followed. They also ensure that products and processes conform to standards and procedures. Audits ensure that product evaluation and process monitoring are performed.

viii. Controlling Change

This task combines human procedures and automated tools to provide a mechanism for change control. The change control mechanism ensures software quality by formalizing requests for change, evaluating the nature of change, and controlling the impact of change. Change control mechanism is implemented during the development and maintenance stages

ix. Measuring Impact of Change

Change is inevitable in the SDLC. However, the change needs to be measured and monitored. Changes in the product or process are measured using software quality metrics. Software qua1ity metrics helps in estimating the cost and resource requirements of a project. To control software quality; it is essential to measure quality and then compare it with established standards. Software qua1ity metrics are used to evaluate the effectiveness of techniques and tools, the productivity of development activities and the qua1ity of products. Metrics enables mangers and developers to monitor the activities and proposed changes throughout the SDLC and initiate corrective actions.

x. Performing SQA Audits

SQA audits scrutinize the software development process by comparing it to

established processes. This ensures that proper control is maintained over the documents required during SDLC. Audits also ensure that the status of an activity performed by the developer is reflected in the status report of the developer.

xi. Keeping Records and Reporting

Keeping records and reporting ensure the collection and circulation of information relevant to SQA. The results of reviews, audits, change control, testing, and other SQA activities are reported and compiled for future reference.

4**) Explain in detail about project tracking and project scheduling.**

Ans: .: Project Scheduling helps to establish a roadmap for project managers together with estimation methods and risk analysis. Project scheduling and Tracking begins with the identification of process models, identification of software tasks and activities, estimation of effort and work and ends with creation of network of tasks and making sure it gets done on time. This network is adapted on encountering of changes and risks.

At the project level, the Project Manager does project tracking and scheduling based on information received from Software Engineers. At an individual level the Software Engineer does it. It is important, as in a complex system many tasks may occur in parallel and have interdependencies that are understandable only with a schedule. A detailed schedule is a useful tool to assess progress on a moderate or large project.

The basic steps followed are, once the tasks dictated by the software process model is refined based on the functionality of the system , effort and duration are allocated for each task and an activity network is created that allows the project to meets its deadlines. The work product of this activity is the project schedule and in order that it is accurate it is required to check all tasks are covered in the activity network, effort and timing are appropriately allocated, interdependencies are correctly indicated, resources are allocated tasks in a right manner and closely spaced milestones are defined to track the project easily.

One of the major challenges in software project management is the difficulty to adhere to schedules. The common reasons for a late delivery of software project are an unrealistic deadline, changing customer requirements, honest underestimate of effort or resources, overlooked risks, unforeseen technical difficulties or human difficulties, miscommunication and failure by project manager to recognize the delay early and take appropriate measures.

Software project scheduling is an activity that distributes estimated effort across the duration of project cycle by allocating effort to each specific task that is associated with all process. The basic principles that guides software project scheduling is compartmentalization of the project into a number of manageable tasks, correct allocation of time, correct effort validation ,defining responsibility for each task to a team member, defining outcomes or work product for each task and defining milestones for a task or group of tasks as appropriate.

A Task set is a collection of software tasks, milestones and deliveries that must be completed for the project to be successfully accomplished. Task sets are defined for being applicable to different type of project and degree of rigor. The types of projects commonly encountered are Concept Development projects, New applications, Development projects, Application enhancement projects, Application maintenance projects and Re-engineering projects. The degree of rigor with which the software process is applied may be casual, structured, strict or quick reaction (used for emergency situation). For the project manager to develop a systematic approach for selecting degree of rigor for the type of project project adaptation criteria are defined and a task set selector value is computed based on the characteristics of the project.

Program evaluation and review technique (PERT) and critical path method (CPM) are two of the commonly used project scheduling methods. These techniques are driven by information such as

estimates of effort, decomposition of the product function, the selection of process model, task set and decomposition of tasks. The interdependencies among tasks are defined using a task network. A task network or activity network is a graphic representation of the task flows for a project. According to basic PERT, expected task duration is calculated as the weighted average of the most pessimistic, the most optimistic and most probable time estimates. The expected duration of any path on the network is found by summing the expected durations of tasks. PERT gives appropriate results when there is a single dominant path in the network. The time needed to complete the project is defined by the longest path in the network which is called critical path. CPM allows software planner to determine the critical path and establish most likely time estimates.

While creating schedule a timeline chart also called as Gantt chart can be generated. This can be developed for entire project or separately for each function or individual.

The information necessary for generation of this is Work Breakdown Structure (WBS – Making a complex project manageable by breaking it into individual components in a hierarchical structure that defines independent tasks), effort, duration and start date and details of assignment of tasks to resources. Along with this most software project scheduling tools produce project tables which is a tabular listing o project tasks, their planned and actual start, end dates and other information. This with timeline chart is valuable to project managers to track the progress.

Tracking the project schedule can be done by conducting periodic project status meeting, evaluating result of reviews conducted at all stages of development life cycle, determining the completion of defined project milestones, comparison of actual and planned dates, using earned value analysis technique for performing quantitative analysis of program.

Error tracking methods can also be used for assessing the status of current project. This is done by collecting error related measures and metrics from past project and using this as baseline for comparison against real time data.

To summarize, in software projects it is crucial to control the actual project schedule against the project plan by keeping in mind the outlined techniques and steps so that variations may not have a significant impact on the project delivery.

5) Explain process metric in detail (6)

Ans: Process Metrics

- Private process metrics (e.g. defect rates by individual or module) are only known to by the individual or team concerned.
- Public process metrics enable organizations to make strategic changes to improve the software process.
- Metrics should not be used to evaluate the performance of individuals.
- Statistical software process improvement helps and organization to discover where they are strong and where are week.

Statistical Process Control

1. Errors are categorized by their origin
2. Record cost to correct each error and defect
3. Count number of errors and defects in each category
4. Overall cost of errors and defects computed for each category
5. Identify category with greatest cost to organization

6. Develop plans to eliminate the most costly class of errors and defects or at least reduce their frequency

6) List and explain activities involved in Software Quality Assurance. (7)

Ans: SQA is the process of evaluating the quality of a product and enforcing adherence to software product standards and procedures. It is an umbrella activity that ensures conformance to standards and procedures throughout the SDLC of a software product. There are a large number of tasks involved in SQA activities. These include:

i. Formulating a quality management plan
ii. Applying software engineering techniques
iii. Conducting formal technical reviews
iv. Applying a multi-tiered testing strategy
v. Enforcing process adherence
vi. Controlling change
vii. Measuring impact of change
viii. Performing SQA audits
ix. Keeping records and reporting

i. Formulating a Quality Management Plan

One of the tasks of SQA is the formulation of a quality management plan. The quality management plan identifies the quality aspects of the software product to be developed. It helps in planning checkpoints for work products and the development process. It also tracks changes made to the development process based on the results of the checks. The quality management plan is tracked as a live plan throughout the SDLC.

ii. Applying Software Engineering

Application of software engineering techniques helps the software designer to achieve high quality specification. The designer gathers information using techniques such as interviews and FAST. Using the information gathered, the designer prepares project estimation with the help of techniques such as WBS, SLOC estimation, or FP estimation.

iii. Conducting Formal Technical Reviews

Formal technical review (FTR) in conducted to assess the quality and design of the prototype. It is a meeting with the technical staff to discuss the quality requirements of a software product and its design quality. FTRs help in detecting errors at an early phase of development. This prevents errors from percolating down to the latter phases and resulting in rework.

iv. Applying a Multi-tiered Testing Strategy

Software testing is a critical task of SQA activity, which aims at error detection. Unit testing is the first level of testing. The subsequence levels of testing are integration testing and system level testing. There are various testing strategies followed by organization. At times, developers perform unit testing and integration testing with independence testing support. There are also occasions where testers perform functional testing and system level testing with developer support. Sometimes beta testing with selected clients is also conducted to test the product before it is finally released.

v. Enforcing Process Adherence

This task of SQA emphasizes the need for process adherence during product development. In addition, the development process should also adhere to procedures defined for product development. Therefore, this is a combination of two tasks, product evaluation and process monitoring

vi. Product Evaluation

Product evaluation ensures that the standards laid down for a project are followed.

During product evaluation, the compliance of the software product to the existing standards is verified. Initially, SQA activities are conducted to monitor the standards and procedures of the project. Product evaluation ensures that the software product reflects the requirements identified in the project management plan.

vii. Process Monitoring

Process monitoring ensures that appropriate steps to follow the product development procedures are carried out. SQA monitors processes by comparing the actual steps carried out with the steps in the documented procedures.
Product evaluation and process monitoring ensure that the development and control processes described in the project management plan are correctly carried out. These tasks ensure that the project-re1ated procedures and standards are followed. They also ensure that products and processes conform to standards and procedures. Audits ensure that product evaluation and process monitoring are performed.

viii. Controlling Change

This task combines human procedures and automated tools to provide a mechanism for change control. The change control mechanism ensures software quality by formalizing requests for change, evaluating the nature of change, and controlling the impact of change. Change control mechanism is implemented during the development and maintenance stages

ix. Measuring Impact of Change

Change is inevitable in the SDLC. However, the change needs to be measured and monitored. Changes in the product or process are measured using software quality metrics. Software qua1ity metrics helps in estimating the cost and resource requirements of a project. To control software quality; it is essential to measure quality and then compare it with established standards. Software qua1ity metrics are used to evaluate the effectiveness of techniques and tools, the productivity of development activities and the qua1ity of products. Metrics enables mangers and developers to monitor the activities and proposed changes throughout the SDLC and initiate corrective actions.

x. Performing SQA Audits

SQA audits scrutinize the software development process by comparing it to established processes. This ensures that proper control is maintained over the documents required during SDLC. Audits also ensure that the status of an activity performed by the developer is reflected in the status report of the developer.

xi. Keeping Records and Reporting

Keeping records and reporting ensure the collection and circulation of information relevant to SQA. The results of reviews, audits, change control, testing, and other SQA activities are reported and compiled for future reference.

Unit 6

1) What is software Risk? Explain their types..

Ans: Risk is an expectation of loss, a potential problem that may or may not occur in the future. It is generally caused due to lack of information, control or time.A possibility of suffering from loss in software development process is called a software risk. Loss can be anything, increase in production cost, development of poor quality software, not being able to complete the project on time. Software risk exists because the future is uncertain and there are many known and unknown things that cannot be incorporated in the project plan. A software risk can be of two types (a) internal risks that are within the control of the project manager and (2) external risks that are beyond the control of project manager. Risk management is carried out to:

1. Identify the risk
2. Reduce the impact of risk
3. Reduce the probability or likelihood of risk
4. Risk monitoring

A project manager has to deal with risks arising from three possible cases:

1. Known knowns are software risks that are actually facts known to the team as well as to the entire project. For example not having enough number of developers can delay the project delivery. Such risks are described and included in the Project Management Plan.
2. Known unknowns are risks that the project team is aware of but it is unknown that such risk exists in the project or not. For example if the communication with the client is not of good level then it is not possible to capture the requirement properly. This is a fact known to the project team however whether the client has communicated all the information properly or not is unknown to the project.
3. Unknown Unknowns are those kind of risks about which the organization has no idea. Such risks are generally related to technology such as working with technologies or tools that you have no idea about because your client wants you to work that way suddenly exposes you to absolutely unknown unknown risks.

Software risk management is all about risk quantification of risk. This includes:

1. Giving a precise description of risk event that can occur in the project
2. Defining risk probability that would explain what are the chances for that risk to occur
3. Defining How much loss a particular risk can cause
4. Defining the liability potential of risk

Risk Management comprises of following processes:

1. Software Risk Identification
2. Software Risk Analysis
3. Software Risk Planning
4. Software Risk Monitoring

2. Write short note on any two:
i**) Software Reengineering.**
ii) Reverse Engineering.
iii) RMMM Plan.

Ans:
It is a set of activity designed to manage change by identifying the work products that are likely to change reestablishing relationships among them; defining mechanisms for managing different version of these work products controlling the changes imposed, and auditing and reporting on the changes made ™When you build computer software change happens and because it happens you need to manage it effectively. ™Change management more commonly called software configuration management(SCM) It's very easy for a stream of uncontrolled changes to turn as well run software project into chaos. For that reason change management is an essential part of good project management and solid software engineering practice.

- Also called software configuration management (SCM)
- It is an umbrella activity that is applied throughout the software process
- It's goal is to maximize productivity by minimizing mistakes caused by confusion when coordinating software development
- SCM identifies, organizes, and controls modifications to the software being built by a software development team

- SCM activities are formulated to identify change, control change, ensure that change is being properly implemented, and report changes to others who may have an interest
- SCM is initiated when the project begins and terminates when the software is taken out of operation
- View of SCM from various roles
    - Project manager -> an auditing mechanism
    - SCM manager -> a controlling, tracking, and policy making mechanism
    - Software engineer -> a changing, building, and access control mechanism
    - Customer -> a quality assurance and product identification mechanism
- The Output from the software process makes up the software configuration
    - Computer programs (both source code files and executable files)
    - Work products that describe the computer programs (documents targeted at both technical practitioners and users)
    - Data (contained within the programs themselves or in external files)
- The major danger to a software configuration is change
    - First Law of System Engineering: "No matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle"
- Problems with paper-based repositories (i.e., file cabinet containing folders)
    - Finding a configuration item when it was needed was often difficult
    - Determining which items were changed, when and by whom was often challenging
    - Constructing a new version of an existing program was time consuming and error prone
    - Describing detailed or complex relationships between configuration items was virtually impossible
- Today's automated SCM repository
    - It is a set of mechanisms and data structures that allow a software team to manage change in an effective manner
    - It acts as the center for both accumulation and storage of software engineering information
    - Software engineers use tools integrated with the repository to interact with it

(ii) Reverse engineering : Reverse engineering is taking apart an object to see how it works in order to duplicate or enhance the object. The practice, taken from older industries, is now frequently used on computer hardware and software. Software reverse engineering involves reversing a program's machine code (the string of 0s and 1s that are sent to the logic processor) back into the source code that it was written in, using program language statements.

Software reverse engineering is done to retrieve the source code of a program because the source code was lost, to study how the program performs certain operations, to improve the performance of a program, to fix a bug (correct an error in the program when the source code is not available), to identify malicious content in a program such as a virus or to adapt a program written for use with one microprocessor for use with another. Reverse engineering for the purpose of copying or duplicating programs may constitute a copyright violation. In some cases, the licensed use of software specifically prohibits reverse engineering.

Someone doing reverse engineering on software may use several tools to disassemble a program. One tool is a hexadecimal dumper, which prints or displays the binary numbers of a program in hexadecimal format (which is easier to read than a binary format). By knowing the bit patterns that represent the processor instructions as well as the instruction lengths, the reverse engineer can identify certain portions of a program to see how they work. Another common tool is the disassembler. The disassembler reads the binary code and then displays each executable instruction in text form. A disassembler cannot tell the difference between an executable instruction and the data used by the program so a debugger is used, which allows the disassembler to avoid disassembling the data portions of a program. These tools might be used by a cracker to modify code and gain entry to a computer system or cause other harm.

3) Explain Software Configuration management. How it is useful for managing the maturity of a company to develop software according to CMM-Model? (7)

Ans: Software Configuration Management(CMM and CMMI)

CMM: Capability Maturity Model

• Developed by Software Engineering Institute (SEI)

• "the quality of a system or product is highly influenced by the quality of the process used to develop and maintain it" CMMI : Capability Maturity Model Integration

• Result of the evolution of three source models:

1. The Capability Maturity Model for Software (SW-CMM)
2. The Systems Engineering Capability Model (SECM)
3. The Integrated Product Development Capability Maturity Model (IPD-CMM)

WHY CMM?

"Process improvement maturity model for the development of products and services. It consists of best practices that address development and maintenance activities that cover the product lifecycle from conception through delivery and maintenance." Provides a means for measuring an organization on process maturity

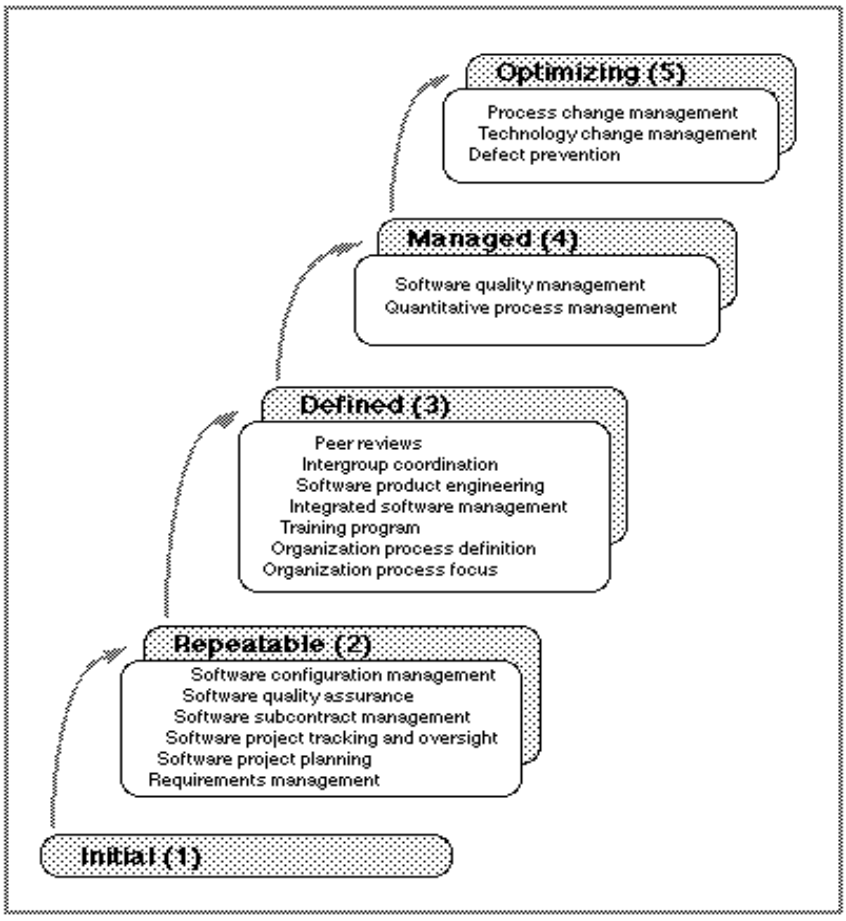


Fig. Maturity level and key process area

Figure lists the key process areas for each maturity level in the CMM. Each *key process area identifies a cluster of related activities that, when performed collectively, achieve a set of goals considered important for enhancing process capability. The key process areas have been defined to reside at a single maturity level. The key process areas are building blocks that indicate the areas an organization should focus on to improve its software process. Key process areas identify the issues that must be addressed to achieve a maturity level.*

The key process areas at Level 2 focus on the software project's concerns related to establishing basic project management controls. Descriptions of each of the key process areas for Level 2 are given below:

- The purpose of Requirements Management is to establish a common understanding between the customer and the software project of the customer's requirements that will be addressed by the software project. This agreement with the customer is the basis for planning (as described in Software Project Planning) and managing (as described in Software Project Tracking and Oversight) the software project. Control of the relationship with the customer depends on following an effective change control process (as described in Software Configuration Management).
- The purpose of Software Project Planning is to establish reasonable plans for performing the software engineering and for managing the software project. These plans are the necessary foundation for managing the software project (as described in Software Project Tracking and Oversight). Without realistic plans, effective project management cannot be implemented.
- The purpose of Software Project Tracking and Oversight is to establish adequate visibility into actual progress so that management can take effective actions when the software project's performance deviates significantly from the software plans.
- The purpose of Software Subcontract Management is to select qualified software subcontractors and manage them effectively. It combines the concerns of Requirements Management, Software Project Planning, and Software Project Tracking and Oversight for basic management control, along with necessary coordination of Software Quality Assurance and Software Configuration Management, and applies this control to the subcontractor as appropriate.
- The purpose of Software Quality Assurance is to provide management with appropriate visibility into the process being used by the software project and of the products being built. Software Quality Assurance is an integral part of most software engineering and management processes.
- The purpose of Software Configuration Management is to establish and maintain the integrity of the products of the software project throughout the project's software life cycle. Software Configuration Management is an integral part of most software engineering and management processes.

4) Explain in detail about project tracking and project scheduling. (6)

Ans.: Project Scheduling helps to establish a roadmap for project managers together with estimation methods and risk analysis. Project scheduling and Tracking begins with the identification of process models, identification of software tasks and activities, estimation of effort and work and ends with creation of network of tasks and making sure it gets done on time. This network is adapted on encountering of changes and risks.

At the project level, the Project Manager does project tracking and scheduling based on information received from Software Engineers. At an individual level the Software Engineer does it. It is important, as in a complex system many tasks may occur in parallel and have interdependencies that are understandable only with a schedule. A detailed schedule is a useful tool to assess progress on a moderate or large project.

The basic steps followed are, once the tasks dictated by the software process model is refined based on the functionality of the system , effort and duration are allocated for each task and an activity network is created that allows the project to meets its deadlines. The work product of this activity is the project schedule and in order that it is accurate it is required to check all tasks are covered in the activity network, effort and timing are appropriately allocated, interdependencies are correctly indicated, resources are allocated tasks in a right manner and closely spaced milestones are defined to track the project easily.

One of the major challenges in software project management is the difficulty to adhere to schedules. The common reasons for a late delivery of software project are an unrealistic deadline, changing customer requirements, honest underestimate of effort or resources, overlooked risks, unforeseen

technical difficulties or human difficulties, miscommunication and failure by project manager to recognize the delay early and take appropriate measures.

Software project scheduling is an activity that distributes estimated effort across the duration of project cycle by allocating effort to each specific task that is associated with all process. The basic principles that guides software project scheduling is compartmentalization of the project into a number of manageable tasks, correct allocation of time, correct effort validation ,defining responsibility for each task to a team member, defining outcomes or work product for each task and defining milestones for a task or group of tasks as appropriate.

A Task set is a collection of software tasks, milestones and deliveries that must be completed for the project to be successfully accomplished. Task sets are defined for being applicable to different type of project and degree of rigor. The types of projects commonly encountered are Concept Development projects, New applications, Development projects, Application enhancement projects, Application maintenance projects and Re-engineering projects. The degree of rigor with which the software process is applied may be casual, structured, strict or quick reaction (used for emergency situation). For the project manager to develop a systematic approach for selecting degree of rigor for the type of project project adaptation criteria are defined and a task set selector value is computed based on the characteristics of the project.

Program evaluation and review technique (PERT) and critical path method (CPM) are two of the commonly used project scheduling methods. These techniques are driven by information such as estimates of effort, decomposition of the product function, the selection of process model, task set and decomposition of tasks. The interdependencies among tasks are defined using a task network. A task network or activity network is a graphic representation of the task flows for a project. According to basic PERT, expected task duration is calculated as the weighted average of the most pessimistic, the most optimistic and most probable time estimates. The expected duration of any path on the network is found by summing the expected durations of tasks. PERT gives appropriate results when there is a single dominant path in the network. The time needed to complete the project is defined by the longest path in the network which is called critical path. CPM allows software planner to determine the critical path and establish most likely time estimates.

While creating schedule a timeline chart also called as Gantt chart can be generated. This can be developed for entire project or separately for each function or individual.

The information necessary for generation of this is Work Breakdown Structure (WBS – Making a complex project manageable by breaking it into individual components in a hierarchical structure that defines independent tasks), effort, duration and start date and details of assignment of tasks to resources. Along with this most software project scheduling tools produce project tables which is a tabular listing o project tasks, their planned and actual start, end dates and other information. This with timeline chart is valuable to project managers to track the progress.

Tracking the project schedule can be done by conducting periodic project status meeting, evaluating result of reviews conducted at all stages of development life cycle, determining the completion of defined project milestones, comparison of actual and planned dates, using earned value analysis technique for performing quantitative analysis of program.

Error tracking methods can also be used for assessing the status of current project. This is done by collecting error related measures and metrics from past project and using this as baseline for comparison against real time data.

To summarize, in software projects it is crucial to control the actual project schedule against the project plan by keeping in mind the outlined techniques and steps so that variations may not have a significant impact on the project delivery.

5)What is Quality Function Deployment (QFD)? Explain its importance for software development process. (7)

Ans: Quality function deployment (QFD) is the translation of user requirements and requests into product designs. The goal of QFD is to build a product that does exactly what the customer wants instead of delivering a product that emphasizes expertise the builder already has

In QFD, quality is a measure of customer satisfaction with a product or a service. QFD is a structured method that uses the seven management and planning tools to identify and prioritize customers' expectations quickly and effectively.

Beginning with the initial matrix, commonly termed the **house of quality**, depicted in Figure 1, the QFD methodology focuses on the most important product or service attributes or qualities. These are composed of customer *wows, wants,* and *musts.*

Once you have prioritized the attributes and qualities, QFD deploys them to the appropriate organizational function for action, as shown in Figure 2. Thus, QFD is the deployment of customer-driven qualities to the responsible functions of an organization.

Many QFD practitioners claim that using QFD has enabled them to reduce their product and service development cycle times by as much as 75 percent with equally impressive improvements in measured customer satisfaction.

QFD use: Measures, sometimes considered in connection with their direction of improvement, are evaluated in each cell in terms of "how important would that response to this (row's) requirement be?" This can help a software development team in a number of ways:

- It checks a team's shared understanding about what the requirements and measures really mean. This may sound simple, but the act of having a short discussion about the ways in which each prospective response could relate to each requirement uncovers differences of opinion and perspective across the team. Better for a team to struggle at this early stage to reach some common view of what's meant by each requirement and measure.
- While the requirements should have already been characterized (e.g., Kano classifications) and prioritized before a QFD, the prioritization of measures during this kind of QFD can focus work on solution generation (stimulating extra creativity on the solution aspects connected with the most important measures).
- Prioritized measures suggest focus for measurement systems analysis and test. The most important measures should be done most carefully – using a measurement system that's been most rigorously built and checked.

6) Write a short note on (i) Change Management (ii) Reverse Engineering (8)

Ans:

It is a set of activity designed to manage change by identifying the work products that are likely to change reestablishing relationships among them; defining mechanisms for managing different version of these work products controlling the changes imposed, and auditing and reporting on the changes made ™When you build computer software change happens and because it happens you need to manage it effectively. ™Change management more commonly called software configuration management(SCM) It's very easy for a stream of uncontrolled changes to turn as well run software project into chaos. For that reason change management is an essential part of good project management and solid software engineering practice.

- Also called software configuration management (SCM)
- It is an umbrella activity that is applied throughout the software process
- It's goal is to maximize productivity by minimizing mistakes caused by confusion when coordinating software development
- SCM identifies, organizes, and controls modifications to the software being built by a software development team
- SCM activities are formulated to identify change, control change, ensure that change is being properly implemented, and report changes to others who may have an interest
- SCM is initiated when the project begins and terminates when the software is taken out of operation

- View of SCM from various roles
  - Project manager -> an auditing mechanism
  - SCM manager -> a controlling, tracking, and policy making mechanism
  - Software engineer -> a changing, building, and access control mechanism
  - Customer -> a quality assurance and product identification mechanism
- The Output from the software process makes up the software configuration
  - Computer programs (both source code files and executable files)
  - Work products that describe the computer programs (documents targeted at both technical practitioners and users)
  - Data (contained within the programs themselves or in external files)
- The major danger to a software configuration is change
  - First Law of System Engineering: "No matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle"
- Problems with paper-based repositories (i.e., file cabinet containing folders)
  - Finding a configuration item when it was needed was often difficult
  - Determining which items were changed, when and by whom was often challenging
  - Constructing a new version of an existing program was time consuming and error prone
  - Describing detailed or complex relationships between configuration items was virtually impossible
- Today's automated SCM repository
  - It is a set of mechanisms and data structures that allow a software team to manage change in an effective manner
  - It acts as the center for both accumulation and storage of software engineering information
  - Software engineers use tools integrated with the repository to interact with it

(ii) Reverse engineering : Reverse engineering is taking apart an object to see how it works in order to duplicate or enhance the object. The practice, taken from older industries, is now frequently used on computer hardware and software. Software reverse engineering involves reversing a program's machine code (the string of 0s and 1s that are sent to the logic processor) back into the source code that it was written in, using program language statements.

Software reverse engineering is done to retrieve the source code of a program because the source code was lost, to study how the program performs certain operations, to improve the performance of a program, to fix a bug (correct an error in the program when the source code is not available), to identify malicious content in a program such as a virus or to adapt a program written for use with one microprocessor for use with another. Reverse engineering for the purpose of copying or duplicating programs may constitute a copyright violation. In some cases, the licensed use of software specifically prohibits reverse engineering.

Someone doing reverse engineering on software may use several tools to disassemble a program. One tool is a hexadecimal dumper, which prints or displays the binary numbers of a program in hexadecimal format (which is easier to read than a binary format). By knowing the bit patterns that represent the processor instructions as well as the instruction lengths, the reverse engineer can identify certain portions of a program to see how they work. Another common tool is the disassembler. The disassembler reads the binary code and then displays each executable instruction in text form. A disassembler cannot tell the difference between an executable instruction and the data used by the program so a debugger is used, which allows the disassembler to avoid disassembling the data portions of a program. These tools might be used by a cracker to modify code and gain entry to a computer system or cause other harm.

7) How we perform project scheduling in software engineering? Explain. (5)

Ans:

General Practices

On large projects, hundreds of small tasks must occur to accomplish a larger goal

  - Some of these tasks lie outside the mainstream and may be completed without worry of impacting on the project completion date
  - Other tasks lie on the critical path; if these tasks fall behind schedule, the completion date of the entire project is put into jeopardy
- Project manager's objectives
  - Define all project tasks
  - Build an activity network that depicts their interdependencies
  - Identify the tasks that are critical within the activity network
  - Build a timeline depicting the planned and actual progress of each task
  - Track task progress to ensure that delay is recognized "one day at a time"
  - To do this, the schedule should allow progress to be monitored and the project to be controlled
- Software project scheduling distributes estimated effort across the planned project duration by allocating the effort to specific tasks
- During early stages of project planning, a macroscopic schedule is developed identifying all major process framework activities and the product functions to which they apply
- Later, each task is refined into a detailed schedule where specific software tasks are identified and scheduled
- Scheduling for projects can be viewed from two different perspectives
  - In the first view, an end-date for release of a computer-based system has already been established and fixed
    - The software organization is constrained to distribute effort within the prescribed time frame
  - In the second view, assume that rough chronological bounds have been discussed but that the end-date is set by the software engineering organization
    - Effort is distributed to make best use of resources and an end-date is defined after careful analysis of the software
  - The first view is encountered far more often that the second

Basic Principles for Project Scheduling

- Compartmentalization
  - The project must be compartmentalized into a number of manageable activities, actions, and tasks; both the product and the process are decomposed
- Interdependency
  - The interdependency of each compartmentalized activity, action, or task must be determined
  - Some tasks must occur in sequence while others can occur in parallel
  - Some actions or activities cannot commence until the work product produced by another is available
- Time allocation
  - Each task to be scheduled must be allocated some number of work units
  - In addition, each task must be assigned a start date and a completion date that are a function of the interdependencies
  - Start and stop dates are also established based on whether work will be conducted on a full-time or part-time basis
- Effort validation
  - Every project has a defined number of people on the team

  - As time allocation occurs, the project manager must ensure that no more than the allocated number of people have been scheduled at any given time
- Defined responsibilities
  - Every task that is scheduled should be assigned to a specific team member
- Defined outcomes
  - Every task that is scheduled should have a defined outcome for software projects such as a work product or part of a work product
  - Work products are often combined in deliverables
- Defined milestones
  - Every task or group of tasks should be associated with a project milestone
  - A milestone is accomplished when one or more work products has been reviewed for quality and has been approved