11. ASSIGNMENT TOPICS WITH MATERIALS

UNIT – I

1. Software myths

Software Management Myths. Pressman describes managers' beliefs in the following mythology as grasping at straws:

- Development problems can be solved by developing and documenting standards. Standards have been developed by companies and standards organizations. They can be very useful. However, they are frequently ignored by developers because they are irrelevant and incomplete, and sometimes incomprehensible.
- Development problems can be solved by using state-of-the-art tools. Tools may help, but there is no magic. Problem solving requires more than tools, it requires great understanding. As Fred Brooks (1987) says, there is no silver bullet to slay the software development werewolf.
- When schedules slip, just add more people. This solution seems intuitive: if there is too much work for the current team, just enlarge it. Unfortunately, increasing team size increases communication overhead. New workers must learn project details taking up the time of those who are already immersed in the project. Also, a larger team has many more communication links, which slows progress. Fred Brooks (1975) gives us one of the most famous software engineering maxims, which is not a myth, "adding people to a late project makes it later."

Software Customer Myths. Customers often vastly underestimate the difficulty of developing software. Sometimes marketing people encourage customers in their misbeliefs.

- Change is easily accommodated, since software is malleable. Software can certainly be changed, but often changes after release can require an enormous amount of labor.
- A general statement of need is sufficient to start coding. This myth reminds me of a cartoon that I used to post on my door. It showed the software manager talking to a group of programmers, with the quote: "You programmers just start coding while I go down and find out what they want the program to do." This scenario is an exaggeration. However, for developers to have a chance to satisfy the customers requirements, they need detailed descriptions of these requirements. Developers cannot read the minds of customers.

Developer Myths. Developers often want to be artists (or artisans), but the software development craft is becoming an engineering discipline. However myths remain:

- The job is done when the code is delivered. Commercially successful software may be used for decades. Developers must continually maintain such software: they add features and repair bugs. Maintenance costs predominate over all other costs; maintenance may be 70% of the development
costs. This myth is true only for shelfware — software that is never used, and there are no customers for next release of a shelfware product.

- Project success depends solely on the quality of the delivered program. Documentation and software configuration information is very important to the quality. After functionality, maintainability, see the preceding myth, is of critical importance. Developers must maintain the software and they need good design documents, test data, etc to do their job.
- You can’t assess software quality until the program is running. There are static ways to evaluate quality without running a program. Software reviews can effectively determine the quality of requirements documents, design documents, test plans, and code. Formal (mathematical) analyses are often used to verify safety critical software, software security factors, and very-high reliability software.

2. CMMI

**Material-1**

- CMM stands for Capability Maturity Model.
- Focuses on elements of essential practices and processes from various bodies of knowledge.
- Describes common sense, efficient, proven ways of doing business (which you should already be doing) - not a radical new approach.
- CMM is a method to evaluate and measure the maturity of the software development process of an organizations.
- CMM measures the maturity of the software development process on a scale of 1 to 5.
- CMM v1.0 was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in Pittsburgh, USA.
- CMM was originally developed for Software Development and Maintenance but later it was developed for:
  - Systems Engineering
  - Supplier Sourcing
  - Integrated Product and Process Development
  - People CMM
  - Software Acquisition
  - Others...
- CMM Examples:
  - People CMM: Develop, motivate and retain project talent.
  - Software CMM: Enhance a software focused development and maintenance capability.

**What is Maturity?**

Definitions vary but mature processes are generally thought to be:

- Well defined
- Repeatable
- Measured
- Analyzed

P. Praveen, Assistant Professor
Improved

And most importantly ... effective. Poor but mature processes are just as bad as no maturity at all!

The CMM helps to solve the maturity problem by defining a set of practices and providing a general framework for improving them. The CMM focus is on identifying key process areas and the exemplary practices that may comprise a disciplined software process.

Immature vs Mature Organization:

There are following characteristics of an immature organization:

- Process improvised during project
- Approved processes being ignored
- Reactive, not proactive
- Unrealistic budget and schedule
- Quality sacrificed for schedule
- No objective measure of quality

There are following characteristics of a mature organization:

- Inter-group communication and coordination
- Work accomplished according to plan
- Practices consistent with processes
- Processes updated as necessary
- Well defined roles/responsibilities
- Management formally commits

Material 2

CMM Integration project was formed to sort out the problem of using multiple CMMs. CMMI Product Team's mission was to combine three Source Models into a single improvement framework to be used by the organizations pursuing enterprise-wide process improvement. These three Source Models are:

- Capability Maturity Model for Software (SW-CMM) - v2.0 Draft C
- Electronic Industries Alliance Interim Standard (EIA/IS) - 731 Systems Engineering
- Integrated Product Development Capability Maturity Model (IPD-CMM) v0.98

CMM Integration:

- builds an initial set of integrated models.
- improves best practices from source models based on lessons learned.
- establishes a framework to enable integration of future models.

The Capability Maturity Model Integration (CMMI) is a capability maturity model developed by the Software Engineering Institute, part of Carnegie Mellon University in Pittsburgh, USA. The CMMI principal is that “the quality of a system or product is highly influenced by the process used to develop and maintain it” CMMI can be used to guide process improvement across a project, a division, or an entire organization.
CMMI provides:

- Guidelines for processes improvement
- An integrated approach to process improvement
- Embedding process improvements into a state of business as usual
- A phased approach to introducing improvements

CMMI Maturity Levels

There are five CMMI maturity levels. However, maturity level ratings are only awarded for levels 2 through 5.

**CMMI Maturity Level 2 – Managed**

- CM – Configuration Management
- MA – Measurement and Analysis
- PMC – Project Monitoring and Control
- PP – Project Planning
- PPQA – Process and Product Quality Assurance
- REQM – Requirements Management
- SAM – Supplier Agreement Management

**CMMI Maturity Level 3 – Defined**

- DAR – Decision Analysis and Resolution
- IPM – Integrated Project Management +IPPD
- OPD – Organizational Process Definition +IPPD
- OPF – Organizational Process Focus
- OT – Organizational Training
- PI – Product Integration
- RD – Requirements Development
- RSKM – Risk Management
- TS – Technical Solution
- VAL – Validation
- VER – Verification

**CMMI Maturity Level 4 – Quantitatively Managed**

- QPM – Quantitative Project Management
- OPP – Organizational Process Performance

**CMMI Maturity Level 5 – Optimizing**

- CAR – Causal Analysis and Resolution
- OID – Organizational Innovation and Deployment
3. Unified Process Model

The Unified Process is not simply a process, but rather an extensible framework which should be customized for specific organizations or projects. The Rational Unified Process is, similarly, a customizable framework. As a result, it is often impossible to say whether a refinement of the process was derived from UP or from RUP, and so the names tend to be used interchangeably.

The name Unified Process as opposed to Rational Unified Process is generally used to describe the generic process, including those elements which are common to most refinements. The Unified Process name is also used to avoid potential issues of trademark infringement since Rational Unified Process and RUP are trademarks of IBM. The first book to describe the process was titled *The Unified Software Development Process* (ISBN 0-201-57169-2) and published in 1999 by Ivar Jacobson, Grady Booch and James Rumbaugh. Since then various authors unaffiliated with Rational Software have published books and articles using the name Unified Process, whereas authors affiliated with Rational Software have favored the name Rational Unified Process.

In 2012 the Disciplined Agile Delivery framework was released, a hybrid framework that adopts and extends strategies from Unified Process, Scrum, XP, and other methods.

**Inception phase**

Inception is the smallest phase in the project, and ideally it should be quite short. If the Inception Phase is long then it may be an indication of excessive up-front specification, which is contrary to the spirit of the Unified Process.

The following are typical goals for the Inception phase:

- Establish
- Prepare a preliminary project schedule and cost estimate
- Feasibility
- Buy or develop it

The Lifecycle Objective Milestone marks the end of the Inception phase.

Develop an approximate vision of the system, make the business case, define the scope, and produce rough estimate for cost and schedule.

**Elaboration phase**

During the Elaboration phase the project team is expected to capture a healthy majority of the system requirements. However, the primary goals of Elaboration are to address known risk factors and to establish and validate the system architecture. Common processes undertaken in this phase include the creation of use case diagrams, conceptual diagrams (class diagrams with only basic notation) and package diagrams (architectural diagrams).

The architecture is validated primarily through the implementation of an Executable Architecture Baseline. This is a partial implementation of the system which includes the core
SOFTWARE ENGINEERING

most architecturally significant components. It is built in a series of small time boxed iterations. By the end of the Elaboration phase, the system architecture must have stabilized and the executable architecture baseline must demonstrate that the architecture will support the key system functionality and exhibit the right behavior in terms of performance, scalability, and cost.

The final Elaboration phase deliverable is a plan (including cost and schedule estimates) for the Construction phase. At this point the plan should be accurate and credible, since it should be based on the Elaboration phase experience and since significant risk factors should have been addressed during the Elaboration phase.

Construction phase

Construction is the largest phase in the project. In this phase the remainder of the system is built on the foundation laid in Elaboration. System features are implemented in a series of short, timeboxed iterations. Each iteration results in an executable release of the software. It is customary to write full text use cases during the construction phase and each one becomes the start of a new iteration. Common Unified Modeling Language (UML) diagrams used during this phase include activity diagrams, sequence diagrams, collaboration diagrams, State Transition diagrams and interaction overview diagrams. Iterative implementation for the lower risks and easier elements are done. The final Construction phase deliverable is software ready to be deployed in the Transition phase.

Transition phase

The final project phase is Transition. In this phase the system is deployed to the target users. Feedback received from an initial release (or initial releases) may result in further refinements to be incorporated over the course of several Transition phase iterations. The Transition phase also includes system conversions and user training.

4. The RAD Model

The RAD (Rapid Application Development) model is based on prototyping and iterative development with no specific planning involved. The process of writing the software itself involves the planning required for developing the product.

Rapid Application Development focuses on gathering customer requirements through workshops or focus groups, early testing of the prototypes by the customer using iterative
concept, reuse of the existing prototypes (components), continuous integration and rapid delivery.

What is RAD?
Rapid application development is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product.

In the RAD model, the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery. Since there is no detailed preplanning, it makes it easier to incorporate the changes within the development process.

RAD projects follow iterative and incremental model and have small teams comprising of developers, domain experts, customer representatives and other IT resources working progressively on their component or prototype.

The most important aspect for this model to be successful is to make sure that the prototypes developed are reusable.

RAD Model Design
RAD model distributes the analysis, design, build and test phases into a series of short, iterative development cycles.

Following are the various phases of the RAD Model –

Business Modeling
The business model for the product under development is designed in terms of flow of information and the distribution of information between various business channels. A complete business analysis is performed to find the vital information for business, how it can be obtained, how and when is the information processed and what are the factors driving successful flow of information.

Data Modeling
The information gathered in the Business Modeling phase is reviewed and analyzed to form sets of data objects vital for the business. The attributes of all data sets is identified and defined. The relation between these data objects are established and defined in detail in relevance to the business model.

Process Modeling
The data object sets defined in the Data Modeling phase are converted to establish the business information flow needed to achieve specific business objectives as per the
business model. The process model for any changes or enhancements to the data object sets is defined in this phase. Process descriptions for adding, deleting, retrieving or modifying a data object are given.

Application Generation
The actual system is built and coding is done by using automation tools to convert process and data models into actual prototypes.

Testing and Turnover
The overall testing time is reduced in the RAD model as the prototypes are independently tested during every iteration. However, the data flow and the interfaces between all the components need to be thoroughly tested with complete test coverage. Since most of the programming components have already been tested, it reduces the risk of any major issues.

5. Spiral model

Spiral Model - Design
The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.

Identification
This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

This phase also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral, the product is deployed in the identified market.

Design
The Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and the final design in the subsequent spirals.

Construct or Build
The Construct phase refers to production of the actual software product at every spiral. In the baseline spiral, when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to the customer for feedback.
Evaluation and Risk Analysis

Risk Analysis includes identifying, estimating and monitoring the technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

The following illustration is a representation of the Spiral Model, listing the activities in each phase.
UNIT-II

1. System Requirements

Requirement Engineering
The process to gather the software requirements from client, analyze and document them is known as requirement engineering.

The goal of requirement engineering is to develop and maintain sophisticated and descriptive ‘System Requirements Specification’ document.

Requirement Engineering Process
It is a four step process, which includes –

- Feasibility Study
- Requirement Gathering
- Software Requirement Specification
- Software Requirement Validation

Let us see the process briefly -

Feasibility study
When the client approaches the organization for getting the desired product developed, it comes up with rough idea about what all functions the software must perform and which all features are expected from the software.

Referencing to this information, the analysts does a detailed study about whether the desired system and its functionality are feasible to develop.

This feasibility study is focused towards goal of the organization. This study analyzes whether the software product can be practically materialized in terms of implementation, contribution of project to organization, cost constraints and as per values and objectives of the organization. It explores technical aspects of the project and product such as usability, maintainability, productivity and integration ability.

The output of this phase should be a feasibility study report that should contain adequate comments and recommendations for management about whether or not the project should be undertaken.

Requirement Gathering
If the feasibility report is positive towards undertaking the project, next phase starts with gathering requirements from the user. Analysts and engineers communicate with the client and end-users to know their ideas on what the software should provide and which features they want the software to include.

Software Requirement Specification
SRS is a document created by system analyst after the requirements are collected from various stakeholders.
SOFTWARE ENGINEERING

SRS defines how the intended software will interact with hardware, external interfaces, speed of operation, response time of system, portability of software across various platforms, maintainability, speed of recovery after crashing, Security, Quality, Limitations etc.

The requirements received from client are written in natural language. It is the responsibility of system analyst to document the requirements in technical language so that they can be comprehended and useful by the software development team.

SRS should come up with following features:

- User Requirements are expressed in natural language.
- Technical requirements are expressed in structured language, which is used inside the organization.
- Design description should be written in Pseudo code.
- Format of Forms and GUI screen prints.
- Conditional and mathematical notations for DFDs etc.

Software Requirement Validation

After requirement specifications are developed, the requirements mentioned in this document are validated. User might ask for illegal, impractical solution or experts may interpret the requirements incorrectly. This results in huge increase in cost if not nipped in the bud. Requirements can be checked against following conditions -

- If they can be practically implemented
- If they are valid and as per functionality and domain of software
- If there are any ambiguities
- If they are complete
- If they can be demonstrated

Requirement Elicitation Process

Requirement elicitation process can be depicted using the following diagram:

- **Requirements gathering** - The developers discuss with the client and end users and know their expectations from the software.
- **Organizing Requirements** - The developers prioritize and arrange the requirements in order of importance, urgency and convenience.
- **Negotiation & discussion** - If requirements are ambiguous or there are some conflicts in requirements of various stakeholders, if they are, it is then negotiated and discussed with stakeholders. Requirements may then be prioritized and reasonably compromised.

P.Praveen,Assistant Professor
SOFTWARE ENGINEERING

The requirements come from various stakeholders. To remove the ambiguity and conflicts, they are discussed for clarity and correctness. Unrealistic requirements are compromised reasonably.

- **Documentation** - All formal & informal, functional and non-functional requirements are documented and made available for next phase processing.

2. Function and Non-Functional requirements

**Functional requirements**

The functional requirements for a system describes what the system should do (defines a function of a system or its component). Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish. These requirements depend on the type of software being developed, the expected users of the software and the general approach taken by the organization when writing requirements. When expressed as user requirements, the requirements described in a fairly abstract way. However, functional system requirements describe the system function in detail, its inputs, expectations, behavior and outputs. Functional requirements for the software system may be expressed in a number of ways. For example, Functional requirements for a library system, used by students to order books and documents from other libraries could be following points:

- The user shall be able to search either all of the initial set of databases or select a subset from it.
- The system shall provide appropriate viewers for the user to read documents in the document store.
- Every order shall be allocated a unique identifier (ORDER_ID) which the user shall be able to copy to the account’s permanent storage area

**Non-functional requirements**

Non-functional requirements are not directly concerned with the specific functions delivered by the system. It is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. It defines system properties and constraints like, reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.

P. Praveen, Assistant Professor
SOFTWARE ENGINEERING

- It is concerned with specifying system performance, security, availability, and other emergent properties. This means they are often more critical than individual functional requirements. These requirements are not just concerned with the software system to be developed. Some non-functional requirements may constrain (restrict) the process that should be used to develop the system. Process requirements may also be specified mandating a particular CASE system, programming language or development method. Non-functional requirements may be more critical than functional requirements, so if these are not met, the system could be useless also.
Unit-III

1. Design Engineering:

Software design is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.

For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms. The output of this process can directly be used into implementation in programming languages.

Software design is the first step in SDLC (Software Design Life Cycle), which moves the concentration from problem domain to solution domain. It tries to specify how to fulfill the requirements mentioned in SRS.

Software Design Levels

Software design yields three levels of results:

- **Architectural Design** - The architectural design is the highest abstract version of the system. It identifies the software as a system with many components interacting with each other. At this level, the designers get the idea of proposed solution domain.

- **High-level Design** - The high-level design breaks the ‘single entity-multiple component’ concept of architectural design into less-abstracted view of sub-systems and modules and depicts their interaction with each other. High-level design focuses on how the system along with all of its components can be implemented in forms of modules. It recognizes modular structure of each sub-system and their relation and interaction among each other.

- **Detailed Design** - Detailed design deals with the implementation part of what is seen as a system and its sub-systems in the previous two designs. It is more detailed towards modules and their implementations. It defines logical structure of each module and their interfaces to communicate with other modules.
Modularization
Modularization is a technique to divide a software system into multiple discrete and independent modules, which are expected to be capable of carrying out task(s) independently. These modules may work as basic constructs for the entire software. Designers tend to design modules such that they can be executed and/or compiled separately and independently.

Modular design unintentionally follows the rules of ‘divide and conquer’ problem-solving strategy this is because there are many other benefits attached with the modular design of a software.

Advantage of modularization:
- Smaller components are easier to maintain
- Program can be divided based on functional aspects
- Desired level of abstraction can be brought in the program
- Components with high cohesion can be re-used again
- Concurrent execution can be made possible
- Desired from security aspect

Concurrency
Back in time, all software are meant to be executed sequentially. By sequential execution we mean that the coded instruction will be executed one after another implying only one portion of program being activated at any given time. Say, a software has multiple modules, then only one of all the modules can be found active at any time of execution.

In software design, concurrency is implemented by splitting the software into multiple independent units of execution, like modules and executing them in parallel. In other words, concurrency provides capability to the software to execute more than one part of code in parallel to each other.

It is necessary for the programmers and designers to recognize those modules, which can be made parallel execution.

Example
The spell check feature in word processor is a module of software, which runs along side the word processor itself.
Coupling and Cohesion

When a software program is modularized, its tasks are divided into several modules based on some characteristics. As we know, modules are set of instructions put together in order to achieve some tasks. They are though, considered as single entity but may refer to each other to work together. There are measures by which the quality of a design of modules and their interaction among them can be measured. These measures are called coupling and cohesion.

Cohesion

Cohesion is a measure that defines the degree of intra-dependability within elements of a module. The greater the cohesion, the better is the program design.

There are seven types of cohesion, namely –

- **Co- incidental cohesion** - It is unplanned and random cohesion, which might be the result of breaking the program into smaller modules for the sake of modularization. Because it is unplanned, it may serve confusion to the programmers and is generally not-accepted.

- **Logical cohesion** - When logically categorized elements are put together into a module, it is called logical cohesion.

- **Temporal Cohesion** - When elements of module are organized such that they are processed at a similar point in time, it is called temporal cohesion.

- **Procedural cohesion** - When elements of module are grouped together, which are executed sequentially in order to perform a task, it is called procedural cohesion.

- **Communicational cohesion** - When elements of module are grouped together, which are executed sequentially and work on same data (information), it is called communicational cohesion.

- **Sequential cohesion** - When elements of module are grouped because the output of one element serves as input to another and so on, it is called sequential cohesion.

- **Functional cohesion** - It is considered to be the highest degree of cohesion, and it is highly expected. Elements of module in functional cohesion are grouped because they all contribute to a single well-defined function. It can also be reused.
2. Coupling
Coupling is a measure that defines the level of inter-dependability among modules of a program. It tells at what level the modules interfere and interact with each other. The lower the coupling, the better the program.

There are five levels of coupling, namely -

- **Content coupling** - When a module can directly access or modify or refer to the content of another module, it is called content level coupling.

- **Common coupling** - When multiple modules have read and write access to some global data, it is called common or global coupling.

- **Control coupling** - Two modules are called control-coupled if one of them decides the function of the other module or changes its flow of execution.

- **Stamp coupling** - When multiple modules share common data structure and work on different part of it, it is called stamp coupling.

- **Data coupling** - Data coupling is when two modules interact with each other by means of passing data (as parameter). If a module passes data structure as parameter, then the receiving module should use all its components.

Ideally, no coupling is considered to be the best.

**Component Level Design:**
As soon as the first iteration of architectural design is complete, component-level design takes place. The objective of this design is to transform the design model into functional software. To achieve this objective, the component-level design represents -the internal data structures and processing details of all the software components (defined during architectural design) at an abstraction level, closer to the actual code. In addition, it specifies an interface that may be used to access the functionality of all the software components.

The component-level design can be represented by using different approaches. One approach is to use a programming language while other is to use some intermediate design notation such as graphical (DFD, flowchart, or structure chart), tabular (decision table), or text-based (program design language) whichever is easier to be translated into source code.

The component-level design provides a way to determine whether the defined algorithms, data structures, and interfaces will work properly. Note that a component (also known as...
as module) can be defined as a modular building block for the software. However, the meaning of component differs according to how software engineers use it. The modular design of the software should exhibit the following sets of properties.

1. **Provide simple interface:** Simple interfaces decrease the number of interactions. Note that the number of interactions is taken into account while determining whether the software performs the desired function. Simple interfaces also provide support for reusability of components which reduces the cost to a greater extent. It not only decreases the time involved in design, coding, and testing but the overall software development cost is also liquidated gradually with several projects. A number of studies so far have proven that the reusability of software design is the most valuable way of reducing the cost involved in software development.

2. **Ensure information hiding:** The benefits of modularity cannot be achieved merely by decomposing a program into several modules; rather each module should be designed and developed in such a way that the information hiding is ensured. It implies that the implementation details of one module should not be visible to other modules of the program. The concept of information hiding helps in reducing the cost of subsequent design changes.

Modularity has become an accepted approach in every engineering discipline. With the introduction of modular design, complexity of software design has considerably reduced; change in the program is facilitated that has encouraged parallel development of systems. To achieve effective modularity, design concepts like functional independence are considered to be very important.

3. **Functional Independence**

   Functional independence is the refined form of the design concepts of modularity, abstraction, and information hiding. Functional independence is achieved by developing a module in such a way that it uniquely performs given sets of function without interacting with other parts of the system. The software that uses the property of functional independence is easier to develop because its functions can be categorized in a systematic manner. Moreover, independent modules require less maintenance and testing activity, as secondary effects caused by design modification are limited with less propagation of errors. In short, it can be said that functional independence is the key to a good software design and a good design results in high-quality software. There exist two qualitative criteria for measuring functional independence, namely, coupling and cohesion.

P.Praveen, Assistant Professor
1. Verification and Validation:

**Verification**

1. **Verification** is a static practice of verifying documents, design, code and program.
2. It does not involve executing the code.
3. It is human based checking of documents and files.
4. **Verification** uses methods like inspections, reviews, walkthroughs, and Desk-checking etc.
5. **Verification** is to check whether the software conforms to specifications.
6. It can catch errors that validation cannot catch. It is low level exercise.
7. Target is requirements specification, application and software architecture, high level, complete design, and database design etc.
8. **Verification** is done by QA team to ensure that the software is as per the specifications in the SRS document.
9. It generally comes first-done before validation.

**Validation**

1. **Validation** is a dynamic mechanism of validating and testing the actual product.
2. It always involves executing the code.
3. It is computer based execution of program.
4. **Validation** uses methods like black box (functional) testing, gray box testing, and white box (structural) testing etc.
5. **Validation** is to check whether software meets the customer expectations and requirements.
6. *It can catch errors that verification cannot catch. It is High Level Exercise.*
7. Target is actual product-a unit, a module, a bent of integrated modules, and effective final product.
8. **Validation** is carried out with the involvement of testing team.
9. It generally follows after **verification**
2. UNIT TESTING:

Unit testing, a testing technique using which individual modules are tested to determine if there are any issues by the developer himself. It is concerned with functional correctness of the standalone modules.

The main aim is to isolate each unit of the system to identify, analyze and fix the defects.

Unit Testing - Advantages:

- Reduces Defects in the Newly developed features or reduces bugs when changing the existing functionality.
- Reduces Cost of Testing as defects are captured in very early phase.
- Improves design and allows better refactoring of code.
- Unit Tests, when integrated with build gives the quality of the build as well.

Unit Testing LifeCycle:

Unit Testing Techniques:

- **Black Box Testing** - Using which the user interface, input and output are tested.
- **White Box Testing** - used to test each one of those functions behaviour is tested.
- **Gray Box Testing** - Used to execute tests, risks and assessment methods.
Black Box Testing

**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.

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 the 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.
UNIT-V

1. **RMMM in RMMM Plan**

   The goal of the risk mitigation, monitoring and management plan is to identify as many potential risks as possible. To help determine what the potential risks are, GameForge will be evaluated using the checklists found in section 6.3 of Roger S. Pressman’s Software Engineering, A Practitioner’s Approach [Reference is the SEPA, 4/e, see risk checklists contained within this Web site]. These checklists help to identify potential risks in a generic sense. The project will then be analyzed to determine any project-specific risks. When all risks have been identified, they will then be evaluated to determine their probability of occurrence, and how GameForge will be affected if they do occur. Plans will then be made to avoid each risk, to track each risk to determine if it is more or less likely to occur, and to plan for those risks should they occur. It is the organization’s responsibility to perform risk mitigation, monitoring, and management in order to produce a quality product.

   The quicker the risks can be identified and avoided, the smaller the chances of having to face that particular risk’s consequence. The fewer consequences suffered as a result of good RMMM plan, the better the product, and the smoother the development process. Risk management organizational role Each member of the organization will undertake risk management. The development team will consistently be monitoring their progress and project status as to identify present and future risks as quickly and accurately as possible. With this said, the members who are not directly involved with the implementation of the product will also need to keep their eyes open for any possible risks that the development team did not spot. The responsibility of risk management falls on each member of the organization, while William Lord maintains this document.

2. **Six Sigma for Software**

   Six Sigma is a highly disciplined process that helps us focus on developing and delivering near-perfect products and services.

P.Praveen, Assistant Professor
Features of Six Sigma

- Six Sigma's aim is to eliminate waste and inefficiency, thereby increasing customer satisfaction by delivering what the customer is expecting.
- Six Sigma follows a structured methodology, and has defined roles for the participants.
- Six Sigma is a data driven methodology, and requires accurate data collection for the processes being analyzed.
- Six Sigma is about putting results on Financial Statements.
- Six Sigma is a business-driven, multi-dimensional structured approach for –
  - Improving Processes
  - Lowering Defects
  - Reducing process variability
  - Reducing costs
  - Increasing customer satisfaction
  - Increased profits

The word *Sigma* is a statistical term that measures how far a given process deviates from perfection. The central idea behind Six Sigma: If you can measure how many "defects" you have in a process, you can systematically figure out how to eliminate them and get as close to "zero defects" as possible and specifically it means a failure rate of 3.4 parts per million or 99.9997% perfect.

Key Concepts of Six Sigma

At its core, Six Sigma revolves around a few key concepts.

- **Critical to Quality** – Attributes most important to the customer.
- **Defect** – Failing to deliver what the customer wants.
- **Process Capability** – What your process can deliver.
- **Variation** – What the customer sees and feels.
- **Stable Operations** – Ensuring consistent, predictable processes to improve what the customer sees and feels.
- **Design for Six Sigma** – Designing to meet customer needs and process capability.

Our Customers Feel the Variance, Not the Mean. So Six Sigma focuses first on reducing process variation and then on improving the process capability.
Myths about Six Sigma

There are several myths and misunderstandings surrounding Six Sigma. Some of them few are given below –

- Six Sigma is only concerned with reducing defects.
- Six Sigma is a process for production or engineering.
- Six Sigma cannot be applied to engineering activities.
- Six Sigma uses difficult-to-understand statistics.
- Six Sigma is just training.

Benefits of Six Sigma

Six Sigma offers six major benefits that attract companies –

- Generates sustained success
- Sets a performance goal for everyone
- Enhances value to customers
- Accelerates the rate of improvement
- Promotes learning and cross-pollination
- Executes strategic change

Origin of Six Sigma

- Six Sigma originated at Motorola in the early 1980s, in response to achieving 10X reduction in product-failure levels in 5 years.
- Engineer Bill Smith invented Six Sigma, but died of a heart attack in the Motorola cafeteria in 1993, never knowing the scope of the craze and controversy he had touched off.
- Six Sigma is based on various quality management theories (e.g. Deming's 14 point for management, Juran's 10 steps on achieving quality).
16. UNIT WISE QUESTION BANK

UNIT-I

Two mark question with answers

1. What is legacy software?
   Ans: In computing, a legacy system is an old method, technology, computer system, or application program, "of, relating to, or being a previous or outdated computer system." Often a pejorative term, referencing a system as "legacy" means that it paved the way for the standards that would follow it.

2. List the phases in unified model?
   Ans: Inception phase
   Elaboration phase
   Construction phase
   Transition phase

3. What is the other name for waterfall model and who invented it?
   Ans: It is also known as linear sequential approach, The Waterfall model is originally invented by Winston W. Royce in 1970.

4. Why we will use formal methods model?
   Ans: Formal methods model encompasses a set of activities that leads to formal mathematical specification of computer software. Formal methods enable a software engineer to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation.

5. Explain the RAD model?
   Ans: Rapid application development is an incremental software process model that emphasizes a short development cycle.
Three mark questions with answers

1. What are the process framework activities?

Ans: A process framework establishes the foundation for a complete software process by identifying a small number of framework activities that are applicable to all software projects, regardless of their size or complexity. Communication, planning, modelling, construction, deployment

2. What are the levels in CMMI model?

Ans:
   a) Level 0: incomplete
   b) Level 1: performed
   c) Level 2: managed
   d) Level 3: defined
   e) Level 4: Quantitatively managed
   f) Level 5: Optimized

3. What is application software?

Ans: Application software consists of standalone programs that solve a specific business need. Application software is a program or group of programs designed for end users. These programs are divided into two classes: system software and application software. While system software consists of low-level programs that interact with computers at a basic level, application software resides above system software and includes applications such as database program

4. What are the characteristics of the software?

Ans: Software is engineered, not manufactured.

* Software does not wear out.

* Most software is custom built rather than being assembled from components

5. What are the various categories of software?

Ans: System software

* Application software

* Engineering/Scientific software

* Embedded software
1. List the types of software myths?

**Ans:** Many causes of a software affliction can be traced to a mythology that arose during the early history of software development. Unlike ancient myths that often provide human lessons well worth heeding, software myths propagated misinformation and confusion. Software myths had a number of attributes that made them insidious; for instance, they appeared to be reasonable statements of fact (sometimes containing elements of truth), they had an intuitive feel, and they were often promulgated by experienced practitioners who "knew the score." Today, most knowledgeable professionals recognize myths for what they are misleading attitudes that have caused serious problems for managers and technical people alike. However, old attitudes and habits are difficult to modify, and remnants of software myths are still believed.

**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).

**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.

**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.
2. Explain in detail the capability Maturity Model Integration (CMMI)?

Ans: CMMI: In recent years, there has been a significant emphasis on “process maturity.” The Software Engineering Institute (SEI) has developed a comprehensive model predicated on a set of software engineering capabilities that should be present as organizations reach different levels of process maturity. To determine an organization’s current state of process maturity, the SEI uses an assessment that results in a five point grading scheme. The grading scheme determines compliance with a capability maturity model (CMM) [PAU93] that defines key activities required at different levels of process maturity. The SEI approach provides a measure of the global effectiveness of a company's software engineering practices and establishes five process maturity levels that are defined in the following manner:

Level 1: Initial. The software process is characterized as ad hoc and occasionally even chaotic. Few processes are defined, and success depends on individual effort.

Level 2: Repeatable. Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.

Level 3: Defined. The software process for both management and engineering activities is documented, standardized, and integrated into an organization wide software process. All projects use a documented and approved version of the organization's process for developing and supporting software. This level includes all characteristics defined for level 2.

Level 4: Managed. Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled using detailed measures. This level includes all characteristics defined for level 3.

Level 5: Optimizing. Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies. This level includes all characteristics defined for level 4.

The five levels defined by the SEI were derived as a consequence of evaluating responses to the SEI assessment questionnaire that is based on the CMM. The results of the questionnaire are distilled to a single numerical grade that provides an indication of an organization's process maturity.

P.Praveen, Assistant Professor
The SEI has associated key process areas (KPAs) with each of the maturity levels. The KPAs describe those software engineering functions (e.g., software project planning, requirements management) that must be present to satisfy good practice at a particular level. Each KPA is described by identifying the following characteristics:

- **Goals**—the overall objectives that the KPA must achieve
- **Commitments**—requirements (imposed on the organization) that must be met to achieve the goals or provide proof of intent to comply with the goals.
- **Abilities**—those things that must be in place (organizationally and technically) to enable the organization to meet the commitments.
- **Activities**—the specific tasks required to achieve the KPA function.
- **Methods for monitoring implementation**—the manner in which the activities are monitored as they are put into place.
- **Methods for verifying implementation**—the manner in which proper practice for the KPA can be verified.

Figure 3.1 The CMM Structure
3. Explain the incremental process models?

Ans: 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 (if required) 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.

4. THE RAD (the rapid application development model):

**Ans:** 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 (e.g., 60 to 90 days) used primarily for information systems applications; the RAD approach encompasses the following phases.

**Business modelling.** The information flow among business functions is modelled 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?

**Data modelling:** information flow defined as part of the business modelling phase is refined into a set of data objects that are needed to support the business. The characteristics (called attributes) of each object are identified and the relationships between these objects defined.

**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. 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. Since the RAD process emphasizes reuse, 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.

The RAD process model the time constraints imposed on a RAD project demand “scalable scope.” If a business application can be modularized in a way that enables each major function to be completed in less than three months (using the approach described previously), it is a candidate for RAD. Each major function can be addressed by a separate RAD team and then integrated to form a whole. Like all process models, the RAD approach has drawbacks [BUT94]:

For large but scalable projects, RAD requires sufficient human resources to create the right number of RAD teams.

RAD requires developers and customers who are committed to the rapid-fire activities necessary to get a system complete in a much abbreviated time frame. If commitment is lacking from either constituency, RAD projects will fail.

Not all types of applications are appropriate for RAD. If a system cannot be properly modularized, building the components necessary for RAD will be problematic. If high performance is an issue and performance is to be achieved through tuning the interfaces to system components, the RAD approach may not work.
Q5) Explain waterfall model?

Ans:

Sometimes called the classic life cycle or the waterfall model, the linear sequential model suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing, and support. Illustrates the linear sequential model for software engineering. Modeled after a conventional engineering cycle, the linear sequential model encompasses the following activities:

**System/information engineering and modeling:** Because software is always part of a larger system (or business), work begins by establishing requirements for all system elements and then allocating some subset of these requirements to software. This system view is essential when software must interact with other elements such as hardware, people, and databases. System engineering and analysis encompass requirements gathering at the system level with a small amount of top level design and analysis. Information engineering encompasses requirements gathering at the strategic business level and at the business area level.

**Software requirements analysis.** The requirements gathering process is intensified and focused specifically on software. To understand the nature of the program(s) to be built, the software engineer ("analyst") must understand the information domain (described in Chapter 11) for the software, as well as required function, behavior, performance, and interface. Requirements for both the system and the software are documented and reviewed with the customer.

**Design:** Software design is actually a multistep process that focuses on four distinct attributes of a program: data structure, software architecture, interface representations, and procedural (algorithmic) detail. The design process translates requirements into a representation of the software that can be assessed for quality before coding begins. Like requirements, the design is documented and becomes part of the software configuration. Code generation. The design must be translated into a machine-readable form. The code generation step performs this task. If design is performed in a detailed manner, code generation can be accomplished mechanistically.
Testing. Once code has been generated, program testing begins. The testing process focuses on the logical internals of the software, ensuring that all statements have been tested, and on the functional externals; that is, conducting tests to uncover errors and ensure that defined input will produce actual results that agree with required results.
Objective type questions with answers

1) RAD Model was purposed by
   a) IBM
   b) Motorola
   c) Microsoft
   d) Lucent Technologies

2) Software is
   a) Set of computer programs, procedures and possibly associated document concerned with the operation of data processing.
   b) A set of compiler instructions
   c) A mathematical formula
   d) None of above

3) Which of the following is not the characteristic of software?
   a) Software does not wear out
   b) Software is flexible
   c) Software is not manufactured
   d) Software is always correct

4) Which of the following is not a process metric?
   a) Productivity
   b) Functionality
   c) Quality
   d) Efficiency

5) Efforts is measured in terms of
   a) Person - Months
   b) Persons
   c) Rupees
   d) Months

6) Infrastructure software are covered under
   a) Generic Products
   b) Customised Products

P.Praveen, Assistant Professor
c) Generic and Customised Products

d) None of the above

7) Management of software development is dependent upon
   a) People
   b) Product
   c) Process
   d) All of above

8) Spiral Model was developed by
   a) Bev Littlewood
   b) Berry Bohem
   c) Roger Pressman
   d) Victor Bisili

9) Which model is popular for student’s small projects?
   a) Waterfall Model
   b) Spiral Model
   c) Quick and Fix model
   d) Prototyping Model

10) Which is not a software life cycle model?
    a) Spiral Model
    b) Waterfall Model
    c) Prototyping Model
    d) Capability maturity Model

Answers

<table>
<thead>
<tr>
<th>Q. No.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>a</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>d</td>
<td>b</td>
<td>a</td>
<td>d</td>
</tr>
</tbody>
</table>
Fill in the blanks

1. The first step in software development life cycle __________
2. The detail study of existing system is referred to as __________
3. Prototyping aims at ______________
4. What is a prototype a mini-model of the ______________
5. Project risk factor is considered in ______________
6. SDLC stands for ______________
7. Build and Fix model has __________
8. Waterfall model is not suitable for ______________
9. RAD stands for ______________
10. The spiral model has two dimensions namely ______ and ______

Answers

<table>
<thead>
<tr>
<th>Q. No.</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>preliminary investigation and analysis.</td>
</tr>
<tr>
<td>2</td>
<td>system analysis</td>
</tr>
<tr>
<td>3</td>
<td>end user understanding and approval</td>
</tr>
<tr>
<td>4</td>
<td>proposed system.</td>
</tr>
<tr>
<td>5</td>
<td>spiral model</td>
</tr>
<tr>
<td>6</td>
<td>Software development life cycle</td>
</tr>
<tr>
<td>7</td>
<td>2 phases</td>
</tr>
<tr>
<td>8</td>
<td>accommodating change</td>
</tr>
<tr>
<td>9</td>
<td>Rapid Application Development</td>
</tr>
<tr>
<td>10</td>
<td>Radial_ and Angular</td>
</tr>
</tbody>
</table>
Unit-II
Two mark questions with answers

Q1) Define Requirements?

**Ans:** descriptions of the services that a software system must provide and the constraints under which it must operate. Requirements can range from high-level abstract statements of services or system constraints to detailed mathematical functional specifications. Requirements Engineering is the process of establishing the services that the customer requires from the system and the constraints under which it is to be developed and operated. Requirements may serve a dual function: As the basis of a bid for a contract As the basis for the contract itself

Q2) What are functional and non-functional requirements?

**Ans Functional requirements** Describe functionality or system services Depend on the type of software, expected users and the type of system where the software is used Functional user requirements may be high-level statements of what the system should do; functional system requirements should describe the system services.

- **Non-functional requirements**
  - Product requirements – Requirements which specify that the delivered product must behave in a particular way, e.g. execution speed, reliability etc. Organisational requirements – Requirements which are a consequence of organisational policies and procedures, e.g. process standards used, implementation requirements etc. External requirements – Requirements which arise from factors which are external to the system and its development process, e.g. interoperability requirements, legislative requirements etc.
  
  Typically non-functional requirements fall into areas such as:

- Accessibility, Capacity, current and forecast, Compliance, Documentation, Disaster recovery, Efficiency, Effectiveness, Extensibility, Fault tolerance, Interoperability, Maintainability, Privacy, Portability, Quality, Reliability, Resilience, Response time, Robustness, Scalability, Security, Stability, Supportability, Testability
Non-functional requirements are sometimes defined in terms of metrics (something that can be measured about the system) to make them more tangible. Non-functional requirements may also describe aspects of the system that don't relate to its execution, but rather to its evolution over time (e.g. maintainability, extensibility, documentation, etc).

Q3) what are User requirements

**Ans:** Should describe functional and non-functional requirements so that they are understandable by system users who don’t have detailed technical knowledge. User requirements are defined using natural language, tables and diagrams.

Q4) What are the prototyping approaches in software process?

**Ans:**

- Evolutionary prototyping – the initial prototype is prepared and it is then refined through number of stages to final stage.
- Throw-away prototyping – a rough practical implementation of the system is produced. The requirement problems can be identified from this implementation.

Q5) What are functional requirements?

**Ans:** Functional requirements are “statements of services the system should provide” how the system should react to particular input and how the system should behave in particular situation.
Three mark questions

Q1) What is Requirements Validations?

Ans: Software validation checks that the software product satisfies or fits the intended use (high-level checking), i.e., the software meets the user requirements, not as specification artifacts or as needs of those who will operate the software only; but, as the needs of all the stakeholders (such as users, operators, administrators, managers, investors, etc.).

There are two ways to perform software validation: internal and external. During internal software validation it is assumed that the goals of the stakeholders were correctly understood and that they were expressed in the requirement artifacts precise and comprehensively. If the software meets the requirement specification, it has been internally validated. External validation happens when it is performed by asking the stakeholders if the software meets their needs.

Different software development methodologies call for different levels of user and stakeholder involvement and feedback; so, external validation can be a discrete or a continuous event. Successful final external validation occurs when all the stakeholders accept the software product and express that it satisfies their needs. Such final external validation requires the use of an acceptance test which is a dynamic test.

However, it is also possible to perform internal static tests to find out if it meets the requirements specification but that falls into the scope of static verification because the software is not running.

Q2) Define Specification validation?

Ans: Not only can the software product as a whole be validated. Requirements should be validated before the software product as whole is ready (the waterfall development process requires them to be perfectly defined before design starts; but, iterative development processes do not require this to be so and allow their continual improvement).

User Requirements Specification validation: User requirements as stated in a document called User Requirements Specification are validated by checking if they indeed represent the will and goals of the stakeholders. This can be done by interviewing them and asking...
them directly (static testing) or even by releasing prototypes and having the users and stakeholders to assess them (dynamic testing).

User input validation: User input (gathered by any peripheral such as keyboard, bio-metric sensor, etc.) is validated by checking if the input provided by the software operators or users meet the domain rules and constraints (such as data type, range, and format).

Q3) Difference Validation vs. verification

Ans:

According to the Capability Maturity Model

- Software Validation: The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.
- Software Verification: The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase

Q4) What are non-functional requirements?

Ans: These define system properties and constraints e.g. reliability, response time and storage

- requirements. Constraints are I/O device capability, system representations, etc. Process requirements may also be specified mandating a particular CASE system, programming language or development method. Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless.

Q5) Discuss DOMAIN REQUIREMENTS

Ans: Derived from the application domain and describe system characteristics and features that reflect the domain. Domain requirements be new functional requirements, constraints on existing requirements or define specific computations. If domain requirements are not satisfied, the system may be unworkable.
Five mark questions

Q1) Explain system models?

Ans: System Models: System models are graphical representation that describes business processes, the trouble to be solved and the system that is to be urbanized. One can use models in the analysis process to develop an understanding of the existing system that is to be replaced or enhanced or to specify the new system that is required. For example,

1) An exterior perspective, where the context or environment of the system is modelled.
2) A behavioural perspective, where the behaviour of the system is modelled.

Types of Model

Different types of system are based on different approaches to abstraction. A data flow model, e.g., concentrates on the flow of data and the functional transformation on that data. It leaves out details of the data structure.

Examples of Types of System Models

1) Data Flow Model: Data flow models show the principal sub-system that make-up a system.
2) Composition Model: A composition or aggregation model shows how entities in the system are composed of other entities.
3) Architectural Model: Architectural models show the principal sub-system that make-up a system.
4) Classification Model: Object class/inheritance diagrams show how entities have common characteristics.
5) Stimulus-Response Model: A stimulus-response model, or state transition diagram. Shows how the system reacts to internal and external events.

Q2) List the types of data models?

Ans: Data Models:

Most great software systems make use of a large database of information. In some cases, this database is autonomous of the software system. An imperative part of system modelling is significant the logical form of the data processed by the system. These are sometimes called semantic data models.
Categories of Data Models:

1) **Flat Model:** This may not sternly qualify as a data model. The flat model surrounds of a single, two-dimensional array of data elements, where all members of a precise column are tacit to be correlated values, and all members of a row are assumed to be related to one another.

2) **Hierarchical Model:** In this model data is structured into a tree-like structure, implying a single upward link in all record to describe the nesting, and a sort field to maintain the records in a particular order in each same-level list.

3) **Network Model:** This model organizes data using two fundamental constructs, called records and sets. Records enclose fields, and sets classify one-to-many relationship between records: one owner, many members.

4) **Relational Model:** Relational Model is a database model based on first-order predicate logic. Its core idea is to depict a database as a collection over a predetermined set of predicate variables, relating constraints on the possible values and combinations of values.

5) **Object-Relational Model:** comparable to a relational database model, but objects, classes and inheritance are straightforwardly supported in database schemas and in the query language.

6) **Semantic Data Model:** A semantic data model in software engineering is a technique to define the meaning of data within the context of its inter-relationships with other data. A semantic data model is an abstraction which defines how the stored symbols relate to real world. A semantic data model is sometimes called a conceptual data model.

**Q3) State Goals and requirements:**

**Ans:** Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify.

**Goal**

A general intention of the user such as ease of use.

The system should be easy to use by experienced controllers and should be organised in such a way that user errors are minimised.

**Verifiable non-functional requirement**

A statement using some measure that can be objectively tested.
Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day.

Goals are helpful to developers as they convey the intentions of the system users.

Requirements interaction:

Conflicts between different non-functional requirements are common in complex systems. Spacecraft system
To minimise weight, the number of separate chips in the system should be minimised. To minimise power consumption, lower power chips should be used. However, using low power chips may mean that more chips have to be used. Which is the most critical requirement?
A common problem with non-functional requirements is that they can be difficult to verify. Users or customers often state these requirements as general goals such as ease of use, the ability of the system to recover from failure or rapid user response. These vague goals cause problems for system developers as they leave scope for interpretation and subsequent dispute once the system is delivered.

Q4) Explain about DOMAIN REQUIREMENTS?

Ans: Derived from the application domain and describe system characteristics and features that reflect the domain. Domain requirements be new functional requirements, constraints on existing requirements or define specific computations.
If domain requirements are not satisfied, the system may be unworkable.

Library system domain requirements:
There shall be a standard user interface to all databases which shall be based on the Z39.50 standard. Because of copyright restrictions, some documents must be deleted immediately on arrival. Depending on the user’s requirements, these documents will either be printed locally on the system server for manually forwarding to the user or routed to a network printer.

Domain requirements problems

Understandability
Requirements are expressed in the language of the application domain; This is often not understood by software engineers developing the system.

Implicitness
Domain specialists understand the area so well that they do not think of making the domain requirements explicit.
Q5) Illustrate on REQUIREMENTS ENGINEERING PROCESSES

Ans: The goal of requirements engineering process is to create and maintain a system requirements document. The overall process includes four high-level requirement engineering sub-processes. These are concerned with:

- Assessing whether the system is useful to the business (feasibility study)
- Discovering requirements (elicitation and analysis)
- Converting these requirements into some standard form (specification)
- Checking that the requirements actually define the system that the customer wants (validation)

The process of managing the changes in the requirements is called requirement management.

The alternative perspective on the requirements engineering process presents the process as a three-stage activity where the activities are organized as an iterative process around a spiral. The amount of time and effort devoted to each activity in iteration depends on the stage of the overall process and the type of system being developed. Early in the process, most effort will be spent on understanding high-level business and non-functional requirements and the user requirements. Later in the process, in the outer rings of the spiral, more effort will be devoted to system requirements engineering and system modeling.

This spiral model accommodates approaches to development in which the requirements are developed to different levels of detail. The number of iterations around the spiral can vary, so the spiral can be exited after some or all of the user requirements have been elicited.

Some people consider requirements engineering to be the process of applying a structured analysis method such as object-oriented analysis. This involves analyzing the system and developing a set of graphical system models, such as use-case models, that then serve as a system specification. The set of models describes the behavior of the system and are annotated with additional information describing, for example, its required performance or reliability.
Objective type questions:

1) SRS stands for?

1. Software requirement specification
2. Software requirement solution
3. System requirement specification
4. None of Above

2) Software engineering aims at developing?

1. Reliable Software
2. Cost Effective Software
3. Reliable and cost effective Software
4. None Of Above

3) A good specification should be?

1. Unambiguous
2. Distinctly Specific
3. Functional
4. All of Above

4) Which of the following is not a process metric?

1. Productivity
2. Functionality
3. Quality
4. Efficiency

5) Efforts is measured in terms of?

1. Person - Months
2. Persons
3. Rupees
4. Months

6) Infrastructure software are covered under?

1. Generic Products
2. Customised Products
3. Generic and Customised Products
4. None of the above
7) Management of software development is dependent upon?
   1. People
   2. Product
   3. Process
   4. All of above

8) During software development which factor is most crucial?
   1. People
   2. Process
   3. Product
   4. Project

9) Milestones are used to?
   1. Know the cost of the project
   2. Know the status of the project
   3. Know the user expectations
   4. None of the above

10) The term module in the design phase refers to?
    1. Functions
    2. Procedures
    3. Sub programs
    4. All of the above

Answers

<table>
<thead>
<tr>
<th>Q. No.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>a</td>
<td>c</td>
<td>d</td>
<td>b</td>
<td>a</td>
<td>A</td>
<td>d</td>
<td>a</td>
<td>a</td>
<td>d</td>
</tr>
</tbody>
</table>
Fill in the blanks

1 An SRS establishes the basis for agreement between the ______ and the ______
2 An SRS provides a reference for ______ of the final product.
3 A high quality SRS reduces the development ______.
4 ______ activity is used to understand the needs, goals and constraints.
5 _______ characteristic of SRS means the entire requirement denotes one interpretation.
6 The components of SRS are: ______
7 Partitioning, abstraction and projection are used for ______
8) COCOMO stands for: ______
9 The medium size projects are also known as ______
10 The form which can be filled up daily or weekly to maintain monitoring and plan activity are known as ______

ANSWERS

<table>
<thead>
<tr>
<th>Q. No.</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Client and the supplier</td>
</tr>
<tr>
<td>2</td>
<td>Validation</td>
</tr>
<tr>
<td>3</td>
<td>Cost</td>
</tr>
<tr>
<td>4</td>
<td>Problem Analysis</td>
</tr>
<tr>
<td>5</td>
<td>Unambiguous</td>
</tr>
<tr>
<td>6</td>
<td>Function Requirement</td>
</tr>
<tr>
<td>7</td>
<td>structuring information.</td>
</tr>
<tr>
<td>8</td>
<td>Constructive Cost Model</td>
</tr>
<tr>
<td>9</td>
<td>Semidetached</td>
</tr>
<tr>
<td>10</td>
<td>Time sheets</td>
</tr>
</tbody>
</table>
Unit-III

Two mark questions with answers

Q1) What is design?

Ans: Design is what virtually every engineer wants to do. It is the place where creativity rules – customer’s requirements, business needs, and technical considerations all come together in the formulation of a product or a system. Design creates a representation or model of the software, but unlike the analysis model, the design model provides detail about software data structures, architecture, interfaces, and components that are necessary to implement the system.

Q2) The goal of design engineering

Ans: Produce a model or representation that exhibits firmness, commodity, and delight. To accomplish this, a designer must practice diversification and then convergence. Another goal of software design is to derive an architectural rendering of a system. The rendering serves as a framework from which more detailed design activities are conducted.

Q3) Quality attributes:

Ans: The FURPS quality attributes represent a target for all software design: Functionality is assessed by evaluating the feature set and capabilities of the program, the generality of the functions that are delivered, and the security of the overall system. Usability is assessed by considering human factors, overall aesthetics, consistency and documentation. Reliability is evaluated by measuring the frequency and severity of failure, the accuracy of output results, and the mean – time –to- failure (MTTF), the ability to recover from failure, and the predictability of the program. Performance is measured by processing speed, response time, resource consumption, throughput, and efficiency. Supportability combines the ability to extend the program (extensibility), adaptability, serviceability; these three attributes represent a more common term maintainability.

Q4) Data design at the Architectural Level:

Ans: The challenge for a business has been to extract useful information from this data environment, particularly when the information desired is cross functional. To solve this challenge, the business IT community has developed data mining techniques, also called knowledge discovery in databases (KDD), that navigate through existing databases in an
attempt to extract appropriate business-level information. An alternative solution, called a data warehouse, adds an additional layer to the data architecture. A data warehouse is a large, independent database that encompasses some, but not all, of the data that are stored in databases that serve the set of applications required by a business.

Q5) What is User Interface Design?

Ans User interface design creates an effective communication medium between a human and a computer. Following a set of interface design principles, design identifies interface objects and actions and then creates a screen layout that forms the basis for a user interface prototype.
**Three mark Questions with answers**

Q1) Describe Interface Design Models

**Ans**

Four different models come into play when a user interface is to be designed. The software engineer creates a *design model*, a human engineer (or the software engineer) establishes a *user model*, the end-user develops a mental image that is often called the *user's model* or the *system perception*, and the implementers of the system create a *implementation model*. The role of interface designer is to reconcile these differences and derive a consistent representation of the interface.

Q2) User Task and Environmental Analysis:

**Ans**

The interface analysis activity focuses on the profile of the users who will interact with the system. Skill level, business understanding, and general receptiveness to the new system are recorded; and different user categories are defined. For each user category, requirements are elicited. In essence, the software engineer attempts to understand the system perception (Section 15.2.1) for each class of users. Once general requirements have been defined, a more detailed task analysis is conducted. Those tasks that the user performs to accomplish the goals of the system are identified, described, and elaborated.

Q3) What is Workflow analysis.

**Ans**

When a number of different users, each playing different roles, makes uses of a user interface, it is sometimes necessary to go beyond task analysis and object elaboration and apply workflow analysis. This technique allows a software engineer to understand how a work process is completed when several people are involved. The flow of events (shown in the figure) enable the interface designer to recognize three day interface characteristics.

Q4) Hierarchical representation.

**Ans**

As the interface is analyzed, a process of elaboration occurs. Once workflow has been established, a task hierarchy can be defined for each user type. The hierarchy is derived by a stepwise elaboration of each task identified for the user. For example, consider the user task requests that a prescription be refilled.

The following task hierarchy is developed:

- Request that a prescription be refilled
- Provide identifying information
Specify name
Specify userid
Specify PIN and password
Specify prescription number
Specify date refill is required

Q5) Mention Application accessibility

Ans Accessibility for users and software engineers who may be physically challenged is an imperative for moral, legal, and business reasons. A variety of accessibility guidelines many designed for Web applications but often applicable to all types of software provide detailed suggestions for designing interfaces that achieve varying levels of accessibility. Others provide specific guidelines or “assistive technology” that addresses the needs of those with visual, hearing, mobility, speech, and learning impairments.
Five mark Questions with answers

1. Explain design concepts?

Ans: Design concepts:

Abstraction
Procedural abstraction – a sequence of instructions that have a specific and limited function
Data abstraction – a named collection of data that describes a data object

Architecture
The overall structure of the software and the ways in which the structure provides conceptual integrity for a system. Consists of components, connectors, and the relationship between them.

Patterns
A design structure that solves a particular design problem within a specific context. It provides a description that enables a designer to determine whether the pattern is applicable, whether the pattern can be reused, and whether the pattern can serve as a guide for developing similar patterns.

Modularity
Separately named and addressable components (i.e., modules) that are integrated to satisfy requirements (divide and conquer principle). Makes software intellectually manageable so as to grasp the control paths, span of reference, number of variables, and overall complexity.

Information hiding
The designing of modules so that the algorithms and local data contained within them are inaccessible to other modules. This enforces access constraints to both procedural (i.e., implementation) detail and local data structures.

Functional independence
Modules that have a "single-minded" function and an aversion to excessive interaction with other modules.
High cohesion – a module performs only a single task
Low coupling – a module has the lowest amount of connection needed with other modules.

Stepwise refinement
Development of a program by successively refining levels of procedure detail.

P. Praveen, Assistant Professor
SOFTWARE ENGINEERING

Complements abstraction, which enables a designer to specify procedure and data and yet suppress low-level details

**Refactoring**

A reorganization technique that simplifies the design (or internal code structure) of a component without changing its function or external behavior

Removes redundancy, unused design elements, inefficient or unnecessary algorithms, poorly constructed or inappropriate data structures, or any other design failures

**Design classes**

Refines the analysis classes by providing design detail that will enable the classes to be implemented

Creates a new set of design classes that implement a software infrastructure to support the business solution

**Types of Design Classes**

User interface classes – define all abstractions necessary for human-computer interaction (usually via metaphors of real-world objects)

Business domain classes – refined from analysis classes; identify attributes and services (methods) that are required to implement some element of the business domain

Process classes – implement business abstractions required to fully manage the business domain classes

Persistent classes – represent data stores (e.g., a database) that will persist beyond the execution of the software

System classes – implement software management and control functions that enable the system to operate and communicate within its computing environment and the outside world.

2. Explain about Architectural styles?

**Ans Software Architectural Style:**

The software that is built for computer-based systems exhibit one of many architectural styles

Each style describes a system category that encompasses

A set of component types that perform a function required by the system

A set of connectors (subroutine call, remote procedure call, data stream, socket) that enable communication, coordination, and cooperation among components
Semantic constraints that define how components can be integrated to form the system. A topological layout of the components indicating their runtime interrelationships.

**Data Flow Style**

Has the goal of modifiability.

Characterized by viewing the system as a series of transformations on successive pieces of input data.

Data enters the system and then flows through the components one at a time until they are assigned to output or a data store.

**Batch sequential style**

The processing steps are independent components.

Each step runs to completion before the next step begins.

**Pipe-and-filter style**

Emphasizes the incremental transformation of data by successive components.

The filters incrementally transform the data (entering and exiting via streams).

The filters use little contextual information and retain no state between instantiations.

The pipes are stateless and simply exist to move data between filters.

**Advantages**

Has a simplistic design in the limited ways in which the components interact with the environment.

Consists of no more and no less than the construction of its parts.

Simplifies reuse and maintenance.

Is easily made into a parallel or distributed execution in order to enhance system performance.

**Disadvantages**

Implicitly encourages a batch mentality so interactive applications are difficult to create in this style.

Ordering of filters can be difficult to maintain so the filters cannot cooperatively interact to solve a problem.

Exhibits poor performance.

Filters typically force the least common denominator of data representation (usually ASCII stream).

Filter may need unlimited buffers if they cannot start producing output until they receive all of the input. Each filter operates as a separate process or procedure call, thus...
incurring overhead in set-up and take-down time. Use this style when it makes sense to view your system as one that produces a well-defined easily identified output. The output should be a direct result of sequentially transforming a well-defined easily identified input in a time-independent fashion.

**Data-Centered Style**
Has the goal of integrating the data
Refers to systems in which the access and update of a widely accessed data store occur
A client runs on an independent thread of control
The shared data may be a passive repository or an active blackboard
A blackboard notifies subscriber clients when changes occur in data of interest
At its heart is a centralized data store that communicates with a number of clients
Clients are relatively independent of each other so they can be added, removed, or changed in functionality
The data store is independent of the clients
Use this style when a central issue is the storage, representation, management, and retrieval of a large amount of related persistent data
Note that this style becomes client/server if the clients are modeled as independent processes.

**Virtual Machine Style**
Has the goal of portability
Software systems in this style simulate some functionality that is not native to the hardware and/or software on which it is implemented
Can simulate and test hardware platforms that have not yet been built
Can simulate "disaster modes" as in flight simulators or safety-critical systems that would be too complex, costly, or dangerous to test with the real system
Examples include interpreters, rule-based systems, and command language processors
Interpreters
Add flexibility through the ability to interrupt and query the program and introduce modifications at runtime
Incur a performance cost because of the additional computation involved in execution
Use this style when you have developed a program or some form of computation but have no make of machine to directly run it on.
3. Write about Designing conventional components?

Ans  Conventional design constructs emphasize the maintainability of a functional/procedural program. Sequence, condition, and repetition.

Each construct has a predictable logical structure where control enters at the top and exits at the bottom, enabling a maintainer to easily follow the procedural flow. Various notations depict the use of these constructs. Graphical design notation.

Sequence, if-then-else, selection, repetition (see next slide). Tabular design notation (see upcoming slide). Program design language.

Similar to a programming language; however, it uses narrative text embedded directly within the program statements.

**Graphical Design Notation**

![Sequence Diagram](image1)

![If-then-else Diagram](image2)

![Selection Diagram](image3)

![Repetition Diagram](image4)

**Tabular Design Notation**

List all actions that can be associated with a specific procedure (or module).

List all conditions (or decisions made) during execution of the procedure.

Associate specific sets of conditions with specific actions, eliminating impossible combinations of conditions; alternatively, develop every possible permutation of conditions.

Define rules by indicating what action(s) occurs for a set of conditions.
4. What are golden rules?

**Ans  Golden Rules**

**Place the User in Control**
Define interaction modes in a way that does not force a user into unnecessary or undesired actions.

The user shall be able to enter and exit a mode with little or no effort (e.g., spell check → edit text → spell check).

Provide for flexible interaction.

The user shall be able to perform the same action via keyboard commands, mouse movement, or voice recognition.

Allow user interaction to be interruptible and "undo"able.

The user shall be able to easily interrupt a sequence of actions to do something else (without losing the work that has been done so far).

The user shall be able to "undo" any action. Streamline interaction as skill levels advance and allow the interaction to be customized. The user shall be able to use a macro mechanism to perform a sequence of repeated interactions and to customize the interface.

Hide technical internals from the casual user. The user shall not be required to directly use operating system, file management, networking, etc., commands to perform any actions. Instead, these operations shall be hidden from the user and performed "behind the scenes" in the form of a real-world abstraction.

Design for direct interaction with objects that appear on the screen.

The user shall be able to manipulate objects on the screen in a manner similar to what would occur if the object were a physical thing (e.g., stretch a rectangle, press a button, move a slider).

**Reduce the User's Memory Load**

Reduce demand on short-term memory.

The interface shall reduce the user's requirement to remember past actions and results by providing visual cues of such actions. Establish meaningful defaults.

The system shall provide the user with default values that make sense to the average user but allow the user to change these defaults. The user shall be able to easily reset any value to its original default value.

Define shortcuts that are intuitive.
The user shall be provided mnemonics (i.e., control or alt combinations) that tie easily to the action in a way that is easy to remember such as the first letter.

The visual layout of the interface should be based on a real world metaphor.

**Make the Interface Consistent**

The interface should present and acquire information in a consistent fashion.

Allow the user to put the current task into a meaningful context.

Maintain consistency across a family of applications.

If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so.

**Q5) Illustrate DESIGN EVALUATION?**

After the design model has been completed, a first-level prototype is created. The prototype is evaluated by the user, who provides the designer with direct comments about the efficacy of the interface. In addition, if formal evaluation techniques are used (e.g., questionnaires, rating sheets), the designer may extract information from these data (e.g., 80 percent of all users did not like the mechanism for saving data files). Design modifications are made based on user input, and the next level prototype is created. The evaluation cycle continues until no further modifications to the interface design are necessary. If a design model of the interface has been created, a number of evaluation criteria can be applied during early design reviews:

The length and complexity of the written specification of the system and its interface provide an indication of the amount of learning required by user of the system. The number of user tasks specified and the average number of actions per task provide an indication on interaction time and the overall efficiency of the system. The number of actions, tasks, and system states indicated by the design model imply the memory load on users of the system. Interface styles, help facilities, and error handling protocol provide a general indication of the complexity of the interface and the degree to which it will be accepted by the user.
Objective type questions with answers

1) Which of the following is a tool in design phase?
   (a) Abstraction  
   (b) Refinement  
   (c) Information Hiding  
   (d) All of Above

2) Information hiding is to hide from user, details?
   (a) that are relevant to him  
   (b) that are not relevant to him  
   (c) that may be maliciously handled by him  
   (d) that are confidential

3) Which of the following comments about object-oriented design of software, is not true?
   (a) Objects inherit the properties of class  
   (b) Classes are defined based on the attributes of objects  
   (c) an object can belong to two classes  
   (d) classes are always different

4) Design phase includes?
   (a) data, architectural and procedural design only  
   (b) architectural, procedural and interface design only  
   (c) data, architectural and interface design only  
   (d) data, architectural, interface and procedural design

5) In the system concepts, term organization?
   (a) implies structure and order  
   (b) refers to the manner in which each component functions with other components of the system  
   (c) refers to the holism of system  
   (d) means that part of the computer system depend on one another

6) In the system concepts, the term integration?
   (a) implies structure and order  
   (b) refers to the manner in which each component functions with other components of the system  
   (c) means that parts of computer system depends on one another
(d) refers to the holism of systems

7) Project indicator enables a software project manager to?
   (a) assess the status of an ongoing project
   (b) track potential risks
   (c) uncover problem areas before they "go critical"
   (d) All of above

8) Once object oriented programming has been accomplished, unit testing is applied for each class. Class tests include?
   (a) Fault based testing
   (b) Random testing
   (c) Partition testing
   (d) All of above

9) A quantitative measure of the degree to which a system, component, or process possesses a given attribute?
   (a) Measure
   (b) Measurement
   (c) Metric
   (d) None of these

10) The model remains operative until the software is retired?
    (a) Waterfall
    (b) Incremental
    (c) Spiral
    (d) None of these

Answers

<table>
<thead>
<tr>
<th>Q. No.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>d</td>
<td>c</td>
<td>c</td>
<td>d</td>
<td>A</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>c</td>
<td>c</td>
</tr>
</tbody>
</table>
Fill in the blanks

1 PDL stands for __________
2 In system design, we do following: __________
3 Design phase includes __________
4 Most common method for designing algorithm is __________
5 Which one is the key term used in design of a system? __________
6 Which of the following is NOT a component of Object oriented software engineering? __________
7 Which is not the level of Cohesion? ______
8 Structured design methodology tries to reduce __________
9 Number of subordinates associated with given module is known as ______
10 Which is not factor for design specification? ______

Answers

<table>
<thead>
<tr>
<th>Q. No.</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Process Design Language</td>
</tr>
<tr>
<td>2</td>
<td>Parallel, Hardware and Software design</td>
</tr>
<tr>
<td>3</td>
<td>Data, architectural, interface, procedural design</td>
</tr>
<tr>
<td>4</td>
<td>Step wise refinement</td>
</tr>
<tr>
<td>5</td>
<td>Module</td>
</tr>
<tr>
<td>6</td>
<td>Architecture</td>
</tr>
<tr>
<td>7</td>
<td>Physical</td>
</tr>
<tr>
<td>8</td>
<td>Coupling</td>
</tr>
<tr>
<td>9</td>
<td>Fan-out</td>
</tr>
<tr>
<td>10</td>
<td>Abstraction</td>
</tr>
</tbody>
</table>
Unit-IV

Two mark questions with answers

Q1) What is White Box testing?
Ans: Also called glass box testing
Involves knowing the internal working of a program
Guarantees that all independent paths will be exercised at least once.
 Exercises all logical decisions on their true and false sides
Executes all loops
Exercises all data structures for their validity
White box testing techniques
Basis path testing
Control structure testing

Q2) Data flow Testing
Ans: Selects test paths according to the locations of definitions and use of variables in a program. Aims to ensure that the definitions of variables and subsequent use is tested. First construct a definition-use graph from the control flow of a program.

Q3) Reconciling Different Metrics Approaches?
Ans: The relationship between lines of code and function points depend upon the programming language that is used to implement the software and quality of the design. Function points and LOC based metrics have been found to be relatively accurate predictors of software development effort and cost.

Q4) Use-Case Oriented Metrics
Ans: Use-cases describe user-visible functions and features that are basic requirements for a system. The use-cases is directly proportional to the size of the application in LOC and to the number of use-cases is directly proportional to the size of the application in LOC and to the number of test cases that will have to be designed to fully exercise the application.
Because use-cases can be created at vastly different levels of abstraction, there is no standard size for a use-case. Without a standard measure of what a use-case is, its application as a normalization measure is suspect.

Q5) What are metrics for software quality?

Ans The overriding goal of software engineering is to produce a high-quality system, application, or product within a timeframe that satisfies a market need. To achieve this goal, software engineers must apply effective methods coupled with modern tools within the context of a mature software process.
Three mark questions with answers

Q1) Object Oriented Metrics:

Ans Conventional software project metrics (LOC or FP) can be used to estimate object oriented software projects. Lorenz and Kidd suggest the following set of metrics for OO projects:

**Number of scenario scripts:** A scenario script is a detailed sequence of steps that describes the interaction between the user and the application.

**Number of key classes:** Key classes are the “highly independent components that are defined early in object-oriented analysis.

**Number of support classes:** Support classes are required to implement the system but are not immediately related to the problem domain.

Q2) Function-Oriented Metrics

Ans Function-oriented software metrics use a measure of the functionality delivered by the application as a normalization value. Since ‘functionality’ cannot be measured directly, it must be derived indirectly using other direct measures. Function-oriented metrics were first proposed by Albrecht, who suggested a measure called the *function point*. Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity.

Proponents claim that FP is programming language independent, making it ideal for application using conventional and nonprocedural languages, and that it is based on data that are more likely to be known early in the evolution of a project, making FP more attractive as an estimation approach.

Q3) Elaborate Software Quality

Ans Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

Factors that affect software quality can be categorized in two broad groups:

Factors that can be directly measured (e.g. defects uncovered during testing)

Factors that can be measured only indirectly (e.g. usability or maintainability)
Q4) **Black box testing**

**Ans** Treats the system as black box whose behavior can be determined by studying its input and related output Not concerned with the internal structure of the program

Q5) **Black Box Testing**

**Ans** It focuses on the functional requirements of the software ie it enables the sw engineer to derive a set of input conditions that fully exercise all the functional requirements for that program. Concerned with functionality and implementation

1) Graph based testing method
2) Equivalence partitioning
Five mark questions with answers

1. What are the Test strategies for conventional software?

Ans Unit testing:

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. 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.

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.

INTEGRATION TESTING:

A neophyte in the software world might ask a seemingly legitimate question once all modules have been unit tested: "If they all work individually, why do you doubt that they'll work when we put them together?" The problem, of course, is "putting them together" interfacing. Data can be lost across an interface; one module can have an inadvertent, adverse affect on another; sub functions, when combined, may not produce the desired major function; individually acceptable imprecision may be magnified to unacceptable levels; global data structures can present problems. Sadly, the list goes on and on.

Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested components and build a program structure that has been dictated by design. There is often a tendency to attempt non incremental integration; that is, to construct the program using a "big bang" approaches. All components are combined in advance. The entire program is tested as a whole. And chaos usually results! A set of errors is encountered. Correction is difficult because isolation of causes is complicated by the vast expanse of the entire program. Once these errors are corrected, new ones appear and the process continues in a seemingly endless loop. Incremental integration is the antithesis of the big bang approach. The program is constructed and tested in small increments, where errors are easier to isolate and correct; interfaces are more likely to be tested completely; and a systematic test approach may be applied. In the sections that follow, a number of different incremental integration strategies are discussed.

**Top-down Integration:**

Top-down integration testing is an incremental approach to construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main control module (main program). Modules subordinate (and ultimately subordinate) to the main control module are incorporated into the structure in either a depth-first or breadth-first manner. depth-first integration would integrate all
components on a major control path of the structure. Selection of a major path is somewhat arbitrary and depends on application-specific characteristics. For example, selecting the left hand path, components M1, M2, M5 would be integrated first. Next, M8 or sary for proper functioning of M2) M6 would be integrated. Then, the central and right hand control paths are built. Breadth-first integration incorporates all components directly subordinate at each level, moving across the structure horizontally. From the figure, components M2, M3, and M4 (a replacement for stub S4) would be integrated first. The next control level, M5, M6, and so on, follows. The integration process is performed in a series of five steps: 1. The main control module is used as a test driver and stubs are substituted for all components directly subordinate to the main control module. 2. Depending on the integration approach selected (i.e., depth or breadth first), subordinate stubs are replaced one at a time with actual components. 3. Tests are conducted as each component is integrated. 4. On completion of each set of tests, another stub is replaced with the real component. 5. Regression testing may be conducted to ensure that new errors have not been introduced. The process continues from step 2 until the entire program structure is built. The top-down integration strategy verifies major control or decision points early in the test process. In a well-factored program structure, decision making occurs at upper levels in the hierarchy and is therefore encountered first. If major control problems do exist, early recognition is essential. If depth-first integration is selected, a complete function of the software may be implemented and demonstrated.

(1) Delay many tests until stubs are replaced with actual modules,

(2) Develop stubs that perform limited functions that simulate the actual module,

(3) Integrate the software from the bottom of the hierarchy upward. The first approach (delay tests until stubs are replaced by actual modules) causes us to lose some control over correspondence between specific tests and incorporation of specific modules. This can lead to difficulty in determining the cause of errors and tends to violate the highly constrained nature of the top-down approach. The second approach is workable but can lead to significant overhead, as stubs become more and more complex. The third approach, called bottom-up testing, is discussed in the next section.
Bottom-up Integration:

Bottom-up integration testing, as its name implies, begins construction and testing with atomic modules (i.e., components at the lowest levels in the program structure). Because components are integrated from the bottom up, processing required for components subordinate to a given level is always available and the need for stubs is eliminated. A bottom-up integration strategy may be implemented with the following steps:

1. Low-level components are combined into clusters (sometimes called builds) that perform a specific software sub function.
2. A driver (a control program for testing) is written to coordinate test case input and output.
3. The cluster is tested.
4. Drivers are removed and clusters are combined moving upward in the program structure. Integration follows the pattern .. Components are combined to form clusters 1, 2, and 3. Each of the clusters is tested using a driver (shown as a dashed block). Components in clusters 1 and 2 are subordinate to Ma. Drivers D1 and D2 are removed and the clusters are interfaced directly to Ma. Similarly, driver D3 for cluster 3 is removed prior to integration with module Mb. Both Ma and Mb will ultimately be integrated with component Mc, and so forth. As integration moves upward, the need for separate test drivers lessens. In fact, if the top two levels of program structure are integrated top down, the number of drivers can be reduced substantially and integration of clusters is greatly simplified.

2. Explain validation testing?

Ans At the culmination of integration testing, software is completely assembled as a package, interfacing errors have been uncovered and corrected, and a final series of software tests validation testing may begin. Validation can be defined in many ways, but a simple (albeit harsh) definition is that validation succeeds when software functions in a manner that can be reasonably expected by the customer. At this point a battle-hardened software developer might protest: "Who or what is the arbiter of reasonable expectations?" Reasonable expectations are defined in the Software Requirements Specification a document that describes all user-visible attributes of the software. The specification contains a section called Validation Criteria. Information contained in that section forms the basis for a validation testing approach.

Validation Test Criteria:
Software validation is achieved through a series of black-box tests that demonstrate conformity with requirements. A test plan outlines the classes of tests to be conducted and a test procedure defines specific test cases that will be used to demonstrate conformity with requirements. Both the plan and procedure are designed to ensure that all functional requirements are satisfied, all behavioural characteristics are achieved, all performance requirements are attained, documentation is correct, and human engineered and other requirements are met (e.g., transportability, compatibility, error recovery, maintainability). After each validation test case has been conducted, one of two possible conditions exist: (1) The function or performance characteristics conform to specification and are accepted or (2) a deviation from specification is uncovered and a deficiency list is created. Deviation or error discovered at this stage in a project can rarely be corrected prior to scheduled delivery. It is often necessary to negotiate with the customer to establish a method for resolving deficiencies.

**Alpha and Beta Testing:**

It is virtually impossible for a software developer to foresee how the customer will really use a program. Instructions for use may be misinterpreted; strange combinations of data may be regularly used; output that seemed clear to the tester may be unintelligible to a user in the field. When custom software is built for one customer, a series of acceptance tests are conducted to enable the customer to validate all requirements. Conducted by the end-user rather than software engineers, an acceptance test can range from an informal "test drive" to a planned and systematically executed series of tests. In fact, acceptance testing can be conducted over a period of weeks or months, thereby uncovering cumulative errors that might degrade the system over time. If software is developed as a product to be used by many customers, it is impractical to perform formal acceptance tests with each one. Most software product builders use a process called alpha and beta testing to uncover errors that only the end-user seems able to find. The alpha test is conducted at the developer's site by a customer. The software is used in a natural setting with the developer "looking over the shoulder" of the user and recording errors and usage problems. Alpha tests are conducted in a controlled environment. The beta test is conducted at one or more customer sites by the end-user of the software. Unlike alpha testing, the developer is generally not present. Therefore, the beta test is a "live" application of the software in an environment that cannot be controlled by the developer. The customer records all problems (real or imagined) that
are encountered during beta testing and reports these to the developer at regular intervals. As a result of problems reported during beta tests, software engineers make modifications and then prepare for release of the software product to the entire customer base.

3. Explain system testing?

**Ans:** Software process and are not conducted solely by software engineers. However, steps taken during software design and testing can greatly improve the probability of successful software integration in the larger system. A classic system testing problem is "finger-pointing." This occurs when an error is uncovered, and each system element developer blames the other for the problem. Rather than indulging in such nonsense, the software engineer should anticipate potential interfacing problems and (1) design error-handling paths that test all information coming from other elements of the system, (2) conduct a series of tests that simulate bad data or other potential errors at the software interface, (3) record the results of tests to use as "evidence" if finger-pointing does occur, and (4) participate in planning and design of system tests to ensure that software is adequately tested. System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that system elements have been properly integrated and perform allocated functions. In the sections that follow, we discuss the types of system tests [BEI84] that are worthwhile for software-based system.

**Recovery testing:**
Many computer based systems must recover from faults and resume processing within a prespecified time. In some cases, a system must be fault tolerant; that is, processing faults must not cause overall system function to cease. In other cases, a system failure must be corrected within a specified period of time or severe economic damage will occur. Recovery testing is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed. If recovery is automatic (performed by the system itself), reinitialization, check pointing mechanisms, data recovery, and restart are evaluated for correctness. If recovery requires human intervention, the mean-time-to-repair (MTTR) is evaluated to determine whether it is within acceptable limits.
Security testing:
Any computer-based system that manages sensitive information or causes actions that can improperly harm (or benefit) individuals is a target for improper or illegal penetration. Penetration spans a broad range of activities: hackers who attempt to penetrate systems for sport; disgruntled employees who attempt to penetrate for revenge; dishonest individuals who attempt to penetrate for illicit personal gain. Security testing attempts to verify that protection mechanisms built into a system will, in fact, protect it from improper penetration. To quote Beizer [BEI84]: "The system's security must, of course, be tested for invulnerability from frontal attack—but must also be tested for invulnerability from flank or rear attack." During security testing, the tester plays the role(s) of the individual who desires to penetrate the system. Anything goes! The tester may attempt to acquire passwords through external clerical means; may attack the system with custom software designed to breakdown any defenses that have been constructed; may overwhelm the system, thereby denying service to others; may purposely cause system errors, hoping to penetrate during recovery; may browse through insecure data, hoping to find the key to system entry. Given enough time and resources, good security testing will ultimately penetrate a system. The role of the system designer is to make penetration cost more than the value of the information that will be obtained.

Stress testing:
During earlier software testing steps, white-box and black-box techniques resulted in thorough evaluation of normal program functions and performance. Stress tests are designed to confront programs with abnormal situations. In essence, the tester who performs stress testing asks: "How high can we crank this up before it fails?" Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency, or volume. For example, (1) special tests may be designed that generate ten interrupts per second, when one or two is the average rate, (2) input data rates may be increased by an order of magnitude to determine how input functions will respond, (3) test cases that require maximum memory or other resources are executed, (4) test cases that may cause thrashing in a virtual operating system are designed, (5) test cases that may cause excessive hunting for disk-resident data are created. Essentially, the tester attempts to break the program. A variation of stress testing is a technique called sensitivity testing. In some situations (the most common occur in mathematical algorithms), a very small range of data contained within the bounds of valid
data for a program may cause extreme and even erroneous processing or profound performance degradation. Sensitivity testing attempts to uncover data combinations within valid input classes that may cause instability or improper processing.

**Performance testing:**
For real-time and embedded systems, software that provides required function but does not conform to performance requirements is unacceptable. Performance testing is designed to test the run-time performance of software within the context of an integrated system. Performance testing occurs throughout all steps in the testing process. Even at the unit level, the performance of an individual module may be assessed as white-box tests are conducted. However, it is not until all system elements are fully integrated that the true performance of a system can be ascertained. Performance tests are often coupled with stress testing and usually require both hardware and software instrumentation. That is, it is often necessary to measure resource utilization (e.g., processor cycles) in an exacting fashion. External instrumentation can monitor execution intervals, log events (e.g., interrupts) as they occur, and sample machine states on a regular basis. By instrumenting a system, the tester can uncover situations that lead to degradation and possible system failure.

4. List design model metrics?

**Ans Design Metrics**
Measurement is done by metrics. Three parameters are measured: process measurement through process metrics, product measurement through product metrics, and project measurement through project metrics.

Process metrics assess the effectiveness and quality of software process, determine maturity of the process, effort required in the process, effectiveness of defect removal during development, and so on. Product metrics is the measurement of work product produced during different phases of software development. Project metrics illustrate the project characteristics and their execution.
Process Metrics

To improve any process, it is necessary to measure its specified attributes, develop a set of meaningful metrics based on these attributes, and then use these metrics to obtain indicators in order to derive a strategy for process improvement.

Using software process metrics, software engineers are able to assess the efficiency of the software process that is performed using the process as a framework. Process is placed at the centre of the triangle connecting three factors (product, people, and technology), which have an important influence on software quality and organization performance. The skill and motivation of the people, the complexity of the product and the level of technology used in the software development have an important influence on the quality and team performance. The process triangle exists within the circle of environmental conditions, which includes development environment, business conditions, and customer/user characteristics.

To measure the efficiency and effectiveness of the software process, a set of metrics is formulated based on the outcomes derived from the process. These outcomes are listed below.

- Number of errors found before the software release
- Defect detected and reported by the user after delivery of the software
- Time spent in fixing errors
- Work products delivered
- Human effort used
- Time expended
- Conformity to schedule
- Wait time
- Number of contract modifications
- Estimated cost compared to actual cost.

Note that process metrics can also be derived using the characteristics of a particular software engineering activity. For example, an organization may measure the effort and time spent by considering the user interface design.

It is observed that process metrics are of two types, namely, private and public. Private Metrics are private to the individual and serve as an indicator only for the specified individual(s). Defect rates by a software module and defect errors by an individual are examples of private process metrics. Note that some process metrics are public to all team members but private to the project. These include errors detected while performing formal technical reviews and defects reported about various functions included in the software.

Public metrics include that was private to both individuals and teams. Project-level defect rates, effort and related data are collected, analyzed and assessed in order to obtain indicators that help in improving the organizational process performance.

5. Process Metrics Etiquette

**Ans** Process metrics can provide substantial benefits as the organization works to improve its process maturity. However, these metrics can be misused and create problems for the organization. In order to avoid this misuse, some guidelines have been defined, which can be used both by managers and software engineers. These guidelines are listed below. Rational thinking and organizational sensitivity should be considered while analyzing metrics data.

Feedback should be provided on a regular basis to the individuals or teams involved in collecting measures and metrics.

Metrics should not appraise or threaten individuals.

Since metrics are used to indicate a need for process improvement, any metric indicating this problem should not be considered harmful.

Use of single metrics should be avoided.

As an organization becomes familiar with process metrics, the derivation of simple indicators leads to a stringent approach called Statistical Software Process Improvement
SOFTWARE ENGINEERING

(SSPI). SSPI uses software failure analysis to collect information about all errors (it is detected before delivery of the software) and defects (it is detected after software is delivered to the user) encountered during the development of a product or system.

Product Metrics

In software development process, a working product is developed at the end of each successful phase. Each product can be measured at any stage of its development. Metrics are developed for these products so that they can indicate whether a product is developed according to the user requirements. If a product does not meet user requirements, then the necessary actions are taken in the respective phase.

Product metrics help software engineer to detect and correct potential problems before they result in catastrophic defects. In addition, product metrics assess the internal product attributes in order to know the efficiency of the following.

- Analysis, design, and code model
- Potency of test cases
- Overall quality of the software under development.

Various metrics formulated for products in the development process are listed below.

- **Metrics for analysis model**: These address various aspects of the analysis model such as system functionality, system size, and so on.
- **Metrics for design model**: These allow software engineers to assess the quality of design and include architectural design metrics, component-level design metrics, and so on.
- **Metrics for source code**: These assess source code complexity, maintainability, and other characteristics.
- **Metrics for testing**: These help to design efficient and effective test cases and also evaluate the effectiveness of testing.
- **Metrics for maintenance**: These assess the stability of the software product.
Objective type questions with answers

1) White box testing, a software testing technique is sometimes called?
   (a) Basic path
   (b) Graph Testing
   (c) Dataflow
   (d) Glass box testing

2) Black box testing sometimes called?
   (a) Data Flow testing
   (b) Loop Testing
   (c) Behavioral Testing
   (d) Graph Based Testing

3) Which of the following is a type of testing?
   (a) Recovery Testing
   (b) Security Testing
   (c) Stress Testing
   (d) All of above

4) The objective of testing is?
   (a) Debugging
   (b) To uncover errors
   (c) To gain modularity
   (d) To analyze system

5) ...... is a black box testing method?
   (a) Boundary value analysis
   (b) Basic path testing
   (c) Code path analysis
   (d) None of above

6) Structured programming codes includes?
   (a) sequencing
   (b) alteration
   (c) iteration
   (d) multiple exit from loops
   (e) only A, B and C
7) An important aspect of coding is?
   (a) Readability
   (b) Productivity
   (c) To use as small memory space as possible
   (d) Brevity

8) Data structure suitable for the application is discussed in?
   (a) data design
   (b) architectural design
   (c) procedural design
   (d) interface design

9) In object oriented design of software, objects have?
   (a) attributes and names only
   (b) operations and names only
   (c) attributes, name and operations
   (d) None of above

10) Function oriented metrics were first proposed by?
    (a) John
    (b) Gaffney
    (c) Albrecht
    (d) Basili

Answers

<table>
<thead>
<tr>
<th>Q. No.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>d</td>
<td>c</td>
<td>d</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>e</td>
<td>a</td>
<td>a</td>
<td>c</td>
</tr>
</tbody>
</table>
**Fill in Blanks:**

1. The goal of coding should not be to reduce the __________ cost, but the goal should be to reduce cost of __________

2. In structured design methodology the hierarchy of modules is represented by the __________

3. Structured programming is often called ________ programming A. goto-less B. object oriented C. procedural D. None of these

4. In static structure of a program the text of the program is in ________ organization.

5. The information hiding principle in modern programming languages by ________

6. The single-entry, single-exit constructs are also called ________

7. In programming style, nesting means __________

8. When ________ type of variables is changed then some side effects are occurs.

9. Comments for a module are often called __________ for the module.

10. The program verification methods fall in which categories __________

**Answers**

<table>
<thead>
<tr>
<th>Q. No.</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Implementation &amp; Later Phases</td>
</tr>
<tr>
<td>2</td>
<td>Goto</td>
</tr>
<tr>
<td>3</td>
<td>structure chart</td>
</tr>
<tr>
<td>4</td>
<td>Linear</td>
</tr>
<tr>
<td>5</td>
<td>data abstraction</td>
</tr>
<tr>
<td>6</td>
<td>control constructs</td>
</tr>
<tr>
<td>7</td>
<td>if-then-else</td>
</tr>
<tr>
<td>8</td>
<td>global</td>
</tr>
<tr>
<td>9</td>
<td>prologue</td>
</tr>
<tr>
<td>10</td>
<td>static and dynamic</td>
</tr>
</tbody>
</table>
Unit-V

Two mark Questions with answers

Q1) REACTIVE VS. PROACTIVE RISK STRATEGIES

Ans: with them, should they become actual problems. More commonly, the software team does nothing about risks until something goes wrong. Then, the team flies into action in an attempt to correct the problem rapidly. This is often called a fire fighting mode.

The project team reacts to risks when they occur

mitigation—plan for additional resources in anticipation of fire fighting fix on failure—resource are found and applied when the risk strikes crisis management—failure does not respond to applied resources and project is in jeopardy

Q2) What is proactive strategy?

Ans begins long before technical work is initiated. Potential risks are identified, their probability and impact are assessed, and they are ranked by importance. Then, the software team establishes a plan for managing risk. Formal risk analysis is performed organization corrects the root causes of risk examining risk sources that lie beyond the bounds of the software o developing the skill to manage change

Q3) RISK IDENTIFICATION

Ans Risk identification is a systematic attempt to specify threats to the project plan. There are two distinct types of risks.

Generic risks and product-specific risks.

Generic risks are a potential threat to every software project.
Product-specific risks can be identified only by those with a clear understanding of the technology, the people, and the environment that is specific to the project that is to be built.

Known and predictable risks in the following generic subcategories

Q4) RISK PROJECTION

Ans Risk projection, also called risk estimation, attempts to rate each risk in two ways—the likelihood or probability that the risk is real and the consequences of the problems associated with the risk, should it occur. The project planner, along with other managers and technical staff, performs four risk projection activities: establish a scale that reflects the perceived likelihood of a risk, delineate the consequences of the risk, estimate the impact of the risk on the project and the product, and note the overall accuracy of the risk projection so that there will be no misunderstandings.

Q5) Assessing Risk Impact

Ans Three factors affect the consequences that are likely if a risk does occur: its nature, its scope, and its timing. The nature of the risk indicates the problems that are likely if it occurs. The scope of a risk combines the severity (just how serious is it?) with its overall distribution. Finally, the timing of a risk considers when and for how long the impact will be felt.
Three mark questions with answers

Q1) RISK REFINEMENT
Ans One way for risk refinement is to represent the risk in condition-transition-consequence (CTC) format. This general condition can be refined in the following manner:

Sub condition 1. Certain reusable components were developed by a third party with no knowledge of internal design standards

Sub condition 2. The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.

Sub condition 3. Certain reusable components have been implemented in a language that is not supported on the target environment

Q2) What is Quality Control?
Ans Quality control involves the series of inspections, reviews, and tests used throughout the software process to ensure each work product meets the requirements placed upon it. A key concept of quality control is that all work products have defined, measurable specifications to which we may compare the output of each process. The feedback loop is essential to minimize the defects produced.

Q3) Quality Assurance
Ans Quality assurance consists of the auditing and reporting functions that assess the effectiveness and completeness of quality control activities. The goal of quality assurance is to provide management with the data necessary to be informed about product quality, thereby gaining insight and confidence that product quality is meeting its goals.

Cost of Quality The cost of quality includes all costs incurred in the pursuit of quality or in performing quality-related activities

Q4) SOFTWARE QUALITY ASSURANCE
Ans Software quality is defined as conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

The definition serves to emphasize three important points:

Software requirements are the foundation from which quality is measured. Lack of conformance to requirements is lack of quality.

P. Praveen, Assistant Professor
SOFTWARE ENGINEERING

Specified standards define a set of development criteria that guide the manner in which software is engineered. If the criteria are not followed, lack of quality will almost surely result. A set of implicit requirements often goes unmentioned (e.g., the desire for ease of use and good maintainability). If software conforms to its explicit requirements but fails to meet implicit requirements, software quality is suspect.

Q5) what are software reviews?

Ans Software reviews are a "filter" for the software engineering process. That is, reviews are applied at various points during software development and serve to uncover errors and defects that can then be removed. Software reviews "purify" the software engineering activities that we have called *analysis*, *design*, and *coding*.

Many different types of reviews can be conducted as part of software engineering. Each has its place. An informal meeting around the coffee machine is a form of review, if technical problems are discussed. A formal presentation of software design to an audience of customers, management, and technical staff is also a form of review.

A formal technical review is the most effective filter from a quality assurance standpoint. Conducted by software engineers (and others) for software engineers, the FTR is an effective means for improving software quality.
Q1. Explain Reactive and proactive risk strategies?

**Ans Reactive Risk Management**
Reactive risk management is often compared to a firefighting scenario. The reactive risk management kicks into action once an accident happens, or problems are identified after the audit. The accident is investigated, and measures are taken to avoid similar events happening in the future. Further, measures will be taken to reduce the negative impact the incident could cause on business profitability and sustainability.

Reactive risk management catalogues all previous accidents and documents them to find the errors which lead to the accident. Preventive measures are recommended and implemented via the reactive risk management method. This is the earlier model of risk management. Reactive risk management can cause serious delays in a workplace due to the unpreparedness for new accidents. The unpreparedness makes the resolving process complex as the cause of accident needs investigation and solution involve high cost, plus extensive modification.

**Proactive Risk Management**
Contrary to reactive risk management, proactive risk management seeks to identify all relevant risks earlier, before an incident occurs. The present organization has to deal with an era of rapid environmental change that is caused by technological advancements, deregulation, fierce competition, and increasing public concern. So, a risk management which relies on past incidents is not a good choice for any organization. Therefore, new thinking in risk management was necessary, which paved the way for proactive risk management. Proactive risk management can be defined as “Adaptive, closed loop feedback control strategy based on measurement, observation of the present safety level and planned explicit target safety level with a creative intellectuality”. The definition relates to the flexibility and creative intellectual power of humans who have a high sense of safety concern. Though, humans are the source of error, they can also be a very important safety source as per proactive risk management. Further, the closed loop strategy refers to setting up of boundaries to operate within. These boundaries are considered to have safe performance level. Accidental analysis is part of the proactive risk management, with which
accident scenarios are built and the key employees and stakeholder who may create the error for an accident, are identified. So, past accidents are important in proactive risk management as well.

Q2. What is the difference between Proactive and Reactive Risk management

Ans Reactive: “A response based risk management approach, which is dependent on accident evaluation and audit based findings.”
Proactive: “Adaptive, closed loop feedback control strategy based on measurement, observation of the present safety level and planned explicit target safety level with a creative intellectuality.”

Purpose of Proactive and Reactive Risk Management
Reactive risk management: Reactive risk management attempts to reduce the tendency of the same or similar accidents which happened in past being repeated in future.
Proactive risk management: Proactive risk management attempts to reduce the tendency of any accident happening in future by identifying the boundaries of activities, where a breach of the boundary can lead to an accident.

Q3. Define Risk mitigation, monitoring and management plan?

Ans Risk mitigation, monitoring and management plan
Risk analysis support the project team in constructing a strategy to deal with risks.
There are three important issues considered in developing an effective strategy:
Risk avoidance or mitigation - It is the primary strategy which is fulfilled through a plan.
Risk monitoring - The project manager monitors the factors and gives an indication whether the risk is becoming more or less.
Risk management and planning - It assumes that the mitigation effort failed and the risk is a reality.

RMMM Plan
It is a part of the software development plan or a separate document.
The RMMM plan documents all work executed as a part of risk analysis and used by the project manager as a part of the overall project plan.
The risk mitigation and monitoring starts after the project is started and the documentation of RMMM is completed.

Mitigation
The cost associated with a computer crash resulting in a loss of data is crucial. A computer crash itself is not crucial, but rather the loss of data. A loss of data will result in not being able to deliver the product to the customer. This will result in a not receiving a letter of acceptance from the customer. Without the letter of acceptance, the group will receive a failing grade for the course. As a result the organization is taking steps to make multiple backup copies of the software in development and all documentation associated with it, in multiple locations.

Monitoring
When working on the product or documentation, the staff member should always be aware of the stability of the computing environment they’re working in. Any changes in the stability of the environment should be recognized and taken seriously.

Management
The lack of a stable-computing environment is extremely hazardous to software development team. In the event that the computing environment is found unstable, the development team should cease work on that system until the environment is made stable again, or should move to a system that is stable and continue working there.

Q4. What is risk refinement?

Ans: Risk refinement:
During early stages of project planning, a risk may be stated quite generally. As time passes and more is learned about the project and the risk, it may be possible to refine the risk into a set of more detailed risks, each somewhat easier to mitigate, monitor, and manage. One way to do this is to represent the risk in condition-transition-consequence (CTC) format. That is, the risk is stated in the following form: Given that <condition> then there is
Using the CTC format for the reuse risk noted we can write:

Given that all reusable software components must conform to specific design standards and that some do not conform, then there is concern that (possibly) only 70 percent of the planned reusable modules may actually be integrated into the as-built system, resulting in the need to custom engineer the remaining 30 percent of components.

This general condition can be refined in the following manner:

**Subcondition 1.** Certain reusable components were developed by a third party with no knowledge of internal design standards.

**Subcondition 2.** The design standard for component interfaces has not been solidified and may not conform to certain existing reusable components.

**Subcondition 3.** Certain reusable components have been implemented in a language that is not supported on the target environment.

The consequences associated with these refined subconditions remains the same (i.e., 30 percent of software components must be customer engineered), but the refinement helps to isolate the underlying risks and might lead to easier analysis and response.

5. **Explain software quality assurance?**

**Ans** Presently there are two important approaches that are used to determine the quality of the software:

1. Defect Management Approach
2. Quality Attributes approach

As mentioned before anything that is not in line with the requirement of the client can be considered as a defect. Many times the development team fails to fully understand the requirement of the client which eventually leads to design error. Besides that, the error can be caused due to poor functional logic, wrong coding or improper data handling. In order to keep a track of defect a defect management approach can be applied. In defect management, categories of defects are defined based on severity. The number of defects is counted and actions are taken as per the severity defined. Control charts can be created to measure the development process capability.
Defect Management Approach
Quality Attribute Approach on the other hand focuses on six quality characteristics that are listed below:

1. Functionality: refers to complete set of important functions that are provided by the software
   - Suitability: whether the functions of the software are appropriate
   - Accurateness: are the functions implemented correctly?
   - Interoperability: how does the software interact with other components of the system?
   - Compliance: is the software in compliance with the necessary laws and guidelines?
2. Security: Is the software able to handle data related transaction securely?

2. Reliability: this refers to the capability of software to perform under certain conditions for a defined duration. This also defines the ability of the system to withstand component failure.
Maturity: Frequency of failure of software

- Recoverability: this gives an idea of a system’s ability to get back into full operation after failure.

3. Usability: refers to the ease of use of a function.
   Understand ability: how easily the functions can be understood
   Learn ability: How much effort the users of different level need to put in to understand the functions.

4. Efficiency: generally depends on good architecture and coding practices followed while developing software.

5. Maintainability: also known as supportability. It is greatly dependant on code readability and complexity and refers to the ability to identify and fix a fault in software:

- Analyzability: identification of the main cause of failure.
- Changeability: defines the effort that goes in modification of code to remove a fault.
- Stability: how stable a system is in its performance when there are changes made to it
- Testability: how much effort goes in testing the system.

6. Portability: Ability of the system to adapt to changes in its environment
   - Adaptable: how easily a system adapts to the changes made in specifications
   - Install ability: how easily a system can be installed.
   - Conformance: this is same as compliance in functionality.
   - Replace ability: how easy it is to replace a component of the system in a given environment.

Cost of Software Quality Cost of quality is important because when you decide to conduct software testing for your product you are actually going to invest your time, money and effort in getting quality checks done. By conducting an analysis of cost of software quality you would know what the return on that investment (ROI) is.

P. Praveen, Assistant Professor
Cost of Software Quality

Cost of quality is calculated by analyzing the conformance costs and non conformance costs.

A conformance cost is related to:

1. Prevention costs: amount spent on ensuring that all quality assurance practices are followed correctly. This includes tasks like training the team, code reviews and any other QA related activity etc.
2. Appraisal costs: this is the amount of money spent on planning all the test activities and then carrying them out such as developing test cases and then executing them.

The non conformance cost on the other hand is the expense that arises due to:

1. Internal failures: it is the expense that arises when test cases are executed for the first time at internal level and some of them fail. The expenses arise when the programme has to rectify all the defects uncovered from his piece of code at the time of unit or component testing.
2. External failures: it is the expense that occurs when the defect is found by the customer instead of the tester. These expenses are much more than what arise at internal level, especially if the customer gets unsatisfied or escalates the software failure.
Objective type questions with answers

1- Tally chart is
   (a) Process monitoring tool
   (b) Data collection tool
   (c) Process planning tool
   (d) None of the above

2- Diamond represents _______ while plotting flow chart.
   (a) Step in activity
   (b) Decision making
   (c) Direction of flow
   (d) None of the above

3- The role of management is to
   (a) provide Resources
   (b) define EMS
   (c) monitor the effectiveness of the system
   (d) All of the above

4- The objective of ISO-9000 family of Quality management is
   (a) Customer satisfaction
   (b) Employee satisfaction
   (c) Skill enhancement
   (d) Environmental issues

5- Total Quality Management (TQM) focuses on
   (a) Employee
   (b) Customer
   (c) Both (a) and (b)
   (d) None of the above

6- Which of the following is responsible for quality objective?
   (a) Top level management
   (b) Middle level management
   (c) Frontline management
   (d) All of the above

7- The following is (are) the machine down time.
   (a) Waste
8-TQM & ISO both focuses on
(a) Customer
(b) Employee
(c) Supplier
(d) All of the above

9-According to Deming, Quality problems are
(a) Due to management
(b) Due to method
(c) Due to machine
(d) Due to material

10-While setting Quality objective, ________ to be considered.
(a) Material quality
(b) Customer need
(c) Market demand
(d) All of the above

Answers

<table>
<thead>
<tr>
<th>Q. No.</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Answer</td>
<td>b</td>
<td>B</td>
<td>d</td>
<td>a</td>
<td>c</td>
<td>a</td>
<td>d</td>
<td>a</td>
<td>A</td>
<td>b</td>
</tr>
</tbody>
</table>
**Fill in the Blanks**

1. TQM promotes __________
2. Kaizen is __________
3. Quality circle can solve problem related to __________
4. Quality circle benefit to __________
5. __________ helps organization reduce employee turnover and absenteeism.
6. CMM stands for __________
7. While setting Quality objective, __________ to be considered.
8. Which is for Environment management __________
9. A formal technical review is a software quality assurance activity performed by __________
10. __________ is the most widely used strategy for statistical quality assurance in industry today

**Answers:**

<table>
<thead>
<tr>
<th>Q. No.</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Employee Participation</td>
</tr>
<tr>
<td>2</td>
<td>small change</td>
</tr>
<tr>
<td>3</td>
<td>Continuous improvement</td>
</tr>
<tr>
<td>4</td>
<td>Employee</td>
</tr>
<tr>
<td>5</td>
<td>Training and development</td>
</tr>
<tr>
<td>6</td>
<td>Capability maturity model</td>
</tr>
<tr>
<td>7</td>
<td>Customer need</td>
</tr>
<tr>
<td>8</td>
<td>ISO-14000</td>
</tr>
<tr>
<td>9</td>
<td>Customer need</td>
</tr>
<tr>
<td>10</td>
<td>Six Sigma</td>
</tr>
</tbody>
</table>
Beyond syllabus Topic:

STLC phases:

STLC stands for Software Testing Life Cycle. STLC is a sequence of different activities performed by the testing team to ensure the quality of the software or the product.

- STLC is an integral part of Software Development Life Cycle (SDLC). But, STLC deals only with the testing phases.
- STLC starts as soon as requirements are defined or SRD (Software Requirement Document) is shared by stakeholders.
- STLC provides a step-by-step process to ensure quality software.
- In the early stage of STLC, while the software or the product is developing, the tester can analyze and define the scope of testing, entry and exit criteria and also the Test Cases. It helps to reduce the test cycle time along with better quality.
- As soon as the development phase is over, the testers are ready with test cases and start with execution. This helps to find bugs in the initial phase.

STLC Phases

STLC has the following different phases but it is not mandatory to follow all phases. Phases are dependent on the nature of the software or the product, time and resources allocated for the testing and the model of SDLC that is to be followed.

There are 6 major phases of STLC –
- **Requirement Analysis** – When the SRD is ready and shared with the stakeholders, the testing team starts high level analysis concerning the AUT (Application under Test).

- **Test Planning** – Test Team plans the strategy and approach.

- **Test Case Designing** – Develop the test cases based on scope and criteria’s.

- **Test Environment Setup** – When integrated environment is ready to validate the product.

- **Test Execution** – Real-time validation of product and finding bugs.

- **Test Closure** – Once testing is completed, matrix, reports, results are documented.

Let us consider the following points and thereby, compare STLC and SDLC.

- STLC is part of SDLC. It can be said that STLC is a subset of the SDLC set.

- STLC is limited to the testing phase where quality of software or product ensures. SDLC has vast and vital role in complete development of a software or product.

- However, STLC is a very important phase of SDLC and the final product or the software cannot be released without passing through the STLC process.

- STLC is also a part of the post-release/ update cycle, the maintenance phase of SDLC where known defects get fixed or a new functionality is added to the software.

The following table lists down the factors of comparison between SDLC and STLC based on their phases –

<table>
<thead>
<tr>
<th>Phase</th>
<th>SDLC</th>
<th>STLC</th>
</tr>
</thead>
</table>
| Requirement Gathering |  - Business Analyst gathers requirements.  
                       - Development team analyzes the requirements.  
                       - After high level, the development team starts analyzing from the architecture and the design |  - Testing team reviews and analyzes the SRD document.  
                       - Identifies the testing requirements - Scope, Verification and Validation key points.  
                       - Reviews the requirements |
<table>
<thead>
<tr>
<th>Stage</th>
<th>Activities</th>
<th>Activities</th>
</tr>
</thead>
</table>
| Design        | - The architecture of SDLC helps you develop a high-level and low-level design of the software based on the requirements.  
- Business Analyst works on the mocker of UI design.  
- Once the design is completed, it is signed off by the stakeholders. | - In STLC, either the Test Architect or a Test Lead usually plan the test strategy.  
- Identifies the testing points.  
- Resource allocation and timelines are finalized here. |
| Development    | - Development team starts developing the software.  
- Integrate with different systems.  
- Once all integration is done, a ready to test software or product is provided. | - Testing team writes the test scenarios to validate the quality of the product.  
- Detailed test cases are written for all modules along with expected behaviour.  
- The prerequisites and the entry and exit criteria of a test module are identified here. |
| Environment Set up | - Development team sets up a test environment with developed product to validate.                                                                 | - The Test team confirms the environment set up based on the prerequisites.  
- Performs smoke testing to make sure the environment is correctly set up. |
<table>
<thead>
<tr>
<th>SOFTWARE ENGINEERING</th>
</tr>
</thead>
<tbody>
<tr>
<td>is stable for the product to be tested.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Testing</th>
</tr>
</thead>
</table>
| ● The actual testing is carried out in this phase. It includes unit testing, integration testing, system testing, defect retesting, regression testing, etc.  
● The Development team fixes the bug reported, if any and sends it back to the tester for retesting.  
● UAT testing performs here after getting sign off from SIT testing. |
| ● System Integration testing starts based on the test cases.  
● Defects reported, if any, get retested and fixed.  
● Regression testing is performed here and the product is signed off once it meets the exit criteria. |

<table>
<thead>
<tr>
<th>Deployment/ Product Release</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Once sign-off is received from various testing team, application is deployed in prod environment for real end users.</td>
</tr>
</tbody>
</table>
| ● Smoke and sanity testing in production environment is completed here as soon as product is deployed.  
● Test reports and matrix preparation are done by testing team to analyze the product. |

<table>
<thead>
<tr>
<th>Maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>● It covers the post deployment supports, enhancement and updates, if any.</td>
</tr>
<tr>
<td>● In this phase, the maintaining of test cases, regression suits and automation scripts take place based on the enhancement and updates.</td>
</tr>
</tbody>
</table>

P.Praveen, Assistant Professor