BTECH DEGREE EXAMINATION, JAN 2023

FIFTH SEMESTER

Information Technology

SOFTWARE ENGINEERING

(2013-14 Regulations)

PART-A

1.Give at least two reason for prototyping is problematic.

It has poor documentation because of continuously changing customer requirements. There may be too much variation in requirements. Customers sometimes demand the actual product to be delivered soon after seeing an early prototype.

1. It can be very hard tom manage prototyping in a large system.

2. a prototype may be adopted when it isn't finished by both analysts and users.

2.State the System Engineering Hierarchy.

A system can be divided into a hierarchy of sets of elements that include **subsystems, components, subcomponents and parts**. · A Subsystem is a system in its own right, except it normally will not provide a useful function on its own, it must be integrated with other subsystems to make a system.

3.Mention any two non-functional requirements on software to be developed.

Non-functional Requirements (NFRs) **define system attributes such as security, reliability, performance, maintainability, scalability, and usability**. They serve as constraints or restrictions on the design of the system across the different backlogs.

4.What is Requirement Engineering?

Requirements engineering is **the area of systems engineering those deals with the process of developing and verifying the system requirements**. Following good requirements engineering practices helps achieve the primary objective of making sure that the delivered system meets the customer's needs.

5.Generalize the concept about Software Re-Engineering.

Software Re-Engineering is **the examination and alteration of a system to reconstitute it in a new form**. The principle of Re-Engineering when applied to the software development process is called software re-engineering. It positively affects software cost, quality, customer service, and delivery speed.

6.Name the commonly used architectural styles.

- Layered (n-tier) architecture.

- Event-driven architecture.
- Microkernel architecture.
- Microservices architecture.
- Space-based architecture.

7.Enumerate different data flow architectures.

There are three Data flow architectures namely

1.Batch Sequential

2.Pipe & filter

3.Process Control architecture.

8.Distinguish between verification and validation?

Verification is a process of determining if the software is designed and developed as per the specified requirements.

Validation is the process of checking if the software (end product) has met the client's true needs and expectations.

9.List out all the data structure errors identified during unit testing.

1.Misunderstood or incorrect arithmetic precedence

2.Mixed mode operations

3.Incorrect initialization

4.Precision inaccuracy

5.Incorrect symbolic representation of an expression.

10.Mention the advantages of CASE tools?

- Provide new systems with shorter development time.
- Improve the productivity of the systems development process.
- Improve the quality of the systems development process.
- Improve worker skills.
- Improve the portability of new systems.
- Improve the management of the systems development process.

PART-B

11.Explain the Water fall model. What are the problems that are sometimes encountered when the water fall model is applied?

The Waterfall Model was the first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**. It is very simple to understand and use. In a waterfall

model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

The Waterfall model is the earliest SDLC approach that was used for software development.

The waterfall Model illustrates the software development process in a linear sequential flow. This means that any phase in the development process begins only if the previous phase is complete. In this waterfall model, the phases do not overlap.

Waterfall Model - Design

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

The following illustration is a representation of the different phases of the Waterfall Model.

The sequential phases in Waterfall model are −

- **Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- **System Design** − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- **Implementation** − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
- **Integration and Testing** − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment of system** − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
- **Maintenance** − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model, phases do not overlap.

Waterfall Model - Disadvantages

The disadvantage of waterfall development is that it does not allow much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

The major disadvantages of the Waterfall Model are as follows −

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

12.List several software process paradigms. Explain how both iterative life cycle model and prototyping model can be accommodated in the spiral process model.

Software paradigms **refer to the methods and steps, which are taken while designing the software**. There are many methods proposed and are in work today, but we need to see where in the software engineering these paradigms stand.

Historically, Software developers have experimented with three major software development paradigms:

1.**procedural**

2.data-driven

3.object-oriented.

This Spiral model is a combination of iterative development process model and sequential linear development model i.e. the waterfall model with a very high emphasis on risk analysis. It **allows incremental releases of the product or incremental refinement through each iteration around the spiral**.

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

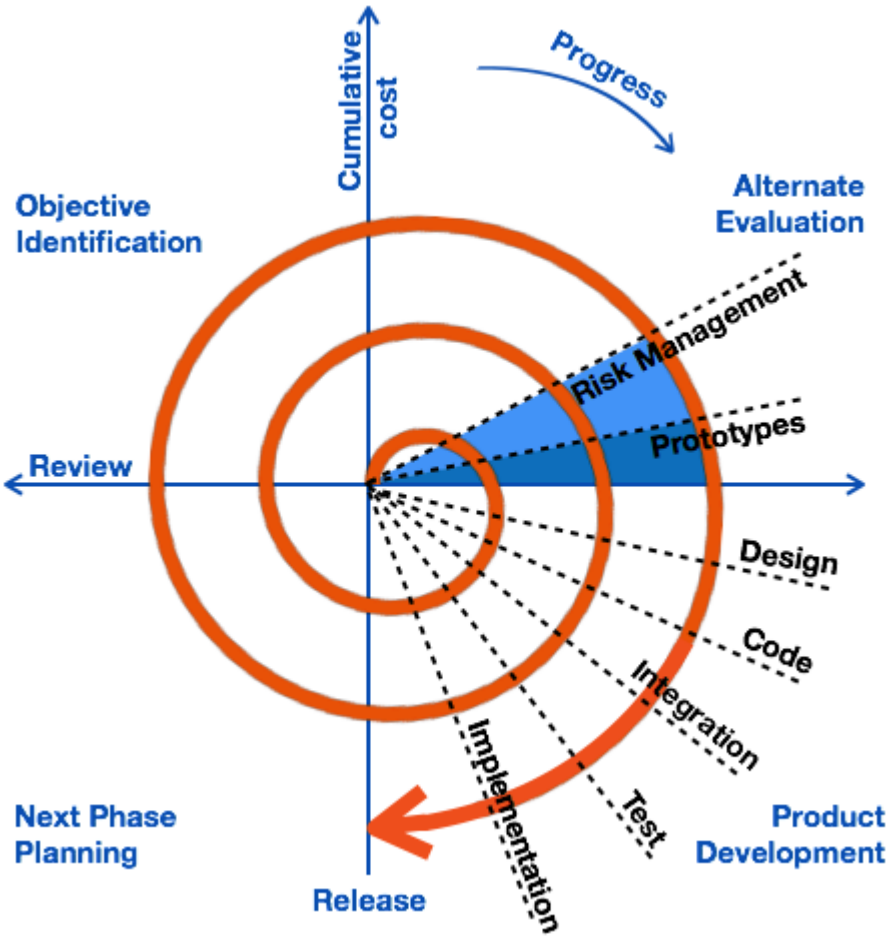
The Construct phase refers to production of the actual software product at every spiral. In the baseline spiral, when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

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.



Based on the customer evaluation, the software development process enters the next iteration and subsequently follows the linear approach to implement the feedback suggested by the

customer. The process of iterations along the spiral continues throughout the life of the software.

13.Consider seven functions with their estimated lines of code. Average productivity based on historical data is 620 LOC/pm and labour rate is RS.8000 per month. Find the total estimates project cost and effort? F1-2340, F2-5380, F3-6800, F4-3350, F5-4950, F6-2140, F7-8400. (Note: use FP, COCOMO, SLOC method to calculate the total estimation).

8 **Consider 7 functions with their estimated lines of code. Average productivity based on historical data is 620 LOC/pm and labour rate is Rs. 8000 per mnth. Find the total estimates project cost and effort? F1 – 2340 , F2 – 5380, F3 – 6800 , F4 –3350 , F5 -4950 , F6 -2140 , F7 – 8400**

There are many techniques that can be used to rigorously estimate or measure effortand cost for a software project, such as:

-Function Point (FP)

-Source Lines of Code (SLOC).

-COnstructive COst MOdel (COCOMO)

-Delphi

SLOC Technique(Source Line of Code Technique)-The SLOC technique is an objective method of estimating or calculating the size of the project.

-The project size helps determine the resources, effort, cost, and duration required to complete the project.

-It is also used to directly calculate the effort to be spent on a project.

-W e can use it when the programming language and the technology to be used are predefined.

-This technique includes the calculation of lines of codes(LOC), documentation of pages, inputs, outputs, and components of a software program.

Counting SLOC-The use of SLOC techniques can be used in the case of the technology or language remains unchanged throughout the project.

Generally, it can be used when you are using third-generation language, such as FORTRAN or COBOL.

-To count the SLOC the following must be considered: The count includes:-

The SLOC delivered to client.

-The SLOC written only by the development team are counted-The declaration statements are counted as source lines of code

The count excludes:-The code created by application generators.

-The comments inserted to improve the readability of program.

-Once, you get the numbers of line of code of SLOC, you can estimate or calculate the total effort and cost to complete the given project.

Example:-Assume estimated lines of code of a system is: 33,600 LOC -Average productivity for system of this type is: 620 LOC/person-month-

There are 7 developers-Labor rate is: $ 8000per person-month Calculate the total effort and cost required to complete the above project

Solution+Way1=>

Total Effort =Total LOC/Productivity = 33600/620=54.19 ≈ 54 person-months=> 7developers

Effort = Total Effort/6= 54/7= 7months=> Total Cost=Total Effort * Labor Rate = 54 * 8000≈ $43,2000+Way2=> Cost per LOC =Labor Rate/Productivity=8000/620=$12.9≈ $13

> Total Cost = Total LOC * Cost perLOC =33,600* 13=$436800

14. Narrate the importance of software specification of requirements.Explain a typical SRS structure and its parts.

The following are the importance of software specification of requirements

- The users and the client get a brief idea about the software while in the initial stages.
- The purposes and the intentions as well as the expected results are properly defined. It hence lays the outline for software design.
- The desired goals are defined thereby easing off the efforts of the developers in terms of time and cost.
- It forms a basis for the agreement between the client and the developer.
- It becomes easier while transferring and using the solution elsewhere or with new customers as the basis of functioning of the software is mentioned.
- It acts as a material for reference at a later stage.
- It acts as the basis for reviews.

Qualities of a good SRS

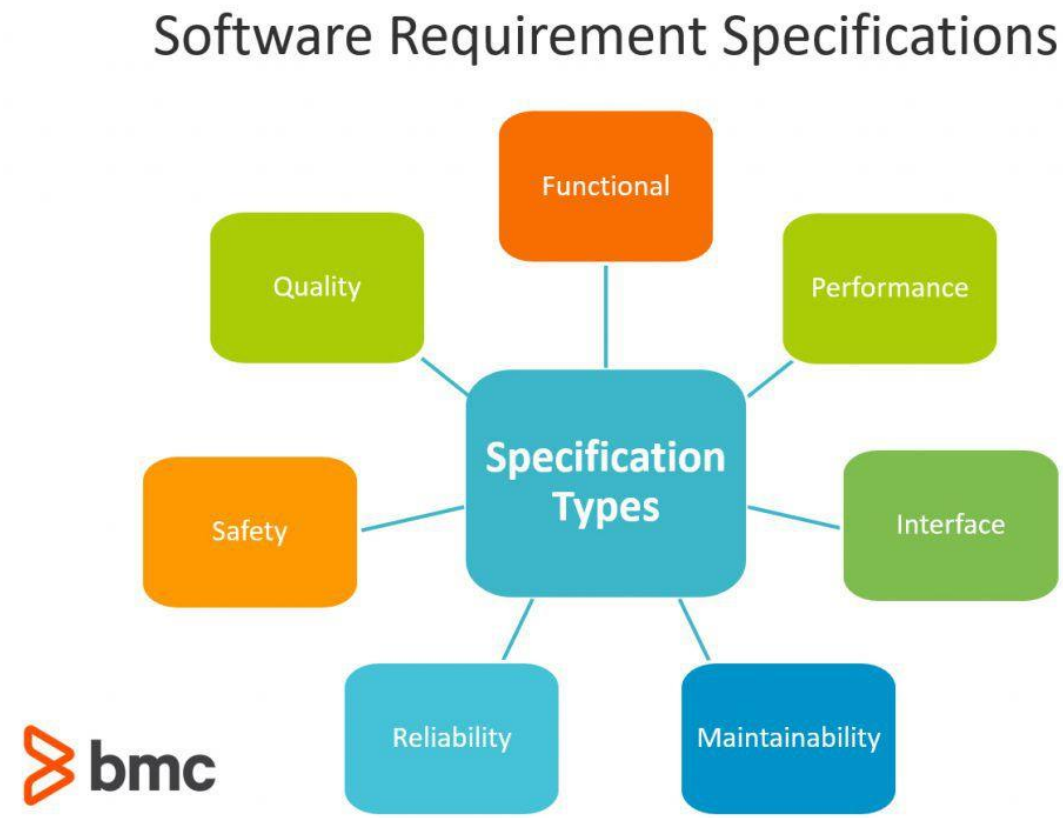
- Correctness: The SRS should be providing the correct scenario and should not just be given for the sake of giving a report.
- Unambiguous in nature: The SRS forms the basis of the agreement and is also tweaked later for reference and better understanding. Hence it should provide concrete information with respect to the scope, requirements and end results of the product so developed.
- Reliability of the document: The SRS should be reliable so that it could be passed on to the further customers of the clients or other branches of the client.
- Verifiability: The SRS should be verifiable i.e. the SRS forms the document that is referred to at the later stage to cross check the workings of the product so developed.

Basic structure of the SRS

The basic structure of any SRS could be:

1. Introduction: The introduction talks about the document and figures out the purpose of the document. It also gives the definitions, system overview and the references to be used while developing the product.
2. Overall description: The SRS should give an overall description with respect to the Product perspective, Product functions, user characteristics. it also briefs about the constraints of using the end product, the assumptions considered about the environment while developing the product and the dependencies if any.
   The SRS gives product perspectives with reference to the system interface, user interface, the hardware and software interfaces.
3. Specific requirements: The various requirements specific to external interface, functions, performance and database are described here. The attributes pertaining to the software system like reliability of the system, security and availability concerns, portability, transferability and maintainability are discussed here.

4. Other requirements: Any other requirements with respect to the software system also need.



The important parts of the Software Requirements Specification (SRS) document are:

Functional requirements of the system.
Non-functional requirements of the system.
Goals of implementation.

15.Eludicate in detail by constructing a context flow diagram level-0 DFD and level-1 DFD for a library management system with neat diagram.

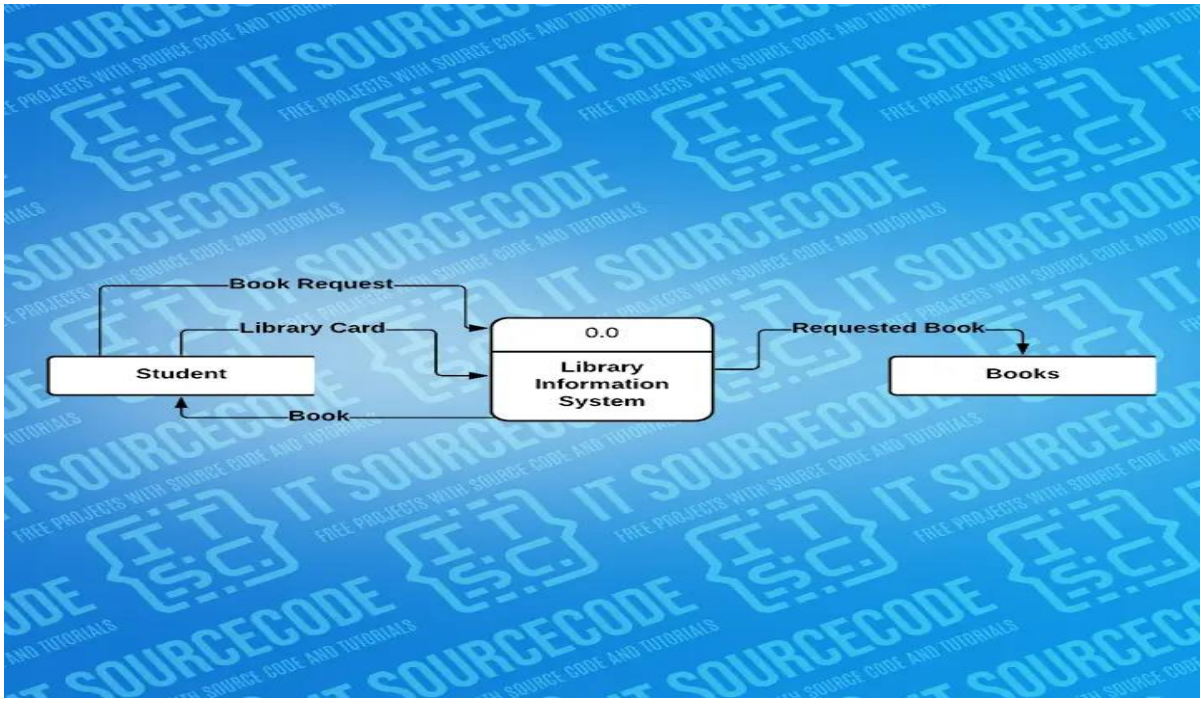
The **DFD for Library Management System** is a representation of the over all flow of data and the transformations made when data are process by the library management system.

In addition, It also represents and describes the **DFD for library information system** through input and output process.

For the **Library Management System**, the inputs include:

- Book Information
- Book Request
- Library Card

A **context diagram (level 0 data-flow diagram)** clarifies the library system's boundaries. It shows how information moves between the system and the external entities. A single process shows the whole concept of the software.



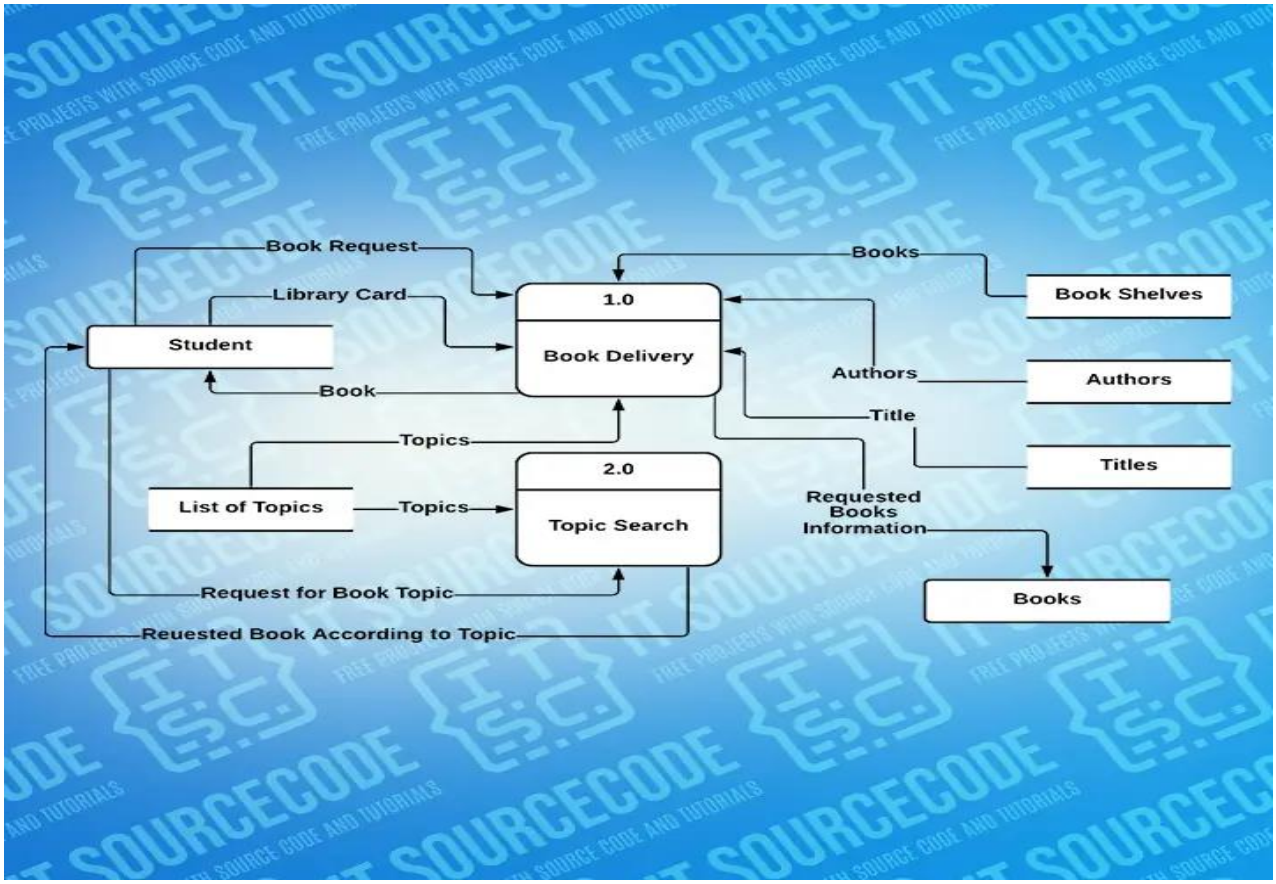
Library Management System DFD – Level 0

The diagram's arrowheads determine the direction of the data input that flows. **Students** (Borrowers) and **Books** are the external entities that cause the library system to deliver a function.

0.0 “Library Management System” is the label of the main process. This indicates that 0.0 is the basis of the preceding levels.

Generally, the library management system DFD level 0 is the starting point of the following diagrams.

A **1st level DFD of the Library** describes each of the major sub-processes that build the entire system. This level is the “expanded perspective” of the context diagram.



Library Management System DFD – Level 1

At this level, the single process is explained by stressing its sub-processes which include:

- Book Delivery
- Topic Search

Book delivery is the process where the system releases books according to the given request. The system will receive input from the external entity and then informs the main user.
Topic search/es will be performed by the system or by the librarian. This enables the admin to know if the requested book is available for purchase or borrowing. Either way, the system will provide an output.

The system data store (database) includes:

- Book Shelves
- Authors
- Titles

16.Describe about the characteristics of a good design. Also, discuss the different types of coupling and cohesion with its design evaluation performances.

For good quality software to be produced, the software design must also be of good quality. Now, the matter of concern is how the quality of good software design is measured? This is done by observing certain factors in software design. These factors are:

1. Correctness
2. Understandability
3. Efficiency
4. Maintainability

Now, let us define each of them in detail,

1) Correctness

First of all, the design of any software is evaluated for its correctness. The evaluators check the software for every kind of input and action and observe the results that the software will produce according to the proposed design. If the results are correct for every input, the design is accepted and is considered that the software produced according to this design will function correctly.

2) Understandability

The software design should be understandable so that the developers do not find any difficulty to understand it. Good software design should be self- explanatory. This is because there are hundreds and thousands of developers that develop different modules of the software, and it would be very time consuming to explain each design to each developer. So, if the design is easy and self- explanatory, it would be easy for the developers to implement it and build the same software that is represented in the design.

3) Efficiency

The software design must be efficient. The efficiency of the software can be estimated from the design phase itself, because if the design is describing software that is not efficient and useful, then the developed software would also stand on the same level of efficiency. Hence, for efficient and good quality software to be developed, care must be taken in the designing phase itself.

4) Maintainability

The software design must be in such a way that modifications can be easily made in it. This is because every software needs time to time modifications and maintenance. So, the design of the software must also be able to bear such changes. It should not be the case that after making some modifications the other features of the software start misbehaving. Any change made in the software design must not affect the other available features, and if the features are getting affected, then they must be handled properly.

Coupling: Coupling is the measure of the degree of interdependence between the modules. A good software will have low coupling.

Types of Coupling:

- **Data Coupling:** If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled. In data coupling, the components are independent of each other and communicate through data. Module communications don't contain tramp data. Example-customer billing system.
- **Stamp Coupling** In stamp coupling, the complete data structure is passed from one module to another module. Therefore, it involves tramp data. It may be necessary due to efficiency factors- this choice was made by the insightful designer, not a lazy programmer.
- **Control Coupling:** If the modules communicate by passing control information, then they are said to be control coupled. It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality. Example- sort function that takes comparison function as an argument.
- **External Coupling:** In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware. Ex- protocol, external file, device format, etc.
- **Common Coupling:** The modules have shared data such as global data structures. The changes in global data mean tracing back to all modules which access that data to evaluate the effect of the change. So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses, and reduced maintainability.
- **Content Coupling:** In a content coupling, one module can modify the data of another module, or control flow is passed from one module to the other module. This is the worst form of coupling and should be avoided.

Cohesion: Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.

Types of Cohesion:

- **Functional Cohesion:** Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.

- **Sequential Cohesion:** An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.
- **Communicational Cohesion:** Two elements operate on the same input data or contribute towards the same output data. Example- update record in the database and send it to the printer.
- **Procedural Cohesion:** Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.
- **Temporal Cohesion:** The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at unit time.
- **Logical Cohesion:** The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different.
- **Coincidental Cohesion:** The elements are not related(unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion. Ex- print next line and reverse the characters of a string in a single component.

17.Discuss in detail about the following:

(a)Black box testing

(b)Regression testing

(c)White box testing

(d)Integration testing

Black Box Testing

Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications. It is also known as Behavioral Testing.

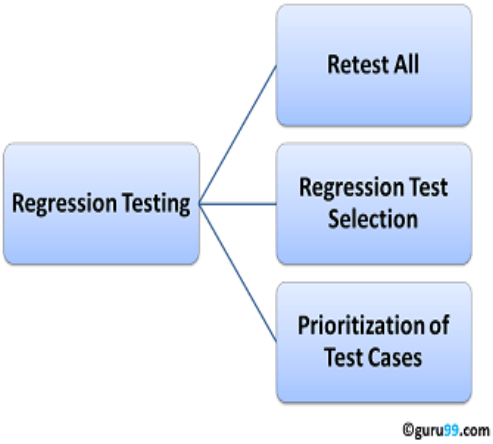
Black Box Testing Techniques

Following are the prominent Test Strategy amongst the many used in Black box Testing

- **Equivalence Class Testing:** It is used to minimize the number of possible test cases to an optimum level while maintains reasonable test coverage.
- **Boundary Value Testing:** Boundary value testing is focused on the values at boundaries. This technique determines whether a certain range of values are acceptable by the system or not. It is very useful in reducing the number of test cases. It is most suitable for the systems where an input is within certain ranges.
- **Decision Table Testing**: A decision table puts causes and their effects in a matrix. There is a unique combination in each column.

Regression Testing is defined as a type of software testing to confirm that a recent program or code change has not adversely affected existing features. Regression Testing is nothing but a full or partial selection of already executed test cases that are re-executed to ensure existing functionalities work fine.

- This testing is done to ensure that new code changes do not have side effects on the existing functionalities. It ensures that the old code still works once the latest code changes are done.



White Box Testing is a testing technique in which software's internal structure, design, and coding are tested to verify input-output flow and improve design, usability, and security. In white box testing, code is visible to testers, so it is also called Clear box testing, Open box testing, Transparent box testing, Code-based testing, and Glass box testing.
It is one of two parts of the Box Testing approach to software testing. Its counterpart, Blackbox testing, involves testing from an external or end-user perspective. On the other hand, White box testing in software engineering is based on the inner workings of an application and revolves around internal testing.

We have divided it into two basic steps to give you a simplified explanation of white box testing. This is what testers do when testing an application using the white box testing technique:

STEP 1) UNDERSTAND THE SOURCE CODE

The first thing a tester will often do is learn and understand the source code of the application. Since white box testing involves the testing of the inner workings of an application, the tester must be very knowledgeable in the programming languages used in the applications they are testing. Also, the testing person must be highly aware of secure coding practices. Security is often one of the primary objectives of testing software. The tester should be able to find security issues and prevent attacks from hackers and naive users who might inject malicious code into the application either knowingly or unknowingly.

STEP 2) CREATE TEST CASES AND EXECUTE

The second basic step to white box testing involves testing the application's source code for proper flow and structure. One way is by writing more code to test the application's source code. The tester will develop little tests for each process or series of processes in the application. This method requires that the tester must have intimate knowledge of the code and is often done by the developer. Other methods include Manual Testing, trial, and error testing and the use of testing tools as we will explain further on in this article.

Integration Testing is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated
Integration Testing focuses on checking data communication amongst these modules. Hence it is also termed as **'I & T'** (Integration and Testing), **'String Testing'** and sometimes **'Thread Testing'**.

Types of Integration Testing

Software Engineering defines variety of strategies to execute Integration testing, viz.

- Big Bang Approach:
- Incremental Approach: which is further divided into the following
  - Top-Down Approach
  - Bottom Up Approach
  - Sandwich Approach – Combination of Top Down and Bottom Up

18.Elobrate the need for software maintenance and maintenance report. Also, discuss the attributes of a good test and a test case design.

Software maintenance is the process of changing, modifying, and updating software to keep up with customer needs. Software maintenance is done after the product has launched for several reasons including improving the software overall, correcting issues or bugs, to boost performance, and more.

Software maintenance is a natural part of SDLC (software development life cycle). Software developers don’t have the luxury of launching a product and letting it run, they constantly need to be on the lookout to both correct and improve their software to remain competitive and relevant.

Using the right software maintenance techniques and strategies is a critical part of keeping any software running for a long period of time and keeping customers and users happy.

Why is software maintenance important?

Creating a new piece of software and launching it into the world is an exciting step for any company. A lot goes into creating your software and its launch including the actual building and coding, licensing models, marketing, and more. However, any great piece of software must be able to adapt to the times.

This means monitoring and maintaining properly. As technology is changing at the speed of light, software must keep up with the market changes and demands.

What are the 4 types of software maintenance?

The four different types of software maintenance are each performed for different reasons and purposes. A given piece of software may have to undergo one, two, or all types of maintenance throughout its lifespan.

The four types are:
Corrective Software Maintenance
Preventative Software Maintenance
Perfective Software Maintenance
Adaptive Software Maintenance
Corrective Software Maintenance

Corrective software maintenance is the typical, classic form of maintenance (for software and anything else for that matter). Corrective software maintenance is necessary when something goes wrong in a piece of software including faults and errors. These can have a widespread impact on the functionality of the software in general and therefore must be addressed as quickly as possible.

Many times, software vendors can address issues that require corrective maintenance due to bug reports that users send in. If a company can recognize and take care of faults before users discover them, this is an added advantage that will make your company seem more reputable and reliable (no one likes an error message after all).

Preventative Software Maintenance

Preventative software maintenance is looking into the future so that your software can keep working as desired for as long as possible.

This includes making necessary changes, upgrades, adaptations and more. Preventative software maintenance may address small issues which at the given time may lack significance but may turn into larger problems in the future. These are called latent faults which need to be detected and corrected to make sure that they won't turn into effective faults.

Perfective Software Maintenance

As with any product on the market, once the software is released to the public, new issues and ideas come to the surface. Users may see the need for new features or requirements that they would like to see in the software to make it the best tool available for their needs. This is when perfective software maintenance comes into play.

Perfective software maintenance aims to adjust software by adding new features as necessary and removing features that are irrelevant or not effective in the given software. This process keeps software relevant as the market, and user needs, change.

Adaptive Software Maintenance

Adaptive software maintenance has to do with the changing technologies as well as policies and rules regarding your software. These include operating system changes, cloud storage, hardware, etc. When these changes are performed, your software must adapt in order to properly meet new requirements and continue to run well.

Characteristics of a good Test

- Fast.
- Complete.
- Reliable.
- Isolated.
- Maintainable.
- Expressive.

An effective test case design will be:

- Accurate, or specific about the purpose.
- Economical, meaning no unnecessary steps or words are used.
- Traceable, meaning requirements can be traced.

- Repeatable, meaning the document can be used to perform the test numerous times.

19. OUT OF QUESTION

20.Illustrate the concept about CMM in detail with their five levels in detail with neat example.

CMM was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in 1987.

- It is not a software process model. It is a framework that is used to analyze the approach and techniques followed by any organization to develop software products.
- It also provides guidelines to further enhance the maturity of the process used to develop those software products.
- It is based on profound feedback and development practices adopted by the most successful organizations worldwide.
- This model describes a strategy for software process improvement that should be followed by moving through 5 different levels.
- Each level of maturity shows a process capability level. All the levels except level-1 are further described by Key Process Areas (KPA's).

Key Process Areas (KPA's):

Each of these KPA's defines the basic requirements that should be met by a software process in order to satisfy the KPA and achieve that level of maturity.

- Conceptually, key process areas form the basis for management control of the software project and establish a context in which technical methods are applied, work products like models, documents, data, reports, etc. are produced, milestones are established, quality is ensured and change is properly managed.

Level-1: Initial –

- No KPA's defined.
- Processes followed are Adhoc and immature and are not well defined.
- Unstable environment for software development.
- No basis for predicting product quality, time for completion, etc.

Level-2: Repeatable –

- Focuses on establishing basic project management policies.
- Experience with earlier projects is used for managing new similar natured projects.
- **Project Planning-** It includes defining resources required, goals, constraints, etc. for the project. It presents a detailed plan to be followed systematically for the successful completion of good quality software.
- **Configuration Management-** The focus is on maintaining the performance of the software product, including all its components, for the entire lifecycle.

- **Requirements Management-** It includes the management of customer reviews and feedback which result in some changes in the requirement set. It also consists of accommodation of those modified requirements.
- **Subcontract Management-** It focuses on the effective management of qualified software contractors i.e. it manages the parts of the software which are developed by third parties.
- **Software Quality Assurance-** It guarantees a good quality software product by following certain rules and quality standard guidelines while developing.

Level-3: Defined –

- At this level, documentation of the standard guidelines and procedures takes place.
- It is a well-defined integrated set of project-specific software engineering and management processes.
- **Peer Reviews-** In this method, defects are removed by using a number of review methods like walkthroughs, inspections, buddy checks, etc.
- **Intergroup Coordination-** It consists of planned interactions between different development teams to ensure efficient and proper fulfillment of customer needs.
- **Organization Process Definition-** Its key focus is on the development and maintenance of the standard development processes.
- **Organization Process Focus-** It includes activities and practices that should be followed to improve the process capabilities of an organization.
- **Training Programs-** It focuses on the enhancement of knowledge and skills of the team members including the developers and ensuring an increase in work efficiency.

Level-4: Managed –

- At this stage, quantitative quality goals are set for the organization for software products as well as software processes.
- The measurements made help the organization to predict the product and process quality within some limits defined quantitatively.
- **Software Quality Management-** It includes the establishment of plans and strategies to develop quantitative analysis and understanding of the product's quality.
- **Quantitative Management-** It focuses on controlling the project performance in a quantitative manner.

Level-5: Optimizing –

- This is the highest level of process maturity in CMM and focuses on continuous process improvement in the organization using quantitative feedback.
- Use of new tools, techniques, and evaluation of software processes is done to prevent recurrence of known defects.
- **Process Change Management-** Its focus is on the continuous improvement of the organization's software processes to improve productivity, quality, and cycle time for the software product.

- **Technology Change Management-** It consists of the identification and use of new technologies to improve product quality and decrease product development time.
- **Defect Prevention-** It focuses on the identification of causes of defects and prevents them from recurring in future projects by improving project-defined processes.