Q.2a. What are the key challenges being faced by software engineering? (3)

Answer: The key challenges facing software engineering are:
1. Coping with legacy systems, coping with increasing diversity and coping with demands for reduced delivery times
2. Legacy systems-Old, valuable systems must be maintained and updated.
3. Heterogeneity-Systems are distributed and include a mix of hardware and software.
4. Delivery-There is increasing pressure for faster delivery of software.
5. Trust – there is a need to trust that software

b. What is meant by risk management? Explain risk management process.
Answer: Risk management is concerned with identifying risks and drawing up plans to minimise their effect on a project. A risk is a probability that some adverse circumstance will occur. Risk Management process consists of:

- Risk identification-Identify project, product and business risks
- Risk analysis-Assess the likelihood and consequences of these risks
- Risk planning-Draw up plans to avoid or minimise the effects of the risk
- Risk monitoring-Monitor the risks throughout the project

With detailed description---------- **(8 Marks)**

c.What are the advantages of incremental development process?
Answer: The incremental development process has a number of advantages:
• Customers do not have to wait until the entire system is delivered before they can gain value from it. The first increment satisfies their most critical requirements so they can use the software immediately.

- Customers can use the early increments as prototype and gain experience that informs their requirements for later system increments.
- There is a lower risk of overall project failure. Although problems may be encountered in some increments, it is likely that some will be successfully delivered to the customer.
- As the highest priority services are delivered first, and later increments are integrated with them, it is inevitable that most important system services receive the most testing. This means that customers are less likely to encounter software failures in the most important parts of the system.

Q.3a. Explain the following terms giving suitable example: (3)
(i) Functional requirement
(ii) Non-functional requirement
(iii) Domain requirement

Answer:
(i) Functional requirements:
Statements of services the system should provide how the system should react to particular inputs and how the system should behave in particular situations.

(ii) Non-functional requirements:
Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
(iii) Domain requirements:
Requirements that come from the application domain of the system and that reflect characteristics of that domain.

b. What are the activities involved during the process of developing a formal specification of a sub-system interface?
Answer: The process of developing a formal specification of a sub-system interface includes the following activities:
• ***Specification structuring:*** Organise the informal interface specification into a set of abstract data types or object classes. One should informally define the operations associated with each class.
• ***Specification naming:*** Establish a name for each abstract type specification, decide whether they require generic parameters and decide on names for the sorts identified.
• ***Operation selection:*** Choose a set of operations for each specification based on the identified interface functionality. You should include operations to create instances of the sort, to modify the value of the instances and to inspect the instance values. You may have to add functions to those initially identified in the informal interface definition.
• ***Informal operation specification:*** Write an informal specification of each operation. You should describe how the operations affect the defined sort.
• ***Syntax definition:*** Define the syntax of the operations and the parameters to each. This is the signature part of the formal specification. You should update the informal specification at this stage if necessary.
Axiom definition: define the semantics of the operations by describing what conditions are always true for different operation combinations.

c. What is a distributed system? Identify & explain the advantages of using a distributed approach to systems developments.

Answer: Page 291 in 8th edition

Q.4a. What is Pair Programming? What are the advantages of pair programming?

Answer:
Pair programming is a concept in which programmers work in pairs to develop the software. They actually sit together at the same workstation to develop the software. Development does not actually involve the same pair of people working together. Rather, the idea is that pairs are created dynamically so that all team members may work with other members in a programming pair during the development process. The use of pair programming has a number of advantages:

• It supports the idea of common ownership and responsibility for the system. This reflects Weinberg's ides of egoless programming where the software is owned by the team as whole and

individuals are not held responsibility for problems with the code. Instead, the team has collective responsibility for resolving these problems.
• It acts as an informal review process because each line of code is looked at by at least two people. Code inspections and reviews are very successful in discovering a high percentage of software errors. However, they are time consuming to organise and, typically, introduce delays into the development process. While pair programming is a less formal process that probably doesn't find so many errors, it is a much cheaper inspection process than formal program inspections.

It helps support refactoring, which is a process of software improvement. A principle of extreme programming (XP) is that the software should be constantly refactored. That is, parts of the code should be rewritten to improve their clarity or structure. The difficulty of implementing this in a normal development environment is that this is effort that is expended for long-term benefit, and an individual who practices refactoring may be judged less efficient than one who simply carries on developing code. Where pair programming and collective ownership are used, others gain immediately from the refactoring so they are likely to support the process.

b. What is requirement elicitation and analysis in requirement engineering process? Why is it difficult to elicit and understand stakeholder requirement?

Answer: Page 170 in 8th edition

Q.5 a. Differentiate between two-tier Client Server approach and three-tier Client Server architecture.

Answer: 2-tier architecture In 2-tier, the application logic is either buried inside the User Interface on the client or within the database on the server (or both). With two tier client/server architectures, the user system interface is usually located in the user's desktop environment and the database management services are usually in a server that is a more powerful machine that services many clients **3-tier architecture** In 3-tier, the application logic (or) process lives in the middle-tier, it is separated from the data and the user interface. 3-tier systems are more scalable, robust and flexible. In addition, they can integrate data from multiple sources. In the three tier architecture, a middle tier was added between the user system interface client environment and the database management server environment. There are a variety of ways of implementing this middle tier, such as transaction processing monitors, message servers, or application servers. The middle tier can perform queuing, application execution, and database staging. For example, if the middle tier provides queuing, the client can deliver its request to the middle layer and disengage because the middle tier will access the data and return the answer to the client The most basic type of three tier architecture has a middle layer consisting of Transaction Processing (TP) monitor technology. The TP monitor technology is a type of message queuing, transaction scheduling, and prioritization service where the client connects to the TP monitor (middle tier) instead of the database server. The transaction is accepted by the monitor, which queues it and then takes responsibility for managing it to completion, thus freeing up the client.

b.Describe design walk throughs and critical design review.

Answer: A design walkthrough is a quality practice that allows designers to obtain an early validation of design decisions related to the development and treatment of content, design of the graphical user interface, and the elements of product functionality. Design walkthroughs provide designers with a way to identify and assess early on whether the proposed design meets the requirements and addresses the project's goal.

For a design walkthrough to be effective, it needs to include specific components.

The following guidelines highlight these key components. Use these guidelines to plan, conduct, and participate in design walkthroughs and increase their effectiveness.

1. Plan for a Design Walkthrough
2. Get the Right Participants
3. Understand Key Roles and Responsibilities
4. Prepare for a Design Walkthrough
5. Use a Well-Structured Process
6. Review and Critique the Product, not the Designer
7. Review, do not Solve Problems

The purpose of critical design review is to ensure that the detailed design satisfies the specifications laid down during system design. The critical design review process is same as the inspection process in which a group of people gets together to discuss the design with the aim of revealing design errors or undesirable properties.

c.What is stepwise refinement? Discuss partitioning & abstraction.

Answer: Stepwise Refinement:-Stepwise Refinement is a top-down design strategy originally proposed by Niklaus Wirth. A program is developed by successively refining levels of procedural detail. A hierarchy is developed by decomposing a macroscopic statement of function in a stepwise fashion until programming language statements are reached.

Refinement is actually a process of elaboration. We begin with a statement of function that is defined at a high level of abstraction. That is, the statement describes function or information conceptually but provides no information about the internal workings of the function or the internal structure of the information. Refinement causes the designer to elaborate on the original statement, providing more and more detail as each successive refinement occurs.

Partitioning:- Problems are often too large and complex to be understood as a whole.

For this reason, we tend to partition such problems into parts that can be easily understood and establish interfaces between the parts so that overall function can be accomplished.

Partitioning decomposes a problem into its constituent parts. We establish a hierarchical representation of function or information and then partition the uppermost element by:-

1) Exposing increasing detail by moving vertically in the hierarchy or
2) Functionality decomposing the problem by moving horizontally in the hierarchy.

Abstraction: - Abstraction permits one to concentrate on a problem at some level of generalization without regard to irrelevant low level details; use of abstraction also permits one

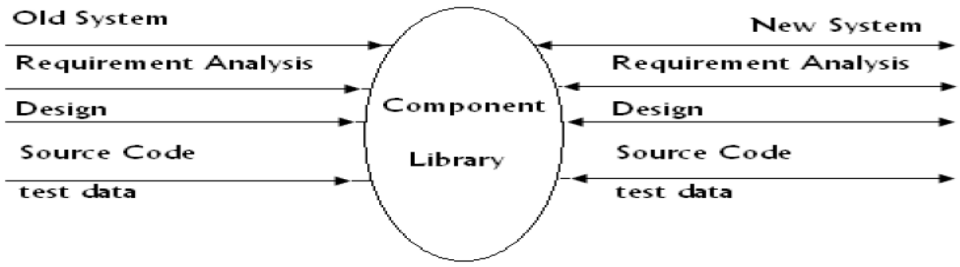
to work with concepts and terms that are familiar in the problem environment without having to transform them to an unfamiliar structure.
• It allows considering the modules at the abstract level without worrying about its details.
• It provides external behavior of the modules.
• It is used for existing modules as well as for modules that are being design.
• It is essential for the problem partitioning.

Q.6a. Explain the reuse maintenance model with the help of a diagram. (6)

Answer: Reuse maintenance model: This model is based on the principal that maintenance could be viewed as an activity involving the reuse of existing program components.

The reuse model has four main steps:

1. Identification of the parts of the old system that are candidates for reuse.
2. Understanding these



b.Explain the following Software Metrics
(i) Lines of Code (ii) Function Count (iii) Token Count

Answer: (i) Lines of code (LOC) is software metric used to measure the size of a software program by counting the number of lines in the text of the program's source code. LOC is typically used to predict the amount of effort that will be required to develop a program, as well as to estimate programming productivity or effort once the software is produced. Advantages:-

1. Scope for Automation of Counting: Since Line of Code is a physical entity; manual counting effort can be easily eliminated by automating the counting process. Small utilities may be developed for counting the LOC in a program. However, a code counting utility developed for a specific language cannot be used for other languages due to the syntactical and structural differences among languages.
2. An Intuitive Metric: Line of Code serves as an intuitive metric for measuring the size of software due to the fact that it can be seen and the effect of it can be visualized. Function Point is more of an objective metric which cannot be imagined as being a physical entity, it exists only in the logical space. This way, LOC comes in handy to express the size of software among programmers with low levels of experience.

Disadvantages

1. Lack of Accountability: Lines of code measure suffers from some fundamental problems. Some think it isn't useful to measure the productivity of a project using only results from the coding phase, which usually accounts for only 30% to 35% of the overall effort.
2. Lack of Cohesion with Functionality: Though experiments have repeatedly confirmed that effort is highly correlated with LOC, functionality is less well correlated with LOC. That is, skilled developers may be able to develop the same functionality with far less code, so one program with less LOC may exhibit more functionality than another similar program. In particular, LOC is a poor productivity measure of individuals, because a developer who develops only a few lines may still be more productive than a developer creating more lines of code.
3. Adverse Impact on Estimation: Because of the fact presented under point (a), estimates based on lines of code can adversely go wrong, in all possibility.
4. Developer's Experience: Implementation of a specific logic differs based on the level of experience of the developer. Hence, number of lines of code differs from person to person. An experienced developer may implement certain functionality in fewer lines of code than another developer of relatively less experience does, though they use the same language.

(ii) Function count:- It measures the functionality from the user point of view, that is, on the basis of what the user requests and receives in return. Therefore it deals with the functionality being delivered, and not with the lines of code, source modules, files etc. measuring size is in this way has the advantage that size measure is independent of the technology used to deliver the function.

Features:-

• Function point approach is independent of the language tools, or methodologies used for implementation.
• They can be estimated from requirement specification or design specification.
• They are directly linked to the statement of requirements.
• They are based on the system user's external view of the system; nontechnical users of the software system have a better understanding of what function points are measuring.

(iii) Token count:- A program is considered to be series of tokens and if we count the number of tokens, some interesting results may emerge. Tokens are classified as either operators or operands. All software science measures are functions of the counts of these tokens. Variables, constants and even labels are operands. Operators consist of arithmetic symbols such as +, -. /, * and command names such as "while", "for", "printf", special symbols such as : =, braces, parentheses, and even function names such as "eof".

The size of the vocabulary of a program, which consists of the number of unique tokens to build a program, is defined as:-

$n = n1 + n2$

Where, n : vocabulary of a program.

n1: number of unique operators

n2: number of unique operands.

c. Differentiate between function oriented design and object oriented design.

Answer: Function oriented design:- Function oriented design strategy relies on decomposing the system into a set of interacting functions with a centralized system state shared by these functions. Functions may also maintain local state information but only for the duration of their execution. Function oriented design conceals the details of an algorithm in a function but system state information is not hidden.

Object oriented design:-Object oriented design transforms the analysis model created using object-oriented analysis into a design model that serves as a blueprint for software construction. It is a design strategy based on information hiding. Object oriented design is concerned with developing an object-oriented model of a software system to implement the identified requirements. Object oriented design establishes a design blueprint that enables a software engineer to define object oriented architecture in a manner that maximizes reuse, thereby improving development speed and end product quality.

Object oriented design Vs function oriented design-

- Unlike function oriented design methods, in OOD, the basic abstractions are not real world function such as sort, display, track etc. but real world entities such as employee, picture, machine etc.
- In OOD, state information is not represented in a centralized shared memory but is distributed among the objects of the system.

Q.7 a. What are essentials of a Component Based Software engineering? List few problems associated with CBSE.

Ans Page 464-465 in 8th edition

b.Explain the various ways in which object classes can be identified in the object identification stage of object-oriented design.

Ans Page 350-351 in 8th edition

Q.8 a. Explain various types of static and dynamic testing tools.

Answer: Static testing tools:

1. **Static analysers** A static analyser operates from a pre-computed database of descriptive information derived from the source text of the program. The idea of a static analyser is to provide allegations, which are claims about the analysed programs that can be demonstrated by systematic examination of all cases.
2. **Code inspectors** A code inspector does a simple job of enforcing standards in a uniform way for many programs. These can be single statement or multiple statement rules. The AUDIT system is available which imposes some minimum conditions on the programs.

3. **Standard enforces** This tool is like a code inspector; expect that the rules are generally simpler. The main distribution is that a full-blown static analyser looks at whole programs, whereas a standard enforcer looks at only single statements.

Dynamic testing tools

1**. Coverage analyzers (execution verifiers)** A coverage analyzer is the most common and important tool for testing. It is often relatively simple. One of the common strategies for testing involves declaring the minimum level of coverage, ordinarily expressed as a percentage of the elemental segments that are exercised in aggregate during the testing process.

2. **Output comparators** -These are used in dynamic testing-both single-module and multiple-module (system level) varieties to check that predicted and actual outputs are equivalent. This is also done during regression testing.

3. **Test data generators**This is a difficult one, and at least for the present is one for which no general solution exists. One of the practical difficulties with test data generation of sets of inequalities that represent the condition along a chosen path.

4. **Test file generators** This creates a file of information that is used as the program and does so based on comments given by the user and/or from the data descriptions program's data definition section.

5. **Test harness systems**This is one that is bound around the test object and that permits the easy modification and control of test inputs and output's and provides for online measurement of CI coverage values.

6. **Test archiving systems:** The goal is to keep track of series of tests ant to act as the basis for documenting that the tests have been done and that no defects were found during the process.

b.Differentiate between failures and faults.

Ans 8(b): Failure:-Failure is the departure of external results of program operation from requirements. So failure is something dynamic. Failure can also be defined as deficiency in performance, attributes and excessive response time.

There are four general ways of characterizing failure occurrence in time:-

1) Time of failure.

2) Time interval between failures.

3) Cumulative failures experienced up to a given time.

4) Failure experienced in a time interval.

Various failure classes are transient, permanent, recoverable, unrecoverable, noncorrupting, and corrupting.

Fault:- fault is the defect in the program that, when executed under particular condition, causes of failure. A fault is a property of the program rather than a property of its execution or behavior.

Various fault classes are data faults, control faults, input/output faults, interface faults, storage management faults, and exception management faults.

c.What do you understand by black box testing? Explain: (8)

(i) Equivalence class (ii) Equivalence partitioning

Answer: Black Box Testing:-Black Box Testing is also called behavioral testing, focuses on the functional requirements of the software. It enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program. It is a complementary approach that is likely to uncover a different class of errors than white box testing.

Black Box Testing attempts to find errors in the following categories:-
1) Incorrect or missing functions.
2) Interface errors.
3) Errors in data structures or external database access.
4) Behavior or performance errors.
5) Initialization and termination errors.

Black Box Testing tends to be applied during later stages of testing because black box testing purposely disregards control structures; attention is focused on the information domain.
Equivalence class:-It represents a set of valid or invalid states for input conditions. An input condition is a specific numeric value, a range of values, a set of related values, or a Boolean condition. Equivalence class may be defined according to the following guidelines:-
1) If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
2) If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
3) If an input condition specifies a member of a set, one valid and one invalid equivalence class are defined.
4) If an input condition is Boolean, one valid and one invalid class are defined.
Equivalence Partitioning:-Equivalence partitioning is black box testing method that divides the input domain of a program into classes of data from which test cases can be derived. An ideal test case single-handedly uncovers a class of errors that might otherwise require many cases to be executed before the general error is observed.

Equivalence partitioning strives to define test case that uncovers classes of errors, thereby reducing the total number of test case that must be developed.

Q.9 a. What is SQA? Discuss different software quality factors.

Answer: Software quality assurance (SQA) consists of a means of monitoring the software engineering processes and methods used to ensure quality. The methods by which this is accomplished are many and varied, and may include ensuring conformance to one or more standards, such as ISO 9000 or a model such as CMMI A software quality factor is a non-functional requirement for a software program which is not called up by the customer's contract, but nevertheless is a desirable requirement which enhances the quality of the software program. Note that none of these factors are binary; that is, they are not “either you have it or you don’t” traits. Rather, they are characteristics that one seeks to maximize in one’s software to optimize

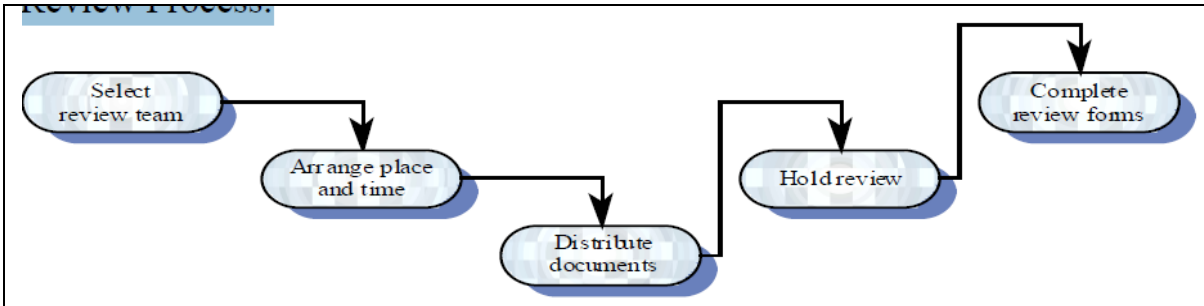
its quality. So rather than asking whether a software product "has" factor *x*, ask instead the *degree* to which it does (or does not). Some software quality factors are listed here:
Understandability: Clarity of purpose. This goes further than just a statement of purpose; all of the design and user documentation must be clearly written so that it is easily understandable. This is obviously subjective in that the user context must be taken into account: for instance, if the software product is to be used by software engineers it is not required to be understandable to the layman.
Completeness: Presence of all constituent parts, with each part fully developed. This means that if the code calls a subroutine from an external library, the software package must provide reference to that library and all required parameters must be passed. All required input data must also be available.
Conciseness: Minimization of excessive or redundant information or processing. This is important where memory capacity is limited, and it is generally considered good practice to keep lines of code to a minimum. It can be improved by replacing repeated functionality by one subroutine or function which achieves that functionality. It also applies to documents.

b.What do you understand by software configuration? Differentiate among release, version and revision of a software product.

Answer: The results (also called as the deliverables) of a large software development effort typically consist of a large number of objects, e.g. source code, design document, SRS document, test document, user's manual, etc. These objects are usually referred to and modified by a number of software engineers throughout the life cycle of the software. The state of all these objects at any point of time is called the configuration of the software product. The states of each deliverable object changes as development progresses and also as bugs are detected and fixed. Release vs. Version vs. Revision A new version of software is created when there is a significant change in functionality, technology, or the hardware it runs on, etc. On the other hand a new revision of software refers to minor bug fix in that software. A new release is created if there is only a bug fix, minor enhancements to the functionality, usability, etc.

c. Write a short note on software quality review and review process

Answer: Software quality review: A group of people carefully examine part or all of a software system and its associated documentation. Code, designs, specifications, test plans, standards, etc. can all be reviewed. Software or documents may be 'signed off' at a review, which signifies that progress to the next development stage has been approved by management. Review Process:



(4 marks)

.

Text book

1. Software Engineering, Ian Sommerville, 7th edition, Pearson Education, 2004