Q2a. What are the key challenges being faced by software engineering?

Ans 2a. 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.

b. What is meant by risk management? Explain risk management process.

Ans 2b. 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

c. What are the attributes of good software?

Ans 2c. The software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.

- Maintainability-Software must evolve to meet changing needs
- Dependability-Software must be trustworthy
- Efficiency-Software should not make wasteful use of system resources
- Usability-Software must be usable by the users for which it was designed

Q3a. Explain the following terms giving suitable example:

(i) Functional requirement (ii) Non-functional requirement
(iii) Domain requirement

Ans 3a. (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 is meant by an object model? Discuss Inheritance model, Aggregation model and Interaction model in brief.

Ans 3b. Object models describe the system in terms of object classes. An object class is an abstraction over a set of objects with common attributes and the services (operations) provided by each object
Various object models may be produced.

Inheritance models:

- Organise the domain object classes into a hierarchy.
- Classes at the top of the hierarchy reflect the common features of all classes.
- Object classes inherit their attributes and services from one or more super-classes. These may then be specialised as necessary.
- Class hierarchy design is a difficult process if duplication in different branches is to be avoided.

Aggregation models:

- Aggregation model shows how classes which are collections are composed of other classes.
- Similar to the part-of relationship in semantic data models.

Q 4a. List the benefits of prototyping. Differentiate between the objectives of evolutionary and throw-away prototyping.

Ans 4a. Benefits of prototyping:

- Misunderstandings between software users and developers are exposed
- Missing services may be detected and confusing services may be identified
- A working system is available early in the process
- The prototype may serve as a basis for deriving a system specification
- The system can support user training and system testing

The objective of *evolutionary prototyping* is to deliver a working system to end-users. The development starts with those requirements which are best understood.

The objective of *throw-away prototyping* is to validate or derive the system requirements. The prototyping process starts with those requirements which are poorly understood.

Q 5a. Describe object request brokers and the principles underlying the CORBA.

Ans 5a. The ORB handles object communications. It knows of all objects in the system and their interfaces. Using an ORB, the calling object binds an IDL stub that defines the interface of the called object. Calling this stub results in calls to the ORB which then calls the required object through a published IDL skeleton that links the interface to the service implementation
CORBA Services:

Naming and trading services

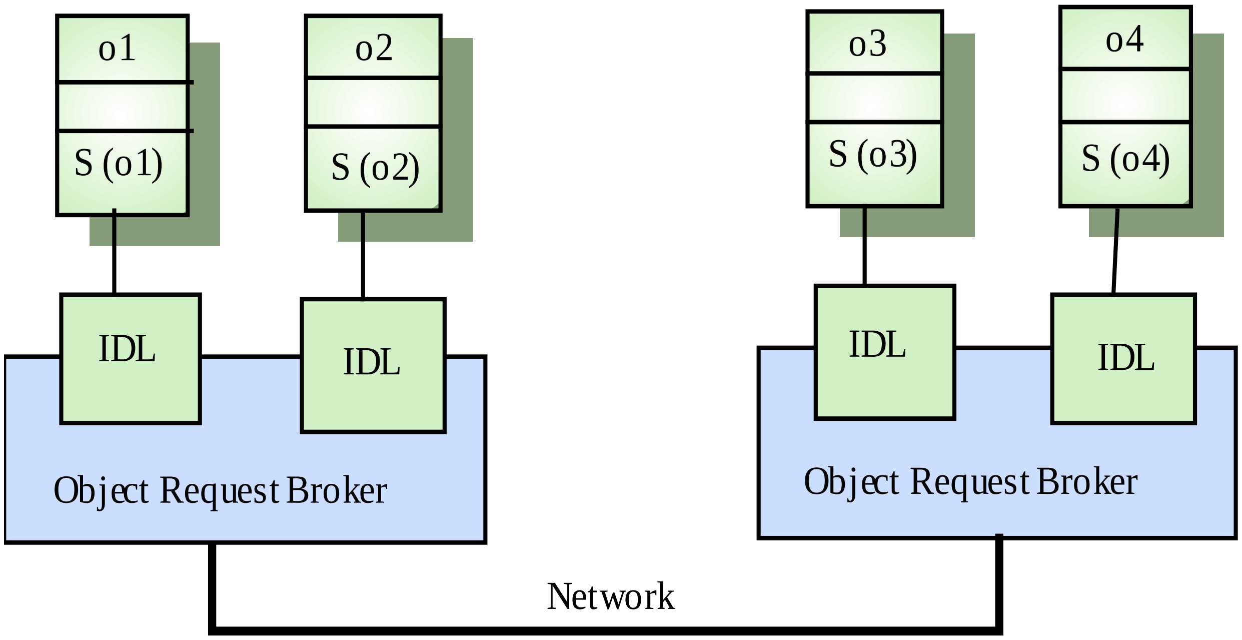
- These allow objects to discover and refer to other objects on the network

Notification services

- These allow objects to notify other objects that an event has occurred

Transaction services

- These support atomic transactions and rollback on failure
-



c. What do you mean by domain specific architectural model? Differentiate between two types of domain specific models.

Ans 5c. Architectural models which are specific to some application domain are called domain specific model.

Two types of domain-specific model are:

- Generic models which are abstractions from a number of real systems and which encapsulate the principal characteristics of these systems
- Reference models which are more abstract, idealised model. Provide a means of information about that class of system and of comparing different architectures

Generic models are usually bottom-up models; Reference models are top-down models

Q 6a. What are various abstractions possible in Component Based Software engineering? List few problems associated with CBSE.

Ans 6a. Component-based software engineering (CBSE) is an approach to software development that relies on reuseIt emerged from the failure of object-oriented development to support effective reuse. Single object classes are too detailed and specificComponents are more abstract than object classes and can be considered to be stand-alone service providers.

Various abstractions:

Functional abstraction -The component implements a single function such as a mathematical function.

Casual groupings -The component is a collection of loosely related entities that might be data declarations, functions, etc.

Data abstractions -The component represents a data abstraction or class in an object-oriented language.

Cluster abstractions -The component is a group of related classes that work together.

System abstraction -The component is an entire self-contained system.

b. What is meant by design patterns? What are the advantages of using design patterns?

Ans 6b. Design patterns are reusable solutions to problems that recur in many applications. A pattern serves as a guide for creating a “good” design. Patterns are based on sound common sense and the application of fundamental design principles. These are created by people who spot repeating themes across designs. The pattern solutions are typically described in terms of class and interaction diagrams. Examples of design patterns are expert pattern, creator pattern, controller pattern etc.

Design patterns are very useful in creating good software design solutions. In addition to providing the model of a good solution, design patterns include a clear specification of the problem, and also explain the circumstances in which the solution would and would not work. Thus, a design pattern has four important parts:

- The problem.
- The context in which the problem occurs.
- The solution.
- The context within which the solution works.

Q7a. Define the following terms with respect to UI design principles:

(i) User Familiarity	(ii) Consistency
(ii) Minimal surprise	(iv) Recoverability
(v) User Guidance	(vi) User Diversity

Ans 7a.

Principle	**Description**
User familiarity	The interface should use terms and concepts which are drawn from the experience of the people who will make most use of the system.
Consistency	The interface should be consistent in that, wherever possible, comparable operations should be activated in the same way.
Minimal surprise	Users should never be surprised by the behaviour of a system.
Recoverability	The interface should include mechanisms to allow users to recover from errors.
User guidance	The interface should provide meaningful feedback when errors occur and provide context-sensitive user help facilities.
User diversity	The interface should provide appropriate interaction facilities for different types of system user.

b. What do you mean by fault tolerance? Where is fault tolerance required?

Ans 7b. Fault tolerance means that the system can continue in operation in spite of software failure. Even if the system seems to be fault-free, it must also be fault tolerant as there may be specification errors or the validation may be incorrect.
In critical situations, software systems must be fault tolerant. Fault tolerance is required where there are high availability requirements or where system failure costs are very high.

c. Differentiate between forward and backward fault recovery techniques.

Ans 7c. Forward recovery-Apply repairs to a corrupted system state

Backward recovery-Restore the system state to a known safe state

Forward recovery is usually application specific - domain knowledge is required to compute possible state corrections

Backward error recovery is simpler. Details of a safe state are maintained and this replaces the corrupted system state

Q 8a. Write a brief note on the following estimation techniques:
(i) Algorithmic cost modelling
(ii) Expert judgement
(iii) Estimation by analogy
(iv) Parkinson's Law
(v) Pricing to win

Ans 8a. Algorithmic: A formulaic approach based on historical cost information and which is generally based on the size of the software.

Expert Judgement: One or more experts in both software development and the application domain use their experience to predict software costs. Process iterates until some consensus is reached.

Estimation by analogy: The cost of a project is computed by comparing the project to a similar project in the same application domain.

Parkinson's Law: The project costs whatever resources are available. Advantages are No overspend.

Pricing to win: The project costs whatever the customer has to spend on it Advantage is that you get the contract.

b. Write some guidelines for interface testing

Ans 8b. Some guidelines for interface testing:

- Design tests so that parameters to a called procedure are at the extreme ends of their ranges.
- Always test pointer parameters with null pointers.
- Design tests, which cause the component to fail.
- Use stress testing in message passing systems.
- In shared memory systems, vary the order in which components are activated.

Q9a. What is the main purpose of SEI Capability Maturity Model (SEI CMM)? Explain five different levels of SEI CMM model.

Ans 9a. SEI CMM can be used two ways: capability evaluation and software process assessment. Capability evaluation and software process assessment differ in motivation, objective, and the final use of the result. Capability evaluation provides a way to assess the software process capability of an organization. The results of capability evaluation indicates the likely contractor performance if the contractor is awarded a work. Therefore, the results of software process capability assessment can be used to select a contractor. On the other hand, software process assessment is used by an organization with the objective to improve its process capability.

Five levels of SEI CMM model:

1. Initial-Essentially uncontrolled
2. Repeatable-Product management procedures defined and used
3. Defined-Process management procedures and strategies defined and used
4. Managed-Quality management strategies defined and used
5. Optimising-Process improvement strategies defined and used

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

Ans 9b. 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 through out 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 brief note on software quality review and review process.

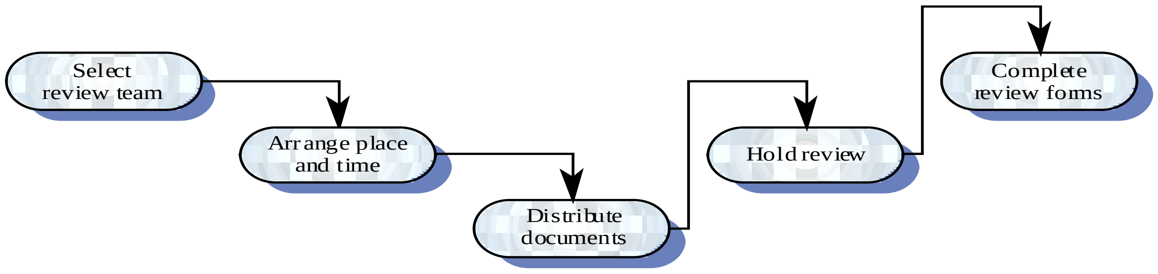
Ans 9c. Software quality review:

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:



TEXTBOOK

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