18CS/IT 501

Hall Ticket Number:

IV/IV B.Tech (Regular / Supplementary) DEGREE EXAMINATION

January, 2021 **Common to CSE and IT**

Fifth Semester **SOFTWARE ENGINEERING**

Time: Three Hours **Maximum :** 60 Marks

Answer ALL Questions from PART-A. (1X10 = 10 Ma
*Answer **ANY FOUR** questions from PART-B.* (4X10=48 Ma

Part - A

1 Answer all questions (1X10=10 Marks)

a) What are goals/objectives of software engineering?
The **goal** for **software** development can be translated, in my opinion, to: raising the throughput, the amount of features delivered (deployed, not implemented or tested) in the unit of time. You can measure this amount in story points, since feature vary in size.

b) State the design principles?
It is combination of five basic designing principles.
- Single Responsibility Principle (SRP) ...
- Open/Closed Principle (OCP) ...
- Liscov Substitution Principle (LSP) ...
- **Interface Segregation Principle** (ISP) ...
- **Dependency Inversion Principle** (DIP)

c) Define Scrum.
Scrum is a framework for project management that emphasizes teamwork, accountability and iterative progress toward a well-defined goal. The framework begins with a simple premise: Start with what can be seen or known. After that, track the progress and tweak as necessary.

d) Define quality assurance ?
Quality assurance can be defined as "part of **quality** management focused on providing confidence that **quality** requirements will be fulfilled." The confidence provided by **quality assurance** is twofold—internally to management and externally to customers, government agencies, regulators, certifiers, and third parties.

e) Write about negotiating requirements.
Validation and **negotiation** during **requirements engineering** is meant to ensure that the documented **requirements** meet the predetermined quality criteria, such as correctness and agreement

f) What is Usecase scenario?
A **use case** represents the actions that are required to enable or abandon a goal. A **use case** has multiple "paths" that can be taken by any user at any one time. A **use case scenario** is a single path through the **use case**.

g) How to create good software.
Good design relies on a combination of high-level systems thinking and low-level component knowledge. In modern **software design**, best practice revolves around creating modular components that you can call and deploy as needed.

h) Define Component.
component provides a particular function or group of related functions.

i) What is abstraction
Abstraction is one of the fundamental concepts of **software engineering**. It is all about hiding complexity in building various parts of your application.

j) Define LOC..
lines of code (SLOC), also known as **lines of code** (**LOC**), is a **software** metric used to measure the size of a computer program by counting the number of lines in the text of the program's source code.

Part - B

2 Explain the following process models and also write their advantages and disadvantages 1

i) Spiral model

Definition: The spiral model is similar to the incremental development for a system, with more emphasis placed on risk analysis. The spiral model has four phases: Planning, Design, Construct and Evaluation. A software project repeatedly passes through these phases in iterations (called Spirals in this model).

Description: These phases are - Planning: This phase starts with the gathering of business requirements. In the subsequent spirals as the product matures, identification of system requirements and unit requirements are done in this phase. This also includes understanding of system requirements by continual communication between the customer and the analyst. At the end of the spiral the product is deployed.

ii) Rapid application development model

Definition: The Rapid Application Development (or RAD) model is based on prototyping and iterative model with no (or less) specific planning. In general, RAD approach to software development means putting lesser emphasis on planning tasks and more emphasis on development and coming up with a prototype. In disparity to the waterfall model, which emphasizes meticulous specification and planning, the RAD approach means building on continuously evolving requirements, as more and more learnings are drawn as the development progresses.

Description: RAD puts clear focus on prototyping, which acts as an alternative to design specifications. This means that RAD works well wherever there's a greater focus on user interface rather than non-GUI programs. The RAD model includes agile method and spiral model.

Below phases are in rapid application development (RAD) model:

1. Business modeling: The information flow is identified between different business functions.

2. Data modeling: Information collected from business modeling is used to define data objects that are required for the business.

3. Process modeling: Data objects defined in data modeling are converted to establish the business information flow to achieve some specific business objective process descriptions for adding, deleting, modifying data objects that are given.

4. Application generation: The actual system is created and coding is done by using automation tools. This converts the overall concept, process and related information into actual desired output. This output is called a prototype as it's still half-baked.

5. Testing and turnover: The overall testing cycle time is reduced in the RAD model as the prototypes are independently tested during every cycle.

3 a) Explain different phases of unified process 5

The Unified Process divides the project into four phases: 5

- **Inception**.
- **Elaboration** (milestone)
- **Construction** (release)
- Transition (final production release)

b) Discuss different software myths.

Myth 1: **Testing** is Too Expensive. ...

Myth 2: **Testing** is Time-Consuming. ...

Myth 3: Only Fully Developed Products are Tested. ...

Myth 4: Complete **Testing** is Possible. ...

Myth 5: A Tested **Software** is Bug-Free. ...
Myth 6: Missed Defects are due to Testers.

4

a) Explain analysis modelling and design modelling principles

Analysis Model is a technical representation of the system. It acts as a link between system description and design **model**. In **Analysis Modelling**, information, behavior and functions of the system is defined and translated into the architecture, component and interface level design in the design **modeling**.

A **design model in software engineering** is an object-based picture or pictures that represent the use cases for a system. Or to put it another way, it's the means to describe a system's implementation and source code in a diagrammatic fashion. This type of representation has a couple of advantages. 5M

b) Discuss about eliciting requirements 5M

The **work product** created as a result of **requirement** elicitation that is depending on the size of the system or **product** to be built. The **work product** consists of a statement need, feasibility, statement scope for the system. It also consists of a list of users participate in the **requirement** elicitation.

Some of the requirement elicitation techniques are as follows.

- Document analysis.
- Observation.
- Interview.
- Prototyping.
- Brainstorming.
- Workshop.
- JAD (Joint Application Development)
- Reverse engineering

5 a) Explain different conventional components for design 5

Also referred to as atomic **design** (we prefer "**Component design**" here at Praxent), **component design's** definition refers to the process of building a digital product or website in pieces. The pieces are the page elements like the header, the search form, and the sidebar call to action, etc.

b) What is DFD? Explain different levels of DFD's with neat diagram? 5

DFD levels are numbered 0, 1 or 2, and occasionally go to even **Level** 3 or beyond. The necessary **level** of detail depends on the scope of what you are trying to accomplish. **DFD Level** 0 is also called a Context Diagram. **It's** a basic overview of the whole system or process being analyzed or modeled.

6 a) Explain cohesion and coupling methods. 5

Coupling shows the relationships between modules. **Cohesion** shows the relationship within the module. **Coupling** shows the relative independence between the modules. **Cohesion** shows the module's relative functional strength. While creating, you should aim for low **coupling**, i.e., dependency among modules should be less.

b) What are the types of golden rules explain?. 5

1 Strive for consistency.

Consistent sequences of actions should be required in similar situations; identical terminology should be used in prompts, men and help screens; and consistent commands should be employed throughout.

2 Enable frequent users to use shortcuts.

As the frequency of use increases, so do the user's desires to reduce the number of interactions and to increase the pace of interaction. Abbreviations, function keys, hidden commands, and macro facilities are very helpful to an expert user.

3 Offer informative feedback.

For every operator action, there should be some system feedback. For frequent and minor actions, the response can be modest while for infrequent and major actions, the response should be more substantial.

4 Design dialog to yield closure.

Sequences of actions should be organized into groups with a beginning, middle, and end. The informative feedback at the

completion of a group of actions gives the operators the satisfaction of accomplishment, a sense of relief, the signal to drop contingency plans and options from their minds, and an indication that the way is clear to prepare for the next group of actions

5 Offer simple error handling.

As much as possible, design the system so the user cannot make a serious error. If an error is made, the system should be able detect the error and offer simple, comprehensible mechanisms for handling the error.

6 Permit easy reversal of actions.

This feature relieves anxiety, since the user knows that errors can be undone; it thus encourages exploration of unfamiliar options. The units of reversibility may be a single action, a data entry, or a complete group of actions.

7 Support internal locus of control.

Experienced operators strongly desire the sense that they are in charge of the system and that the system responds to their acti Design the system to make users the initiators of actions rather than the responders.

8 Reduce short-term memory load.

The limitation of human information processing in short-term memory requires that displays be kept simple, multiple page displays be consolidated, window-motion frequency be reduced, and sufficient training time be allotted for codes, mnemonics and sequences of actions.

7 Explain software architecture? 1

The top 5 software architecture patterns: How to make the right choice

- Layered (n-tier) **architecture**.
- Event-driven **architecture**.
- Microkernel **architecture**.
- Microservices **architecture**.
- Space-based **architecture**.

8 a) Describe SQA plan. 5

The software quality assurance **plan** (SQAP) is a comprehensive **plan** that directs the work of the **SQA** function for a year. ... The project **plan** is a comprehensive document that serves the software project throughout the project life time: the development and operation stages

b) Write about formal technical reviews 5

9 a) Explain about different types of testing 5

. **Types of Testing:-**

- **Unit Testing**. It focuses on the smallest unit of software design. ...
- **Integration Testing**. The objective is to take **unit tested** components and build a program structure that has been dictated by design. ...
- **Regression Testing**. ...
- **Smoke Testing**. ...
- **Alpha Testing**. ...
- **Beta Testing**. ...
- **System Testing**. ...
- Stress Testing.

b) What are advantages and disadvantages of software testing 5

Advantages	Disadvantages
Well suited and efficient for large code segments.	Limited coverage, since only a selected number of test scenarios is actually performed.
Code access is not required.	Inefficient testing, due to the fact that the tester only has limited

	knowledge about an application.
Clearly separates user's perspective from the developer's perspective through visibly defined roles.	Blind coverage, since the tester cannot target specific code segments or errorprone areas.
Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems.	The test cases are difficult to design.