Victoria University
of Bangladesh

MID Term Assessment

Md Bakhtiar Chowdhury

ID: 2121210061

Department: CSE

Semester: Summer 2023

Batch: 21th

Course Title: Software Engineering

Course Code: CS1 321

Submitted To:

Umme Khadiza Tithi

Lecturer, Department of Computer Science & Engineering

Victoria University of Bangladesh

Submission Date: 18 August, 2023

Answer to the question no 1(a)

# Define Software Engineering?

Answer: Software Engineering is a systematic and disciplined approach to designing, developing, testing, deploying, and maintaining software applications, systems, and solutions. It encompasses a set of principles, practices, methodologies, and tools that aim to ensure the creation of high-quality software that meets user needs, is reliable, scalable, and maintainable, and is completed within specified timeframes and budgets.

Key aspects of software engineering include:

- Requirements Engineering
- Design
- Implementation
- Testing
- Deployment
- Maintenance
- Documentation
- Project Management
- Quality Assurance
- Configuration Management
- Software Development Life Cycle (SDLC)

Answer to the question no 1(b)

# Define SDLC Activities and Sprial model?

Answer:

SDLC Activities: The Software Development Life Cycle (SDLC) comprises a series of activities that guide the process of creating software from conception to deployment and maintenance. These activities provide a structured approach to software development, ensuring that the resulting software meets quality standards, user requirements, and business goals. The typical SDLC activities include:

1. **Requirements Gathering and Analysis:** Understanding user needs and defining the software's functional and non-functional requirements.
2. **System Design:** Creating a detailed design that outlines the software's architecture, components, modules, and interactions.
3. **Implementation:** Writing, coding, and programming the software based on the design specifications.
4. **Testing:** Conducting various levels of testing (unit, integration, system, acceptance) to identify and fix defects and ensure the software meets quality standards.
5. **Deployment:** Releasing the software for end-user use, which involves installation, configuration, and sometimes migration from existing systems.
6. **Maintenance:** Continuously monitoring, updating, and enhancing the software to address bugs, accommodate changes, and improve performance.
7. **Documentation:** Creating comprehensive documentation that covers design, functionality, usage instructions, and maintenance procedures.
8. **Project Management:** Overseeing resources, schedules, risks, and communication to ensure the project's success.

Spiral Model:

The Spiral Model is a software development process model that combines elements of iterative development and risk management in a cyclical fashion. It was introduced by Barry Boehm in the 1980s as a response to the limitations of traditional linear development models like the Waterfall. The Spiral Model consists of a series of cycles, each involving the following four major phases:

Planning: Defining project goals, objectives, requirements, constraints, and identifying potential risks.

Risk Analysis: Evaluating potential risks and uncertainties associated with the project. Risks are assessed, and strategies are developed to manage them.

Engineering: Designing, coding, and building the software based on the requirements and design specifications.

Evaluation: Reviewing the progress of the project, assessing the current iteration of the software, and obtaining feedback from stakeholders.

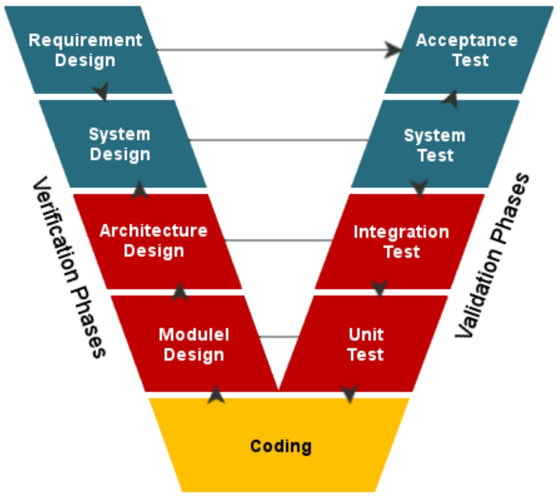
The Spiral Model emphasizes the need for risk assessment and mitigation at every phase of the project. Each cycle involves planning, risk analysis, engineering, and evaluation, with each subsequent cycle building upon the previous one. As the project progresses, the software becomes more refined, and risks are addressed iteratively.

The Spiral Model is particularly suitable for projects with high complexity, uncertainty, and evolving requirements. It allows for flexibility in accommodating changes and adjustments based on feedback and changing conditions. However, it requires careful management of risks and may involve higher costs due to its iterative nature.

Answer to the question no 1(c)

# Describe V-model?

Answer: The V-Model, also known as the Validation and Verification Model, is a software development and testing framework that emphasizes the relationship between different phases of development and corresponding testing activities. The model is called "V-Model" due to its visual representation that resembles the letter "V," with development activities on the left side of the V and testing activities on the right side. The V-Model is often seen as an extension of the traditional Waterfall model, where testing activities are aligned with each development phase. Here's how the V-Model works:



1. Requirements Phase:

- Development: During this phase, the software requirements are gathered and documented in detail.

- Testing: Corresponding test planning and requirement analysis activities take place. Testers create a test plan based on the gathered requirements.

2. System Design Phase:

- Development: The high-level system design is created based on the requirements.

- Testing: System-level test cases are created based on the design specifications.

3. Detailed Design Phase:

- Development: Detailed designs of individual components or modules are produced.

- Testing: Test cases for individual components or modules are created.

4. Implementation Phase:

- Development: Actual coding and programming of the software take place.

- Testing: Unit testing is carried out to ensure that individual components or modules work as intended.

5. Integration Phase:

- Development: Integration of individual components or modules to build the complete system.

- Testing: Integration testing is performed to verify the interactions between different components/modules.

6. Validation Phase:

- Development: System validation is carried out to ensure that the software meets the intended requirements.

- Testing: Validation testing, also known as system testing, is performed to validate the entire system against the original requirements.

7. Verification Phase:

- Development: The software is prepared for delivery to the customer or end-users.

- Testing: Final verification activities, including user acceptance testing (UAT), are performed to ensure that the software is ready for deployment.

The V-Model's strength lies in its clear alignment of development and testing activities, which helps identify defects early in the development process. It emphasizes the importance of verifying and validating the software against requirements throughout the development life cycle. However, like other linear models, the V-Model may struggle to accommodate changes in requirements during the later stages of development, which has led to the adoption of more iterative and flexible methodologies such as Agile and DevOps.

Answer to the question no 1(d)

# Define Graphical user Interface Elements?

Answer:

Graphical User Interface (GUI) elements are visual components used in software applications to enable users to interact with the software using a graphical interface. GUI elements provide a user-friendly way for users to perform tasks, input data, navigate the application, and receive feedback. These elements make software more intuitive and user-centered.

Here are some common GUI elements:

1. Buttons: Clickable elements that trigger actions when clicked. They often have labels or icons that indicate their purpose.

2. Text Boxes: Input fields where users can enter text or numeric data. Text boxes can be used for various purposes, such as search, login, and data entry.

3. Checkboxes: Interactive boxes that users can select or deselect. They are commonly used to toggle settings or options.

4. Radio Buttons: A set of options from which users can select only one choice. Radio buttons are often used for mutually exclusive selections.

5. Dropdown Lists: Lists that display options when clicked, allowing users to select one option from the list.

6. Menus: Lists of commands or options that are displayed when users click on a menu bar or icon. Menus can be hierarchical (submenus) and provide access to various functions.

7. Toolbars: Rows of icons or buttons that provide quick access to frequently used actions or tools.

8. Tabs: Organizational elements that allow users to switch between different sections or views of the application.

9. Panels: Separate areas within the interface that contain related information or controls. Panels can be used for grouping and organizing content.

10. Sliders: Controls that allow users to select a value within a range by moving a slider along a track.

11. Dialog Boxes: Modal windows that provide additional information, prompts, or options. They typically require user interaction before proceeding.

12. Progress Bars: Visual indicators that show the progress of an ongoing task.

13. Icons: Visual representations of objects, functions, or actions. Icons are often used to represent actions in a more compact form.

14. Notifications: Messages that provide feedback, alerts, or information to the user.

15. Images and Graphics: Visual elements that enhance the appearance of the interface or convey information visually.

16. Hyperlinks: Text or graphic elements that, when clicked, navigate to another location or resource within the application or on the web.

17. Context Menus: Right-click menus that provide context-specific options based on the user's current interaction.

GUI elements are designed to enhance user experience, improve usability, and guide users through the application's functionalities. Properly designing and arranging these elements contributes to the overall usability and user satisfaction of software applications.

Answer to the question no 2(a)

# Define project manager?

Answer:

A Project Manager is a professional responsible for planning, executing, and overseeing all aspects of a project to ensure its successful completion. Project managers are integral to various industries, including software development, construction, manufacturing, healthcare, and more. Their role involves coordinating resources, managing teams, and ensuring that projects meet their objectives, adhere to timelines, and stay within budget. Key responsibilities of a project manager include:

Certainly, here are the key responsibilities of a project manager in bullet points:

- Project Planning
- Resource Management
- Team Leadership
- Risk Management
- Communication
- Budget Management
- Timeline Management
- Quality Control
- Scope Management
- Stakeholder Management
- Documentation
- Problem Solving
- Closure and Evaluation

Answer to the question no 2(b)

# Describe role Of project manager and responsibilities Of project manager?

Answer:

The Role of a Project Manager involves overseeing the entire project lifecycle, from initiation to closure. Project managers are responsible for ensuring that projects are completed successfully, meeting objectives, staying within scope, on time, and within budget. They play a crucial role in coordinating resources, managing teams, and communicating effectively with stakeholders. Here's an overview of the role and associated responsibilities:

Role of a Project Manager:

- **Leadership**: Providing guidance and direction to the project team, motivating them to achieve project goals.

- **Coordination**: Ensuring efficient allocation of resources, both human and material, to accomplish project tasks.

- **Communication:** Facilitating clear and effective communication among team members, stakeholders, and clients.

- **Risk Management**: Identifying potential risks and developing strategies to mitigate or manage them.

- **Problem Solving:** Addressing challenges and issues that arise during the project, making decisions to keep the project on track.

- **Decision Making:** Making informed decisions related to project scope, changes, and priorities.

- **Stakeholder Management:** Building and maintaining positive relationships with project stakeholders and managing their expectations.

- Planning: Creating a detailed project plan, including tasks, timelines, milestones, and resource requirements.

- Quality Assurance: Ensuring that project deliverables meet established quality standards**.**

- Budget Control: Monitoring project expenses and ensuring that the project stays within the allocated budget.

- Scope Management: Preventing scope creep and managing changes to project scope.

- Time Management: Monitoring project progress against the schedule and making adjustments as needed.

- Documentation: Keeping accurate records of project plans, decisions, and communications.

- Closure and Evaluation: Concluding the project, conducting evaluations, and documenting lessons learned for future projects.

Responsibilities of a Project Manager:

- Define project goals, objectives, and scope.

- Develop a comprehensive project plan.

- Identify and allocate necessary resources.

- Assemble and lead project teams.

- Monitor and manage project progress.

- Communicate project status to stakeholders.

- Mitigate risks and resolve issues.

- Make timely decisions to keep the project on track.

- Ensure quality and adherence to requirements.

- Manage project budgets and expenses.

- Handle changes and scope adjustments.

- Maintain stakeholder relationships.

- Ensure effective communication within the team.

- Prepare for project closure and evaluation.

- Document project processes and outcomes.

In summary, a project manager is responsible for overseeing all aspects of a project, from planning and execution to closure and evaluation. They ensure that projects are completed successfully while managing resources, risks, scope, and communication. Effective leadership, communication, problem-solving, and organizational skills are essential for a project manager to excel in their role.

Answer to the question no 2(c)

# What are objects of Software Design?

Answer:

In software design, the term "objects" can refer to two different concepts: "objects" in the context of object-oriented programming (OOP) and "objects" in the context of design objectives. Let's explore both:

1. Objects in the Context of Object-Oriented Programming (OOP):

In object-oriented programming, an "object" is an instance of a class, which is a blueprint for creating objects. Objects encapsulate data (attributes) and behaviors (methods/functions) related to a specific concept or entity. They are the fundamental building blocks of object-oriented design and allow for the organization, modularity, and reusability of code. Objects represent real-world entities or concepts and interact with each other to create a functional software system.

2. Objects as Design Objectives:

In the context of software design, "objects" can also refer to the goals, objectives, or principles that guide the design process. These design objectives help ensure that the resulting software is of high quality, maintainable, and aligned with user needs. Some common design objectives include:

- Modularity: Designing software in a way that breaks it down into smaller, manageable modules or components that can be developed, tested, and maintained independently.

- Reusability: Creating software components that can be reused across different projects or parts of the same project, reducing development effort and promoting consistency.

- Maintainability: Designing software that is easy to understand, modify, and maintain over time. This involves using clear and structured designs, as well as appropriate documentation.

- Scalability: Designing software with the capability to handle increased workloads or user demands without major changes to the architecture.

- Flexibility: Designing software that can adapt to changing requirements or technological advancements without significant redesign.

- Efficiency: Designing software to perform tasks quickly and with minimal resource consumption, such as memory and processing power.

- User Experience (UX): Designing software with a focus on providing a positive and intuitive user experience, including user interface design and usability considerations.

- Reliability: Designing software that consistently performs as expected, minimizing errors and failures.

- Security: Designing software with measures to prevent unauthorized access, data breaches, and other security vulnerabilities.

- Interoperability: Designing software that can interact and integrate smoothly with other systems or software components.

These design objectives help guide the decisions made during the software design process, ensuring that the resulting software meets both technical and user-related requirements. They contribute to creating software that is effective, efficient, and aligned with the overall project goals.

>>>>>>END<<<<<<