FR. CONCEICAO RODRIGUES COLLEGE OF ENGG.

Fr. Agnel Ashram, Bandstand, Bandra (W) Mumbai 400 050.

SEMESTER / BRANCH: V/COMPUTER Engineering

SUBJECT: Software Engineering **(CSC502)/ First Assignment**

Date: 19-08-23 Due Date : 25-08-23

CSC502.1: Recognize software requirements and various process models. (Understanding) **CSC502.2**: Develop project Plan, schedule and track the progress of the given project (Applying)

Questions :

Q1) What is the significance of recognizing software requirements in the software engineering process?

Recognizing software requirements is a crucial and fundamental aspect of the software engineering process. Software requirements define what a software system needs to accomplish, how it should behave, and what functionalities it should provide. The significance of recognizing software requirements lies in the following key points:

1. Understanding User Needs: Software requirements capture the needs, expectations, and goals of users and stakeholders. Recognizing and documenting these requirements ensure that the software system addresses the actual needs of its intended users.

2. Basis for Design and Development: Requirements serve as the foundation for designing and developing the software. They provide the context and guidelines for software architects and developers to create a system that meets the desired functionality and quality.

3. Scope Definition: Clear requirements help define the scope of the software project. This prevents scope creep, which occurs when the project's objectives and features continuously expand beyond what was initially planned.

4. Communication: Well-defined requirements facilitate effective communication among team members, stakeholders, developers, and users. Everyone involved can have a common understanding of what the software should achieve.

5. Risk Management: Identifying requirements early in the process helps identify potential risks and challenges. Addressing these issues early reduces the likelihood of encountering problems during later stages of development.

6. Resource Allocation: Requirements provide insight into the resources (such as time, budget, and personnel) needed to develop the software. This helps in better planning and allocation of resources.

7. Quality Assurance: By specifying desired behavior and functionality, requirements provide a basis for testing and quality assurance efforts. Testing can be focused on verifying that the software meets its intended functionality.

8. Customer Satisfaction: Meeting user needs and expectations is a key driver of customer satisfaction. Accurate recognition of requirements contributes to a successful software product that satisfies its users.

9. Change Management: Requirements act as a baseline against which changes can be evaluated. When new needs arise or changes are requested, the original requirements provide a reference point for assessing the impact.

10. Legal and Regulatory Compliance: Many software systems need to comply with legal and regulatory standards. Properly recognizing requirements ensures that the software meets these standards.

11. Documentation and Maintenance: Well-documented requirements provide essential documentation for future maintenance and updates. They serve as a reference point for understanding the software's intended behavior and purpose.

Q2) Describe the main characteristics of different process models used in software development.

1. Waterfall Model:
- Sequential and linear approach.
- Divides the project into distinct phases (requirements, design, implementation, testing, deployment).
- Each phase must be completed before the next one begins.
- Emphasizes documentation and planning.
- Suitable for projects with well-defined and stable requirements.

2. Iterative Model:
- Involves repeating cycles of development.
- Each cycle produces a working version of the software.
- Each iteration builds upon the previous one, incorporating feedback.
- Useful for evolving requirements and when early delivery is desired.

3. Incremental Model:
- Splits the project into smaller increments or modules.
- Each increment is developed independently and can be delivered separately.
- Suitable for projects with changing requirements and when partial functionality is acceptable.

4. Agile Model (e.g., Scrum, Kanban):
- Emphasizes flexibility, collaboration, and customer feedback.
- Uses short development iterations called "sprints."
- Focuses on delivering incremental value to customers.

- Allows for changes and adjustments throughout development.

5. Spiral Model:
- Combines iterative and incremental development with risk assessment.
- Each cycle involves four phases: planning, risk analysis, engineering, and evaluation.
- Good for projects with high risk, evolving requirements, or complex functionality.

6. V-Model (Validation and Verification Model):
- Correlates each development stage with a corresponding testing phase.
- Testing is integral to each phase.
- Ensures that each development phase has a corresponding validation phase.
- Emphasizes rigorous testing to ensure quality.

7. Big Bang Model:
- Minimal planning and structure.
- Developers start building the system without clear requirements.
- Suitable for small projects or proof-of-concept prototypes.
- High risk due to lack of planning.

8. Rapid Application Development (RAD):
- Focuses on rapid prototyping and user involvement.
- Uses a series of prototyping cycles to refine the software.
- Well-suited for projects requiring fast delivery and frequent changes.

9. Feature-Driven Development (FDD):
- Divides the software into manageable, well-defined features.
- Each feature is developed, tested, and integrated.
- Emphasizes collaboration and clarity of features.

10. Unified Process (UP):
- Iterative and incremental approach based on the Unified Modeling Language (UML).
- Divides the development cycle into phases and iterations.
- Focuses on architecture, design, and iterative development.

11. Extreme Programming (XP):
- Emphasizes close collaboration between developers and customers.
- Involves frequent releases, continuous testing, and incremental changes.
- Prioritizes simplicity, communication, and responsiveness to change.

Q3) How does the Capability Maturity Model (CMM) contribute to improving software development processes?

The Capability Maturity Model (CMM) is a framework that outlines a set of best practices and guidelines for improving software development processes within an organization. The primary goal of the CMM is to help organizations enhance the quality of their software development processes, increase predictability, and improve the overall efficiency of

software development efforts. The CMM achieves these goals through several key contributions:

1. Process Standardization:
The CMM provides a structured framework for organizations to assess and standardize their software development processes. It defines a set of maturity levels that describe the progression of process improvement, from ad hoc and chaotic practices to well-defined and optimized processes.

2. Continuous Improvement:
The CMM emphasizes the importance of continuous improvement. Organizations are encouraged to evaluate their current processes, identify areas for enhancement, and systematically work towards higher maturity levels.

3. Performance Measurement:
The CMM encourages the establishment of metrics and measurements to assess the effectiveness of processes. This enables organizations to track progress, identify bottlenecks, and make data-driven decisions for process improvements.

4. Risk Management:
By defining structured processes, the CMM helps organizations identify and mitigate risks. As processes become more mature, organizations are better equipped to manage and reduce the impact of risks on software development projects.

5. Predictability:
As organizations move up the maturity levels of the CMM, they develop more reliable and predictable processes. This leads to improved project planning, estimation, and delivery, reducing the likelihood of cost and schedule overruns.

6. Consistency and Reproducibility:
The CMM guides organizations in creating repeatable processes that can be consistently applied across projects. This consistency leads to higher quality deliverables and a more efficient use of resources.

7. Employee Skills Development:
As organizations work towards higher maturity levels, they invest in training and skill development for their employees. This results in a more capable workforce with better understanding of effective software development practices.

8. Customer Satisfaction:
The CMM's focus on process improvement often leads to better alignment with customer requirements and expectations. Delivering high-quality software on time and within budget contributes to higher customer satisfaction.

9. Competitive Advantage:
Achieving higher maturity levels can give organizations a competitive edge. Clients and partners may prefer to work with organizations that have established and mature software development processes.

10. Best Practice Sharing:
The CMM framework provides a common language and structure for discussing software development processes. This facilitates knowledge sharing and collaboration within and across organizations.

Q4) Explain the differences between prescriptive process models and evolutionary process models.

Prescriptive process models and evolutionary process models are two broad categories of software development methodologies, each with its own approach and characteristics. Here are the key differences between these two types of process models:

Prescriptive Process Models:

Also known as plan-driven or predictive models, prescriptive process models emphasize careful planning and execution of a well-defined process from the beginning to the end of the project. These models prescribe a set of predetermined steps to be followed in a sequential manner. Examples of prescriptive models include the Waterfall model and the V-Model.

1. Sequential Approach: Prescriptive models follow a sequential and predefined order of phases, with each phase building upon the completion of the previous one.

2. Detailed Planning: Planning is done extensively upfront, including requirements gathering, design, development, testing, and deployment. The entire project is planned before execution begins.

3. Stability: Prescriptive models work best when project requirements are stable and well-defined at the beginning of the project.

4. Minimal Changes: Changes to requirements are typically discouraged or require formal change management due to the linear nature of the process.

5. Documentation: Extensive documentation is emphasized at each phase to ensure clear communication and traceability.

6. Risk Management: Risk management is often performed during the planning phase, and risk mitigation strategies are established.

7. Suitability: Prescriptive models are suitable for projects with well-defined requirements, low levels of uncertainty, and where the project can be accurately estimated upfront.

Evolutionary Process Models:

Evolutionary process models, also known as adaptive models, emphasize flexibility and adaptability to changing requirements and circumstances. These models involve iterative and incremental development, allowing the software to evolve gradually. Examples of evolutionary models include Agile methodologies (e.g., Scrum, Kanban) and the Spiral model.

1. Iterative and Incremental: Evolutionary models involve iterative cycles where development, testing, and feedback occur in repeated phases. Each iteration adds new functionality.

2. Continuous Adaptation: Evolutionary models embrace changing requirements and encourage flexibility in adapting to new insights and evolving user needs.

3. Early Feedback: Users are involved early in the process, providing feedback on working prototypes or increments of the software.

4. Less Upfront Planning: While some planning is involved, evolutionary models focus more on adapting to feedback and adjusting the development process as the project progresses.

5. Emphasis on Collaboration: Collaborative teamwork, frequent communication, and customer involvement are critical aspects of evolutionary models.

6. Risk Management: Risk management is a continuous activity, with regular assessment and adaptation based on feedback.

7. Suitability: Evolutionary models are suitable for projects with changing requirements, high levels of uncertainty, and a need for quick iterations and user feedback.

Q5) Provide examples of situations where using a specific process model would be more
suitable.

- **Waterfall Model:**

  The Waterfall model is suitable when the project requirements are well-defined and stable, and there is little likelihood of significant changes throughout the project. It's also appropriate for projects with a strict budget and timeline. For example:

  - Developing a simple mobile app with a fixed set of features and requirements.
  - Creating a documentation-heavy project where each phase requires thorough documentation before moving on to the next.

- **Agile Model:**

  Agile methodologies (e.g., Scrum, Kanban) are suitable when there is a need for flexibility, frequent collaboration, and the ability to adapt to changing requirements. It's great for projects with evolving or unclear requirements and for teams that value continuous improvement. For example:

  - Developing a complex software system with multiple features that may change during development.
  - Designing a website with a dynamic user interface that requires regular updates based on user feedback.

- **Spiral Model:**

  The Spiral model is suitable when the project involves significant risk assessment and mitigation. It's useful for projects with changing requirements and a need for frequent prototypes and evaluations. For example:

  - Developing a new medical device that requires rigorous testing and validation at each iteration.
  - Creating a large-scale enterprise software system that needs continuous user feedback and improvements.

- **Incremental Model:**

  The Incremental model is suitable when the project can be divided into smaller manageable modules or components. It's great for projects where certain modules can be developed independently and then integrated. For example:
  - Building an online e-commerce platform where individual modules like user authentication, product catalog, and payment gateway can be developed separately.
  - Developing a video game where different levels or stages can be developed incrementally and then combined.

- **V-Model (Validation and Verification Model):**

  The V-Model is suitable when there is a strong emphasis on validation and verification activities throughout the project lifecycle. It's especially useful for projects with stringent quality and testing requirements. For example:

  - Developing safety-critical systems like automotive control systems or medical equipment.
  - Building software for industries with strict regulatory compliance, such as financial services or healthcare.

- **Rapid Application Development (RAD) Model:**

  The RAD model is suitable when there's a need to quickly develop a prototype or working version of the software for demonstration or evaluation. It's beneficial for projects that require a fast turnaround time and iterative development. For example:

  - Creating a proof-of-concept for a new product idea to present to potential investors.
  - Developing a software tool for a specific event or campaign that has a tight deadline.

Q6) Compare and contrast the Waterfall model and Agile methodologies in terms of project planning and progress tracking.

1. ***Waterfall Model:***

- Project Planning:

1. Sequential Phases: The Waterfall model follows a linear and sequential approach. Each phase (requirements, design, implementation, testing, deployment, maintenance) is completed one after the other.
2. Detailed Planning: Extensive planning is done at the beginning of the project to define requirements, scope, and resources. Once planning is complete, it's difficult to make significant changes without impacting the entire project.

- Progress Tracking:

1. Milestone-Based: Progress is tracked based on completing milestones at the end of each phase. Each phase has its own set of deliverables.
2. Late Feedback: Feedback from stakeholders is usually gathered towards the end of the project, which may result in potential changes being difficult and costly to accommodate.

- Advantages:

1. Clear project scope and requirements.
2. Well-defined milestones and deliverables.
3. Predictable timeline and budget (assuming no major scope changes).

- Disadvantages:

1. Limited flexibility to accommodate changing requirements.
2. Late-stage discovery of defects or issues can be costly to rectify.
3. Lack of customer involvement during development can lead to misalignment with actual needs.

2. Agile Methodologies:

- Project Planning:

1. Iterative Approach: Agile methodologies, such as Scrum and Kanban, follow an iterative approach with short development cycles (sprints or iterations).
2. Adaptive Planning: Initial planning is done with a high-level vision, and detailed planning is carried out for the upcoming iteration. Plans can be adjusted based on ongoing feedback and changing priorities.

- Progress Tracking:

1. Iterative Progress: Progress is tracked at the end of each iteration, with working increments of the product delivered at the end of every cycle.
2. Frequent Feedback: Agile methodologies prioritize continuous feedback from stakeholders and end-users, allowing for early detection of issues and quicker adaptations.

- Advantages:

1. Flexibility to accommodate changing requirements and priorities.
2. Frequent customer involvement leads to better alignment with user needs.
3. Early defect detection and continuous improvement.

- Disadvantages:

1. Can be challenging to estimate the overall project timeline and scope accurately.
2. May require more frequent communication and collaboration with stakeholders.
3. Complexities in managing changing priorities and balancing workloads.

Q7) Apply process metrics to evaluate the efficiency and effectiveness of Waterfall , Agile (both Scrum & Kanban) methodologies, considering factors such as development speed, adaptability to change and customer satisfaction.

Development Speed:

Development speed is a measure of how quickly a methodology produces deliverables and progresses through the development lifecycle.

- Waterfall:

- Waterfall projects might have slower development speed due to the linear nature of the process.
- Metrics: Average time per phase, time taken to move from one phase to another.

- Scrum:
- Scrum emphasizes time-boxed iterations (sprints) that lead to regular, potentially shippable increments.
- Metrics: Velocity (amount of work completed per sprint), lead time (time taken from request to delivery), cycle time (time taken for a single item to move from start to finish).

- Kanban:
- Kanban focuses on optimizing flow by reducing work-in-progress (WIP) and ensuring items move through the process smoothly.
- Metrics: Cycle time, throughput (items completed per unit of time), WIP limit adherence.

Adaptability to Change:
Adaptability to change measures how well a methodology handles evolving requirements and responds to feedback.

- Waterfall:
- Less adaptable to change due to its rigid sequential nature.
- Metrics: Change requests after the initial planning phase, percentage of completed requirements that changed.

- Scrum:
- Scrum is adaptable through sprint planning, backlog grooming, and sprint reviews that allow continuous feedback and adjustments.
- Metrics: Changes requested mid-sprint, number of scope changes during a sprint.

- Kanban:
- Kanban's focus on flexibility allows teams to adapt to changing priorities and requirements as they arise.
- Metrics: Lead time for scope changes, time taken to react to changing priorities.

Customer Satisfaction:
Customer satisfaction measures how well the methodology delivers value and meets user needs.

- Waterfall:
- Customer satisfaction may be impacted by the long duration before the final product is delivered.
- Metrics: User acceptance testing results, customer feedback at the end of the project.

- Scrum:
- Regular feedback and incremental delivery in Scrum help ensure customer satisfaction throughout the project.
- Metrics: Customer feedback at sprint reviews, Net Promoter Score (NPS).

- Kanban:
- Kanban's focus on delivering value continually contributes to maintaining customer satisfaction.
- Metrics: Customer feedback on individual items, frequency of customer interactions.

Q8) Justify the relevancy of the following comparison for software development models.

Features	Waterfall Model	Incremental Model	Prototyping Model	Spiral Model
Requirement Specification	Beginning	Beginning	Frequently Changed	Beginning
Understanding Requirements	Well Understood	Not Well Understood	Not Well Understood	Well Understood
Cost	Low	Low	High	Expensive
Availability of reusable component	No	Yes	Yes	Yes
Complexity of System	Simple	Simple	Complex	Complex
Risk Analysis	Only at beginning	No risk analysis	No risk analysis	Yes
User involvement in all phases of SDLC	Only at beginning	Intermediate	High	High
Guarantee of Success	Less	High	Good	High
Overlapping Phases	Absent	Absent	Present	Present
Implementation Time	Long	Less	Less	Depends on Project
Flexibility	Rigid	Less flexible	Highly flexible	Flexible
Changes Incorporated	Difficult	Easy	Easy	Easy
Expertise Required	High	High	Medium	High
Cost Control	Yes	No	No	Yes
Resource Control	Yes	Yes	No	Yes

The provided comparison of software development models (Waterfall, Incremental, Prototyping, Spiral) based on various features is relevant because it highlights the distinct characteristics and trade-offs associated with each model. This comparison is valuable for making informed decisions about which development model to use for a particular project based on the project's requirements, constraints, and goals.

1. Requirement Specification:

- Relevant because it addresses how each model handles requirement specification at the beginning or throughout the project, which impacts early planning and design decisions.

2. Understanding Requirements:

- Relevant because it indicates the level of understanding of project requirements at the start of the project, influencing how well the chosen model accommodates changes and uncertainties.

3. Cost:
- Relevant because it helps stakeholders gauge potential project costs, especially when comparing cost-efficiency among different models.

4. Availability of Reusable Component:
- Relevant because it addresses the utilization of existing components, libraries, or modules, which impacts development speed, quality, and maintenance.

5. Complexity of System:
- Relevant because it reflects how well a model suits projects of varying complexities, helping choose an appropriate approach based on project scope.

6. Risk Analysis:
- Relevant because it highlights the extent to which risk analysis is integrated into each model, influencing risk management practices during development.

7. User Involvement in all Phases of SDLC:
- Relevant because it points out the level of user engagement throughout development, which affects the product's alignment with user needs and preferences.

8. Guarantee of Success:
- Relevant because it addresses the level of confidence in achieving project success, guiding decision-making when risk tolerance is a factor.

9. Overlapping Phases:
- Relevant because it indicates whether certain phases overlap, which can impact project efficiency and time-to-market.

10. Implementation Time:
- Relevant because it discusses the duration of implementation for different models, helping in project scheduling and planning.

11. Flexibility:
- Relevant because it evaluates the flexibility of each model to accommodate changes and adapt to evolving project needs.

12. Changes Incorporated:
- Relevant because it highlights how easily changes are integrated into the project, influencing the project's ability to respond to evolving requirements.

13. Expertise Required:
- Relevant because it identifies the expertise level needed to successfully implement each model, assisting in resource allocation and skill assessment.

14. Cost Control and Resource Control:

- Relevant because these factors inform how well the chosen model manages project costs and resources.

Rubrics :

Indicator	Average	Good	Excellent	Marks
Organization (2)	Readable with some mistakes and structured (1)	Readable with some mistakes and structured (1)	Very well written and structured (2)	
Level of content(4)	Minimal topics are covered with limited information (2)	Limited major topics with minor details are presented(3)	All major topics with minor details are covered (4)	
Depth and breadth of discussion(4)	Minimal points with missing information (1)	Relatively more points with information (2)	All points with in depth information(4)	
Total Marks(10)				