Learning Journal Week#4

Student Name: Aditya Sawant
Course: SOEN6841 Software Project Management
Journal URL: https://github.com/ads2602/SOEN-6841-Software-Project-Management-W25
Dates Rage of activities: 23/02/2025 to 15/03/2025
Date of the journal: 16/03/2025

Key Concepts Learned:

Throughout this period, I've been focusing chapters 9 Software Lifecycle management to chapter 12 Software construction these all chapters are part of SLCM software lifecycle management, and each of them explain us various topics which are essential for good project management practises. In (chapter 9) Introduction to Software Life-Cycle Management we came across what is software engineering management, it basically balances technical engineering (design, construction) with process selection, improvement, and standardization. Various Life-Cycle Models such as, **Waterfall Model**: Sequential phases (requirements, design, construction, testing, release, maintenance). Best for stable, large-scale projects, **Iterative Model**: Short cycles (e.g., Scrum, Agile) for incremental delivery, reducing risks of unclear requirements and **Concurrent Engineering**: Parallel development by distributed teams to compress timelines, were also covered. What are the **Metrics & Work Products**: Measured attributes (e.g., requirement testability, code modularity) and artifacts (SRS documents, design models) and what are their significance was explained along with the importance of **Quality Assurance**: Exit criteria for phases (e.g., formal reviews) to prevent defects from propagating downstream. (Chapter 10) Software Requirement Management a detailed Software requirement and Requirement Types was explained like what are **Functional Requirements**: Core features (e.g., "volunteer task assignment"), **Non-Functional Requirements**: Performance, security, usability (e.g., real-time GPS tracking). Various requirement development techniques like: Elicitation techniques (interviews, use cases), prioritization, and validation via stakeholder reviews etc were explained. Further the concept of Change Control: Versioning, impact analysis, and communication to distributed teams, and Quality Control: Ensuring requirements are testable, unambiguous, and aligned with stakeholder needs were discussed also learnt the importance of SRS document. Software Design Management (Chapter 11) is all about Software Design Management with methods such as top-down and bottom-up. The chapter discusses design characteristics and techniques such as object-oriented programming, prototypes, ER models, and system analysis. I learned the concept of refactoring and reusing designs. **Quality Assurance**: Design reviews for flaws, maintainability, and alignment with requirements. Software Construction (Chapter 12) Project Construction speaks about coding standards like Modularity, clarity, reliability, and safety (e.g., secure API integration), frameworks, methods, and techniques. Software construction follows the iterative model approach, covering aspects like continuous integration, pair programming, and configuration management.

Application in real projects:

Various core concepts from the SLCM are used in each and every software projects. Every software project manager should follow these core concepts for a well defined and structured project, these practises are industry standard these processes help maintain and progress the project to completion.

- **Software Lifecycle Implementation:** Every software project follows a structured lifecycle from requirements gathering to design, development, testing, deployment, and maintenance. Selecting the right lifecycle model based on project needs (e.g., Waterfall for well-defined projects, Agile for flexible and evolving projects) helps optimize project execution.
- **Requirement Analysis and Management:** Understanding and documenting requirements effectively is critical in any project. Functional and non-functional requirements play a vital role in defining the system's scope, usability, and performance. Proper requirement validation through stakeholder engagement ensures that project goals align with business needs.
- **Design and Architectural Best Practices:** A well-structured design ensures scalability, maintainability, and performance. Object-oriented principles, modular design, and architectural patterns (such as MVC or microservices) are widely used in enterprise applications to enhance reusability and flexibility.
- **Coding Standards and Software Construction:** Implementing best coding practices such as modularity, clarity, and maintainability improves software quality. Techniques like pair programming, continuous integration, and automated testing streamline the development process and ensure high code reliability.
- **Change Management and Version Control:** Managing software changes effectively is essential for project stability. Tools like Git, SVN, and Jira help in versioning, tracking issues, and ensuring smooth collaboration among distributed teams. Proper change control mechanisms prevent scope creep and ensure project deliverables remain on track.
- **Quality Assurance and Testing:** Implementing various testing strategies, including unit testing, integration testing, and user acceptance testing, ensures software reliability and robustness. Defining clear exit criteria at each phase prevents defects from propagating downstream, reducing rework and project delays.
- **Industry Standards and Frameworks:** Many modern projects leverage existing frameworks to expedite development. For instance, Java-based projects utilize Spring Boot for backend services, while frontend applications often use React or Angular. Choosing the right framework and adhering to industry standards significantly enhances development efficiency.

Peer Interactions:

During this period, I had insightful discussions with my classmates regarding software lifecycle management and its real-world applications. Exchanging knowledge about different lifecycle models, requirement analysis techniques, and coding standards helped me gain a broader perspective. Our discussions revolved around practical challenges in implementing iterative models in large-scale projects and the effectiveness of various requirement elicitation methods. Additionally, while working on project assignments, we collaborated on refining design approaches and ensuring that our solutions aligned with industry best practices. We also had a debate on whether to use agile methodology or some other design approach for designing that we would have used for our disaster volunteer platform project. We also had a project pitch during this period it was a great activity to present the idea in front of the entire class. We used the above learnt concepts in using the concepts of requirements analysis and design management to understand the complexity of the idea and determining the justification for the iterative method that we choose earlier for our project development.

Challenges Faced:

One of the key challenges I faced was understanding how different software lifecycle models fit into various project scenarios. While the waterfall model is suitable for well-defined projects, iterative models offer flexibility but require continuous stakeholder involvement. Another challenge was in software requirements management, specifically in defining non-functional requirements that are often ambiguous but crucial for system performance and security. Understanding the user requirements in English and refactoring it to the developers understanding was a difficult concept. The idea here is to understand the right requirements and to reach the same requirements without missing any artifact to the developers and get the right software built. Ensuring that requirement changes were properly documented and controlled within our project was a learning curve. In software construction, adhering to best coding practices and modularization required additional effort to balance readability, maintainability, and performance.

Personal development activities:

To deepen my understanding, I explored case studies on software lifecycle management and software design best practices namely the Spotify's Agile model. I also referred to additional resources beyond the course material to understand the evolving trends in agile development and DevOps integration with SLCM. Practicing coding exercises related to software construction helped reinforce my understanding of modularity, clarity, and secure API integration. Additionally, I reviewed past project experiences to identify areas where these concepts could have been applied more effectively, like I also explore the tool like the SonarQube for code quality analysis. I improved my knowledge with more intense understanding when I was writing the report for my project idea for the project plan and feasibility study which used the concepts learnt above. I was able to add all the concepts and mould my project and explain it in terms of the software project management principles.

Goals for the Next Week:

Achieved the previous week's goal even learnt one extra chapter, moving forward, I aim to apply my learning to our ongoing project by refining our requirement documentation and incorporating best design practices. I also plan to review the upcoming chapters (Chapter 13 and Chapter 14) to gain insights into software deployment and maintenance strategies. Preparing for the second exam is another priority, and I intend to focus on revisiting key concepts and solving related exercises. Lastly, I will continue engaging with peers and exploring real-world applications to strengthen my understanding of software project management principles. Further, I must explore more on the subject with more articles and papers published.