Final Learning Journal

Student Name: Sushanth Ravishankar
Course: SOEN 6481 (Software Project Management)
Journal URL: https://github.com/sushanth16012002/SOEN6841_40267400
Dates Rage of activities: 17/03/2025 – 30/03/2025
Date of the journal: 30/03/2025

Final Reflections:
Overall Course Impact:
Software project management is divided into four key phases: initiation, planning, monitoring and control, and closure. Each stage builds upon the previous one and adapts as the project progresses. The initiation phase involves creating an initial baseline for the project by estimating effort, cost, and timeline. A market analysis is conducted to understand the purpose and benefits of developing the software. The outputs or artifacts from this stage are often used to gain customer approval. For outsourced projects, cost and effort estimates are sent for approval, and at this point, the project requirements may still be unclear or not fully defined.
The planning phase begins once the requirements are available. This phase includes managing suppliers and communication, as well as estimating cost, schedule, effort, and managing resources, tools, and issues. Tools like COCOMO (Basic, Intermediate, II) and function point analysis are commonly used to estimate the effort. The more complex and high-quality the project, the greater the effort required. The critical path method is employed to determine the project duration by identifying the longest sequence of tasks that must be completed—this can be essential to meeting deadlines. A work breakdown structure is used to divide the project into manageable pseudo tasks (such as Designing, Implementation, Testing), which are further split into individual tasks with defined start and end dates. The completion of each pseudo task marks a milestone.
There are two main types of planning: top-down and bottom-up. Top-down planning is ideal when strict timelines are crucial to ensuring the software's competitiveness in the market. Bottom-up planning is more suitable when the requirements are clear, but the schedule is either flexible or not a primary concern. This planning stage is critical because any miscalculation—either underestimating or overestimating—can significantly affect the next phase: monitoring and control. Monitoring is basically ensuring whether the project is going as planned. Accurate and frequent monitoring is important and is achieved by measuring products, cost, schedule, quality and so on from the ongoing software processes. Earned Value Analysis is used to identify deviations from the project plan by measuring the amount of work that should have been completed based on the budget spent. This helps assess if the project is moving in the correct direction. Control involves taking actions to correct these deviations and keep the project on track.
Risk management involves recognizing potential events that could affect the cost, schedule, or quality of a project during its execution. This process includes identifying risks, evaluating them, and prioritizing them based on their impact and likelihood. By assessing each risk, teams can determine which ones require the most attention. For every identified risk, a mitigation strategy should be in place to reduce its chances of occurring. Additionally, a contingency plan outlines the actions to take if the risk does materialize.
Project closure refers to the final phase where specific actions are taken as the project nears completion. This includes reviewing whether the project followed the planned course, storing project-related data, documenting lessons learned, and releasing any unused resources. All deliverables are analyzed to help enhance future software development efforts. The insights gained help identify more efficient methods, areas needing improvement, mistakes made, and strategies for better client communication. It also allows for evaluation of how risks were managed and which contingency or mitigation plans proved most effective.
All phases of software project management are influenced by the chosen software lifecycle model, such

as Waterfall or Iterative. A software lifecycle outlines the sequence of processes followed during software development. Common lifecycle models include waterfall, iterative, and incremental, each based on its own principles. Therefore, selecting the right model requires careful evaluation. Software lifecycle metrics are used to assess both the development process and the resulting product. These metrics primarily help evaluate work products and also identify connections between different deliverables, their level of association, and overall impact.
Requirement engineering involves refining, verifying, and validating the requirements after the initial input is collected from the customer. Software requirements are generally categorized into functional and non-functional (such as performance or quality attributes). These requirements are gradually improved through continuous engagement with customers and stakeholders during the development process.

Application in Professional Life:
The insights gained can be effectively utilized to manage software projects. When leading a development project, I can use a project charter to bring stakeholders into alignment and apply Function Point Analysis to produce precise estimates for size and cost. However, crafting a detailed charter requires a deep understanding of the project, which might not be fully clear at the start. Familiarity with the project closure phase aids in the efficient release of unused resources and in documenting key lessons learned. These lessons are valuable for ongoing learning and help in making better decisions in future projects, ultimately enhancing my capabilities as a software project manager. In practical terms, resource planning and budgeting often include buying software licenses and assigning developers to specific modules. Monitoring helps in tracking completed features or LOCs, while lessons learned can involve comparing actual person-months spent on a feature to the initial estimates.
Applying the concepts and skills from this course can strengthen my performance in handling project closure and refining requirements. Understanding which lifecycle model helped me understand the given project which is crucial. These approaches not only enhance workflow efficiency but also contribute to building a more adaptable and robust team environment.

Peer Collaboration Insights:
Through collaborative learning, I discovered how theoretical concepts apply to real-world projects, highlighting the importance of thoughtful decision-making. Rushed or poorly judged choices can lead to expensive changes later on. Since every project has its own unique complexities, it’s essential to thoroughly analyze them before making decisions. This process typically involves multiple discussions, consensus-building, and deep analysis. Classroom discussions with experienced project managers helped me grasp the intricate skills and knowledge needed to successfully manage software projects. These experiences provided valuable insights into managing complexity, assessing software quality, and using past projects as useful references. Additionally, I learned that using a reliable configuration and version control system is key to ensuring smooth collaboration and avoiding errors during simultaneous development.

Personal Growth:
I developed a stronger understanding of the complexities involved in project management, realizing overtime that it’s not a simple or linear process. I now understand the role of project managers playing a crucial role in ensuring a project's success. They oversee and coordinate the efforts of developers while ensuring that the final software product meets quality standards and is delivered on time and within budget.