Learning Journal 4

Student Name: Bharti Chhabra
Course: Software Project Management
Journal URL: https://github.com/BhartiChh/SOEN-6481_software-project-management
Dates Rage of activities: 3 November - 9 November
Date of the journal: 9 November, 2024

Key Concepts Learned: Chapter 9 starts with an overview of Software Engineering that highlighted that Technical (requirements specification, software design, construction and maintenance) and managerial tasks (process selection, improvement, standards, process quality across organization) are both included in software engineering. It entails overseeing development procedures, productivity, cost effectiveness, and quality control. Current approaches are guided by best practices from past initiatives, although software engineering is still evolving. **Software Development Life Cycles** includes **Waterfall Model** (a linear method with separate stages), **Iterative Model** (where requirements vary over iterations), and **Concurrent Engineering** (overlapping phases majorly construction and testing for faster progress) are some of the life-cycle models that are employed. From well specified government initiatives to adaptable business solutions, each model meets certain project demands. Moving from waterfall to iterative model requires 3 levels, **organization level** where the complete product management is taken care of, **lower level** concerns major releases of the product, and **lowest lower level** where most of actual product development is done using iterations**. Challenges in requirement gathering** include unclear requirements, difficulty in getting requirements, difficulty in understanding requirements, and translating those requirements into a suitable software design .**Challenges in software design** include difficulty in modeling due to changes or unclear requirements, limitations of representation of requirements into system design, etc. **Challenges in software construction** include lack of team work, rework due to changes in design, lack of clarity in design, etc. **Challenges for software testing** include too many defects in the software application that increases load on software testers, lack of test strategy, lack of test planning, etc. Reviews and other quality control procedures guarantee flawless work results at every stage of the SDLC. Software testing, also known as functional testing, is of 2 types: **white box testing** (at functions level) and **black box testing** (at system level). Comprehensive framework is needed for effective testing. **Software releases** can be alpha, beta, and final, depending upon purpose of release. Verifying phase completion using **exit criteria** , delivering work products free of defects, and following formal review criteria are all part of the **quality assurance process**. While work products (such as requirements and design papers) are produced throughout the SDLC to ensure consistency and clarity, metrics are utilized to gauge productivity, quality, and project success. Chapter 10 revolves around **Requirement Development and Management**. Gathering and honing both functional (features and capabilities) and non-functional (performance, security) needs is the first step in effective requirement management. There could be **multiple sources** of requirements: users, standards, customer feedback, business policies, etc. All the requirements need to be converted into software features (logical entities). All these features need to be categorized under some hierarchy (top level to bottom level features).**Changes in requirements** needs to be tackled with change control to verify its impact on teams and components. Whenever any changes happen, then there should be provision for automated e-mails to be sent to all concerned teams. **Configuration and version control** system should be located centrally and should be easily accessible to all teams. **Validation of requirements** is done to make sure they meet customer expectations. Version management and change control are essential, particularly for remote teams. Teams can avoid rework by managing version control and staying informed about requirement changes with the use of centralized repositories and automatic notifications. Agile approaches, like Scrum, place an emphasis on accommodating flexibility by managing a backlog of requirements. To reduce the possibility of changes over subsequent iterations, each iteration selects a subset of the backlog's prioritized requirements. Software Requirements Practical Strategies were discussed later. **Quality assurance** involves checking for errors and maintainability as well as confirming completeness, clarity, and testability. Project delays and misunderstandings are avoided with precise, well-defined specifications.

Application in Real Projects: Choosing the right life-cycle model is crucial for project planning. Misalignment with project goals may result from selecting the incorrect model. For multifaceted projects, hybrid model that incorporates aspects of several life-cycle models. Appropriate model enables customized project management that fits the needs of the client and the team's skills. Quality assurance checkpoints at each phase ensure that each deliverable meets set standards before moving to the next phase.QA takes a significant amount of time and manpower, and teams are frequently under time pressure to cut back on QA procedures.QA could be made more efficient by employing predictive analytics to identify potential failure areas and automated quality assurance to manage repetitive testing operations. Metrics such as defect rates or code stability could be continuously monitored to ensure project health. "Box-checking" mentality instead of an emphasis on significant advancement can result from an over-reliance on metrics. Another concern is measurement fatigue, which occurs when a team becomes overwhelmed by tracking too many metrics. Planning could be improved using predictive modeling on past project metrics to anticipate problems in upcoming projects. Incomplete definitions and misconceptions are common during requirement collecting. Accuracy and participation may be improved by using collaborative requirement modeling technologies that allow stakeholders to view and modify requirements in real time. This method minimizes misunderstandings early on by allowing for ongoing stakeholder feedback. By centralizing requirement changes, version control helps teams avoid misalignment and rework in projects where needs vary. Regular modifications can cause team weariness and interfere with workflows. When used with a project's version control, an automated requirement update notification system may help remote teams maintain communication.

Peer Interactions: We talked about different SDLC models in study group session and exchanged opinions on the models we had used in operation. A peer described how overlapping phases accelerated delivery in large-scale telecom project using the Concurrent Engineering Model. This showed me that if the proper dependencies are maintained, concurrent approaches could increase project efficiency. I got challenged to think of different workflows where I could overlap phases for efficiency after learning about the Concurrent Engineering Model used in multidisciplinary projects. Another peer offered methods for managing changing requirements in Agile settings. They talked about a technology that indicates changes that impact numerous stakeholders and emphasized the use of a priority matrix in backlog management to avoid scope creep. I have reconsidered how to prioritize and visualize changes as a result of the recommendation to use change-impact monitoring tools.

Challenges Faced: Selecting the optimal SDLC model for a project can be challenging, particularly for hybrid projects that might profit from a combination of different methodologies (e.g., mixing Agile and Waterfall). It is frequently difficult to commit to a single model because of this ambiguity. It would be beneficial to have further guidelines on when to successfully mix SDLC models. By overlapping phases, the Concurrent Engineering Model offers efficiency; nevertheless, managing dependencies and coordinating activities across teams increase complexity .Creating a strong dependency tracking system is crucial to reducing conflicts. When requirements change and stakeholders make requests for additions during ongoing iterations, scope creep is a common problem in Agile projects. I need to get better at striking the correct balance between control and suppleness. Using change request paperwork and becoming proficient in backlog prioritization might be beneficial. Inconsistencies can arise when team members work with out-of-date information due to a lack of efficient version control and change notification systems. I need to get better at setting up automated notifications in version control systems for notifications.

Personal development: I finished an advanced training on Jira. The goals of this training were to improve backlog management, manage dependencies, and optimize project timeframes. This training has really me in my comprehension of how to organize and rank work in intricate projects, particularly while adjusting to changing specifications. I joined a software project management peer study group that meets once a week to talk about issues like change control, requirement handling, and risk management. Working with peers has shown to be a very successful strategy for obtaining a variety of viewpoints and creative ideas. It has forced me to re-evaluate my strategy and modify tactics that have worked for other people. I spent time reading books and articles about integrating Waterfall and Agile approaches in order to learn more about hybrid project management models.

Goals for the Next Week: Examine hybrid SDLC models with an emphasis on practical applications that blend

Waterfall and Agile methodologies. Practice monitoring, documenting, and communicating requirement changes by spending time with platforms like as Jira and Confluence. In order to practice organizing updates and responding to change requests, I will also create a sample communication plan for a fictitious project. Set aside an hour to practice DevOps concepts, with a particular emphasis on CI/CD pipeline setup. I'll focus on creating a simple pipeline with continuous deployment, automated testing, and version control integration. I'm advancing my career in software project management by learning how to build up and maintain CI/CD pipelines, which will help me in positions where tight coordination with development teams is essential.