**Learning Journal**

**Student Name:** Sannidhi Shetty

**Course:** SOEN-6841 Software Project Management

**Journal URL:** [**https://github.com/sannidhi470/SPM**](https://github.com/sannidhi470/SPM)

**Week 3:** Feb 11 – Feb 17

**Date:** Feb 15

**Key Concepts Learned:**

Below are the concepts I learnt from this week’s session:

**CHAPTER 5: Configuration Management**

Software projects tend to receive frequent change requests depending on the needs of the client/end users. These changes need to be incorporated into the developed software which results into different versions. This process of managing change requests to generate different versions of software product is handled by Configuration Management.

It deals with the questions:

* *WHO? 🡪 Who will make the changes?*
* *WHY? 🡪 Why were the changes made?*
* *WHEN? 🡪 When are the changes made?*
* *WHAT? 🡪 What are the changes?*

**Sources of the changes:**

* Changes can occur due to longer lifecycle of the project. Longer the project runs, more are the chances that the software will be prone to changes.
* Solutions to issues identified can result into more changes to take place.
* Funding/Budget changes
* Technology keeps advancing rapidly which can lead to change requests.
* Customer expectation keeps evolving resulting into catering those in software being developed.

Changes also arise due to the poor practices of development team:

* The bug that was fixed with greater expense reappears.
* The developed feature is not found, or latest version of the source code could not be found.
* After full testing, code doesn’t work/ wrong version of the code was tested.
* Programmers working on wrong version of the code.
* Configuration Management not done properly wherein wrong versions of configuration items are baselined, delivered, or developed in a product.
* Confusion amongst developers to determine which module contains the developed change request.

**Four Key Functions of CM:**

* **Configuration Identification**

The main Question asked: *What is my System Configuration?*

It includes:

* Defining baselines and identifying configuration items
* Documenting baselines, Configuration items and requirements
* Identify data requirements.
* Define and identify acceptance requirements for the change request.
* Establish schema for change request identification.
* **Configuration Control**

The main Question asked: *How do I control changes to my configuration?*

It includes:

* It provides mechanisms in the form of documentation, procedure to prepare, evaluate, approve or disapprove the configuration items identified.
* Identifying who can initiate change request and who is responsible for evaluating, approving and tracking the baseline.
* Check in/ Check out process to effectively track and make changes. This also maintains revision history which is important.
* Defining criteria to include software components for the proposed change request.
* Change impact analysis for each requested change.
* Regression testing to ensure that the changes made is affecting existing features/functionalities.
* Process that SCCB (Software Configuration Control Board) should take to approve the requested change.
* Procedure to follow to update all affected software.
* **Configuration Status Accounting**

The main Question asked: *What changes I have made to the system? What changes are remaining to be implemented?*

It includes:

* Define a mechanism to keep record of the evolution of the system to achieve the identified configuration item/baseline.
* Is the specification approved?
* Is the change request approved by Change control board?
* Which version of configuration item implements approved change request?
* What is different in the new version of the system?
* What was the cause of the change requested reported?
* Generate report on the traceability of all changes to the configuration item throughout the lifecycle of product.
* **Configuration Auditing**

The main Question asked: *Does the system I am developing satisfy the requested change?*

It includes:

* Defining a mechanism to identify if the current state of the system matches to the system pictured in the baseline by verifying if the software is built according to the standards and requirements of change request.
* Ensuring CM procedures are followed and providing a mechanism to establish the baseline.

**Change Control Policy:**

* Every change request must follow the established process for documentation.
* Change control board should approve the requested change and then only it will be acted upon
* Contents of the change request database should be visible to all project members.
* No work should be performed on unapproved change request other than the feasibility check for the approval process.
* The original text of the change request should not be modified or deleted from the database.
* The incorporated change must trace back to the requested change request.

**Chapter 6: Software Project Plan**

A Software Project Plan is a detailed document that determines the objectives, tasks, schedules, resources, and risks associated with a software project. It provides a structured approach to project management and ensures project goals are achieved efficiently and effectively. The plan acts as a document throughout the project lifecycle that helps in planning, executing, monitoring, and controlling the project to deliver the requested software product.

**Parts of a Software Project Plan:**

**Project Scope:** This is to determine the feasibility of the project as to what can be achieved and what can be not. Knowing the scope at the initial phase of the project helps in setting a baseline for developing the project. It clearly outlines what needs to be developed and not deviate into areas that weren't originally intended.

**Requirements:** This step in the plan is to analyze the product development requirements like data, resources, and tools. Clear requirements while planning eliminates misunderstandings and guide the development process efficiently.

**Schedule:** Project scheduling in a software project plan is nothing but setting a schedule to complete a certain task. It ensures that tasks are completed in a timely manner and helps in delivering the product within estimated timeline.

**Budget:** In software project plan, this step is to allocate he budget for a software project based on financial resources for various expenses like salaries, equipment, and software licenses. It's crucial for ensuring that the project is developed within budget and covers all the resources needed for the project.

**Resource Allocation:** This involves assigning resources to the project to determine who will work on what and when, ensuring that team members are appropriately assigned to tasks based on their skills and availability. Resource allocation is done based on effort estimation of the project being developed. Effective resource allocation maximizes productivity and minimizes bottlenecks.

**Risk Management:** Risk management as covered in previous chapter involves identifying potential risks that could impact the project and to develop strategies to identify, analyze, mitigate, or avoid them. By proactively addressing risks, the project team can minimize frequent hindrance in the development and stay focused.

**Types of Software Project Plans:**

**Top-Down Approach:** It begins with high-level planning and gradually breaks down tasks into smaller, more manageable components (Work Breakdown Structure (WBS)). This approach provides a clear direction as the inputs and outputs are clearly defined while allowing for flexibility in execution.

**Bottom-Up Approach:** The Bottom-Up Approach in software project planning is essentially the opposite of the top-down approach. It focuses on starting with detailed tasks or components and then aggregates them into larger project structures instead of starting with high-level planning and breaking down tasks in top-down approach.

**Inputs for Making a Software Project Plan:**

**Project Scope:** As mentioned earlier, it outlines the goals and objectives of the project, providing a foundation for planning and decision-making. This gives a baseline to start with the software project development.

**Requirements:** Clear and detailed requirements is important for software planning as it guides the development process and ensures the requirements identified aligns with the user needs.

**Budget:** The financial resources allocated to the project determine its scope and feasibility. Determining budget influences decision regarding resource allocation and effort estimation.

**Duration:** As per the user needs, duration for the entire development, testing and deployment process is analyzed to understand the expected duration of the project. It helps in scheduling tasks, allocating resources, and cater user expectations. Delivering the software on time is crucial in the software development lifecycle.

**Start and End Dates:** These mark the boundaries of the project timeline, establishing milestones and deadlines for task completion. Milestones are the end point of any project task.

**Techniques for Making a Software Project Plan:**

**Gantt Charts:** It is a visual representation of project schedules to demonstrate the sequence of tasks, their durations, and dependencies. It ensures effective project management throughout the development and to allocate resources for the project.

**Network Diagrams:** It is similar to Gantt chart but in network diagrams, it depicts task sequences and relationships between these tasks to identify critical paths and potential bottlenecks in executing the project.

**PERT/CPM Charts:** It helps analyze project timelines, identify critical tasks, and estimate project completion dates, facilitating effective schedule management and risk assessment.

**Earned Value Management:** This method integrates cost associated with the project, project schedule, and performance metrics to assess project progress. It helps in forecasting future performance of the software which ensures effective decision-making and resource allocation.

**Critical Chain Method:** This method helps identify and prioritize critical tasks which are important for the software which will minimize delays and maximize project efficiency.

Top of Form

**Reflections on Case Study/course work:**

* Case study 5 focuses on Configuration Management. The software vendor employs an incremental iteration development model for their system, engaging both internal and offshore teams. They incorporated a central configuration management system that ensures seamless collaboration across locations, ensuring 24/7 availability. Access rights are granted based on roles where administrators manage documents and super-user role is for overarching control. For the ease in regression testing, Automated smoke testing software is integrated into all branches, verifying code compatibility with the existing build upon check-in. They have a feature of failure notifications that prompt immediate corrective actions by developers and if the issues persist, it is escalated to the global program manager. Developers make sure to keep local builds synchronized with the central system. They conduct end to end tests before deploying code changes. This architecture guarantees continuous integration and makes sure that the change requests are handled in an efficient manner considering the policies defined.
* Attending this weeks’ class on Software Configuration Management and Software project plan gave me insight onto how to deal with incoming change request and the different procedure and policies to consider in CM. I understood the importance of Software project plan as it sets the baseline for the entire development of the software and is a crucial step in the software lifecycle.

**Collaborative Learning:**

* Had a study session with a classmate to prepare for the midterm and to discuss Chapter 5 and 6. We discussed how we incorporated incoming change requests from clients on our respective firms we worked in. On my project specifically, my project manager used to receive change request document from the client according to the format given in the chapter. It included the module of the software that needs to be changed and explanation of the functionality to implement. Once the project Manager approves the changes, it was sent to Business analyst to reframe the document as per the source code and developed software so that the developers would clearly understand the requirement in change request. As for project planning, we used to follow top-down approach to divide the task into smaller components. Flow charts were used to visualize the process of developing the software.
* Had project meeting with team members to finalize the first project deliverable and to discuss pitch presentation for next class.

**Further Research/Readings:**

* I encountered "Configuration Management Principles and Practice" by Anne Mette Jonassen Hass that offers a thorough exploration of configuration management in software development. The book covers essential concepts covered in the chapter such as configuration identification, change management, configuration status accounting and audits. It focuses on the importance of maintaining consistency and reliability throughout the development lifecycle. I liked how the author provides practical guidance on implementing configuration management processes in the software by considering real world case studies selecting tools and conducting audits and reviews throughout the development process.

**Adjustments to Goals:**

* The goal for last week was to complete the first deliverable of the project. We worked and documented it throughout the week with regular team meetings. We were able to submit the deliverable on time.
* Goal for next week is continue preparing for midterms by going through the covered chapters in class and read extra materials on the topics to enhance my practical knowledge on the concepts.