**Learning Journal**

**Student Name:** Yug Kotak

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

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

**Week 3:** Feb 11 – Feb 17

**Date:** Feb 17

**Key Concepts Learned:**

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

**CHAPTER 5: Configuration Management**

Software projects often encounter frequent change requests based on the requirements of clients or end users. These modifications necessitate integration into the existing software, leading to the creation of distinct software versions. The responsibility for managing these change requests and generating various software versions is undertaken by Configuration Management.

This process involves addressing fundamental queries:

* *WHO? → Identifying the individuals responsible for making the changes.*
* *WHY? → Understanding the reasons behind the proposed changes.*
* *WHEN? → Determining the timing of the changes.*
* *WHAT? → Specifying the nature of the changes.*

**Causes of changes in software projects include:**

1. **Extended Project Lifecycle:** Longer project durations increase the likelihood of software being susceptible to alterations.
2. **Issue Resolutions:** Addressing identified problems may lead to additional changes.
3. **Financial Adjustments:** Changes can stem from alterations in funding or budget allocations.
4. **Technological Advancements:** Rapid technological progress can result in requests for changes.
5. **Evolving Customer Expectations:** Shifting customer preferences contribute to modifications in the software development.

Changes can also arise from suboptimal development practices:

1. **Recurrence of Fixed Bugs:** Previously resolved bugs may reappear, incurring additional costs.
2. **Difficulty in Locating Developed Features:** Challenges in finding developed features or the latest source code version.
3. **Code Testing Issues:** Post full testing, issues may arise, or the wrong code version might have been tested.
4. **Incorrect Code Versions:** Programmers may inadvertently work on the wrong code versions.
5. **Configuration Management Challenges:** Issues with Configuration Management may result in baselining, delivering, or developing incorrect versions of configuration items.
6. **Developer Confusion:** Ambiguity among developers about which module contains the developed change request.

**Four Primary Functions of CM:**

1. **Configuration Identification**

* Key Question: What is my System Configuration?
* It involves:
* Clearly defining baselines and recognizing configuration items.
* Documenting baselines, configuration items, and associated requirements.
* Identifying data requirements.
* Specifying and recognizing acceptance requirements for change requests.
* Establishing a structured schema for the identification of change requests.

1. **Configuration Control**

* Key Question: How do I control changes to my configuration?
* It involves:
* Providing mechanisms, including documentation and procedures, to prepare, assess, approve, or disapprove identified configuration items.
* Identifying individuals authorized to initiate change requests and determining responsibilities for evaluating, approving, and monitoring baselines.
* Implementing a check-in/check-out process to systematically monitor and implement changes, maintaining a crucial revision history.
* Establishing criteria for incorporating software components in proposed change requests.
* Conducting change impact analyses for each requested change.
* Performing regression testing to ensure that the implemented changes do not adversely affect existing features or functionalities.
* Outlining the process that the Software Configuration Control Board (SCCB) should follow to approve requested changes.
* Defining procedures for updating all affected software in response to approved changes.

1. **Configuration Status Accounting**

* Key Question: What changes have been made to the system? What changes are pending implementation?
* It involves:
* Establishing a mechanism to record the system's evolution and achieve the identified configuration item/baseline.
* Addressing queries such as whether the specification has been approved, if the change request has received approval from the Change Control Board, and which version of the configuration item implements the approved change request.
* Identifying differences in the new version of the system.
* Analyzing the cause of reported change requests.
* Generating a comprehensive report on the traceability of all changes to the configuration item throughout the product's lifecycle.

1. **Configuration Auditing**

* Key Question: Does the system I am developing fulfill the requested change?
* It involves:
* Establishing a mechanism to determine whether the current state of the system aligns with the system envisioned in the baseline, ensuring that the software is constructed in accordance with the standards and requirements of the change request.
* Ensuring adherence to Configuration Management (CM) procedures and implementing a mechanism to establish the baseline.

**Change Control Policy:**

* All change requests must adhere to the prescribed documentation process.
* Approval from the change control board is mandatory before any requested change is implemented.
* All project members should have access to the contents of the change request database.
* No actions should be taken on unapproved change requests, except for conducting a feasibility check as part of the approval process.
* The original text of the change request in the database should remain unmodified or deleted.
* Every incorporated change should be traceable back to the initially requested change.

**Chapter 6: Software Project Plan**

A Software Project Plan is a comprehensive document outlining the goals, tasks, schedules, resources, and risks associated with a software project. It establishes a systematic approach to project management, ensuring the efficient and effective achievement of project objectives. Throughout the project lifecycle, the plan serves as a guiding document for planning, executing, monitoring, and controlling the project to successfully deliver the requested software product.

**Parts of a Software Project Plan:**

* **Project Scope:**
  + Objective: To assess the project's feasibility and establish a baseline for development.
  + Clarifies what needs to be developed, preventing deviations into unintended areas.
* **Requirements:**
  + Purpose: Analyzing product development requirements, including data, resources, and tools.
  + Clear requirements during planning eliminate misunderstandings and guide the development process efficiently.
* **Schedule:**
  + Objective: Setting a timeline for completing specific tasks.
  + Ensures tasks are completed in a timely manner, facilitating the delivery of the product within the estimated timeline.
* **Budget:**
  + Purpose: Allocating the budget for the software project based on financial resources.
  + Crucial for ensuring the project is developed within budget and covers all necessary resources like salaries, equipment, and software licenses.
* **Resource Allocation:**
  + Involves assigning resources to determine who will work on specific tasks and when.
  + Ensures team members are appropriately assigned based on their skills and availability, maximizing productivity and minimizing bottlenecks.
* **Risk Management:**
  + Covers the identification of potential risks impacting the project.
  + Strategies are developed to identify, analyze, mitigate, or avoid risks, allowing the project team to proactively address challenges and stay focused on development.

**Types of Software Project Plans:**

**Top-Down Approach:** This method initiates with high-level planning and progressively dissects tasks into more manageable components, following a Work Breakdown Structure (WBS). The approach offers a clear direction, with well-defined inputs and outputs, while maintaining flexibility in execution.

**Bottom-Up Approach:** In software project planning, the Bottom-Up Approach is essentially the reverse of the top-down approach. It starts with detailed tasks or components and aggregates them into more extensive project structures, as opposed to commencing with high-level planning and breaking down tasks.

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

**Project Scope:** As previously mentioned, this outlines the project's goals and objectives, serving as a foundation for planning and decision-making. It establishes a baseline for commencing software project development.

**Requirements:** Clear and detailed requirements are crucial for effective software planning, guiding the development process and ensuring alignment with user needs.

**Budget:** The financial resources allocated to the project determine its scope and feasibility. Budget determination influences decisions related to resource allocation and effort estimation.

**Duration:** Based on user needs, the duration for the entire development, testing, and deployment process is assessed to understand the expected project duration. This aids in scheduling tasks, allocating resources, and meeting user expectations, emphasizing the importance of timely software delivery.

**Start and End Dates:** These define the project timeline boundaries, setting milestones and deadlines for task completion. Milestones represent the endpoint of various project tasks.

**Approaches for Developing a Software Project Plan:**

1. **Gantt Charts:** These charts visually represent project schedules, illustrating the sequence of tasks, their durations, and dependencies. They ensure efficient project management throughout development and aid in resource allocation.
2. **Network Diagrams:** Similar to Gantt charts, network diagrams portray task sequences and relationships, identifying critical paths and potential bottlenecks in project execution.
3. **PERT/CPM Charts:** These charts assist in analyzing project timelines, identifying critical tasks, and estimating project completion dates. They facilitate effective schedule management and risk assessment.
4. **Earned Value Management:** This method integrates project costs, schedules, and performance metrics to evaluate project progress. It aids in forecasting future software performance, ensuring effective decision-making and resource allocation.
5. **Critical Chain Method:** This method identifies and prioritizes critical tasks crucial for software development, minimizing delays and maximizing project efficiency.

Top of Form

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

* Case study 5 centers on Configuration Management. The software vendor adopts an incremental iteration development model for their system, involving both internal and offshore teams. They implemented a central configuration management system, ensuring seamless collaboration across locations and continuous 24/7 availability. Access rights are assigned based on roles, with administrators managing documents and a super-user role for overarching control. Automated smoke testing software is integrated into all branches for convenient regression testing, verifying code compatibility with the existing build upon check-in. The system includes a feature for failure notifications, triggering immediate corrective actions by developers. If issues persist, they are escalated to the global program manager. Developers ensure local builds are synchronized with the central system, conducting end-to-end tests before deploying code changes. This architecture ensures continuous integration and efficient handling of change requests in line with defined policies.
* Participating in this week's class on Software Configuration Management and Software Project Plan provided me with insights into managing incoming change requests and the various procedures and policies to consider in CM. I gained an understanding of the crucial role of a Software Project Plan in setting the baseline for the entire software development process, emphasizing its significance in the software lifecycle.

**Collaborative Learning:**

* Engaged in a study session with a fellow classmate to prepare for the midterm and discuss Chapters 5 and 6. We shared insights on how we handled incoming change requests from clients in our respective workplaces. In my project, the client submitted change request documents to the project manager following the format outlined in the chapter. The document specified the software module to be changed and provided an explanation of the functionality to implement. Upon project manager approval, the document was forwarded to the business analyst for refinement to align with the source code and developed software, ensuring clear understanding for the developers. Regarding project planning, we adhered to a top-down approach, breaking down tasks into smaller components. Flow charts were instrumental in visualizing the software development process.
* Conducted a project meeting with team members to finalize the first project deliverable.

**Further Research/Readings:**

* Came across "Configuration Management Principles and Practice" by Anne Mette Jonassen Hass, offering a comprehensive exploration of configuration management in software development. The book delves into essential concepts covered in the chapter, including configuration identification, change management, configuration status accounting, and audits. Emphasizing the importance of maintaining consistency and reliability throughout the development lifecycle, the author provides practical guidance on implementing configuration management processes in software through real-world case studies, tool selection, and conducting audits and reviews throughout the development process.

**Adjustments to Goals:**

* Last week's goal was to complete the first deliverable of the project. The team worked collaboratively, documented the deliverable, and held regular meetings, successfully submitting it on time.
* The goal for the upcoming week is to continue preparing for midterms by revisiting covered chapters in class and reading additional materials on the topics to enhance practical knowledge of the concepts.