Software Project Management

Learning Journal

Name: Adarsh Manojkumar Pawar

Course: Software Project Management

Journal link: https://github.com/AdarshPawar/LearningJournal.git

Date: 16/02/2024

Week 4: February 11 - February 17

Concepts Learned

Chapter 5 - Configuration Management

In this chapter, we are introduced to configuration management system, an essential component of a software project where the various artifacts produced during the development cycle are safely and securely kept and are easily accessible whenever required.

- During the development of software, a huge number of artifacts are created and each artifact serves a specific purpose and helps the team communicate, collaborate, and keep track of progress throughout the project. Due to the inherent nature of software development, there can be several different versions of the same artifact created.
- Now the question arises for the development team, on which one of them is the right version and how every team member of every group can access and work on it?
- This is where the implementation of a good configuration management system comes in handy. A good configuration management system should not only be able to store, archive, identify, retrieve, and release work products and information items for the entire project team but also provide the following functionalities -

I. Centralized configuration management system -

i. A centralized configuration management system serves as a single repository where all artifacts are stored and managed.

ii. This ensures that all team members have access to the latest version of project assets, such as source code, documentation, and configuration files, from one centralized location.

II. Secured access mechanism with role-based access control -

i. Secured access with role-based access control regulates user permissions within the configuration management system based on predefined roles and permissions.

ii. Users are assigned specific roles, determining their level of access to different artifacts. This mechanism ensures that only authorized personnel can perform specific actions, mitigating the risk of unauthorized access, modifications and data breaches.

III. **Continuous integration of software build with smoke test facility -**

i. Continuous integration involves automatically integrating code changes into a shared repository and running automated builds and tests.

ii. The inclusion of a smoke test facility allows for the rapid execution of basic tests to validate critical functionalities of the software.

iii. Smoke tests serve as an initial quality assurance measure, identifying major defects early in the development cycle and providing timely feedback to developers.

IV. Easy branching mechanism to branch out an entire software version -

i. Branching mechanism enables developers to create separate branches or copies of the code to work on independent features or fixes.

ii. This mechanism enables parallel development and experimentation of features without disrupting the main branch code. It simplifies the process of creating, merging, and managing branches, and ensures code isolation.

V. Audit facility -

i. An audit facility maintains a detailed and extensive record of all user activities and system events within the configuration management system.

ii. It also maintains a detailed history of modifications, ensuring that both the old and new versions of documents are accessible for reference. Additionally, a timestamp accompanies each change, providing precise information about when modifications were made to any document.

Chapter 6 - Project Planning

In this chapter, we are introduced to Software Project Plan, an elaborate document that outlines the goals, tasks, timelines, resources, and potential risks involved in a software project. Its main purpose is to establish a systematic framework for managing the project, ensuring that its objectives are met in a timely and efficient manner.

Throughout the project's lifecycle, this plan serves as a guiding document, facilitating the planning, execution, monitoring, and control of activities to deliver the desired software product. There are 2 different types of project plans -

I. Top-Down Approach

I. This method starts with broad planning and gradually breaks tasks into smaller, more manageable parts using a Work Breakdown Structure (WBS).

II. It offers a clear direction as inputs and outputs are clearly defined, yet allows for flexibility in execution.

III. It is typically used in product development scenarios, where project timelines and feature releases are pre-established.

II. Bottom-Up Approach

i. The Bottom-Up Approach begins with detailed tasks or components and then combines them into larger project structures, rather than starting with broad planning and breaking tasks down.

ii. It is is commonly used in custom software development, where project timelines and feature releases are determined based on project requirements and estimation.

Different parts of software project plan -

I. Project Scope planning : This entails figuring out what can be achieved and what cannot be achieved in the project. Understanding the scope early on helps establish a starting point for the project's development. It clearly defines what needs to be done and prevents the project from straying into areas that were not originally intended.

II. Supplier Management Planning: This part focuses on managing relationships with external vendors or suppliers who provide goods or services essential for the project. It includes activities such as selecting suppliers, negotiating contracts, and monitoring performance.

III. Requirements planning: This step involves analyzing the requirements for product development, such as data, resources, and tools. Having clear requirements during planning prevents misunderstandings and guides the development process efficiently.

IV. Configuration Management Planning: Configuration management involves managing changes to project artifacts, such as code, documents, and other deliverables, while ensuring version control and consistency across the project.

V. Time/Schedule planning: Project scheduling is simply setting a timeline to complete specific tasks. It ensures that tasks are completed on time and helps deliver the product within the estimated timeframe.

VI. Cost/Budget planning: In the software project plan, this step involves allocating funds for various expenses like salaries, equipment, and software

licenses. It's crucial for ensuring that the project stays within budget and covers all necessary resources.

VII. **Tool Planning:** Tool planning involves selecting and deploying appropriate tools and technologies to support project activities efficiently. It includes evaluating available tools, acquiring licenses, and providing training to team members.

VIII. **Resource planning:** This includes assigning resources to the project to determine who will work on what and when. It ensures that team members are assigned tasks based on their skills and availability. Effective resource allocation maximizes productivity and minimizes bottlenecks.

IX. **Risk Management:** Risk management involves identifying potential risks that could affect the project and developing strategies to address them. By addressing risks proactively, the project team can minimize disruptions and stay focused on development.

Techniques for making software project plan -

I. **Gantt Charts:** Gantt Charts are visual tools that represent project schedules in a clear and organized manner, showcasing the sequence of tasks, their durations, and dependencies. By providing a visual roadmap, Gantt Charts aid in effective project management by allowing project managers to allocate resources efficiently and monitor progress throughout the development process. They serve as a valuable communication tool, helping team members understand project timelines and responsibilities, thus promoting collaboration and coordination.

II. **Network Diagrams:** Network Diagrams are similar to Gantt charts but focus on depicting task sequences and relationships between tasks. By illustrating these connections, Network Diagrams help identify critical paths and potential bottlenecks in executing the project, enabling project managers to optimize resource allocation and scheduling for smoother project execution. They provide a comprehensive overview of project flow and interdependencies, facilitating better decision-making and risk management.

III. **PERT/CPM Charts:** PERT/CPM Charts assist in analyzing project timelines by identifying critical tasks and estimating project completion dates. By breaking down complex projects into manageable components, PERT/CPM Charts facilitate effective schedule management and risk assessment, allowing project managers to prioritize tasks and allocate resources accordingly. They

offer insights into project dependencies and potential delays, enabling proactive measures to mitigate risks and ensure project success.

IV. **Earned Value Management:** Earned Value Management integrates project cost, schedule, and performance metrics to assess project progress accurately. By comparing planned costs and schedules with actual performance, Earned Value Management enables project managers to forecast future project performance, facilitating informed decision-making and resource allocation to ensure project success. It provides stakeholders with a comprehensive understanding of project health and performance trends, fostering transparency and accountability.

V. **Critical Chain Method:** The Critical Chain Method focuses on identifying and prioritizing critical tasks essential for software development. By minimizing delays and maximizing project efficiency, this method helps project managers streamline project execution and meet project objectives within stipulated timelines. It emphasizes resource optimization and focuses on removing constraints to enhance project flow, ultimately improving project outcomes and customer satisfaction.

Reflections on case study

I wanted to get a clearer picture of configuration management in an enterprise setting when I came across this case study where a large financial institution worked on the development of a mobile banking application. The project scope included account management, fund transfers, bill payments, and mobile deposit services.The software required frequent updates and enhancements to meet evolving user requirements and regulatory changes.

Action

- The project team implemented a robust configuration management process using version control systems, Git.
- Each software component, including code, documentation, and configuration files, was meticulously tracked and managed using Git repositories.
- Branching and merging strategies were established to manage concurrent development efforts and ensure code integrity.

Result

- The implementation of configuration management enhanced collaboration among team members by providing a centralized repository for sharing and tracking project artifacts.
- Clear ownership and accountability were established for each configuration item, ensuring that changes were properly documented and authorized.

Action

- Configuration items were systematically labeled and linked to their corresponding business requirements, architectural designs, and test cases.

Result

- The traceability and auditing practices established in configuration management not only ensured compliance and accountability but also facilitated continuous improvement.
- Through regular audits, discrepancies or inconsistencies in configuration items were identified and addressed promptly, leading to enhanced data integrity and reliability.

Collaborative Learning

- My friend and I planned to study for the midterms with the objective of going over chapter 5. We discussed our experiences with different version control systems in our respective companies.
- While my company relies on Perforce for managing our extensive codebase and binary files, my friend's company has adopted Git for its distributed nature and flexible branching model. My friend highlighted Git's ability to work offline and sync changes seamlessly, as well as its lightweight branching model, which allows for rapid iteration and collaboration.
- Despite the benefits of Perforce in enforcing access controls and managing large binary files, I was intrigued by Git's popularity among developers and its extensive ecosystem of third-party tools. The conversation shed light on the diverse ways in which version control systems can impact development workflows and productivity, prompting me to consider exploring Git further for potential enhancements to our current practices with Perforce.

Further Research/ Readings

I recently came across "Configuration Management Best Practices: Practical Methods that Work in the Real World" by Bob Aiello and Leslie Sachs, which offers a detailed guidance on implementing effective configuration management practices.

The book also covers essential topics such as version control, change management, and release management, providing practical methods and strategies for establishing robust CM processes.

What I found particularly helpful were the real-world examples and case studies used throughout the book, which illustrated key concepts and demonstrated how CM practices can be applied in different organizational contexts. The authors' emphasis on the importance of CM in ensuring software quality and compliance with industry standards resonated with me, and their perspectives made it easy to understand and implement CM best practices.

Adjustment to goals

- Last week, our aim was to finish the first deliverable of the project. We spent the week working on it, meeting as a team regularly, and making sure everything was inline with the rubrics and the in-charge TA . Luckily, we managed to hand in the work on schedule.
- The plan for the next week is to prepare for the upcoming midterms. Go over the course material in detail, make a list of questions to ask the professor and refer to the reference material.