|  |  |
| --- | --- |
| **Student Name:** Yehia Metwally | **Course:** SOEN 6481 |
| **Dates Range of activities:** 9 February – 23rd February 2025 | **Date of the journal:** 22nd February 2025 |
| **Journal URL:**  https://github.com/yehiametwally55/SOEN-6841.git | |

**Learning Journal 3**

***Key concepts Learned:***

This week, I learned some key concepts from Chapter 4, 5 and 6 while I was studying for the midterm exam that I wasn’t fully familiar with before. In Chapter 4, the idea that risks can be categorized into budget, schedule, resource, quality, and technology risks helped me see how different aspects of a project can be vulnerable to failure. The solution of adding buffer time stood out as well, as I wasn’t aware of the different types like task-level buffers, project-level buffers, and feeding buffers. This makes a lot of sense as it ensures that even if one part of the project is delayed, the rest doesn’t get derailed, and the quality of the work is preserved. I also learned about the importance of implementing a knowledge management system to store information and insights gathered by team members throughout the project. This can be particularly helpful when you need to take over someone’s tasks or understand how a specific feature was implemented, especially in software projects where tasks are interdependent and rely on integration with each other.

Moving to Chapter 5, while I have previously worked with configuration management systems as part of my role as a software developer, one new and particularly interesting concept for me was the idea of installing an automatic smoke testing tool. I hadn’t used or heard of something like Cruise Control before, but I can now see how useful it would be to have such a tool that constantly checks the source code whenever a new piece of code is checked into the system. The fact that it continues to generate error messages until the wrong code is rectified could save so much time and ensure that the building remains stable. This would add an extra layer of quality control to projects and make the development process smoother, catching issues earlier.

Moving to Chapter 6, In this chapter, I learned about Software Project Planning, which involves defining project objectives, scope, timelines, and resources. One of the key takeaways was understanding the different types of planning, including Top-down and Bottom-up planning. Additionally, the Work Breakdown Structure (WBS) was introduced as a systematic way to break down project tasks while maintaining dependencies. I also explored various project scheduling techniques like the Critical Path Method (CPM) and Goldratt’s Critical Chain, which help optimize project timelines. Another crucial aspect covered was resource allocation, where the correct assignment of personnel and workload distribution ensures project efficiency. Moreover, the chapter emphasized the importance of milestones and deliverables in tracking progress, as well as supplier and communication planning, which help manage external dependencies and ensure clear communication across all stakeholders. Finally, I learned about quality assurance and budgeting, two critical components that ensure the project meets quality standards while staying within financial constraints.

***Application in Real projects:***

In Chapter 4, I learned about the importance of identifying risks early on, which is something I hadn't fully appreciated before. In freelance work, I’m responsible for everything from selecting the technology stack to managing client expectations. One example that stood out was the concept of technology risks, such as choosing a framework like React or Angular that might later prove incompatible with a project's long-term needs. I now understand how important it is to conduct a feasibility study before committing to a stack, weighing factors like future support and scalability.

Additionally, the concept of scope creep as it’s something I frequently deal with when clients continuously ask for new features. By defining project boundaries early on through a well-defined project charter, I can better manage these requests and keep the project on track.

The case study in chapter 5 on the U.S.-based software vendor really emphasized the importance of having a configuration management system, especially in projects with incremental development. The idea of implementing automatic smoke testing stood out to me as a practical way to maintain code quality. I’ve worked in environments without such tools, and I can see how much smoother the development process would be if we had something that automatically flagged code issues as soon as new code was checked in. This would not only save time on manual testing but also ensure that the project stays stable and reliable throughout the development cycle. One interesting idea I recently stumbled upon was the use of AI-driven predictive analytics to manage risks. Tools like Jira and GitLab now offer these features, and I hadn’t considered how useful they could be.

For chapter 6, Applying these concepts to real projects, I realized how project management tools such as Gantt charts and activity networks help in visualizing task dependencies and schedules. The choice between Agile and Waterfall planning depends on the software development model, influencing how iterations are managed. Risk and contingency planning also play a crucial role in mitigating unexpected delays by incorporating time buffers. Additionally, in projects involving supplier management, clear contracts and expectations must be set to maintain consistency in product quality.

***Peer interactions***

In one of the meetings for our group project we had a discussion on some of the concepts covered. One of the most engaging topics was scope creep. We debated whether it’s always negative or if, in some cases, it can be managed in a way that improves the overall project outcome. Some group members felt strongly that scope creep should be avoided because it leads to budget overruns and delays. On the other hand, others argued that if handled properly, scope creep could actually enhance the project by allowing it to evolve and better meet the user’s needs. This discussion made me realize that, while scope creep can be risky, it doesn’t always have to be a bad thing if it's handled properly. We also shared personal stories of past projects where things went wrong, particularly when projects crashed unexpectedly. Many of us agreed that having automatic smoke testing tools could have prevented some of those failures. Hearing these interestingly funny experiences reinforced the importance of early detection and continuous testing.

***Challenges Faced:***

In one of the meetings for our group project, we had a discussion on proactive vs. reactive risk management. Some of my peers emphasized the importance of being proactive to avoid issues later, while others pointed out that focusing too much on potential risks can lead to over-planning and wasting time. They suggested that addressing risks as they arise might be more efficient. While I believe proactively assessing risks is important, I can see how addressing risks as they come could be a better approach in certain projects or environments. I personally torn between the two when thinking about this balance. On one hand, proactively identifying risks seems like the safest route, continuously assessing risks could help prevent major issues down the road.

However, I also agree with the argument that focusing too much on what might go wrong can slow down the project, especially in fast-paced environments. A reactive approach, dealing with risks as they occur, could save time upfront and allow for more flexibility, as some of my peers suggested. The challenge I'm still grappling with is finding the right balance between these two approaches. How can I ensure that we’re well-prepared for potential issues without overcomplicating the project with excessive risk planning?

***Personal development activities:***

During this period, I focused on enhancing my professional development by doing exercise 3.2, which was Agile projects may have less effort required compared to traditional projects. What factors are responsible for this phenomenon? I explored the advantages of agile that make it require less effort such as iterations, Constant feedback, less useless documentation and remote teamwork documentation. And I solved the review question 3.2 which was describe the COCOMO technique for deriving effort and cost estimates for software projects? So I explored the difference between basic, intermediate and detailed COCOMO and their use cases.

***Goals for the next Week:***

My goal is to focus on automatic smoke testing tools and look into how I can implement them into my current projects, aiming to improve my code quality and streamline my development process further. This would improve my ability to automate testing and enhance the quality of my work but also increase efficiency