ERP System Postmortem

Team 2

GINA CODY School of Engineering and Computer Science

Department of Computer Science and Software Engineering

Concordia University

SOEN 390 - Winter 2021

Introduction :

This postmortem will discuss everything we experienced during the development process of our ERP system. The ERP system is an enterprise resource planning tool that helps companies manage resources and interaction between different departments in an efficient way. At the beginning of the development process, the ERP was a bit ambiguous for most of us, but in the end of each sprint and after a lot of brainstorming the final image of the product became more and more clearer. In our opinion, we did produce a very fair amount of minimum requirements. During SOEN390 we managed to develop different parts of the ERP system such as the Manufacturing department and Accounting department. Furthermore, we used several tools in order to produce our final products, some that we were familiar with before (Java, spring framework) and some other that we had to learn from the top (MySQL, Jira).

During this semester we had a couple of obstacles that we had to overcome that hindered our internal deadlines and production quality. This was mainly due to the following issues: a steep learning curve, communication difficulties, poor enforcement to the internal deadlines, methodologies not being followed, inconsistencies with some documents. However, the good always outbalances the bad because we had the following advantages: we chose the right tools, relied on pair programming, had good team dynamic, set up frequent meetings, and we relied on brainstorming techniques to be on the same page.

What went wrong

1 - Learning Curve:

The learning curve for some tools was very steep since everyone has their own rhythm and also other responsibilities to other classes. This can delay some internal delivery deadlines between team members as a lot of parts done by the frontend can depend on the backend. One of the main reasons that contributed to having a steep learning curve is a lot of us have different experiences with different language tools. Furthermore, since all of us are close to graduation we have a big workload that prevents us from being 100% focused on our tasks and investing time to learn new tools.

The way to address this issue is to have pair programming methodology so that people struggling with some concepts can be brought up to speed with the rhythm of the group and setting up meetings to check on the progress with respect to the requirement of each sprint. We also have had streams (live demo) done by the team leader to show us how some tools work.

2 - Communication:

Communication was often a common root obstacle encountered when discussing rising problems and existing conflicts within the project. As a matter of fact, several team members have never worked together, and most importantly, the entire team consisting of nine individuals, never collaborated as a whole unit. Thus, it took some time to adapt to each

other's methods and working styles, as well as taking into account the different time zones, preferred method of communication, etc.

As the team decided to separate the tasks between a frontend and backend team, there was inevitably a lack of relay between both parties. Though some common ground has been settled at the start of each sprint, because some changes were necessary along the way as the project progressed, the failure to communicate changes or intentions successfully made it hard for the team to find cohesion and consistency. It therefore resulted in last-minute updates to the application, several back and forth, unexpected surprises, and amongst others, reverting to previous understandings.

This was mainly addressed with meetings between the leads, where the discussion steered around finding a solution to respect both frontend and backend's work, while taking into account productivity, efficiency and efficacy. Because the topics in question were often redundant as different members came into play at every iteration, a lot of cycles were made before finally achieving the needed changes.

Instead of trying to find a time to accommodate all nine team members, it would be more efficient to conduct meetings when *most* members are available. This allows for more frequent meetings, and more regular discussions of the project's progress, current blocks, comments, and more. Furthermore, by making sure members are assigned to a specific task, redundant discussions are avoided, since common grounds will be achieved more quickly.

3 - Enforcing internal deadlines:

The issue was that internal deadlines were not respected. During the sprint, we had set a date around a week ahead to check-in on everyone's progress, this would be the halfway point so we expected people to have at least 50% of the work done. However, most of the time, work was not done by this internal deadline. Our internal deadlines seemed to have no impact on the push for work to be completed.

The way we addressed this issue was to have a more formal meeting on the day of the internal deadline where we'd have to demo our progress. This forces us to come up with something at least semi functional. This helped for the 4th sprint. Additionally, we tried to be more conscious of prior commitments from each member as well as foresee possible obstacles in developing such features. We tried to plan the internal deadlines more realistically with these in mind. We also believe that talking about the impacts of missing the deadline could propel us to improve our time management.

So we believe that forcing members to demo their progress on the day of the internal deadline as well as evaluating the feasibility of the deadline date could have sped up our progress each sprint or at least reconsider our expectations.

4 - Methodology (git, branches):

One of the main issues regarding our GitHub repository was that we had several branches that were never deleted even after merging to the master. We used feature-based development which breaks up the branches based on the needs. Yet, it became difficult to manage all branches, and we often had conflicts while merging with other branches. Furthermore, the naming convention for our branches was inconsistent. The CI pipeline was built such that only the branches starting with “backend/” would conduct all the maven tests, since “frontend/” branches should have no impact on the backend code and shouldn’t need to execute the regression tests. For this reason, for the first few sprints, everyone followed a specific syntax: “backend /[Jira issue number - feature name]” or “frontend/[Jira issue number - feature name]”. As the sprints went by, we deviated and didn’t follow the convention anymore which made it even more difficult to find the feature on the corresponding branch.

The same issue occurred with our commit messages. For organization purposes, our goal was to start each commit message with the Jira task number. However, this was only followed for the first few sprints and was forgotten as we approached the end of the project.

To avoid confusion and manage branches properly, we should have deleted them every time we merged. Additionally, we should have protected our master branch to avoid conflicts. Lastly, enforcing a rule to follow the naming convention for branches and commit messages should have been done.

5 - Consistency:

Consistency is difficult because it requires constant effort and commitment. SOEN390 was the team’s first experience working in a large group with members possessing not only different skill sets, but lacking familiarity with each other. The reason as to why these inconsistencies occurred is due to other factors of what went wrong during our project. Communication issues and poor enforcement of internal deadlines were our weaknesses. Though the inconsistencies were notified by either our TA or teammates, the addressing of the problems were belated.

Each sprint required certain deliverables to be created and/or updated. Starting with the Risk Management Plan (RMP), one of our solutions on how to avoid/tackle risks was not indicated in the table and therefore presented an inconsistency to what the team presented during the first demo. The Release Plan, which is a layout of our tasks and corresponding deadlines, was not scheduled in a logical order. The inconsistency in the dates had to be changed and was taken into consideration for every iteration onwards, as the Release Plan was one of the deliverables which required constant updates. There were also instances throughout the sprints where the use case diagrams that corresponded to the user stories for that iteration were not updated on the SAD. In addition to the SAD, the following deliverables needed constant updates or were added after Sprint 2: RMP, User Stories backlogs, Release Plan, Defects tracking report, and Testing plan. Lastly, mutual understanding and constant communication with each other could be and should be prioritized to do better next time.

What went right

1 - Right Tools:

The technological stack used is often paramount in defining the success of the project. At the beginning of the project, we had a team meeting to discuss the technologies we are familiar with, and what shall be the best options for the task at hand. In the end, we employed the following stack:

- Spring, a Java framework for backend: Most of the team was familiar with Java, and Spring being a fully fledged web framework was perfect for developing integrated, enterprise level softwares. Although the learning curve of the framework was rather steep, by the end of the project everyone was adequately familiar with it, and all the features satisfactorily implemented.
- React, a JavaScript library for frontend: It is a very popular library with tonnes of documentation, has high performance and fast development speed. The frontend team was familiar with JS, and was able to pick up React because of its simplicity, particularly under the guidance of the frontend lead.
- MySQL: For ERP systems, relational databases are the best choice. MySQL was chosen for its simplicity, maturity and familiarity.
- GitHub for version control: Everyone was familiar with it and it's an industry standard, so it was a natural choice.
- Jira for issue and project tracking: Another industry standard. Although no one was familiar with it, it was very intuitive and allowed the team to keep track of the progress with its agile boards.
- TravisCI for continuous integration: Its seamless integration with GitHub made its selection a no-brainer.

In retrospect, the stack proved sufficiently adequate for the purpose of this project. Perhaps PostgreSQL could have been chosen instead of MySQL as it warrants better performance, but MySQL worked for the most part.

The only issues that occurred due to the choice of tools were not due to their limitations, but the team's abilities to pick up the stack. The beginning of the project was rather slow because the team was learning the stack employed. This pick-up time could have been reduced with the members already familiar with the stack guiding the members unfamiliar with it from the beginning.

2 - Pair Programming/Feedback:

One of the things that went right in our team is frequent pair programming. There were members who had different kinds of technical expertise and pair programming was a way to pass on their knowledge to the rest of the team through visual demonstration. This allowed us to be more confident in our developed solutions and also gave us an opportunity to understand each other's skills and to learn from our peers. Moreover, we also had fewer coding mistakes since there was always a second member looking over it. Through pair programming, we would get instant feedback from our teammates. Pair programming would

have been a more efficient and useful learning experience if the peers who were coding were alternated frequently over the course of the project so that everyone benefits from an opportunity to work with various people.

3 - Team Dynamic:

One of the strong points that we had for this project was our team dynamic. As a team, we were able to communicate honestly and clearly with each other. Everyone on the team was very approachable and were always willing to answer any queries. Furthermore, all the team members got along well and as the semester went on, we were able to form closer relationships with each other. Throughout the course of the project, we learned about each other's strengths and weaknesses, allowing us to capitalize on them and in turn optimize our workflow. The pair programming sessions and brainstorming sessions in particular played big factors in learning about each other's skill sets.

4 - Frequent Meeting:

As the team was divided into two main parties, either backend and frontend, it made it easier to handle the tasks with a micro approach. During the first few sprint iterations of the project, both teams had frequent meetings to keep up to date with the newly acquired knowledge on the technologies, new for some. It also allowed the team to discuss matters at quicker intervals, resolving potential problems at an earlier rate. The meetings encompassed pair programming sessions, brainstorming, clarifications, feature discussions, and much more. The team members from each party were aware of the others' progress and vice versa, which made it easier to track the progress.

Though frequent meetings were often scheduled for the first half of the project, the more the semester advanced, the harder it was to keep the perk. As different deadlines piled up, it made it harder to handle. Therefore, it would be important to keep a weekly reserved slot for such meetings in each member's schedule to ensure the continuity of the meetings.

5 - Brainstorming Sessions:

One of the things that worked well during each sprint is the brainstorming process, where we had to meet with some team members and come up with ways to either solve an issue we encountered, understand a requirement better, or come up with a structure for our products. For instance, the sprint where we relied a lot on brainstorm techniques was sprint 3 where we had a bit of misunderstanding about the accounting notions and what are the components of a "Ledger". So every now and then we would need to meet to discuss the requirement, need the correct method, class and structure needed in order to have something more or less realistic.

Although brainstorming is a good technique to create a good understanding for the subject at hand, it is also time consuming and requires a commun availability with other team members, which is a luxury in our case.

Conclusion

In general, this mini-capstone course taught us many things. It is the first time for all of us to work with a team of that size. It was not easy and smooth, at least in the beginning. Day by day, each member and each group of members started to adjust and benefit from their strengths. listening to each other's comments as well as helped a lot in utilizing the best of our efforts.

Almost everyone learned a new framework, a library or a workaround in order to have a working product. Some of us played more than one role for the sake of delivering a system that matches the requirements and the sprint schedule. The use of productivity and task management tools is a no brainer at this point. It is now evident that it is almost impossible to manage without using such tools, so learning and using them was very beneficial.

The main takeaway is knowing that building an agile team is definitely hard in the beginning and that it takes time, but we all witnessed a progress towards being in sync and on the same frequency. Another important takeaway was the experience of dealing with a product owner. Preparing and doing the demos is probably something we will be doing a lot in the real life situations when working in the industry.