POST MORTEM

ERP System

Team 5

Celia Cai (ID: 40098535)
Kelvin Chow Wan Chuen (ID: 40029677)
Daniel Gauvin (ID: 40061905)
Michael Lee (ID: 40054375)
Armando Mancino (ID:40078466)
Muhammad Shah Newaz (ID: 25067022)
Pasha Pishdad (ID: 40042599)
Piravien Suntharalingam (ID:40035136)
Julien Xu (ID: 40095332)

SOEN 390: Software Engineering Team Design Project
Concordia University
2021-04-20

Mission Complete and Lessons Learned: La Bicicletta

Introduction

The purpose of the course was to refine our technical and communication skills by working in teams to build an Enterprise Resource Planning (ERP) tool for a bicycle manufacturing plant. An ERP system is a tool used to centralize and organize the different processes of an organization in one place. For this project, this meant creating a software that tracks the entire production workflow of a bicycle manufacturing plant: from the initial sale of a bicycle, to the design and assembly of the final bicycle. In addition to this, the developed tool was also keeping track of the inventory and incoming/outgoing cash flows.

As a group, we expected that we would need to do a lot of research regarding the general details of bicycles and ERP systems as a whole. However, we were surprised about the level of complexity involved in constructing a bicycle. On the surface, a bicycle seems like a relatively simple invention. Yet, it was only after diving into the subject that we were able to fully appreciate the level of engineering and complexity that goes into bicycle manufacturing.

The code for the project used a Model-View-Controller structure written using the Laravel framework. Laravel is a full stack framework that allows front-end and back-end development. It also includes the database structure. As everything was in a single workspace, it was easier to keep track of the codebase.

The entire team is undoubtedly proud of the final product that was built. We were able to meet all of the minimum requirements and were able to make a product that functions well as a whole. Being able to see the fruits of one's labour is not only satisfying, but essential for our well being; this was an experience felt unanimously by the team.

What went wrong

1 - Laravel Challenges

For this project, we decided to use the powerful PHP framework, Laravel. One of the challenges the team encountered is the level of difficulty it took to set up due to the many prerequisites. For example, the course required to use docker, a heavy application of no less than 1GB needed to be installed and the installation of Laravel also required other software such as composer and artisan. Altogether, these software easily surpassed 2GB of storage which caused problems for our members.

Another issue that was present was that some members were using macOS and others were using Windows, making two separate installation processes. The windows process also required the use of the Linux subsystem (WSL) which none of us were familiar with previously.

Finally, the biggest challenge throughout the project was correctly using Laravel (ie. its syntax and structure). In other words, the learning curve for the framework was steep. For example, the file organization of the framework was unclear at first. We knew from its documentation that it followed an MVC pattern, however, it was difficult to figure out where the models, views and controllers were. Another example is the use of the blade syntax for views. Laravel uses this syntax as communication between the controllers and the views, which we also had to learn.

While there were extensive challenges for this framework, these challenges would also have been present if the team used another framework, such as Django or Angular. Most of these challenges were expected, and caused a slight delay in the development of the ERP system.

2 - Different Expectations Between the Product Owner and the Team

After getting feedback for Sprint 1, we realized that our team and the product owner had different expectations about certain deliverables, mainly surrounding documentations. For example, we thought we covered all requirements for the basic testing plan, however the product owner expected to see more. This most likely occurred due to miscommunication and misinterpretation during our message exchanges.

In order to avoid this situation again, we can set up meetings with the product owner to thoroughly discuss the expectations he has for our team. This way, we can ensure that the deliverables we are submitting are what he is looking for.

3 - Meeting Efficiency

Meetings were generally conducted at the beginning of a sprint to divide tasks (1.5-2.5hrs), a week prior to the deadline to catch up (1-2hrs), and on the day of the deadline to prepare the submission (4-5hrs). While there was no issue with the scheduling of the meetings, oftentimes they may have dragged on for too long. One of the challenges working remotely is finding ways to have everyone involved in discussions. Being 9 team members, it was often difficult for everyone to speak at once. As such, we only had 3-4 people talking per subject.

At any given time, we generally had 3 features being implemented in parallel. This meant that most people were tasked with helping with the implementation of a single feature, and not all 3. While team meetings are good to keep everyone in the loop, it may have been difficult to always stay engaged when the topic doesn't pertain to your work. Therefore, it sometimes felt like it was unnecessary for everyone to attend large chunks of the meeting.

For future projects, it may be better to split into 2-3 separate groups. The number of subgroups can be determined by the amount of features we are implementing at the time. Each group can have their own meetings to discuss the plan to implement the feature into the system. This should give everyone a better opportunity to voice their opinions and ideas. Furthermore, when we have a team meeting, it should be used more to bring everyone up to speed, rather than to plan. Each subgroup could even have a leader and all leaders could congregate to ensure that there is consistency throughout the system.

4 - Random Task Division

When dividing the tasks between team members, we often did it randomly. This randomness sometimes caused some members having to wait until other members finished their part in order to start their own. This created inefficiency in productivity between members as those who had to wait, often did not have anything to do prior to waiting and ended up having to rush their part as they were left without fewer days to complete their assigned tasks.

One way to solve this issue is to clearly identify the tasks that depend on other tasks. One way to do this is by plotting a network diagram which visually illustrates the tasks that can be executed in parallel and the tasks that depend on another. By doing so, we can divide the tasks in such a way that all parallel tasks are executed by a different person. If the task division was done randomly, 1 person could have been assigned to 3 parallel tasks, and the others would have had to wait for that 1 person to finish. However, by ensuring that different people are working on parallel tasks, they can be completed faster and the waiting time for those who are assigned to dependent tasks would be shorter.

5 - Github Approval Without Proper Review

When it comes down to adding your peer review before merging code, it is almost as important as writing the code itself. In some cases, it was either skipped or not properly assessed. This was caused by reasons such as carelessness or time constraints due to assignments, examinations and projects from other courses. Unfortunately, this caused delays in production and development of the code since bugs were discovered after the merge or sometimes merges had to be reversed completely because it would delete important files.

Proper verification is required to make sure the code works and meets the requirements. Furthermore, some basic testing is required and could have been done earlier on the project before merging to improve performance or to find the bugs that could have otherwise been missed. It is important since some members relied on others to start their part of a task. Just taking more time to properly review a request is the least that could have been done. Unfortunately, those measures were taken a bit too later during the project.

What went right

1 - Communication

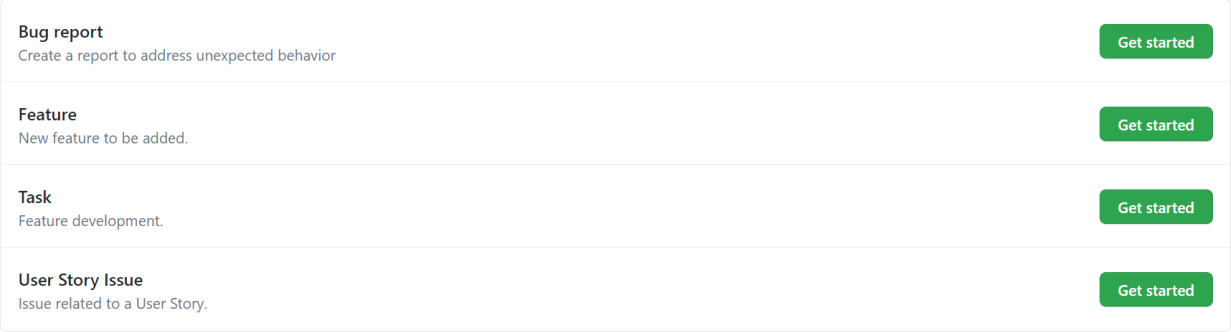
One of the strongest points that this team had was proper communication. While it is true that the meetings often went overboard, and dragged on sometimes, at the end of every meeting, every member knew what their responsibility was for either the next meeting or for the end of the sprint. Meetings were also documented in the Wiki section of the project.

Outside of the meetings, the team was active on Discord, a messaging platform. Discord allowed us to neatly organize our meeting times, technologies, documents, and many other things into different tabs and sections. This was one of the tools that allowed us to stay organized and keep everyone on the same page in the status of the project.

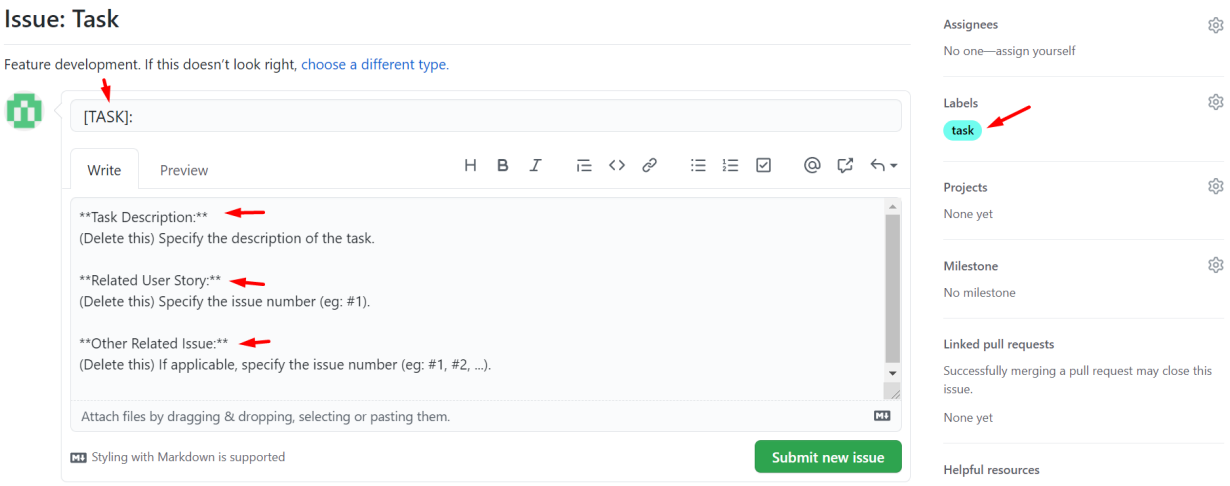
2 - GitHub Issue Templates and Conventions

From the beginning of the project, our team has set up GitHub issue templates and PR, branch and commit naming conventions. The purpose was to have an organized, legible and consistent repository.

<u>GitHub issue templates:</u>
In our repository, we created 4 templates for 4 different purposes: to report bugs, create a feature issue, a task issue and a user story issue.

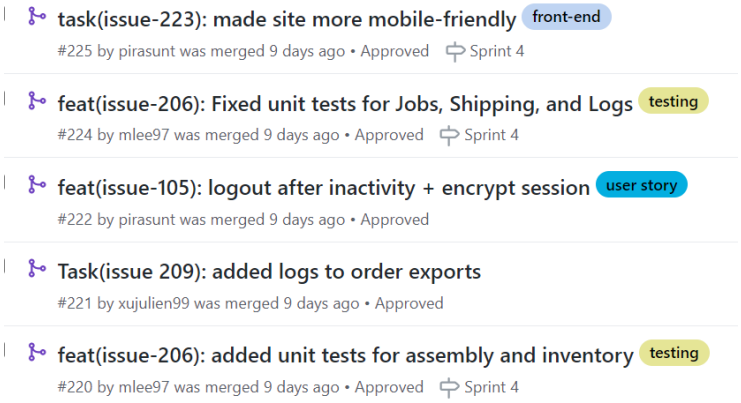


The GitHub issue templates allowed us to lay out all the necessary sections that would be useful in the description. For example, in the task issue template, there were predefined headers in the description called “Task Description”, “Related User Story” and “Other Related Issue”. This encourages members to properly describe the issues and creates uniform issue descriptions. Without them, some might just create the issues without adding enough information. It also allows us to add predefined labels that could save us a lot of time.



PR, branch and commit naming conventions:

All our conventions are documented in our GitHub Wiki under the page “GitHub Conventions”. Similarly to the issue templates, the conventions allowed a uniform looking repository. For example, as illustrated in the figure below, the pull requests look consistent.



The conventions also encouraged meaningful namings. For example, with the conventions, the team members create descriptive branch names which helps other team members easily find other member’s branches. In addition, some members have expressed that the imposed commit naming convention allowed him to easily track his commits when he needs to look at the history of commits.

3 - Proper Documentation

Reliable documentation is a crucial part when it comes to software development. Not only does it allow us to keep track of our progress, but it also allows us to keep track of its quality for every individual sprint as software developers and for stakeholders to get updates on our progression for reliable feedback. Not only is it good practice, but also a requirement from this class. All the information is easily accessible for better development, maintenance and knowledge transfer.

Multiple documents were completed and created along the course of this project regarding various aspects such as:

- Release notes
- README file
- System documentation (function and non-functional requirements)
- Identifying the target audience for the ERP system
- Agile documentation (Individuals and interactions, working software, customer collaboration, responding to change)
- QA and Testing
- Customer feedback
- Creation of style guide (github)
- Risk management plan
- Evolving documentation (prioritization of documents in development process)

Of course, improvements could have been done to the documentation during the production of the web application. Mainly concerning our research and analysis of an ERP system. We should have initially amassed more information concerning its users which led us to a rough start. A proper way would be by defining our users/targeting audience, discussing and exchanging more information with our customer from the beginning of the project to get a better grasp of the problem that needs to be solved. This would have helped us remove the confusion that came later on concerning the types of users that needed to be created for the ERP system.

4 - Managing Scope, Schedule and Cost

As future software engineers, managing scope, schedule and cost plays a key part in a successful project. The team did a good job balancing the amount of time invested in the project and the amount of work put in. While it is important to work on the project on a consistent basis, it is of equivalent importance to take a few breaks from time to time to avoid burnouts and ensure that everyone is working at their maximum capacity. The cost of the project, which is the time invested in our case, was reasonable and fair.

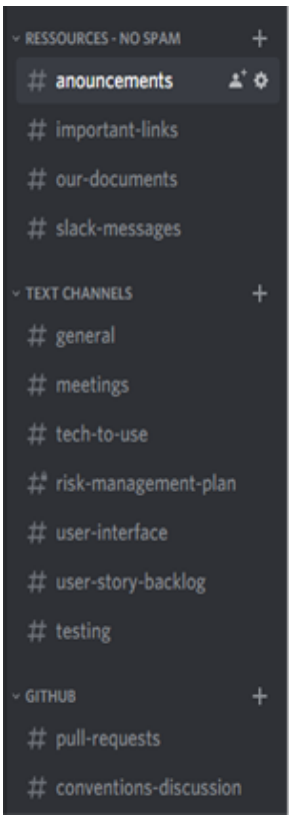
Our team also understood each other when it came to our schedules. Being all part of the same program, our due dates for assignments and exams usually coincided which made it easier to know when it was the right time to focus on the project. Some people had to go to their jobs or had to go to class so we shared our availability at the start of the project which definitely made our planning and scheduling easier.

When it comes to scope, our team never had any plans to go beyond the product owner’s expectations so we decided to settle for the minimal requirements of an ERP system. Some features that weren’t specified in the requirements were cut in order to focus on delivering a consistent, reliable and efficient product.

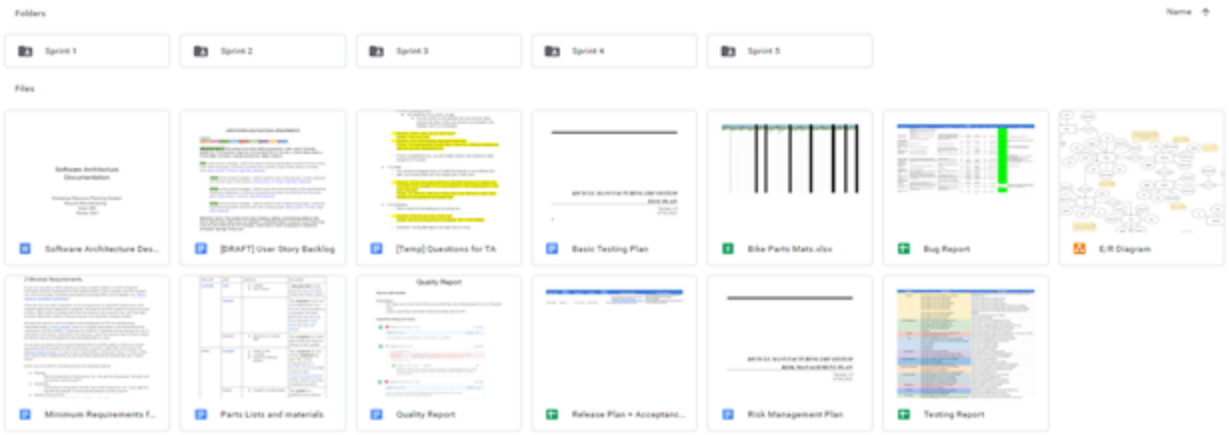
5 - Organization

Organization is one of the most important aspects to having a software project running smoothly and according to plan. This is due to the fact that it serves to help prevent future problems since information is more known on where to be found when needed. Additionally, with a project of this magnitude, there are so many small requirements that were necessary to be met and so the organization played a key role in that. Our team was extremely efficient in the sense that everything was organized perfectly.

A unique aspect of our organization was the discord server that was operated for communication purposes. It is planned in such a way that it also stored valuable information that was very useful at all times. Additionally, the resources channels which had the most important information were regulated to prevent too many messages from flooding the valuable information to be seen, while the text channels were meant for higher volume communication. A breakdown of the organization of channels is shown below.



Announcements: The announcements section was used to send out very important information such as important sprint details.
Important links: The important links channel was used to send out important links with important information. For example, it could be a link to articles or videos about Laravel that we found useful.
Our-documents: Used to provide links to our documents, such as our personal group google drive.
Slack-messages: Used to post screenshots of slack messages from our TA and professors in order to ensure that everyone in the team is aware of them.
General: A discussion regarding almost everything. It was used to discuss any other topic that did not fit into any other text channels.
Meetings: When there was a meeting being organized, we discussed it in this channel. We also ran polls to get input from every member.
Tech-to-use: Includes early discussion regarding technologies to use for this project. It also has important information regarding Laravel and other helpful information regarding docker, etc.
Pull-requests: Any pull requests that went through on github were also linked via the discord with details regarding the pull request. Here, discussions can be done regarding whether there are issues to the pull request.
Conventions-discussion: Used to discuss any errors or changes to our conventions.



Moreover, our google drive was also very organized, containing everything we needed in one place. Since the project contained numerous evolving documents that were constantly being updated, the originals were in the main folder, with sprint specific ones being placed in their respective sprint folder as shown below. Lastly, recordings of end of sprint meetings were placed here for reflection if needed.

Finally, another helpful aspect of the organization was the meetings being documented. Within the GitHub wiki, each meeting was summarized with their important details which was helpful throughout the sprints to revisit what was covered, and to get the breakdown of tasks again.

Conclusion

One of the things that were challenging was using Laravel which was due to the more complicated set up mainly because of different operating systems used by our team and high storage needed for installation. Another challenge we had as a team was the difference with what we perceived as required components for our software and what product owners expected from us. Another problem we had which was mainly caused by doing the meetings remotely was going over the expected time for meetings. Also, since we were 9 people and we divided the tasks, some parts of the meetings were not related to everyone's tasks. Another issue that came up was that sometimes we had to wait for other team members to finish their tasks as we could not make tasks completely independent.

One of the main strengths of our team was our communication and amount of meetings that allowed us to perform our project. We also made templates at the beginning for creating tasks, issues, user stories and reporting bugs which made everything easier and more consistent. We made proper documentation as we progressed for each sprint which made maintenance and development faster and more consistent. As for our scheduling, we made our available times from the first sprint so we easily managed to set our meetings.

What we learned from doing this project was having more meetings with the product owner to clarify everything from the beginning. Another thing is that we should create separate meetings too between our team members so those who are working on a separate task can have their group meetings so we can progress faster. Also we should assign tasks in a way that can be done in parallel. And the last thing is to pay more attention to approving pull requests and check them completely so we end up with less bugs.