**Summary and reflections**

This chapter will introduce how the project was managed by specific tools and methodologies in Section 1. Reflective remarks will also be demonstrated in Section 2 of this chapter.

**1 Project management**

This section introduces three main supporting tools for managing the project and how the team distributes jobs and utilises agile methodology.

**1.1 Supporting Tools**

The team utilised some valuable tools to manage the team and project. Section 1.1.1 will introduce the usage of Git. In section 1.12, a documentation tool is explained. Section 1.1.3 shows how the team is managed with the help of GitHub.

**1.1.1 Version Control – Git**

Git is a version control tool that can trace any code changes. Each commit will record all the code status at a specific time node. The utilisation of Git allows the team to easily spot positions of bugs and notice what has affected the software. With commit messages, it also makes communication more efficient. Git provides a convenient feature, branch as well. Branch simplifies team collaboration. Each time a member needs to develop a new feature, a new branch would help ensure the new feature will not affect the main software.

**1.1.2 Documentation – Style Guidist**

The team takes JavaScript as the programming language. However, we noticed that JavaDoc is a powerful tool for demonstrating the usage of code with proper documentation. For further maintenance, the team uses Style Guidist as the documentation tool. With an extra Markdown file in each folder, Style Guidist will automatically generate a JavaScriptDoc in a web. It would help the future maintenance team to understand and help anyone who is interested in our design.

**1.1.3 Teamwork Organisation and Remote Repository – GitHub**

GitHub works not only as a remote repository but also as a teamwork organisation tool. GitHub stores branches and records on the cloud as a remote repository to share the code among all the members. CI is also applied by a feature called Action on GitHub. Action will automatically run all the tests and build the project when GitHub receives a push operation to automate testing and prevent potential long-term errors. GitHub also provides issues, milestones and Kanban. The team assigns tasks and raises questions and bugs by issues. Kanban will automatically take issues and assign them into TODO, In Progress and Done columns. Milestone will display each sprint’s tasks and encourage members to complete tasks soon. These three features help a lot in task assignment and time management. All the members could have a clear view of the current state.

**1.2 Task and Responsibility Distribution**

Design and develop software from scratch not only need the ability of programming. The skills of UI design, structure design and testing are also required. The team explored each member’s unique skills and strength, which allows the team to take advantages of every member. E.g., people who are creative design the software structure first. Experienced UI designers are responsible for designing UI. Members who are familiar with programming will implement designed features. After the design was confirmed, the team of designers will also join the quality assurance team. Clear responsibilities make members focused, and the whole team become efficient. Labor division is as follows.

Team Leader: Shiliang Chen

UI Design Team: Yiming Tang, Yani Huang

Technical Team: Shiliang Chen, Yijie Lu, Yuting Jiang, Ruizi Han

Quality Assurance Team: Yiming Tang, Yani Huang

Report Editor: Yuting Jiang, Yijie Lu

Meetings are conducted twice a week, formally and informally. Chairperson and secretary are changed in turn. Tasks are distributed after the meeting, and minutes are in appendix X.

**1.3 Agile Methodology with Requirements Documentation**

The team decided to apply the agile methodology for the development of software. However, in the beginning, the team has made detailed requirements documentation to clarify all the basic requirements from stakeholders. The requirements documentation also functions as a checklist with our supervisor.

During the development, the team wrote user stories to specify features that needed to be built. With those user stories, the team discussed features to achieve in each week’s informal meeting and develop the features during the week’s sprint. In the formal meeting of the week, we confirmed that week’s achievements with the supervisor. In this case, the level of customer engagement was high, and any unsatisfied things could be fixed very soon.

The team also utilises stand-up, which is a short meeting of about 5 minutes held every day except those days with meetings. Stand-up allows everyone to report daily progress and problems encountered, which helps the team clear tasks in time and handle issues and problems in time.

Peer programming was applied strictly during the spring festival. Each task was assigned to two of the technical team members. One would be responsible for the coding, and the other will take care of monitoring and checking whether there was anything wrong with commenting, naming, etc. Peer programming improves code quality considerably and increases efficiency by exchanging ideas between peers.

**2 Reflection**

The team learned experience from the project. This section will share some of the reflective remarks from this project and future expectations on it.

**2.1 Requirements Engineering**

In order to build software from scratch, requirements engineering is the most fundamental thing but also most important. The team finds requirements engineering is a relatively prosperous part of the project. Team 10 learned how to perform a survey, focus group and interview through this process. Survey gave us an overall user preference for the software, while the focus group allows users to share more specific ideas about the software. Survey and focus group helped the team decide the software's platform and style and come up with practical and innovational features. The Focus group even aspired us to design a tutorial section for newcomers to programming. The interview collected opinions from a lecturer, and the interviewee helped us spread the survey widely to conclude more general ideas.

As for the weakness in requirements engineering, the questionnaire contains some questions which may not be that useful in the software design, such as "Through what you learn algorithms?" Since 207 students have involved in the survey, more focused questions may help more.

**2.2 Technical Research**

Technical research was not conducted smoothly. It was assigned to be a week's research on programming languages and tools we can use. However, because the task was made vaguely, i.e., everyone goes to do technical research, it turned out that no one had done anything useful related to technical research. After that, the team discussed aspects of technical research and distributed work to different people. Technical research was done and enough for the interim report at the end. However, it was far from enough to help us decide the programming language at that time. The team has worked on complete the part as asked to do it, but not in order to actually choose some technical things properly. This was a big mistake and caused more waste of time before actual programming since all of us found it hard to do coding with the tools decided in the interim report.

As a reflective remark, we would do comprehensive and focused technical research to choose the most suitable programming language and tools for a project.

**2.3 Software Engineering Methodology**

It is not easy for team members who are not experienced in software engineering to decide which software engineering method to use, traditional or agile. Firstly, we reviewed the related lectures of the Software Engineering module and listed both methods' advantages and disadvantages. Also, we referred to the advice of the supervisor. Finally, we decided to combine two methods to develop this software. The traditional development method was used to detailly record the requirement documents, while the agile development method was used in the design and development stages. Agile helped improve customer engagement. We were able to contact and confirm any details with our supervisor on time. According to the supervisor's advice and suggestions after each sprint, the software was accumulatively being more robust and equipped with more features.

The software we built is components based and is relatively suitable for the agile method. We could always see the software's progress, which made the team more energetic and willing to develop. As the method was agile, even requirements change could be handled well.

**2.4 Time Management Issues**

**Meeting Time Management**

The supervisor emphasised that each formal meeting must be controlled within 30 minutes. However, sometimes we may have lots of content expected to discuss with the supervisor. That sometimes led to the meeting for longer than 30 minutes. To solve this problem, one member pays attention to the time in each formal meeting. The member will remind everyone if time is exceeded. Further, we carefully choose more important content to discuss in formal meetings.

**Late of Starting Coding**

At the middle stage, our overall progress seemed a bit slow compared to other groups. The reason may be that we need to do more work to collect user requirements. Some members have raised concerns about whether we could finish the project on time. After consulting the module convenor, we were told to focus on ourselves because it is meaningless to compare with different groups. We should trust our time plan and follow it.

Because we did technical research again after interim report for finding a more suitable programming language, coding was postponed. The team also spent another three weeks learning React and Electron. It turns out that software development finished later than planned, but the overall time plan was reasonable. More importantly, we modified the plan several times to adapt to any updates. This made the process more flexible and agile.

**2.5 Team Management Issues**

**Late of Finishing Tasks**

Some of the team members could not finish their tasks on time. This may potentially affect the progress of the whole team because tasks have dependencies. The first attempt is to separate tasks to reduce coupling and assign individual tasks to those who may take more time. This attempt was no ultimate solution and still caused some issues since there could still be one task left which the whole team was waiting for. The second attempt was trying to treat the root. The team leader had conversations on the issue and tried to find out the reason behind it. This attempt worked better and encouraged the team member to take responsibility. Hence, a team could not lack communication. Spotting issues and addressing them immediately is vital for further work.

**Meeting and Task assignment**

As we applied agile methodology, a stand-up meeting was conducted every day except a break day of week, informal and formal meetings. Daily stand-up pushed us to keep pace with the overall progress and finish tasks on time. This also allows every member to have a global view of the project. During the Spring Festival, stand-up functions well in keeping everyone learning React and related knowledge. As for task assignment, different members are responsible for different work in this project. However, it is difficult for members with cooperation to keep track of the progress of each other. To solve that problem, we decided to use Kanban in GitHub. At the same time, we assign issues in GitHub to ensure that each member knows about tasks.

**Disagreement**

A severe disagreement occurred during the prototype design stage. Different members had different opinions about the process of learning sorting algorithms. This kind of situation is not a surprise. We have predicted it in the risk management section. To deal with this problem, all members must keep calm. Then, we decide to finish the sequence diagram together, and the disagreement above solved finally.

**2.6 Technical Issues**

**Way of Peer Programming**

We learned that peer programming had several merits. However, during the break, it was hard to program together. One team member came up with an idea to use an editor sharing tool to perform peer programming. Because Visual Studio Code has a large base of extensions, we successfully found an extension called Live Share. Through Live Share, we were able to edit on the same workplace and even share the terminal. Since the project is web-based, a browser sharing tool was also used to view real-time changes. By performing peer programming, the code quality was to some extent improved. Typos and minor errors like lack of semi-colon were avoided.

**Testing**

The team realised the importance of testing and spent a week learning how to perform testing on React JavaScript project. At the early stage, TDD was not taken seriously by some of the team members. One group of two in the team did not follow the instruction of TDD and wrote code directly without writing any unit test. The component displays appropriately at first, but coding is painstaking as no clear plan was made—the group of two modified their design multiple times. After TDD was stressed to be vital, the group added unit tests for the component but found a title in it was wrong. Compared to human eyes and testing manually, automated unit testing helps design the code and prevents potential mistakes by checking components each time they are modified.

**2.7 Lexical Issues**

The supervisor pointed out that there were several grammar errors in the documentation files. Because all team members of team 10 are Chinese, there indeed are difficulties in writing formal reports in English. To deal with this problem, we decided to double-check all documentation and pay more attention to vocabulary usage and grammar errors when writing reports.

**2.8 Future Work**

**Language Support**

Team 10 cares about people who speak different languages. According to the survey, most participants claim that they need a Chinese version to assist them in understanding the content. Sections such as correctness would be somehow difficult to understand without explanation in their native language. We have considered it as one of the additional requirements. However, translation needs to be conducted with professional assistant and references. It is hard to find such help and reference, so we have not done the Chinese version at the end. We hope to develop a Chinese iCanSort to help more students who are struggling with learning sorting algorithms.

**Accessibility Support**

Team 10 noticed that web contents have a special attribute called aria with which browsers can read out content to help people who cannot see the content. For future work, we would like to develop an idea to help people with disability in visions.

**Multiple Platform Support**

As a web-based project, it is highly possible to be deployed in a web server and allow people to use this web app by accessing a website. Also, it could be compatible with smartphones, which needs further work.

**3 Risk Management**

