1 Introduction

Agile development is an efficient development method. Through agile development, the team can deliver high-quality software faster and better meet customer needs. It advocates rapid feedback and flexible adjustment so that the team can adapt to changing needs and market conditions in a timely manner. Agile Development also encourages close cooperation and communication between development teams and stakeholders to ensure common understanding and consensus. In this report, we will show the project development process of this group using agile methods. The name of this project is: Learning Journey System.

We used agile methods to participate in the whole development process. In the process of developing the system, we made full use of the skills we learned in the software engineering course, worked together as a development team, tried our best to complete user requirements, and completed the development of Student Learning Journey system. This article will introduce in detail the project management, requirements, analysis design and implementation testing in our development process to describe our development process in detail.

1.2 Scope and purpose of our system

The users of our system are the students of the BUPT & QMUL partnership. We have designed this system to make it easier for students to view and manage their academic information. Students can view and amend their personal information, add and remove information about their skills, projects and honours during their studies, and view their exam results. We have also designed a sentimental feature that will allow them to view their journey through campus and look back at their journey.

1.3 Roles and Responsibilities:

In agile development, everyone is involved in all aspects of the project. Therefore, everyone's role is uncertain in our team, but each member’s responsibility still has a focus. Zehao Xing is the team leader, he is responsible for overseeing the prototype design, making decisions, and organizing and facilitating meetings to drive the progress of the project. He also did a lot in the control and entity classes. Yitong Hu is responsible for the primary design, and implementation of the learning journey system modules, and the definition of the entity class. Shuo Xiu is mainly responsible for the structural design and implementation of the user interface. Haoxian Ye focused on market research, defining the project requirements and writing stories. Yichang Zhang focused on testing and bug fixing. Huanyu Chao is responsible for determining the development sequence and breaking down tasks for our team.

The above is just a brief introduction to the team members. Please refer to the weekly report for specific responsibilities.

2.3时间估计和计划：

我们在项目的一开始便规划了完整的计划，包括明确的目标和各种迭代更改的假设，并且在该时间管理计划的指导下成功准时地完成了任务。我们采用了Trello应用来规划并记录我们的工作进程，而不是传统的表格与文件。Trello能够显示我们每次会议的记录与下次的计划、已完成的任务、要做的议程与悬而未决的任务。我们实现了对项目进程灵活的调整与高效的推进，及时对各种预想中与意料外的困难进行攻克。

2.4决策制定：

我们采用先进的敏捷开发方式，团队队员参与全过程的开发，又各有注重负责的部分。我们常常会在组长的定期组织下在研讨间或宿舍开展即时便捷的讨论，共同对代码等发表意见；我们也可以把自己的意见和观点记录在我们Trello的留言板上，方便成员随时随地查看。

2.3 Time Estimation and Planning:

At the beginning of the project, we developed a comprehensive plan that included clear objectives and assumptions for various iteration changes. Under the guidance of this time management plan, we successfully completed tasks on schedule. Instead of traditional spreadsheets and files, we utilized the Trello application to plan and track our work progress. Trello enables us to visualize records from each meeting, plan for the next steps, track completed tasks, and manage pending and unresolved issues. This approach allowed us to adapt and progress flexibly with the project's timeline while efficiently addressing anticipated and unforeseen challenges in a timely manner.

2.4 Decision Making:

We adopt an advanced agile development approach where team members are involved in the entire development process while having their respective areas of responsibility. We frequently engage in instant and convenient discussions in meeting rooms or dormitories, organized regularly by the team leader, to provide input on code and other matters. Additionally, we can document our opinions and perspectives on our Trello's message board, allowing members to access them anytime and anywhere for reference.

2.5 decision making

Due to our agile development approach, our team makes decisions democratically. Design pattern issues don't need to be reported to the scrum master and can be addressed by the developer directly. Any design problems will be identified during later testing and implementation. If there's an issue with a new function, it should be brought to the attention of the scrum master and discussed during the weekly meeting. In summary, team members have autonomy with some limitations.

2.6risk handling

In this project, we may encounter risks related to both the project and the product. To address project risks, we have adopted a strategy of allowing ample time for team members. By allocating sufficient time, we ensure that if any individual is unable to complete their tasks on schedule, others can step in and accomplish them within the given timeframe. As a result, our project schedule differs slightly from the recommended one.

Regarding product risks, our approach involves multiple iterations and evaluations. At the conclusion of each iteration, we engage in discussions within the Feishu platform to identify the shortcomings of that particular phase. We thoroughly evaluate our design and implementation during these discussions. Any issues discovered are then addressed and improved upon in subsequent iterations. In the event of a significant problem emerging during an iteration, we refrain from altering the plan midway but instead prioritize rectifying it in the next iteration.

**3. Requirements**

**3.1** **requirements Finding Techniques**

**Questionnaire:** The first approach we discussed for identifying the requirements is questionnaire (which is the part of Fact-finding Techniques). We spend an organizational meeting to discuss, decide the questions together and publish the questionnaire. Then we obtain the feedback as our requirements decision background information.

**Observation:** As the students of BUPT, we start from our study lives experience and use some products, which is about the study or the information of student and get some potential requirements.

**Brainstorm:** after collecting enough information of requirements, we screened and summarized series requirement from the brainstorming.

**3.2** **User stories**

As a convenient workflow tool, we combined Flybook to complete our preliminary epics through taskbar and market research.

As the figure show, before the user stories, we collect the requirements in a mind map, which including our epics of the project and the first version of the priority by the DSDM-MoSCoW and the estimating the point for each epics. The mind map is not static, we constantly update it with the iteration change.

We first classify the epics as six parts as the mind map show, which are different modules, and such operation makes it intuitive and flexible.

Then we break down the epics to the relative stories and estimate the story point (Fibonacci Sequence), in fact we generate more stories than the product backlog show since we delete and modify lots of stories according to the discussion in the regular meeting.

After the stories determined primarily, we estimate and prioritize the passed stories refer to the mind map.

As an Agile Software Development project:

In the first iteration meeting, we discussed the initial stories that needed to be implemented according to the product backlog, broke them down into small tasks for quick and efficient implementation, and allocated them rationally by team members.

In the subsequent regular meeting, before completing the process of the first meeting, we would first share with each other the difficulties and solutions encountered in the development process of this iteration, as well as some completed functions and some new ideas generated by each iteration and make technical adjustments for our next round of development based on this.

Together, we evaluate an existing implementation of the story and gain reusable experience from it, and we evaluate the priority of the new story in the iteration and make a preliminary estimate for the story point and so on.

As our iterations continue to complete, our decomposition of an epics becomes more thorough, and our estimation of a story becomes more accurate.

**3.3** **Prototypes**

In our group, Mr. Xing Zehao has relatively good aesthetic skills and also has some experience in prototyping in previous projects. As the team leader of the prototyping project, he led the group in the design of the software prototype. During the software development process, the requirements of the software and the requirements of the developers change from time to time, so the prototype design was modified or reworked several times during the development process.

We only made three generations of prototypes, of which generations 1-2 were less accurate models, and it was only in the third generation that we made relatively beautiful models that met the aesthetic needs of the users.

The first generation was roughly drawn with pencil and paper and from the second generation we started prototyping using the prototyping program Figma. This is a lightweight UI design software that the team used to collaborate and prototype the various features of the software. The real-time updates of the prototypes in the program allowed us to communicate in real time during the prototype education and to suggest changes more quickly. Here we have only uploaded some key prototypes, which are integrated into the Figma web link at the end of the article.

These prototypes were of great importance to us for the subsequent development of the website. With these prototypes we were able to clearly understand how the software should be designed and the logical layout of the prototype images helped us to easily understand the logical structure of the user interface. The logical layout of the software also allowed us to design software that was better laid out and looked better.

**3.4** **iterations planning and time planning**

Refer to the suggested timeline, we generate the planning table:

这里可以讨论一下要填的事项和时间点，可修改

|  |  |  |  |
| --- | --- | --- | --- |
| **time** | **theme** | **tasks** | **outcomes** |
| 13-15 March | Meet group members, appoint a group leader and discuss the  project handout |  |  |
| 16-24 March | Gather real requirements. Story writing workshop. Outcomes: product backlog and prototype |  |  |
| 27-31 March | Iteration 1. Outcomes: Working Software v1 |  |  |
| 3-4 April | Iteration 2. Outcomes: Working Software v2 |  |  |
| 17-18 April | Iteration 3. Outcomes: Working Software v3 |  |  |
| 1-12 May | Iteration 3. Outcomes: Working Software v3 |  |  |
| 15-26 May | Software final delivery |  |  |

In the preparation phase before the iteration, we determined the questionnaire through regular meetings, made relevant requirements observations, obtained corresponding data, conducted brainstorming, and prepared the product backlog, glossary, and prototype.

During the iteration, we design and implement the software and improve the backlog and other related content written during the preparation phase.

Initial iterations are code that implements high-priority features easily and quickly, and subsequent iterations are refinements and modifications of features based on previous iterations and experience, as well as attempts at additional features. Each iteration includes relevant tests.

**4.3 Design Principles**

**Single Responsibility Principle (SRP):** To conform with SRP, we tried to distinguish classes by modules and services from the beginning of the design. Each class corresponds to only one service and only modifies the entity class associated with the service. For example, the "InfoApp" class is used to display students' information, the "ProjApp" class is used to display the projects that students participate in, etc.

**Open-Closed Principle (OCP):** Our software modules (classes, methods, etc.) are “open for extension” and “closed for modification”. For example, we have an “ExperienceApp” superclass, so that we can make other subclasses behave in new and different ways as requirements change or to meet new needs, such as the different functions between the “GradeApp” class and the “HonorApp” class, and it does not require changing the code of the module.

**The Liskov Substitution Principle (LSP):** To adhere to the LSP, our overriding methods in subclasses have the same signature (input parameters and return type) as the methods in the superclass, and we conform to "Inheritance should only weaken preconditions and strengthen postconditions.", also the exceptions thrown by the overridden methods in subclasses are the same as or subclasses of the exceptions thrown by the methods in the superclass.

For other design principles, such as the Interface-Segregation Principle (ISP) and the Dependency-Inversion Principle (DIP), most classes conform to these principles, but a small number of classes still do not.

5.1 Execution Strategy:

To effectively execute our testing strategy, we will follow the following steps:

1. Requirement Analysis: Thoroughly study the requirement documentation to ensure a clear understanding of the project's functionalities and non-functional requirements.

2. Test Case Design: Develop detailed test cases based on the requirements, including input data, expected outputs, and desired behavior.

3. Test Code Development: Write unit tests and integration tests to validate the code.

4. Test Automation: Utilize automation testing tools to execute and manage test cases, generate test reports, and logs.

5. Regular Test Execution: Perform regular test suite execution after code modifications, during the integration build process, and prior to release.

6. Defect Tracking and Resolution: Promptly record and track any defects discovered during testing, collaborating with the development team to address and resolve issues.

7. Continuous Improvement: Continuously improve and optimize the testing process based on test results and feedback received.

By employing a combination of unit testing, integration testing, and regression testing, we can effectively verify the correctness, stability, and reliability of the code.

5.2 Iteration Plan:

Following the Scrum development methodology, our code iteration plan is as follows:

Our minimum implementation unit is a task. In each iteration, we encourage team members to synchronize their code with Git after completing 2-3 tasks.

We adopt a flexible implementation approach as requirements are constantly changing. Therefore, prior to the project's commencement, we have a brief plan outlining important milestones, which is refined over time.

Our iteration plan organizes tasks based on the team's available time and the priority of requirements. Within each iteration cycle, we focus on the following aspects:

1. Implementation of Core Functionality: We prioritize the completion of the project's core functionalities, ensuring the development of the basic framework and essential modules.

2. Bug Fixes: We address known bugs to maintain code stability and ensure correct functionality.

3. User Interface Improvements: Based on user feedback and design requirements, we enhance the layout and style of the user interface.

Each iteration cycle typically spans 1 week, although the length may vary depending on project complexity and other factors.

At the end of each iteration cycle, team members synchronize their code with Git and conduct relevant testing and evaluations to ensure code quality and functional integrity. We remain flexible in adjusting and improving the iteration plan based on project progress and evolving requirements, aiming to better align with project goals and meet customer needs.

5.3 Testing Strategy:

In order to make our system work better, we combined several different testing methods to write the system: firstly we used the core TDD testing method to determine the user requirements and then we wrote some core functional test classes before writing the code to pass the requirements. Next we developed a series of software testing rules. Yichang Zhang was the engineer who led our group in testing and managed a range of testing elements. Under his leadership, we defined a series of testing guidelines:

1. User requirements-based testing: After implementing the software implementation, we need to test whether the functionality meets the user requirements

2. Defect-based testing: We need to test for any faults, errors, and defects that will be detected, and we need to be aware of these when testing.

3. Policies: depending on the structure of our project: any method that involves reading and writing to JSON files should be white-box tested. Any method that involves jumping should be tested.

In order to make our system work better, we combined several different testing methods to write the system: firstly we used the core TDD testing method to determine the user requirements and then we wrote some core functional test classes before writing the code to pass the requirements. Next we developed a series of software testing rules. Yichang Zhang was the engineer who led our group in testing and managed a range of testing elements. Under his leadership, we defined a series of testing guidelines:

1. Requirements testing: After implementing the software implementation, we need to test whether the functionality meets the user requirements

2. Defect testing: We need to test for any faults, errors, and defects that will be detected, and we need to be aware of these when testing.

3. Policies: depending on the structure of our project: any method that involves reading and writing to JSON files should be white-box tested. Any method that involves jumping should be tested.

White Box Testing:

According to our test policies, we encapsulated all the methods involved in reading json files into the DB file, so our white box testing was mainly for the methods in the DB. We then built a test harness before development based on the required operational functionality and subsequently modified the tests as requirements changed, ensuring that every possibility would be tested to ensure robustness

Black Box Testing

We mainly use black box testing to do user request-based testing and integration testing after the functionality has been implemented. For example, for functions involving jumps, we will focus more on their actual performance. For example, for the login function, we will test the performance of the program when an incorrect password is entered, and for the project addition function, we will test whether the system can determine and reject the addition when certain fields are not entered. Through this comprehensive black box testing, we can ensure that the application meets the needs of the user and that it works well.

5.4 Application of TDD:

We followed the following steps:

1. Write Test Case: Based on our requirements and functional specifications, we wrote a simple and specific test case to validate a particular aspect of the code.

2. Run Test Case: Upon running the test case for the first time, we confirmed that the test failed. This is because we haven't written the corresponding functionality code yet.

3. Write Functionality Code: We wrote sufficient code to make the test case pass. The focus was on fulfilling the requirements of the test case.

4. Run Test Case: We reran the test case to confirm that it passed. If the test did not pass, we continued improving the functionality code.

5. Refactor Code: We refactored the functionality code to enhance code quality, readability, and maintainability while ensuring that the test cases still passed.

6. Repeat the Steps: We repeated the above steps continuously, writing test cases for each new feature or requirement, writing code to make them pass, and then refactoring the code.

By following the Test-Driven Development (TDD) approach, we ensured that our code was developed incrementally and validated against specific test cases. This helped us maintain code quality, catch defects early, and improve the overall design and structure of our code.