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.

Using this system, students of QMUL&BUPT cooperation projects can easily query and modify personal information, score inquiry, student awards, projects, ability to view and modify, and even view their student journey and automatically generate a resume.

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 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。

2.5 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.

5.3测试策略

在代码测试策略中，我选择了单元测试、集成测试和回归测试这三种测试类型。

1. 单元测试：

单元测试是对代码中最小的可测试单元进行测试，以验证其功能和逻辑的正确性。对于每个函数或方法，我们编写了多个测试用例，包括正常情况下的输入和边界条件下的输入，以覆盖各种情况。我们使用了测试框架（如JUnit）自动运行这些测试用例，并生成测试报告。单元测试的目标是尽早发现和修复代码中的逻辑错误。

2. 集成测试：

集成测试是测试多个组件或模块之间的交互。我们将通过创建集成测试套件来测试这些组件的协同工作，并确保它们能够正确地交互和传递数据。我们可以使用模拟对象或真实的依赖项来模拟组件之间的通信。集成测试的目标是确保系统的各个部分能够无缝地协同工作，完成所需的功能。

3. 回归测试：

回归测试是在代码修改后重新运行之前通过的测试，以确保新的更改没有破坏现有的功能。我们构建了一个完整的测试套件，包括单元测试和集成测试，并使用自动化测试工具来运行这些测试。在每次代码修改后，我们会运行回归测试套件，以验证修改的代码是否引入了新的错误或破坏了现有功能。回归测试的目标是确保系统在不断演化的过程中依然保持稳定和可靠。

5.1执行策略

为了有效执行测试策略，我们将遵循以下步骤：

1. 分析需求：仔细研究需求文档，确保理解项目的功能和非功能要求。

2. 设计测试用例：根据需求编写详细的测试用例，包括输入数据、预期输出和预期行为。

3. 编写测试代码：编写单元测试和集成测试代码。

4. 自动化测试：使用自动化测试工具运行和管理测试用例，生成测试报告和日志。

5. 定期执行测试：在代码修改后、集成构建过程中和发布前定期执行测试套件。

6. 跟踪和解决缺陷：如果测试中发现缺陷，及时记录并跟踪缺陷，并与开发团队合作解决问题。

7. 持续改进：根据测试结果和反馈进行测试流程的改进和优化。

通过综合运用单元测试、集成测试和回归测试，我们可以有效地验证代码的正确性、稳定性和可靠性

5.2迭代计划

根据Scrum开发方法，我们的代码迭代计划如下：

我们的最小实现单元是任务。在每个迭代中，我们鼓励团队成员在完成2-3个任务后将代码同步到Git。

我们采用灵活的实现策略，而需求是不断变化的，因此，在整个项目开始之前，我们只对重要的里程碑做了一个简短的计划，并随着时间的推移进行改进。

我们的迭代计划按照团队的可用时间和需求的优先级来安排任务。在每个迭代周期内，我们将关注以下方面：

1. 核心功能的实现：我们将着重完成项目的核心功能，确保其基本框架和关键模块的开发。

2. Bug修复：我们会修复已知的Bug，以确保代码的稳定性和功能的正确性。

3. 用户界面的改进：我们会根据用户反馈和设计要求，对用户界面进行布局和样式的改进。

每个迭代周期通常为1周，但根据项目的复杂性和实际情况，迭代周期的长度可能有所不同。

在每个迭代周期结束时，团队成员将同步他们的代码到Git，并进行相应的测试和评估，以确保代码质量和功能的完整性。根据项目的进展和需求的变化，我们将灵活地调整和改进迭代计划，以便更好地满足项目的目标和客户的需求。

5.4 TDD的应用

我们遵循了以下步骤

编写测试用例：根据我们的需求和功能规格，我们编写了一个简单且具体的测试用例，用于验证代码的某个方面。

运行测试用例：初次运行测试用例，确认测试失败。这是因为还没有编写与测试用例相对应的功能代码。

编写功能代码：我们为测试用例编写足够的代码，使得测试用例能够通过。

运行测试用例：再次运行测试用例，确认测试通过。如果测试未通过，继续改进功能代码。

重构代码：对功能代码进行重构，以提高代码质量、可读性和可维护性，同时确保测试用例仍然通过。

重复上述步骤：持续重复上述步骤，为每个新功能或需求编写测试用例，然后编写代码使其通过，并进行重构。

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.

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 our code testing strategy, I have selected three types of tests: unit testing, integration testing, and regression testing.

1. Unit Testing:

Unit testing involves testing the smallest testable units of code to validate their functionality and logic. For each function or method, we have written multiple test cases, including inputs for normal cases and boundary conditions to cover various scenarios. We use a testing framework, such as JUnit, to automate the execution of these test cases and generate test reports. The goal of unit testing is to identify and fix logical errors in the code as early as possible.

2. Integration Testing:

Integration testing focuses on testing the interactions between multiple components or modules. We create integration test suites to test the collaboration of these components and ensure they interact and pass data correctly. We can use mock objects or real dependencies to simulate communication between components. The objective of integration testing is to ensure seamless coordination among different parts of the system to achieve the desired functionality.

3. Regression Testing:

Regression testing involves rerunning previously passed tests after code modifications to ensure that new changes have not broken existing functionality. We build a comprehensive test suite that includes both unit tests and integration tests, and use automation testing tools to execute these tests. After each code modification, we run the regression test suite to verify if the modified code has introduced any new errors or broken existing functionality. The goal of regression testing is to ensure the system remains stable and reliable as it evolves.

By incorporating unit testing, integration testing, and regression testing, we aim to validate the correctness, stability, and reliability of the code throughout its development lifecycle.

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.

**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.