

Software Development Practices

PBL Final Report

Team Teletubbies

Kang Yewon Division of Software 2019054693

Kim Yejin Division of Software 2019069034

Jun Minji Division of Software 2019025823

Jeong Yeonju Division of Software 2019014739

Han Subin Division of Software 2019071994

Index

I . Problem Definition

II. Software Development Research

III. Software Development Best Practice Proposal

IV. Software Development Best Practice Verification
Results and Knowledge Acquisition

V. Conclusion

VI. Git Address & Jenkins Address

I . Problem Definition

SETB's team-level software development is not systematic and inefficient, requiring a lot of time and human effort to verify and ensure the quality of customized products. Therefore, new and efficient software product development best practices should be established.

II. Software Development Research

Software development refers to a series of processes, such as requirements, design, implementation, testing, and maintenance.

Applying process models in simple projects carried out by individuals can become grandiose. However, for complex projects under way by many, applying process models makes it easier to understand development progress and review projects. Furthermore, while simple things can be implemented by developers alone, maintenance, expansion, and modification requires others. In other words, cooperation and collaboration are necessary when developing software. The advantage of solid team collaboration is clear. When a problem occurs with one person, it is easy for another person to solve the problem. As changes in customer needs and requirements accelerate, software also needs to be developed and repaired. Therefore, collaboration is important.

1. Software Development Process – Software Development Life Cycle

1) Definition

Software development life cycle refers to a model that defines the entire process from software development to disposal as a single life cycle and organizes the step-by-step process. The software development life cycle is not a way to develop software from a perspective of development. Therefore, different types of software products require different work content and order of work. Choosing a software development life cycle that does not fit the type of product may result in missing the required work or in an inappropriate work order, making development inefficient or reducing the quality of the product.

2) Characteristics of software development life cycle

- Project management: Support project management to facilitate project execution
- Efficient use of resources: Project cost calculation and schedule planning support

3) Background of the emergence of the software development life cycle

In terms of software size, large-scale software development and crisis-induced systematic software development was necessary. Furthermore, measures were needed to effectively conduct high-quality software development in terms of software quality.

4) Components of the software development life cycle

- Plan: Identify the functionality of the system and review the business feasibility
- Analysis: Specifically analyze and understand user requirements
- Design: Structural and interface design between modules forming the system skeleton
- Implementation: Steps to express design content in programming language
- Testing: Test steps to detect and correct errors in an executable program
- Operation/maintenance: Steps to correct/complement problems that can occur during the user's own operation

2. Software Development Process Model

In order to build a process model, we first need to think about where to focus on the process. This allows us to understand the overall flow of the project and to clearly understand the development progress by applying the process model. The outputs produced at each stage make it easy to review the project. A model is needed that can safely share and modify code, allow all members of the organization to participate in communication, and eliminate unnecessary rules that interfere with the modern software development life cycle.

Models include waterfall models, prototype models, spiral models, RAD models, Agile, and Ddev ops.

1) Waterfall model

The waterfall model is carried out in such a way that each task in the project flows down step by step, like a waterfall, and the results completed at the end of the previous step are handed over to the group responsible for the next step. The waterfall model is a documented process. First, we create a requirement problem to build a project and complete the implementation. The completed software is delivered to the Quality Assurance Team (QA) for testing. When the distribution is finally determined to be verified, maintenance by the operations team after distribution is considered a matter of future consideration. Each team performs its duties in isolation. If a problem occurs during the test, the project returns to the development team and the development phase begins again.

Traditional software development cycles mostly follow this model. It is preferred in large organizations with clear hierarchies. It is easy to identify project progress and responsibilities throughout the process from requirements to design, implementation, testing, and completion of quality assurance. The waterfall model is a sequential process in which the project proceeds only after all the requirements have been collected, so it takes a very long time to reflect the changes. This can lead to situations in which the deployed software does not meet the needs.

Feasibility Study -> Plan -> Needs Analysis -> Design -> Implementation -> Testing -> Maintenance

2) Prototype

It is a model for clearly identifying users' requirements. It is a gradual development method that prototypes core or high-risk functions to predict the outcome that will be completed in the end. The prototype is built with a focus on the interface between the user and the system. The skeleton will be used in the implementation process. The model is designed to compensate for the errors found after the development of the waterfall model is completed.

It is easy to derive requirements and understand the system. Also, communication is improved. However, there is a possibility that users may misunderstand the prototype as a finished product, and there may be overhead and developer complaints due to the discarded prototype.

Collect Needs -> Quick Design -> Build a Prototype -> Customer Assessment
-> Alignment -> Implementation

3) Spiral model

The model was created by selectively accommodating the advantages of waterfall models and prototype models and adding risk analysis. The results are gradually completed through several development processes as if it were following a helix. Among the various functions that make up the system, it first develops the functions that will determine success or failure, and then adds the other functions later. The objective is to manage and minimize the possible risks during development. It is possible to proceed by adding missing or added requirements and does not require a precise maintenance process.

The work of the model consists of screws with multiple cycles. Each cycle of a repeated spiral begins with setting a goal. It also identifies other ways to achieve the goal and the existing constraints. Next, weigh and evaluate several alternatives. The focus of the assessment is the concept of project risk. The next step uses methods such as benchmarking, simulation, and prototyping to develop strategies to solve problems and risks. Then develop the software and plan the next steps.

An important feature of the spiral model is that after planning and analyzing the requirements for development, there are steps to review the risk factors and the next best option. It contains the philosophy of finding and preparing for failure and risk factors at the beginning of the project. Ultimately, the software development process was viewed from a risk management perspective.

The graphic representation of the spiral model results in a screw shape that is repeatedly thrown out front. In other words, the system is completed through several development cycles, not just one development cycle of planning, design, development, and evaluation, but by dividing the system into several parts. Theoretically, the external helical cycle is less dangerous because the system has become more specific throughout the iterative process.

Spiral models are evaluated in the best way to develop software for large-scale systems. In particular, this method enables developers or users to understand and take countermeasures against the risks arising from each stage

of expansion. Thus, it is a way to prevent the risk of a project failing in advance.

Spiral models are nonlinear and repeatedly developed, making them a way to increase the robustness of software quality. At the end of a cycle, requirements that are not included may be added as requirements for the development of the next phase. This is the robustness of the software and can be achieved by continuous improvement.

Plan -> Risk Analysis -> Development -> Customer Assessment

4) RAD(Rapid Application Development)

A sequential process model for developing software during a short development cycle of one to three months is performed using CASE tools, etc. for rapid development. Rapid development and full functional implementation are possible when the requirements are fully understood and project scope is clearly established. However, it can fail if there is no responsible member. It is also not suitable for high technical risks, with adequate modules and possibilities preconditioned.

5) Agile

The Agile model, a collaborative methodology, presupposes that each project is managed separately and that appropriate methods are applied to the project to meet the given requirements. In Agile, designers, developers, and test engineers team up and work together from project design to completion. Team members need to know what other members are doing. Thus, if one person is left out of the project, another member can quickly take on the task. The project is split into small task units called Sprint or Iteration. Each sprint has a very clear to do list, short-term goals, and completion dates, and all team members proceed together at the same time. If you divide a task into sprints, changing the requirements will not take months to discard the code you wrote. Sprint can proceed only when the work is clearly defined from the beginning stage. Thus, the scope of the project can be prevented from creeping up. If there is a problem during the sprint, the problem is postponed to the next sprint and the development continues. Only the goal of the Sprint must be accomplished. Each sprint accepts the customer's evaluation and demand for

results, and carries out the development work by giving priority to the customer.

Agile models can create cohesive teams that focus on their respective tasks and interactions, as they aim to create working software that can be tested during a project. Agile is designed to address customer requirements, software requirements, and communication issues between development and testing teams. The customer or someone who reflects the customer's sound can revert to the process, and reflecting the issue again does not delay the design/development process. As a result, the introduction of agile methodology can quickly reflect changes and realize high quality in a short time.

Scrum, XPeXtreme Programming, 칸반Kanban, Lean, 크리스탈Crystal, ASD(Adaptive Software Development), DSDM(Dynamic System Development), DAD(Disciplined Agile Delivery), and so on

6) DevOps

DevOps is a combination of development and operation, and it is a development methodology that cooperates with development and operation personnel.

DevOps focuses on narrowing the gap between the development/test team and the operations/IT infrastructure team. DevOps is designed to manage processes from start to finish within the IT department. In addition, quality control is applied while maintaining a viable state from the beginning of development. Thus, it improves the quality of the software and reduces the time it takes to distribute the software. Like Agile, DevOps is also interested in constantly testing and distributing software while focusing on change adaptation. DevOps does not have a clearly defined framework, but instead puts collaboration itself first. In other words, once one bottleneck is cleared, the next constraints are improved. If an issue occurs at any point, it is handed over to QA and development team again.

Faster work speeds enable better adaptation to market changes and efficiently generate business results. Furthermore, the faster the deployment of new features and bug fixes, the more competitive they can gain. It is possible to test whether changes are working properly and safely through methods such as

ensuring the quality of application updates and infrastructure changes, continuous integration, and continuous delivery. It is possible to operate and manage infrastructure and development processes according to size. Automated compliance policies, fine-grained control and configuration management techniques are available.

It is used in companies such as elegant brothers and smart study.

3. Software Development Deployment Environment

Software development should be more efficient by building a productive collaborative environment and leveraging contextual collaborative tools.

1) Modeling

Class diagrams, etc. can be created through modeling tools. Visual paradigm, starUML, etc.

debug, compilation, and distribution through ideodecode, an integrated development environment. Visual Studio, Eclipse, IntelliJ, etc.

In addition, the text editor includes atom, visual transcript studio, video transcript code transcript, subbubbuniformal text transcripts, etc.

3) DBMS

A set of software tools that provide acce

2) Development environment

It can handle related tasks such as coding, ss to data within a database. Oracle DB, MySQL, PostgreSql, mongoDB, and so on.

4) VCS

Version management system VCS is largely divided into local VCS, centralized VCS, and distributed VCS. Local VCS can be implemented as a database alone by managing versions within the local computer without a server. The centralization consists of a client-server structure. Distributed VCS typically has Git.

5) Project, issue management tools

Tools that improve work efficiency by managing the project as a whole include Jira, BugZila, Mantis, ASANA, and slack, etc.

6) Integration and deployment

Tools for developing and distributing products include Jenkins and Github Action.

7) Quality control tools

- Sonarcube

Sonarcube is a static analysis application that helps to continuously improve and maintain code quality. The analysis function is excellent and the improvements are detailed. It even guides the potential risks of the code, helping to improve the proper coding habits and quality of the code. It can run in a variety of environments such as Linux, Window, and Mac. It supports code analysis for more than 20 program languages. The quality of the developed code can be checked through the administrator and can be continuously managed. Standardized code quality requirements can be established through the quality gate. Integrated with CI engines such as Jenkins for analysis. Analysis is possible through a number of plugins linked to IDE.

4. Latest trends in utilization tools

As a result of interviews with on-site developers, Github and Jira are used a lot. Furthermore, we were able to identify which software development tools are in use today by looking at the 2019 survey by Stack Over Flow. MySQL received the most responses for DBMS and Visual Studio code for development environments.³ Latest trends in utilization tools

As a result of interviews with on-site developers, Github and Jira are used a lot. Furthermore, we were able to identify which software development tools are in use today by looking at the 2019 survey by Stack Over Flow. MySQL received the most responses for DBMS and Visual Studio code for development environments.

III. Software Development Best Practices Proposal

1. Software Development Process Model

Waterfall is a method that can be carried out sequentially from demand analysis to planning, development, testing, and release. Development begins after making all documents on the entire project, such as demand analysis and planning. This can cause unpredictable issues or additional requests to occur in the middle of development to be difficult to handle, resulting in customer requirements not meeting all or less complete. We cannot use waterfalls because we think it is most important to meet the customer's requirements as much as possible.

Agile, on the other hand, can be developed with agility and ability to meet requirements. The Sprint phase allows the bug to be fixed at the same time as development, and the necessary modifications can be made. Each team will be able to analyze problems through weekly meetings and reflect them by adding them in the middle of development, which will further enhance customer satisfaction. Therefore, we choose Agile.

Agile's two most famous tools are Scrum and Kanban. Scrum divides developable units into as small units as possible based on Sprint. It also estimates how long it will take to develop before development, plans, and develops accordingly. It will be able to develop quickly and efficiently within a given development period. Furthermore, the issues that occur in the middle are unconditionally backlogs of the next sprint, as they only run with the goal of a given task during planning. That is, if an issue occurs within a Sprint, it is not considered for that Sprint period.

On the other hand, Kanban has no fixed Sprint period. There is no fixed period of time when to handle issues in order of urgent work units in a continuous flow of work and when to deal with issues like Scrum. In addition, issues can be dealt with by adding them directly to the Kanban board when the issue occurs.

Since there are many exceptions or errors in development, I thought there would be many cases beyond the expected development time, and I think it is better to reflect on new issues as soon as possible and deal with them. Therefore, we choose to select the Kanban board.

2. Project Management

We thought about which project management tools are appropriate for us because of the high use of Redmine, Trello, and JIRA in the field. We thought that it would be a tool that can understand each person's work and current status at a glance due to the nature of work that requires efficient and fast processing of work. We also thought that if there were errors in the middle of development, we should be able to generate issues in real time, and that it would be a tool that would be able to link with github and manage releases and versions together. So we planned to use JIRA for the project.

In the case of Redmine, e-mail notifications can be made according to the progress of the work, so each person can handle their work within the time limit, and they can also manage their schedules. However, it was not easy to change the status of work because it was harder to grasp the current status of work than JIRA.

In the case of Trello, we could intuitively check the status of the issue, but we decided not to use it because the issue search function was not accurate and it was difficult to manage the history of the issues.

The biggest reason for choosing JIRA is that it can see the status of its operations at a glance. Backlog on the board, tasks selected for development, tasks in progress, and tasks completed are divided into four parts, which can be easily seen at a glance. In addition, drags make it easy to change the status of the work and make it easy for the reps and reporters to specify and verify it. In conjunction with github, the commitment related to the task can be displayed and the bug or work can be shown using icons. Project pages allow you to create relevant reports, such as project requirements reports, meeting reports, and so on, connecting to and releasing relevant tasks, and managing versions. I chose these points because I thought they fit us well.

3. Software Configuration Management

Software Configuration Management tools include Git, SVN, CVS, and Perforce. Since there are many situations where people work from home these days, collaboration should have been possible anytime, anywhere. I also thought that

history management should be easy in case of a possible situation and that it would be possible to systematically manage and maintain it.

CVS has the problem of not being rolled back if an error occurs during commitment, so it cannot be used because it may be difficult to recover in the event of an accident.

SVN is reflected in the central repository and is shared with everyone as soon as one person commutes. We thought that if more than two people modify and commit at the same time, there is a high possibility of conflict and if there is an error in the central storage, all the work would be paralyzed and it would not fit our work that had to be done quickly.

In the case of Perforce, I thought it would be okay to use it because it is fast and easy to search history, but it cannot be used because it is difficult to trace history if the file name is changed.

Therefore, we also choose Git because of its advantages of being heavily used by a large number of companies, having fast processing speed, easy history management for transient tasks, and being able to collaborate anywhere.

4. Software Quality Management

Quality control steps are to be carried out to ensure that all software requirements for users are met. First, the requirements added through discussion and essential requirements were summarized in the requirements statement before development could begin. Accordingly, each team decided to share their work and proceed with development and post it on GitHub to conduct code review and evaluation. We repeat the process of finding and correcting error-causing parts through code review. After development, check how much it matches the requirements, and if you think it is not suitable for the missing part or requirement, revise it again to repeat these processes.

5. Testing

Among the three types of integrated tests, functional tests, and unit tests, we considered which testing method is appropriate for SETB.

In computer programming, unit testing is a procedure that verifies that a particular module in the source code is functioning exactly as intended. That is,

the procedure for creating a test case for all functions and methods. The purpose of the unit test is to isolate each part of the program to ensure that each part is functioning correctly. This allows us to identify and correct it in a short time. Building a continuous unit test environment will allow us to verify and verify that the code and its execution are as intended, no matter what changes may be made.

Integration Test is a test performed to find problems with compatibility between modules in the process of integrating modules. Integration testing is similar to unit testing, with one big difference. Unit testing is independent of other components, while integrated testing is not. Integrated testing is usually more complex and takes longer than unit testing. Integrated testing is used when the unit test alone does not feel sufficient. We did not choose the integrated test because we thought the unit test was enough.

End-to-end test is to test the functionality of the system from beginning to end, using databases, networks, hardware, systems, etc., as is actually performed on site. I thought it was not suitable for this project because there were many items to check, such as whether the data was correctly contained in the database and the server had to respond within a certain time.

Therefore, the unit test tool that I thought was appropriate is Junit. Junit is one of the unit test frameworks run on a character or GUI basis. Junit tells the developer where the error occurred without debugging to "system.out.println", and infers the optimized code and provides it to the developer. You can also leave the test results as classes rather than text, leaving a record of the test classes. The use of Junit increases the efficiency of Test-Driven-Development (TDD).

6. Issue management (Jira)

There is a software called Jira as a tool for managing issues. Using Jira, you can share new issues with your team members immediately, and efficient work distribution can be made through designation of managers. Since there is a record of resolving the issue, the process of handling similar issues can be reviewed later. It is easy to modify and manage the main functions by raising the requirements to the story and tying the main functions together with Epic.

By tying up issues with Epic, you can see which Epic belongs to which Epic on the Kanban board, and how far the progress of the Epic-specific issues has been completed. Priority can be set for each issue. If the issue sorting criteria are prioritized in the Issues window, the priorities are sorted in high order and the high ones can be resolved first. You can create sub-tasks in the story. By creating sub-tasks, the development process can be smoother.

7. Integration and deployment (Jenkins)

Jenkins is a tool that automates the ongoing integration of build, test, and distribution in software development. When multiple developers develop a single program, they frequently upload their work to storage, such as Git in the shared area, to prevent version conflicts, enabling continuous integration. The use of Jenkins automatically automates builds, tests, and distributions, which have the advantages of verifying compilation errors in a project standard compilation environment, checking compliance with coding conventions, and simplifying various deployment tasks.

8. Tool and Environment

Java's development tools include IDE, Eclipse, Intel, and VS Code. There are features such as source code editors, local build automation, and debugger, as are common IDE features. Among them, the team members thought it would be better to use the IDE that they usually use, so they decided to proceed with the development by choosing one of the compatible Eclipse and IntelliJ. There is no problem with collaboration as the export functionality of importing (import) or exporting projects generated from one of the IDE projects works well. In case of an error, a solution can be found through the tool tip.

IV. Software Development Best Practice Verification Results and Knowledge Acquisition

1. Project Plan

After analyzing the requirements, we decided to divide the tasks and develop them separately to implement the functions. After development, several tests

were conducted to find errors and then to perform maintenance.

Week 1 (11/24–11/30): Analyse requirements, meet additional requirements, and select software development tools that work well with the project.

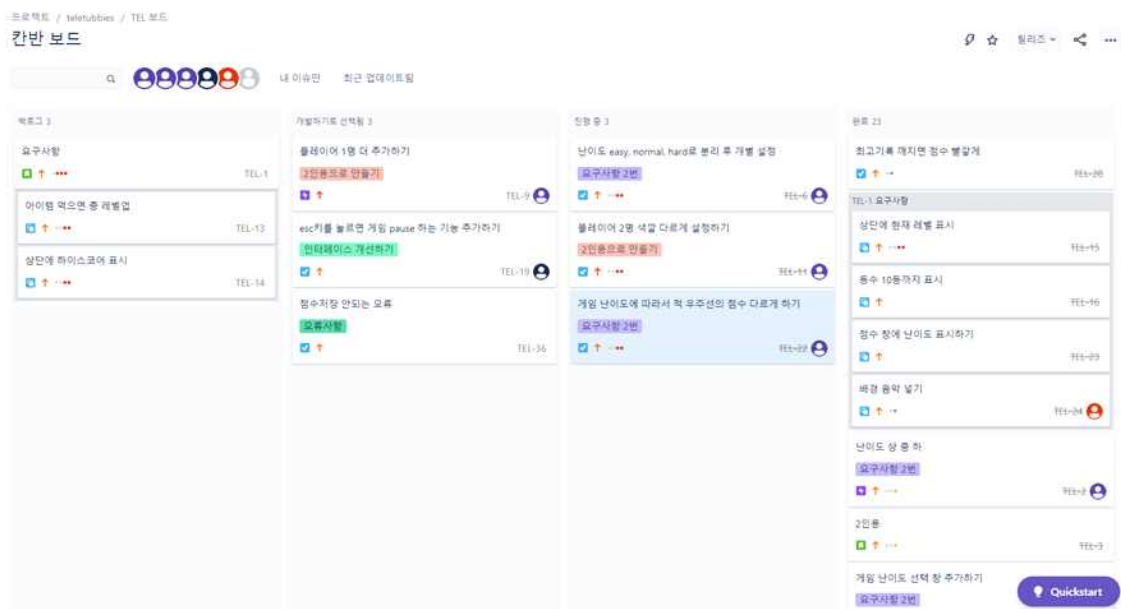
Week 2 (12/1-12/7) : Sharing team work and starting development with project management tools

Week 3 (12/8/12/12): Testing, maintenance, integration and distribution

2. Best Practice Validation Results and Knowledge Acquisition

1) KANBAN BOARD

As the development period is set at four weeks, it may be effective to write a scrum, but the Kavanese board, which is easy to change, was used to develop by adding errors or additional requirements during the development. The use of the KANBAN board provides a quick view of all tasks that need to be done, tasks in progress, and tasks completed, thus, the KANBAN board was adopted.



2) Issue Management Tools – JIRA

One of the reasons for adopting JIRA is that it can see the current status of its business at a glance. The board was divided into backlogs, tasks selected for development, tasks in progress, and tasks completed at a glance. The drag between the parts made it easy to change the work status and to easily specify

and verify the person in charge and the reporter. Furthermore, in conjunction with Github, when individual issues were clicked, the contents of commitments related to those issues could be checked.

The second reason for adopting JIRA is the advantage of efficient time allocation in the development process. Time management was very important because this project was a tight project that had to complete everything in a short period of four weeks. The JIRA showed how much each work was completed in percentage, and prioritized each requirement to ensure that development was not disrupted. These features allowed us to determine where we needed to spend more time and to complete the development in time.

The final reason for adopting the JIRA is to write a report. Issues generated within the JIRA could be linked together to produce multiple reports. In particular, while writing a report on development requirements, it was possible to write a report with high delivery power as the commitment linked to the issue could be added directly to the report.

Requirements

	Requirement	Details	Importance	Jira Issue
1	2인용으로 만들기	1. player1과 player2는 다른 색의 우주선을 갖는다. 2. player1과 player2의 점수를 각각 기록한다.	HIGH	<div>TEL-9: 플레이어 1명 더 추가하기 완료됨</div> <div>TEL-11: 플레이어 2명 색깔 다르게 설정하기 완료됨</div> <div>TEL-12: 플레이어 각각 점수 매기기 완료됨</div> <div>TEL-17: 플레이어 인원 선택화면 완료됨</div> <div>TEL-26: 플레이어 한명 죽으면 죽은 상태 유지 완료됨</div> <div>TEL-27: 난이도 선택 화면에서 뒤로가기 기능 추가 완료됨</div>
2	게임 난이도 설정	1. Easy, Normal, Hard 3단계로 난이도를 구성한다. 2. 각각의 레벨은 적, 총알, 우주선의 속도가 다르게 한다. 3. 기존의 게임과 다른 난이도를 가지도록 한다. 4. 난이도 별로 게임 점수를 개별적으로 저장한다.	HIGH	<div>TEL-2: 난이도 상 중 하 완료됨</div> <div>TEL-7: 난이도별 점수 다르게 저장할 변수 분리 완료됨</div> <div>TEL-6: 난이도 easy, normal, hard로 분리 후 개별 설정 완료됨</div> <div>TEL-22: 게임 난이도에 따라서 적 우주선의 점수 다르게 하기 완료됨</div> <div>TEL-4: 게임 난이도 선택 창 추가하기 완료됨</div>
3	인터페이스 개선	1. ESC키를 이용하여 게임을 일시 정지하는 기능을 추가한다. 2. 점수를 리셋하는 기능을 추가한다.	HIGH	<div>TEL-8: 점수 리셋하는 방법 만들기 완료됨</div> <div>TEL-20: 점수 리셋창 만들기 완료됨</div> <div>TEL-19: esc키를 누르면 게임 pause 하는 기능 추가하기 완료됨</div>
4	추가 요구사항	1. 마우스나 키보드 자판을 누르면 게임을 시작하는 기능을 추가한다. 2. 게임 중 아이템을 떨어뜨려 아이템 획득 시에 총 레벨을 더해준다. 3. 게임 화면 상단에 하이스코어와 레벨을 표시한다. 4. 레벨이나 난이도가 올라갈 시에 적이 쏘는 총의 개수를 늘린다. 5. 등수를 10등까지 표시한다. 6. 배경 음악을 삽입한다. 7. 점수 창에 난이도를 표시한다.	NORMAL	<div>TEL-15: 상단에 현재 레벨 표시 완료됨</div> <div>TEL-24: 배경 음악 넣기 완료됨</div> <div>TEL-23: 점수 창에 난이도 표시하기 완료됨</div> <div>TEL-16: 등수 10등까지 표시 완료됨</div> <div>TEL-14: 상단에 하이스코어 표시 BACKLOG</div> <div>TEL-13: 아이템 먹으면 총 레벨업 BACKLOG</div>

<Part of Requirements Report>

3) Software Configuration Management Tool – GITHUB

Although a variety of tools exist for configuration management, GITHUB was actively used in this project. I thought I could use many functions while working on the project because I learned various applications in the theoretical class at the beginning of the semester. In fact, during development, different development teams could easily manage files using different branches. Furthermore, a pull request could be posted on each branch to prevent complications during the merge, and a commentary in the code review could reveal new feature suggestions, problems in the code or errors occurring.

4) Class Diagram

A class diagram is a UML that expresses a component of a class or relationship between classes. The diagram tools within IntelliJ were used to identify the structure of the game project and to easily identify dependencies between classes. The diagram made it easy to identify the inheritance and dependencies of existing classes and classes added during development. During the course of development, we updated and checked the class diagram to ensure that the newly added classes were correctly referenced, and that there was also an unnecessary dependency.

5) Testing Tool – JUNIT

After developing Invaders, many bugs were discovered and fixed by running them on their own several occasions. However, I thought this was not enough, so I used JUnit as a testing tool for units. JUnit chose this because it is easy to use compared to other testing tools and shows good compatibility with the development environment based on JAVA.

However, it did not actually complete the test successfully. The reason for this project was that most of the variables changed during the game, but it was not possible to identify the variables that varied depending on the situation when tested on a unit basis. Thus, after testing only a portion of the game's development, they tried to find new testing methods and proceed with the testing.



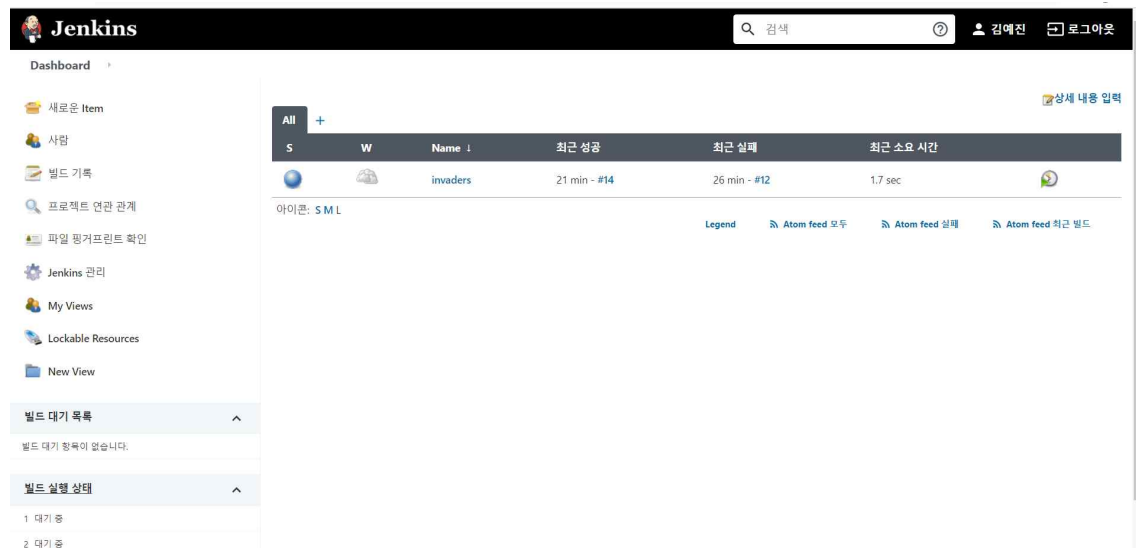
<Invaders Class Diagram>

6) Integrated Deployment Tool – Jenkins

We chose Jenkins, which has the advantage of automating ongoing integration such as build, test, and distribution as integration and deployment. It even connected with the GitHub Repository in Jenkins to automatically build the commit, but unfortunately, it was not distributed. Jenkins was installed on the ec2 Amazon Linux server to run Jenkins on the Jenkins container. The webbook of GitHub was used to make an automated build in Jenkins.

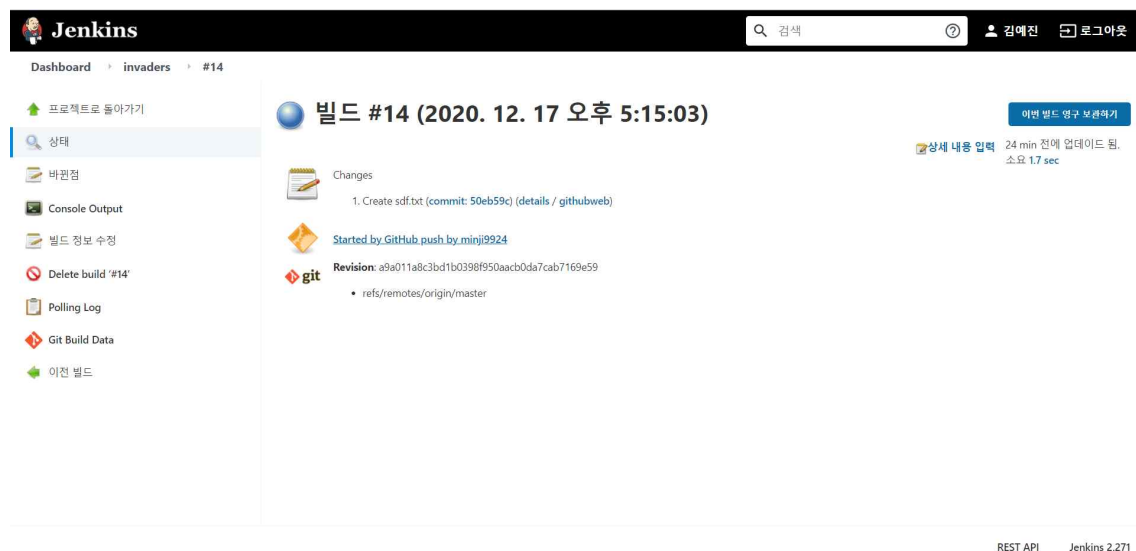
Tomcat was installed on ec2 server for deployment. Since the Jenkins port was 8080, Tomcat's port was designated 8088, but when built at the end, it failed to distribute an error in maven, pom.xml.xml. However, instead of

distributing it at Jenkins, the .war file could be extracted and uploaded to github to find an alternative.



The screenshot shows the Jenkins Dashboard. On the left is a sidebar with navigation links like '새로운 Item', '사람', '빌드 기록', etc. The main area displays a table of builds for the 'invaders' job. The table has columns for status (S, W), name, and timing. The first build is 'invaders' with a status of 'S' (Success), a name of 'invaders', and a timing of '21 min - #14'. Below the table, there are links for 'Legend', 'Atom feed 모두', 'Atom feed 실패', and 'Atom feed 최근 빌드'.

S	W	Name ↓	최근 성공	최근 실패	최근 소요 시간
		invaders	21 min - #14	26 min - #12	1.7 sec



The screenshot shows the Jenkins Build #14 page. The left sidebar has links like '프로젝트로 돌아가기', '상태', '바뀐점', etc. The main area displays the build details for '빌드 #14 (2020. 12. 17 오후 5:15:03)'. It shows the build was started by a GitHub push by 'minij9924'. The revision is 'a9a011a8c3bd1b0398f950aacb0da7cab7169e59' from the 'refs/remotes/origin/master' branch. The build was successful, with a timing of '24 min 전에 업데이트 됨. 소요 1.7 sec'.

빌드 #14 (2020. 12. 17 오후 5:15:03)

Changes

- 1. Create sdf.txt (commit: 50eb59c) (details / githubweb)

Started by GitHub push by minij9924

Revision: a9a011a8c3bd1b0398f950aacb0da7cab7169e59

- refs/remotes/origin/master

3. Project Epilogue

In this project, Agile and KANBAN were able to develop efficiently for a short period of time and felt that the improved collaboration was also well done. However, it was regrettable that the team developed using Jira, so it was not possible to assign a single issue to multiple people because it could not be

multiple designations of the person in charge and the reporter. It was also regrettable that in order to connect the GitHub commitment to each issue, the issue code must be included in the GitHub.

GitHub also had minor discomfort. As several people wrote the code together, there were times when conflicts occurred due to minor spacing problems. Furthermore, the initial use of the gif was unfamiliar and difficult to adapt due to the large number of functions and tools. It didn't matter much when I started the project because I was used to it.

Finally, this project failed to integrate IDE while developing. At first, there was no problem at all, but by touching the code associated with the highest score, we found that the two IDE paths were different. Although there were some difficulties in setting the route and not too inconvenient, the next time you participate in a project developed by multiple people, you will incorporate IDE.

V. Conclusion

In the case of KANBAN, the progress of the issue can be visually identified through KANBAN board. It also reduces project management overhead because it is easy to identify what is needed for the next task. Therefore, it is suitable for best practices.

By selecting Jira as a project management tool and issue management tool, we were able to maximize the advantages of Kavanaugh. With Jira, you can manage the Canban Board, easily change your work status, and easily create reports. In addition, it is suitable for best practices in that it can immediately share issues with team members and review the issue process.

Because SETB currently has software development on a team basis, git is suitable as a feature management tool. git is a distributed versioning management tool that is suitable for best practices because it is easy to manage history and systematically.

When the first best practice was proposed, code reviews were simply provided to each other through github in software quality management. However, it would be better to perform an automatic review of code analysis using code quality control tools such as SonarQube. SonarQube works with github and automatically modifies and analyzes code.

Furthermore, the most efficient method would be to configure Jenkins + Github web-book + SonarQube to modify code and register Github's PullRequest to automatically analyze the modified code and even post the information in the comments of the PullRequest.

VI. Git Address & Jenkins Address

<https://github.com/minji9924/invaders>

<http://3.34.45.230:8080/> (ID: teletubbies Password : 0000)