소프트웨어개발실무 PBL 최종보고서

팀 텔레토비

강예원 소프트웨어학부 2019054693

김예진 소프트웨어학부 2019069034

전민지 소프트웨어학부 2019025823

정연주 소프트웨어학부 2019014739

한수빈 소프트웨어학부 2019071994

목차

Ⅰ. 문제 정의

SETB의 팀 단위 소프트웨어 개발이 체계적이지 못하고 효율성이 낮아 맞춤형 제품의 품질 검증 및 보장을 위해서 많이 시간과 인적 노력이 필요한 상황이다. 따라서 새롭고 효율적인 소프트웨어 제품 개발 Best Practice를 구축해야 한다.

Ⅱ. 소프트웨어 개발 과정 조사

소프트웨어 개발은 요구사항, 설계, 구현, 테스팅, 유지 보수 등의 일련의 과정을 말한다.

개인이 진행하는 간단한 프로젝트에서 프로세스 모델을 적용하는 것은 거창해질 수 있다. 하지만, 다수가 진행하는 복잡한 프로젝트의 경우 프로세스 모델을 적용함으로써 개발 진행 상황을 파악하며 프로젝트에 대한 검토가 쉬워진다. 또한, 개발자 혼자서 간단한 것은 구현이 가능하겠지만 유지 관리, 확장, 수정 등을 위해서는 다른 이가 필요하다. 즉, 소프트웨어를 개발할 때 다른 사람의 협조와 협업은 필요하다. 팀 협업이 견고할 때 생기는 장점은 분명하다. 어느 한 사람에게 문제가 발생하면 다른 사람이 문제를 해결해주기가 쉽다. 고객의 니즈와 요구사항의 변화가 빨라짐에 따라 소프트웨어 역시 빠르게 개발과 보수가 이루어져야 한다. 그렇기 때문에 협업은 중요하다.

1. 소프트웨어 개발 과정 – 소프트웨어 개발 생명 주기(Software Development Life Cycle)

1) 정의

소프트웨어 개발 생명 주기란 소프트웨어 개발부터 폐기까지 전 과정을 하나의 생명 주기로 정의하고 단계 별 공정을 체계화한 모델을 일컫는다. 소프트웨어 개발 생명 주기는 개발을 바라보는 시각으로 소프트웨어를 개발하는 방법이 아니다. 따라서, 소프트웨어 제품의 종류가 다를 때에는 다른 작업 내용과 작업 순서가 요구된다. 제품의 종류에 맞지 않은 소프트웨어 개발 생명 주기를 선택할 경우, 필요한 작업을 빠뜨릴 수도 있고 부적합한 작업 순서로 진행되어 개발이 비효율적이거나 제품의 품질을 저하시킬 수도 있다.

2) 소프트웨어 개발 생명 주기의 특징

- 프로젝트 관리: 원활한 프로젝트 수행을 위한 프로젝트 관리 지원
- 효율적인 자원 사용: 프로젝트 비용의 산정, 일정 계획 수립 지원

3) 소프트웨어 개발 생명 주기의 등장 배경

소프트웨어 규모 측면에서 소프트웨어 대규모화 및 위기로 인한 체계적인 소프트웨어 개발이 필요하였다. 또한, 소프트웨어 품질 측면에서 고품질 소프트웨어 개발을 효과적으로 수행하기 위한 방안이 필요하였다.

4) 소프트웨어 개발 생명 주기의 구성요소

- 계획: 시스템이 갖추어야 할 기능 파악 및 비즈니스 타당성 검토
- 분석: 사용자의 요구사항을 구체적으로 분석, 파악
- 설계: 시스템 골격을 형성하는 모듈 간의 구조 및 인터페이스 설계
- 구현: 정해진 설계 내용을 프로그래밍 언어로 표현하는 단계
- 테스트: 실행 프로그램의 오류를 발견하고 수정하는 테스트 단계
- 운영/유지 보수: 사용자가 직접 운영하고 운영 시 발생 가능한 문제점을 수정/보완하는 단계

2. 소프트웨어 개발 과정 모델

프로세스 모델을 세우기 위해 먼저 과정 중 어디에 중점을 둘 것인지 생각해야 한다. 이에 따라 프로세스 모델을 적용함으로써 프로젝트가 어떤 식으로 진행될 것인지 전체적인 흐름을 파악할 수 있으며 개발 진행 상황을 명확히 파악할 수 있다. 각 단계에서 생성되는 산출물들을 통해 프로젝트에 대한 검토가 쉬워진다. 코드를 안전하게 공유 및 수정할 수 있고, 조직 구성원 모두가 의사소통에 참여할 수 있고, 현대식 소프트웨어 개발 생명 주기에 방해가 되는 불필요한 규칙들을 제거할 수 있는 모델이 필요하다.

모델의 종류로는 폭포수 모델, 프로토타입 모델, 나선형 모델, RAD 모델, 애자일, D데브옵스 등이 있다.

1) 폭포수 모델

폭포수 모델은 폭포수처럼 프로젝트 내의 각 작업이 단계 별로 차근차근 흘러내려가고, 앞선 단계를 끝내면서 완성한 결과물을 다음 단계를 담당하는 그룹에게 넘기는 방식으로 진행된다. 폭포수 모델은 문서 집중식 프로세스이다. 가장 먼저 요구사항 문제를 만들어 프로젝트를 구축하고 구현을 완료한다. 완성된 소프트웨어는 품질 보증팀(QA)에게 전달되어 테스트 과정을 거친다. 마지막으로 검증되었다고 판단되면 배포를 하는데, 배포 후 운영팀에서 유지보수 하는 것은 차후에 생각할 문

제로 간주한다. 팀 별로 각각 고립된 상태에서 업무 수행을 한다. 만약 테스트 중에 문제가 발생한다면, 프로젝트는 개발팀에게 돌아가고 개발 단계가 다시 시작된다.

전통적인 소프트웨어 개발 주기가 대부분 이 모델을 따른다. 계층 구조가 명확한 대형 조직에서 선호한다. 요구사항부터 설계, 구현, 테스트, 품질 보증 완료까지 전 과정에서 프로젝트 진척도와 책임 소재를 파악하기 쉽다. 폭포수 모델은 요구사항을 모두 취합하고 나서야 프로젝트가 진행되는 순차적 프로세스이므로 변경 사항을 반영하려면 시간이 매우 오래 걸린다. 따라서 배포된 소프트웨어가 니즈를 충족하지 못하는 상황이 발생하기도 한다.

타당성 검토 -> 계획 -> 요구 분석 -> 설계 -> 구현 -> 테스팅 -> 유지 보수

2) 프로토타입 모델(Prototype)

사용자의 요구사항을 명확하게 파악하기 위한 모델이다. 핵심 기능이나 위험성이 높은 기능을 시제품으로 만들어 최종에 완성될 결과물을 예측하는 점진적 개발 방법이다. 시제품은 사용자와 시스템 사이의 인터페이스에 중점을 두어서 만든다. 추후 구현 과정에서 사용될 골격이 된다. 폭포수 모델의 단점인 개발이 완료된 후 오류가 발견되는 점을 보완하기 위해 만들어진 모델이다.

요구사항 도출과 시스템 이해가 용이하다. 또한, 의사소통이 향상된다. 하지만, 사용자가 시제품을 완제품으로 오해할 가능성이 생기며 폐기되는 시제품에 따른 오버헤드 및 개발자의 불만이 발생할 수 있다.

요구 수집 -> 빠른 설계 -> 프로토타입 구축 -> 고객 평가 -> 조정 -> 구현

3) 나선형 모델(Spiral)

폭포수 모델과 프로토타입 모델의 장점을 선택적으로 수용하고 위험 분석을 추가하여 만든 모델이다. 나선을 따라 돌 듯이 여러 번의 개발 과정을 거쳐서 점진적으로 결과를 완성한다. 시스템을 이루는 여러 기능 중에 성패를 좌우할 만한 기능들을 먼저 개발하고 다른 기능들을 후에 추가해 나간다. 개발 중에 발생 가능한 위험을 관리하고, 최소화하는 것이 목적이다. 누락되거나 추가된 요구사항을 첨가하여 진행 가능하며 정밀하고 유지 보수 과정이 필요 없다.

모델의 작업들이 복수의 주기를 가진 나사와 같이 구성되어 있다. 반복되는 나선형의 각 주기는 목표를 설정하는 일부터 시작한다. 또한 목표를 성취하기 위한 다른 방안과 존재하는 제약 사항을 파악한다. 다음은 여러 대안들을 저울질하고 평가해 본다. 평가의 초점은 프로젝트의 위험이라는 개념이다. 다음 단계는 문제와 위험을 해결하는 전략을 개발하는데 벤치마킹, 시뮬레이션, 프로토타이핑과 같은 방법을

이용한다. 그 후에 소프트웨어를 개발하고 다음 단계를 계획한다.

나선형 모델의 중요한 특징은 개발을 위한 계획 및 요구분석 후에 위험 요소와 차선책에 대하여 검토하는 단계가 있다는 점이다. 프로젝트의 초기에 실패 요인과 위험 요소들을 찾아내어 대비하자는 철학이 담겨 있다. 결국 소프트웨어 개발 프로세스를 위험관리 측면에서 보았다.

나선형 모델을 그림으로 나타내면 반복적으로 앞트로 나가는 나사 모양이 된다. 즉, 계획, 설계, 개발, 평가의 개발 주기가 한 번에 그치는 것이 아니라 시스템을 여러 부분으로 나누어 여러 번의 개발 주기를 거치면서 시스템이 완성된다. 이론적으로는 외부 나선 주기가 위험이 적은데 그 이유는 반복 과정을 거치면서 시스템이 더욱 구체화되었기 때문이다,

나선형 모델은 대규모 시스템의 소프트웨어를 개발하는 데 가장 적합한 방법으로 평가받고 있다. 특히 이 방법은 개발자나 사용자가 각 확장 단계에서 발생될 위험에 대한 이해와 대책이 가능하다. 따라서 프로젝트가 실패로 끝날 수 있는 위험을 사전에 막는 방법이 된다.

나선형 모델은 비선형적이며 반복적으로 개발이 진행되므로 소프트웨어 품질 중 강인성을 높일 수 있는 방법이 된다. 한 사이클이 끝난 후 포함되지 못한 요구 사항을 다음 단계의 개발을 위한 요구 사항으로 첨가할 수 있다. 이것이 소프트웨어의 강인성이며 계속적인 개선에 의하여 정확성을 획득할 수가 있다.

계획 ->위험 분석 -> 개발 -> 고객 평가

4) RAD 모델(Rapid Application Development)

1~3개월의 짧은 개발 주기 동안 소프트웨어를 개발하기 위한 순차적 프로세스 모델로 빠른 개발을 위해 CASE 도구 등을 활용하여 수행한다. 요구사항의 완전한 이해와 프로젝트 범위의 명확한 설정 시 신속한 개발 및 완전한 기능 구현이 가능하다. 하지만, 책임감 있는 구성원이 없을 경우에 실패할 수 있다. 또한, 적절한 모듈과 가능성이 전제되어있으며 기술적 위험이 높을 경우 부적합하다.

5) 애자일

협업형 방법론인 애자일 모델은 각 프로젝트는 분리하여 관리하고, 주어진 요구사항을 충족할 수 있는 알맞은 방법을 프로젝트에 적용하는 것을 전제로 한다. 애자일에서는 설계자, 개발자, 테스트 엔지니어가 한 팀을 이루고, 프로젝트 설계부터 완성까지 함께 작업한다. 팀 구성원은 다른 구성원이 무엇을 하는지 알아야 한다. 따라서 프로젝트에서 어느 한 사람이 빠지면 다른 구성원이 신속하게 해당 업무를 맡

을 수 있다. 프로젝트는 스프린트(Sprint) 또는 이터레이션(Iteration)이라고 하는 작은 작업 단위로 쪼개진다. 각 스프린트는 할 일 목록, 단기 목표, 완료일이 매우 명확하게 정해지고, 팀 구성원 모두가 동시에 함께 진행한다. 작업을 스프린트로 작게 나누면, 요구사항이 변경된다고 해서 몇 달이나 걸려서 작성한 코드를 버리는 일이 생기지 않는다. 스프린트는 작업 내용이 시작 단계부터 명확하게 정해져야 진행될 수 있다. 따라서 프로젝트 범위가 슬그머니 커지는 경우를 방지할 수 있다. 스프린트 중에 문제가 생기면, 그 문제는 다음 스프린트로 미루고, 계속 개발을 진행한다. 오직 정해진 스프린트의 목표 만을 완수하면 된다. 스프린트마다 결과에 대한 고객의 평가와 요구를 수용하며 고객이 우선 순위를 부여하여 개발 작업을 진행한다.

애자일 모델은 프로젝트 진행 중에 테스트할 수 있는 작동하는 소프트웨어 작성을 목표로 제시하기 때문에, 각자의 임무와 상호 작용에 집중하는 응집력 높은 팀을 만들 수 있다. 애자일은 고객 요구사항, 소프트웨어 요구사항 사이 그리고 개발팀과 테스트팀 사이의 커뮤니케이션 문제를 해결하고자 고안되었다. 고객 또는 고객의 소리를 반영하는 누군가가 프로세스에 되돌릴 수 있고, 이슈를 다시 반영한다고 해서 설계/개발 프로세스가 지연되지 않는다. 그 결과, 애자일 방법론을 도입하면 변경사항을 신속하게 반영하고 짧은 시간 안에 높은 품질을 실현할 수 있다.

스크럼(Scrum), XP(eXtreame Programming), 칸반(Kanban), Lean, 크리스탈(Crystal), ASD(Adaptive Software Development), DSDM(Dynamic System Development), DAD(Disciplined Agile Delivery) 등의 모델이 있다.

6) 데브옵스

데브옵스는 development와 operation의 합성어로 개발 담당자와 운영담당자가 연계하여 협력하는 개발 방법론이다.

데브옵스는 개발/테스트 팀과 운영/IT 인프라 팀 사이의 간격을 좁히는 것에 초점을 둔다. 데브옵스는 IT부서 내에서 프로세스를 시작부터 끝까지 관리할 수 있도록 고안되었다. 또한, 개발 초기부터 실행이 가능한 상태로 유지하며 퀄리티 컨트롤을 적용한다. 따라서, 소프트웨어의 질적 향상과 소프트웨어를 배포하는데 걸리는 시간을 줄인다. 애자일과 마찬가지로, 데브옵스 역시 변경 적응에 초점을 두면서도, 소프트웨어를 끊임없이 테스트하고 배포하는 것에도 관심을 가진다. 데브옵스에는 명확하게 정의된 프레임워크가 없지만 그 대신 협업 자체를 최우선으로 한다. 다시 말해서 병목 하나를 해소하고 나면, 그 다음 제약 사항을 개선한다. 어느 시점이든 이슈가 발생하면 QA와 개발팀에게 다시 넘긴다.

작업 속도가 빨라지면서 시장 변화에 더 잘 적응하고 효율적으로 비즈니스 성과

를 창출할 수 있다. 또한, 새로운 기능의 배포와 버그 수정 속도가 빨라질수록 경쟁 우위를 차지할 수 있다. 애플리케이션 업데이트와 인프라 변경의 품질 보장, 지속적 통합 및 지속적 전달과 같은 방식을 통해 변경 사항이 제대로 안전하게 작동하는지 테스트가 가능하다. 규모에 따라 인프라와 개발 프로세스 운영, 관리가 가능하다. 자동화된 규정 준수 정책, 세분화된 제어 및 구성 관리 기술이 사용 가능하다.

우아한 형제들, 스마트 스터디와 같은 기업에서 사용되고 있다.

3. 소프트웨어 개발 구축 환경

생산성 높은 협업 환경을 구축하고 상황에 맞는 협업 도구를 활용하여 보다 더 효율적으로 소프트웨어를 개발해야 한다.

1) 모델링

모델링 도구를 통해 클래스 다이어그램 등을 작성할 수 있다. Visual Paradigm, starUML 등이 있다.

2) 개발 환경

통합개발 환경인 IDE를 통해 코딩, 디버그, 컴파일, 배포 등 관련 작업을 처리할 수 있다. IDE에는 Visual Studio, Eclipse, IntelliJ 등이 있다.

또한, 텍스트 편집기에는 Atom, Visual Studio Code, Sublime Text 등이 있다.

3) DBMS

데이터베이스 내의 데이터를 접근할 수 있도록 해주는 소프트웨어 도구의 집합이다. Oracle DB, MySQL, PostgreSql, mongoDB 등이 있다.

4) VCS

버전 관리 시스템 VCS는 크게 로컬 VCS, 중앙집중 VCS, 분산 VCS으로 나눠진다. 로컬 VCS는 서버 없이 로컬 컴퓨터 내에서 버전을 관리하여 데이터베이스 만으로도 구현이 가능하다. 중앙 집중식은 클라이언트-서버 구조로 이루어져 있다. 분산 VCS에는 대표적으로 Git이 있다.

5) 프로젝트, 이슈 관리 도구

프로젝트를 전반적으로 관리하여 업무의 효율을 높이는 도구에는 Jira, BugZila, Mantis, ASANA, slack 등이 있다.

6) 통합 및 배포

제품을 개발하고 배포할 수 있는 도구로는 Jenkins, Github Action 등이 있다.

7) 품질 관리 도구

- 소나큐브

소나큐브는 지속적으로 코드의 품질을 높이고 유지 보수하는데 도움을 주는 정적 분석 애플리케이션이다. 분석 기능이 탁월하고 개선사항도 자세하게 안내해준다. 코드의 잠재적인 위험 사항까지 안내해주어 올바른 코딩 습관과 코드의 품질 향상에 많은 도움을 준다. Linux, Window, Mac 등 다양한 환경에서 모두 구동이 가능하다. 20개가 넘는 프로그램 언어에 대한 코드 분석을 지원한다. 개발된 코드의 품질을 어드민을 통해 확인해 볼 수 있고 지속적으로 관리가 가능하다. 품질 게이트를 통해 표준화된 코드 품질 요구사항을 설정할 수 있다. Jenkins와 같은 CI 엔진과 통합되어 분석이 가능하다. IDE와 연동되는 다수의 Plugin을 통해 분석이 가능하다.

4. 활용 도구에 대한 최신 트렌드

현업 개발자와의 인터뷰를 진행한 결과, Github와 Jira를 많이 사용한다고 한다. 또한, Stack Over Flow의 2019년 조사 결과를 보며, 현재 많이 쓰이고 있는 소프트웨어 개발 도구가 무엇인지 파악할 수 있었다. DBMS에서는 MySQL이, 개발환경으로는 Visual Studio code가 가장 많은 응답을 받았다.

Ⅲ. 소프트웨어 개발 Best Practice 제안

1. 소프트웨어 개발 프로세스 모델

폭소수는 요구분석부터 기획, 개발, 테스트, 출시까지 순차적으로 진행할 수 있는 방식이다. 요구분석, 기획 등 전체 프로젝트에 대한 모든 문서를 만든 후에 개발을 들어가는 것이다. 이는 개발 중간중간에 예측할 수 없는 이슈나 추가적인 요청사항이 생기면 그 이슈들을 처리하기 힘들고 그로 인해 고객의 요구사항이 모두 충족되지 않거나 완성도가 낮은 결과를 얻을 수 있다. 우리는 고객의 요구사항을 최대한 충족하는 것을 가장 중요하게 생각했기 때문에 폭포수는 사용할 수 없다.

반면 애자일은 기민하고 민첩하게 요구사항들을 충족하면서 개발할 수 있다. 스프린트 단계로 이루어져 개발과 동시에 버그를 고치고, 필요한 것을 수정할 수 있다. 팀별로 주마다 회의를 통해 문제점을 분석하고 그에 대한 이슈들을 개발 중간에 추

가하여 반영할 수 있어 고객의 만족도를 더욱 높일 수 있을 것이다. 따라서 애자일을 선택하기로 한다.

애자일의 가장 유명한 2가지 도구로는 스크럼과 칸반이 있다. 스크럼은 스프린트 기반으로 개발 가능한 단위를 최대한 작은 단위로 쪼갠다. 또한 개발 전에 개발하는데 개발 시간이 얼마나 걸릴지 추정하고 계획하여 그에 맞춰서 개발한다. 이는 주어진 개발 기간 내에 빠르고 효율적으로 개발할 수 있을 것이다. 또한, 플래닝 때 정해진 작업만을 목표로 달리기 때문에 중간에 발생하는 이슈는 무조건 다음 스프린트의 백로그로 들어가게 된다. 즉 한 스프린트 내에서 이슈 발생 시 해당 스프린트 기간 동안 고려하지 않는다.

이와 달리 칸반은 스프린트 기간이 정해져 있지 않다. 연속적인 일의 흐름에서 급한 작업 단위 순서대로 이슈를 처리하고 스크럼처럼 이슈를 언제부터 언제까지 처리할 건지 정해진 기간이 없다. 또한 이슈 발생시 이슈를 칸반 보드에 바로바로 추가하여 처리할 수 있다.

개발할 때 예외나 오류가 많이 발생하기 때문에 예상한 개발 시간을 넘어서는 일이 많을 거라 생각하였고 새로운 이슈가 발생하면 최대한 빠르게 반영하여 이슈를 처리하는 것이 좋다고 생각한다. 따라서 칸반 보드를 선택하기로 한다.

2. 프로젝트 관리

현업에서 Redmine, Trello, JIRA 등을 많이 사용한다고 하여 이 중에서 우리에게 적합한 프로젝트 관리 도구는 무엇인지 생각해보았다. 우리는 업무를 효율적으로 빠르게 처리해야 하는 업무 특성 상 각자의 업무와 현황을 한눈에 파악할 수 있는 도구였으면 좋겠다고 생각하였다. 또한, 개발 중간에 오류사항이 생기면 바로 이슈를 실시간으로 생성할 수 있어야 하며, github와 연동이 되고 릴리즈와 버전 관리도 같이 할 수 있는 도구였으면 좋겠다고 생각하였다. 그래서 우리는 프로젝트에 JIRA를 사용하기로 계획하였다.

Redmine의 경우 일감 진행에 따른 이메일 통보가 가능해 각자의 업무를 빠짐없이 기한 내에 처리할 수 있고, 일정관리도 할 수 있다. 하지만 JIRA보다 업무 현황을 한눈에 파악하기 어려워 업무 상태를 쉽게 변경할 수 없다고 보았다.

Trello의 경우 직관적으로 이슈의 상태를 확인할 수 있었지만 이슈 검색 기능은 정확하지 않고 이슈들의 히스토리 관리가 어렵다는 점 때문에 사용하지 않기로 하였다.

JIRA를 선택한 가장 큰 이유는 업무 현황을 한눈에 파악할 수 있다는 점이다. 보드에 백로그, 개발하기로 선택된 업무, 진행 중인 업무, 완료된 업무로 4가지의 파

트로 나누어져 있어 한눈에 잘 보인다. 또한 드래그를 통해 쉽게 업무 상태를 변경할 수 있고 담당자와 보고자도 쉽게 지정하고 확인할 수 있다. github와 연동하여 해당 업무에 관련된 커밋을 표시할 수 있으며 아이콘을 이용해 버그인지 작업인지도 보여줄 수 있다. 프로젝트 페이지를 통해 프로젝트 요구사항 보고서, 회의 보고서 등 관련된 보고서를 작성하고 해당 업무를 연결하여 확인하기 편하고 릴리즈하여 버전관리 또한 할 수 있다. 이러한 점들이 우리와 잘 맞다고 생각하여 선택하게 되었다.

3. 형상관리

형상관리 툴에는 Git, SVN, CVS, Perforce가 있다. 요즘 재택근무를 하는 상황이 많이 발생하기 때문에 언제 어디서나 협업이 가능해야 했다. 또한 혹시 모를 상황에 대비하여 히스토리 관리가 편해야 하고 체계적으로 관리하고 유지가 가능했으면 좋겠다고 생각하였다.

CVS의 경우 커밋 중 오류가 발생하면 롤백이 되지 않는다는 문제점이 있어 혹시 모를 사고가 일어났을 경우 복구하기 힘들 수도 있다고 생각하여 사용할 수 없다.

SVN의 경우 중앙저장소에 반영이 되어 한 사람이 커밋을 하는 순간 모든 사람에게 공유가 된다. 이는 두 명 이상이 동시에 수정하고 커밋하였을 경우 충돌이 일어날 확률이 높고 중앙저장소에 에러가 생기면 모든 작업이 마비가 되어 빠르게 작업해야 하는 우리 업무와는 맞지 않다고 생각하였다.

Perforce의 경우 빠르고 히스토리 검색도 편하여 사용해도 괜찮겠다고 생각하였으나 파일명이 바뀌면 히스토리 추적이 곤란하다는 단점이 있어 사용할 수 없다.

따라서 우리는 Git을 선택하였다. 다수의 기업에서 Git을 많이 사용하기도 하고 처리 속도가 빠르며 일시적인 작업에 대한 이력 관리가 쉽고 어디서나 협업이 가능하다는 장점 때문에 선택하였다.

4. 소프트웨어 품질 관리

사용자의 소프트웨어 요구사항을 모두 충족하였는지 확인하기 위해 품질 관리 단계를 진행하기로 하였다. 우선 개발을 시작하기 전 필수 요구사항들과 토의를 통해 추가한 요구사항들을 요구사항명세서로 정리하기로 하였다. 그에 따라 각자 업무를 분담하여 개발을 진행하고 깃허브에 올려 다른 팀원들이 코드리뷰와 평가를 하기로 하였다. 코드 리뷰를 통해 오류가 발생하는 부분을 찾고 수정하는 과정을 반복한다. 개발을 마친 후 요구사항과 어느 정도 일치하는지 확인해보고 빠뜨린 부분이나 요구사항과 적합하지 않다고 생각이 들면 다시 수정하여 이 과정들을 다시 반복한다.

5. 테스팅

통합 테스트, 기능 테스트, 유닛 테스트 세가지 테스트 종류 중 SETB사에 적합한 테스팅 방식은 무엇인지 생각해 보았다.

유닛 테스트(unit test)는 컴퓨터 프로그래밍에서 소스 코드의 특정 모듈이 의도된 대로 정확히 작동하는지 검증하는 절차이다. 즉, 모든 함수와 메소드에 대한 테스트 케이스(Test case)를 작성하는 절차를 말한다. 유닛 테스트의 목적은 프로그램의 각 부분을 고립시켜서 각각의 부분이 정확하게 동작하는지 확인하는 것이다. 이를 통해서 단시간 내에 이를 파악하고 바로잡을 수 있도록 해준다. 지속적인 유닛 테스트 환경을 구축하면 어떠한 변화가 있더라도 코드와 그 실행이 의도대로 인지를 확인하고 검증할 수 있게 된다.

통합테스트(Integration Test)는 모듈을 통합하는 과정에서 모듈 간 호환성의 문제를 찾아내기 위해 수행되는 테스트이다. 통합 테스트는 유닛 테스트와 비슷한데, 큰 차이점이 하나 있다. 유닛 테스트는 다른 컴포넌트들과 독립적인 반면 통합 테스트는 그렇지 않다. 통합 테스트는 대게 유닛 테스트를 작성하는 것보다 복잡하고 오랜 시간이 걸린다. 통합 테스트는 유닛 테스트만으로 충분하다고 느끼지 못할 때 사용된다. 우리는 유닛 테스트만으로도 충분하다고 생각했기 때문에 통합 테스트를 선택하지 않았다.

E2E 테스트(End-to-end Test)는 현장에서 실제로 수행되는 것처럼 데이터베이스, 네트워크, 하드웨어, 시스템 등을 이용하여, 시스템의 기능을 처음부터 끝까지 테스트하는 것이다. 데이터베이스에 자료가 정확하게 들어가 있는지, 서버는 일정시간안에 응답해야 하는 등 점검해야 할 항목이 많기 때문에 이번 프로젝트에 적합하지 않다고 생각했다.

따라서 적합하다고 생각한 유닛 테스트 도구는 Junit이다. Junit은 단위테스트 framework 중 하나로서 문자 혹은 GUI 기반으로 실행된다. Junit을 사용하면 "system.out.println"으로 디버깅을 하지 않아도 어디에서 오류가 났는지를 알려주고, 최적화된 코드를 유추해 개발자에게 제공한다. 또한 테스트 결과를 텍스트가 아닌 클래스로 남겨 테스트 클래스의 기록을 남겨줄 수 있다. Junit을 사용하면 TDD(Test-Driven-Development, 테스트 기반 개발) 방식의 효율화를 높여준다.

6. 이슈관리(지라)

이슈를 관리하는 도구로 지라라는 소프트웨어가 있다. 지라를 이용해 팀원들과 함께 새로 생긴 이슈를 바로 공유할 수 있고, 담당자 지정을 통해 효율적인 업무 분

배를 할 수 있다. 이슈 해결에 대한 기록이 남기 때문에 후에 비슷한 이슈가 발생했을 때 처리 과정을 되짚어 볼 수 있다. 요구사항을 스토리로 올리고, 메인 기능들을 에픽으로 묶어서 기능별로 수정하고 관리하기가 쉽다. 에픽으로 이슈를 묶게 되면 칸반보드에서도 이슈가 어떤 에픽에 속해있는 지를 알 수 있고, 에픽 별 이슈의 진행상황 또한 어느 정도 완료되었는지 확인할 수 있다. 각 이슈별로 우선 순위를 정할 수 있다. 이슈 창에서 이슈 정렬 기준을 우선 순위로 하게 되면, 우선 순위가 높은 순으로 정렬되고 높은 것 먼저 이슈를 해결할 수 있다. 스토리에서 하위작업을 생성할 수 있다. 하위작업을 생성함으로써 개발 진행과정이 더 원활하게 이뤄질 수 있다.

7. 통합 등에 대한 기법(jenkins)

젠킨스(Jenkins)는 소프트웨어 개발 시 빌드, 테스트, 배포 등의 지속적인 통합을 자동화해주는 툴이다. 다수의 개발자들이 하나의 프로그램을 개발할 때 버전 충돌을 방지하기 위해 각자 작업한 내용을 공유 영역에 있는Git등의 저장소에 빈번히 업로드함으로써 지속적 통합이 가능하도록 해 준다. 젠킨스를 사용하게 되면 빌드와 테스트, 배포가 자동화되어 프로젝트 표준 컴파일 환경에서의 컴파일 오류 검증과 코딩 규약 준수여부 체크, 각종 배치 작업의 간략화 등의 장점이 있다.

8. 도구 및 환경

자바의 개발 도구에는IDE인 이클립스와 인텔리제이, VS Code 등이 있다. 일반적인 IDE의 기능들처럼 소스 코드 편집기, 로컬 빌드 자동화, 디버거 등의 기능이 있다. 그 중에서도 팀원들이 평소 사용하는 IDE를 사용하는 것이 좋다고 생각하여 호환이 되는 이클립스와 인텔리제이 둘 중 하나를 선택하여 개발을 진행하기로 하였다. 둘 중 하나의 IDE에서 생성된 프로젝트를 다른 IDE 프로젝트로 가져오거나(import) 내보내는(export) 기능이 잘 작동하므로 협업에 문제가 없다. 에러가 났을 경우, 툴팁을 통하여 해결책을 알 수 있다.

IV. 소프트웨어 개발 Best Practice 검증 결과 및 습득 지식

1. 프로젝트 주차별 계획

요구사항 분석을 한 후 업무 분담을 하여 각자 개발을 하여 기능을 구현하기로 하였다. 개발이 끝난 후 여러 번의 테스팅을 통해 오류 사항을 찾아낸 후 유지보수를 하기로 계획하였다.

1주차(11/24~11/30) : 요구사항 분석, 추가 요구사항 회의, 프로젝트와 잘 맞는 소프트웨어 개발 도구 선택

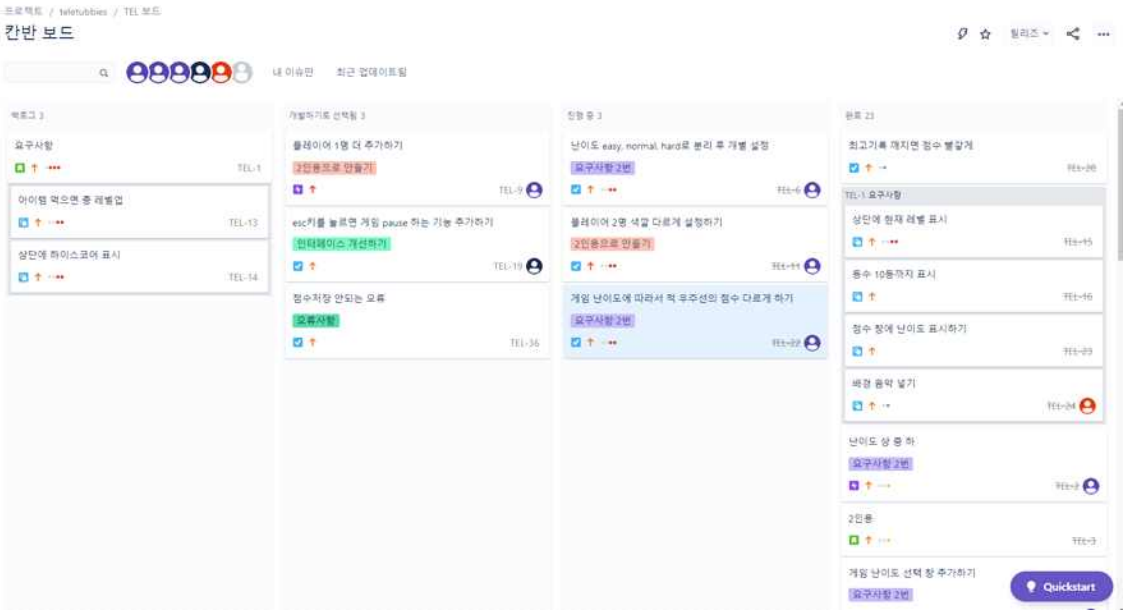
2주차(12/1~12/7) : 프로젝트 관리 도구를 이용하여 팀원 업무 분담 및 개발 시작

3주차(12/8~12/12) : 테스팅, 유지보수, 통합 및 배포

2. Best Practice 검증 결과 및 습득 지식

1) 칸반 보드

개발 기간이 4주로 정해져 있는 만큼, 스크럼을 쓰는 것이 효과적일 수 있으나 개발하면서 발생하는 오류나 추가되는 요구사항을 추가하여 개발하기 위해 변경이 쉬운 칸반 보드를 사용하였다. 칸반 보드를 사용하면 해야 하는 모든 업무와 진행 중인 업무, 완료한 업무를 한눈에 파악할 수 있어서 칸반 보드를 채택하였다.



2) 이슈 관리 도구 – JIRA

JIRA를 채택한 이유 중 하나는 업무 현황을 한눈에 파악할 수 있다는 점이다. 보드는 백로그, 개발하기로 선택된 업무, 진행 중인 업무, 완료된 업무 총 네 부분으로 나뉘어 한눈에 볼 수 있었다. 각 부분 간의 드래그를 통해 쉽게 업무 상태를 변경할 수 있었고 담당자와 보고자를 쉽게 지정하고 확인할 수 있었다. 또한, Github과 연동하여 개별적인 이슈 클릭 시 해당 이슈들과 관련된 커밋의 내용을 확인할 수 있었다.

JIRA를 채택한 두 번째 이유는 개발 과정에서 시간 배분을 효율적으로 할 수 있다는 장점이다. 본 프로젝트는 4주라는 짧은 시간 동안 모든 것을 마무리해야 하는

촉박한 프로젝트였기 때문에 시간 관리가 매우 중요했다. JIRA는 각각의 작업 현황이 어느 정도 마무리되었는지 퍼센트를 통해 보여주었고, 요구사항마다 우선순위를 설정하여 개발에 차질이 없도록 하였다. 이러한 기능들을 통해 어느 부분에 시간을 더 투자해야 하는지를 판단할 수 있었고, 기한 내에 개발을 완료할 수 있었다.

JIRA를 채택한 마지막 이유는 보고서 작성에 있다. JIRA 내에서 생성한 이슈들을 연결하여 여러 보고서 작성을 할 수 있었다. 특히 개발 요구사항 보고서를 작성하면서 관련 개발 과정을 일일이 적지 않아도, 이슈와 연결된 커밋을 직접 보고서에 추가할 수 있어서 전달력이 높은 보고서를 작성할 수 있었다.

Requirements

	Requirement	Details	Importance	Jira Issue
1	2인용으로 만들기	1. player1과 player2는 다른 색의 우주선을 갖는다. 2. player1과 player2의 점수를 각각 기록한다.	HIGH	TEL-9: 플레이어 1명 더 추가하기 완료됨 TEL-11: 플레이어 2명 색깔 다르게 설정하기 완료됨 TEL-12: 플레이어 각각 점수 매기기 완료됨 TEL-17: 플레이어 인원 선택화면 완료됨 TEL-26: 플레이어 한명 죽으면 죽은 상태 유지 완료됨 TEL-27: 난이도 선택 화면에서 뒤로가기 기능 추가 완료됨
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	TEL-2: 난이도 상 중 하 완료됨 TEL-7: 난이도별 점수 다르게 저장할 변수 분리 완료됨 TEL-6: 난이도 easy, normal, hard로 분리 후 개별 설정 완료됨 TEL-22: 게임 난이도에 따라서 적 우주선의 점수 다르게 하기 완료됨 TEL-4: 게임 난이도 선택 창 추가하기 완료됨
3	인터페이스 개선	1. ESC키를 이용하여 게임을 일시 정지하는 기능을 추가한다. 2. 점수를 리셋하는 기능을 추가한다.	HIGH	TEL-8: 점수 리셋하는 방법 만들기 완료됨 TEL-20: 점수 리셋창 만들기 완료됨 TEL-19: esc키를 눌르면 게임 pause 하는 기능 추가하기 완료됨
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	TEL-15: 상단에 현재 레벨 표시 완료됨 TEL-24: 배경 음악 넣기 완료됨 TEL-23: 점수 창에 난이도 표시하기 완료됨 TEL-16: 등수 10등까지 표시 완료됨 TEL-14: 상단에 하이스코어 표시 BACKLOG TEL-13: 아이템 먹으면 총 레벨업 BACKLOG

<요구사항 보고서 중 일부>

3) 형상관리 도구 – GITHUB

형상관리 도구는 다양한 툴이 존재하지만 본 프로젝트에서는 GITHUB을 적극적으로 이용하였다. 학기 초반 이론 수업에서 다양한 활용법을 학습했기 때문에 프로젝트를 하면서 많은 기능을 사용할 수 있을 것으로 생각했다. 실제로 개발을 하면서 개발팀마다 다른 branch를 사용하여 파일들을 쉽게 관리할 수 있었다. 또한, branch 마다 pull request를 올려 merge 시 conflict를 방지할 수 있었고, code review의 comment를 통해 새로운 기능 제안, 코드 내의 문제점이나 발생하는 오류를 확인할 수 있었다.

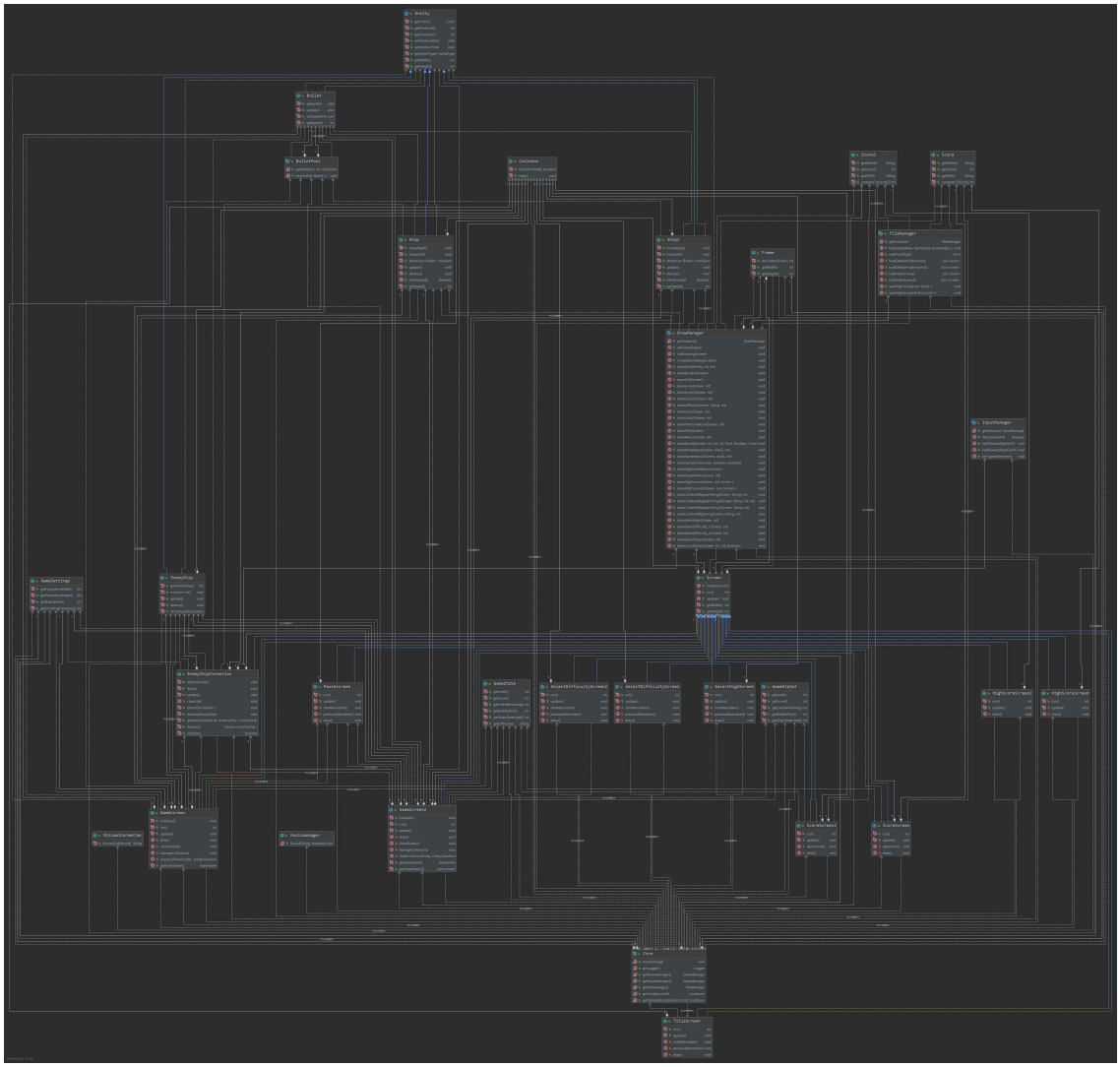
4) 클래스 다이어그램

클래스 다이어그램은 클래스의 구성요소나 여러 클래스 간의 관계를 표현하는 UML이다. Intellij 내의 다이어그램 툴을 이용하여 게임 프로젝트의 구조를 확인하고 클래스 간의 의존 관계를 쉽게 파악할 수 있었다. 다이어그램을 통해 기존에 존재하던 클래스와 개발 중 추가한 클래스의 상속 관계, 의존 관계를 쉽게 파악할 수 있었다. 개발 진행 중간에 클래스 다이어그램을 갱신하고 확인하면서 새롭게 추가된 클래스들이 올바르게 참조되고 있는지를 확인하였고, 불필요한 의존 관계가 있는지도 확인할 수 있었다.

5) 테스팅 도구 – JUNIT

Invaders 개발을 마치고 여러 번 직접 실행을 해봄으로써 많은 버그를 발견하고 고치는 과정을 반복하였다. 하지만 이것만으로는 충분하지 못하다고 생각하여, 유닛 단위 테스팅 도구로 junit을 사용하였다. Junit은 다른 테스팅 도구들과 비교하면 사용하기 쉽고, JAVA 기반으로 개발 환경과도 좋은 호환성을 보여주기 때문에 이를 선택하였다.

하지만 실제로 테스팅을 성공적으로 완료하지는 못했다. 그 이유는 본 프로젝트는 게임을 진행하면서 바뀌는 변수들이 대부분이었지만, 유닛 단위로 테스팅을 하게 되면 상황에 따라 달라지는 변수들을 확인할 수 없었기 때문이다. 하여, 게임 개발 내용의 일부만 테스팅을 진행한 후에 새로운 테스팅 방법을 찾아 테스팅을 진행하려고 노력했다.

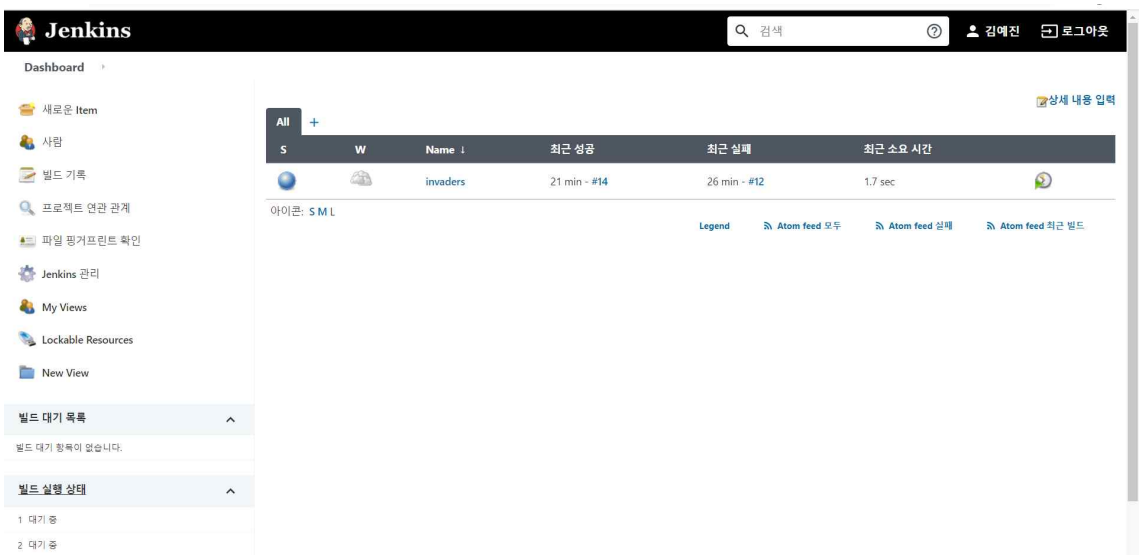


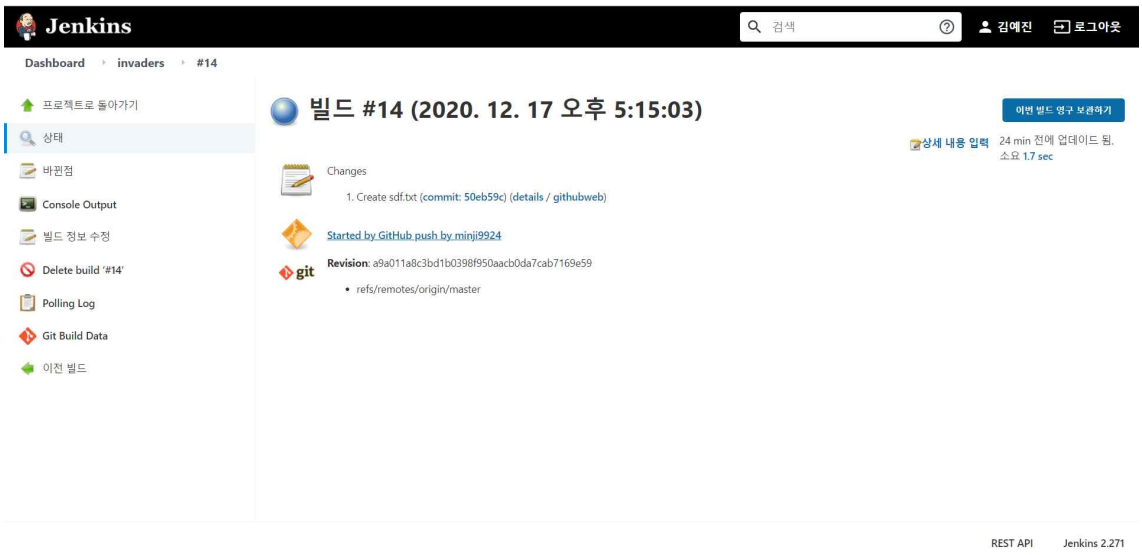
<Invaders 클래스 다이어그램>

6) 통합 배포 도구 – 젠킨스

통합 및 배포로는 빌드, 테스트, 배포 등의 지속적인 통합을 자동화할 수 있는 젠킨스를 선택하였다. 젠킨스에 깃헙 레포지토리와 연결을 하여 자동으로 커밋이 빌드가 되는 것 까지 하였으나 아쉽게도 배포는 하지 못했다. 젠킨스를 ec2 아마존 리눅스서버에서 도커를 설치해 젠킨스 컨테이너에서 젠킨스를 실행하였다. 깃헙의 web hook을 사용하여 젠킨스에 자동화된 빌드가 되도록 하였다.

배포를 하기 위해 ec2서버에 톰캣을 설치하였다. 젠킨스 포트가 8080이였기 때문에 톰캣의 포트는 8088로 지정을 해 주었지만 마지막에 빌드를 해 보니 maven, pom.xml에서 에러가 나 배포를 하지는 못했다. 하지만 젠킨스에서 배포를 하는 대신 .war 파일을 추출하여 github에 올림으로서 대안방안을 찾을 수 있었다.





3. 프로젝트를 진행하면서 느낀점

본 프로젝트에서 애자일, 칸반을 사용하면서 짧은 시간 동안 효율적인 개발을 할 수 있었고 향상된 협업이 역시 잘 이루어졌다는 것을 느꼈다. 그러나, 지라를 사용하면서 팀별로 개발을 하였기에 담당자와 보고자를 복수 지정할 수 없어 한 이슈를 여러 명에게 할당할 수 없는 점이 아쉬웠다. 또한, 각 이슈에 깃헙 커밋을 연결하기 위해서는 깃헙에서 커밋 시 이슈 코드를 포함해야만 연결된다는 점이 아쉬웠다.

깃헙 역시 사소한 불편함이 존재했다. 코드를 여러 명이 함께 짜다 보니 사소한

띄어쓰기 문제로 충돌이 일어나는 경우가 있었다. 또한, 처음에 깃헙을 사용할 때는 사용법이 익숙하지 않았고, 기능과 도구가 많아 적응하기 어려웠다. 프로젝트를 시작할 때는 사용이 익숙해서 큰 문제가 되지 않았다.

마지막으로 이번 프로젝트에서는 개발을 진행하면서 IDE를 통합하지 못했다. 처음에는 문제가 전혀 없었는데 최고점수와 관련된 코드를 만지면서 두 IDE의 경로 설정 방법이 다르다는 것을 알게 되었다. 경로 설정에서 약간의 어려움이 있었고 크게 불편하지는 않았지만, 다음에 여러 명이 개발하는 프로젝트에 참가한다면 IDE를 통합할 것이다.

Ⅴ. 결론

칸반의 경우 칸반 보드를 통해 가시적으로 이슈 진행 상황을 파악할 수 있다. 또한, 다음 작업을 위해서 무엇이 필요한지 파악하기가 용이하기에 프로젝트 관리 오버헤드를 감소시킨다. 따라서, Best Practice에 적합하다.

프로젝트 관리 도구와 이슈 관리 도구로 Jira를 선택함으로써, 칸반의 장점을 극대화할 수 있었다. Jira로 칸반보드를 관리하고 쉽게 업무 상태를 변경하며 보고서를 쉽게 작성할 수 있다. 또한, 팀원들과 이슈를 바로 공유하고 이슈 처리 과정을 되짚어 볼 수 있다는 점에서 Best Practice에 적합하다.

SETB사는 현재 팀 단위로 소프트웨어 개발이 이루어져 있기 때문에 형상 관리 도구로 git이 적합하다. git은 분산형 버전 관리 도구로 히스토리 관리가 용이하고 체계적으로 관리가 가능하기에 Best Practice에 적합하다.

처음 Best Practice를 제안했을 때에는, 소프트웨어 품질 관리에 있어서 단순히 코드 리뷰를 github를 통해 서로 리뷰를 해주도록 하였다. 하지만, SonarQube와 같은 코드 품질 관리 도구를 사용하여 코드 분석을 자동 리뷰를 수행하는 것이 좋을 것 같다. SonarQube의 경우 github와 연동이 되고 코드 수정과 분석을 자동으로 해준다.

또한, Jenkins + Github web-hook + SonarQube를 구성하여 코드를 수정하고 Github의 PullRequest를 등록하여 수정한 코드에 대해 자동으로 분석하고 그 정보를 PullRequest의 댓글에 다는 것까지 하는 것이 가장 효율적인 방법이 될 것이다.

Ⅵ. Git 주소 & 젠킨스 주소

https://github.com/minji9924/invaders

http://3.34.45.230:8080/ (아이디 : teletubbies 비밀번호 : 0000)