석사학위논문

# 소규모 조직을 위한
# 오픈소스 기반 지속적 통합 프로세스

Open Source Based Continuous Integration Process
for Small Organizations

**상명대학교 대학원**

컴퓨터과학과 컴퓨터과학전공

오 승 원

2018년 2월

석사학위논문

# 소규모 조직을 위한
# 오픈소스 기반 지속적 통합 프로세스

Open Source Based Continuous Integration Process
for Small Organizations

**상명대학교 대학원**

컴퓨터과학과 컴퓨터과학전공

오 승 원

2018년 2월

# 소규모 조직을 위한
# 오픈소스 기반 지속적 통합 프로세스

Open Source Based Continuous Integration Process
for Small Organizations

지도교수 한 혁 수

본 논문을 석사학위 논문으로 제출함

**상명대학교 대학원**

컴퓨터과학과 컴퓨터과학전공

오 승 원

2018년 2월

오 승 원의
석사학위 논문을 인준함

심사위원장 민 승 욱 ㊞

심사위원 한 혁 수 ㊞

심사위원 이 석 필 ㊞

상명대학교 대학원

2018년 2월

# 차 례

# 표 차 례

# 그 림 차 례

# 국 문 요 약

## 소규모 조직을 위한 오픈소스 기반 지속적 통합 프로세스

소프트웨어 프로젝트는 소프트웨어의 비가시성이라는 특성 때문에 진행 중 발생하는 결함에 대해 파악하기 힘든 어려움을 내재한다. 이러한 문제들은 소프트웨어 프로젝트 실패의 주요 원인이 되어왔다. 소프트웨어 공학 분야에서는 이에 대한 해결책으로 프로젝트 관리 이론을 채택하고 다양한 방법론과 CMMI와 같은 프로세스 모델을 개발하는 등의 노력을 기울여 왔다. 또한 개발/관리/지원 프로세스를 원활하게 구축하고 유지하기 위해 도구들을 개발하고 이를 적용하고 있다.

현재 시스템에서 소프트웨어가 차지하는 비율이 높아지고 그 규모가 점차적으로 커지게 되면서, 대부분의 프로젝트는 여러 참여자들이 함께 시스템을 개발하는 협업 환경이 이루어지게 되었다. 각자 개발한 서브시스템이 통합되고 전체적으로 테스트를 하면 최종 산출물이 완성되는 형태로 진행되는데, 일반적으로 통합 과정에서 개별적 구현에서는 파악하지 못한 에러가 발생하게 된다. 이러한 오류가 발생한다면 시간은 부족하고 수정해야 하는 양은 많아져 결국 프로젝트 실패로 이어질 수 있다.

이러한 문제점을 극복하기 위해서 프로젝트의 초기 단계부터 지속적으로 통합과 빌드, 검증 작업을 수행하여 문제점을 조기에 발견하는 지속적 통합이 제안되었고, 그 효용성이 입증되었다. 지속적 통합의 내재화가 소규모 조직에서도 필요하지만 지속적 통합 활동들은 정의와 구축부터 적용까지의 과정에 전문성을 요구하고, 처음 시도하는 경우에는 많은 시행착오를 겪어야하기 때문에 이를 지원할 수 있는

방안이 요구된다. 소규모 조직의 지원 방안으로 상용 솔루션을 활용하는 방법이 있는데, 비용이 높아서 문제이기도 하지만 아직 한 회사의 솔루션만으로는 지속적 통합의 모든 활동들을 지원하지 못하고 도구 간의 연동 또한 쉽지 않다. 이 모든 점을 고려해보면 오픈소스 도구를 사용하는 방법이 가장 적합하다. 하지만 이 방법은 적합한 도구의 선정, 학습, 연계 방안 파악 등 어려움이 있어 구현이 어렵다. 그러므로 소규모 조직의 지속적 통합 환경 구축 및 프로세스 적용을 위해서는 이를 위한 특정 가이드라인이 필요하다.

따라서 본 논문에서는 소규모 조직을 위한 오픈소스 기반 지속적 통합 프로세스를 제안하기 위하여 지속적 통합과 소규모 조직의 소프트웨어 개발 특성에 관하여 연구하였다. 연구한 내용을 기반으로 오픈소스 도구를 선정하고, 소규모 조직이 사용하기 쉽도록 도구 기반의 지속적 통합 프로세스를 총 4단계로 나누어, 기반 구축 단계와 빌드, 정적 분석, 단위 테스트를 위한 3단계로 구성하여 제공하였다. 기반 구축 단계는 소스 코드 형상관리 프로세스 구축으로 정의하였고, 1단계는 지속적 통합 빌드 프로세스 구축, 2단계는 정적 분석 프로세스 구축 및 지속적 통합 반영, 3단계는 단위 테스트 프로세스 구축 및 지속적 통합 반영으로 정의하였다. 또한, 제공한 프로세스의 효용성을 입증하기 위하여 학부 프로젝트에 2년간 적용하였으며 적용 과정과 결과를 서술하였다.

# 1. 서론

소프트웨어 프로젝트는 참여자 즉, 사람 중심(Human-Intensive)의 작업이기 때문에 작업의 일관성을 유지하기 어렵고, 소프트웨어의 비가시성이라는 특성 때문에 프로젝트 진행 중 발생하는 결함에 대해 파악하기 힘든 어려움을 내재하고 있다. 이러한 문제들은 소프트웨어 프로젝트 실패의 주요 원인이 되어왔다.

소프트웨어 공학 분야에서는 프로젝트의 성공을 위해 PMP(Project Management Professional)와 같은 프로젝트 관리 이론을 채택하고, 객체 지향 방법론, 프로덕트 라인 등의 다양한 방법론과, CMMI(Capability Maturity Model Integration) 같은 프로세스 모델을 개발하는 등의 노력을 기울여 왔다. 또한 개발/관리/지원 프로세스를 원활하게 구축하고 유지하기 위해 도구들을 개발하고 이를 적용하고 있다[1].

시스템에서 소프트웨어가 차지하는 비율이 높아지고, 그 규모가 점차적으로 커지게 되면서, 대부분의 프로젝트는 여러 참여자들이 함께 소프트웨어를 개발하게 되었다. 프로젝트의 최종 산출물인 소프트웨어 시스템은 설계 과정에서 여러 개의 서브 시스템(Sub System)으로 나누어지고, 각 서브 시스템들은 여러 개발 팀에게 할당되어, 각자 맡은 서브 시스템을 개발하는 협업 환경이 이루어졌다. 각 팀이 개발한 서브 시스템들은 최종산출물을 만들기 위해서, 적절한 시기에 통합(Integration)이 되고 전체적으로 테스트해야 한다. 일반적으로 이 통합 과정에서 인터페이스 관련 오류 등 개별적 구현에서는 파악되지 못한 에러들이 발생하게 되는데, 이러한 통합 관련 결함들이 프로젝트 후반부에 발견되면 시간은 부족하고 수정해야 하는 양은 많아져 결국 프로젝트 실패로 이어지는 경우가 많았다. 그로 인해 통합 활동이 소프트웨어 개발에서 성공과 실패를 가르는 중요한 활동이 되었다.

이러한 문제점을 극복하기 위해서, 프로젝트의 초기 단계부터 지속적으로 통합 작업을 수행하여 문제점을 조기에 발견하는 지속적 통합(Continuous Integration)이 제안되었고, 그 효용성이 입증되었다. 대부분의 조직들이 이를 Best practice로 채택하고 프로젝트에 적용하고 있다[2].

소프트웨어 프로젝트 성공의 중요한 활동이라고 인정받고 있는 지속적 통합을 수행하기 위해서는 다음과 같은 작업들을 프로세스 내 활동으로 구성해야 한다[3].

- 각 개별 팀이 개발한 모듈들의 최신 버전을 공용 저장소에 저장한다.
- 지속적으로 특정 시간에 공용 저장소 내의 모듈들을 자동으로 빌드(Software Build)한다.
- 결함을 조기에 발견하기 위한 활동으로 정적 분석을 수행한다.
- 소스 코드를 검증하기 위해 단위 테스트와 코드 커버리지 분석을 수행한다.
- 프로젝트 산출물 간의 추적을 유지한다.
- 매일 빌드 결과에 대해 확인한다.

지속적 통합을 위해 필요한 활동들은 특정 담당자의 업무로 할당하기에 잦은 반복 작업들로 이루어져 있고, 또한 각 활동의 수행 시간이 오래 소요되기 때문에 활동에 적합한 지원 도구들을 활용하는 것이 일반적이다. 지속적 통합을 위해 일반적으로 정형화된 활동과 도구들을 정리해보면, 다음과 같다.

- 지속적 통합 빌드 프로세스와 지원 도구
  - 지속적 통합 빌드 프로세스 정의
  - 활동과 지원 도구를 포함한 구축 프로세스 정의
  - 지속적 통합 빌드 환경 구축
  - 구축 환경과의 연동
  - 지속적 통합 빌드 결과 확인

- 정적 분석 프로세스와 지원 도구
  - 정적 분석 프로세스 정의

  - 활동과 지원 도구를 포함한 구축 프로세스 정의
  - 정적 분석 환경 구축
  - 구축 환경과의 연동
  - 정적 분석 결과 확인을 통한 소스 코드 품질 수준 파악

- 단위 테스트 프로세스와 지원 도구
  - 단위 테스트 프로세스 정의
  - 활동과 지원 도구를 포함한 구축 프로세스 정의
  - 단위 테스트 환경 구축
  - 구축 환경과의 연동
  - 단위 테스트 수행 및 코드 커버리지 확인을 통한 소스 코드 검증

소규모 조직에서도 소프트웨어 프로젝트의 성공을 위해서는 지속적 통합을 프로세스로 내재화하는 것이 필요하다. 하지만, 위에서 설명한 지속적 통합 활동들은 정의와 구축부터 적용까지의 과정에 전문성을 요구하고, 처음 시도하게 되면 상당히 많은 시행착오를 겪어야 하기 때문에 이를 지원할 수 있는 방안이 요구된다.
소규모 조직의 지원 방안으로 상용 솔루션을 활용하는 방법이 있다. 상용 솔루션은 비용이 높아서 문제이기도 하지만, 아직 한 회사의 솔루션만으로는 지속적 통합의 모든 활동들을 지원하지 못하기 때문에 여러 회사의 도구들을 사용해야 하는데, 도구 간의 연동도 쉽지가 않다. 이에 대한 대안으로 도구 구축과 함께 컨설팅을 받는 방법이 있다. 이 방법의 장점은 환경 구축 및 도구 간 연동까지 한꺼번에 제공되기 때문에 단시간 내 간편하게 지속적 통합 프로세스 적용을 할 수 있다는 점이다. 하지만 고가의 비용을 요구하기 때문에 소규모 조직에게는 비용적으로 큰 부담이 된다.
소규모 조직의 특성을 고려해보면 오픈소스 도구를 사용하는 방법이 가장 적합하다. 하지만 이 방법은 적합한 도구의 선정, 도구에 대한 학습, 도구 간 연계 방안 파악 등

의 많은 어려움이 있다.

도움을 얻기 위해, 정보통신산업진흥원의 소프트웨어개발 품질관리 매뉴얼[4]과 같은 관련 자료를 참고하거나 혹은 오픈소스 도구 관련 정보를 인터넷 검색하여 도구에 관한 정보 혹은 구축 방법에 대해 파악할 수는 있지만 지속적 통합 프로세스를 구축하기 위한 활동의 순서나 프로세스 정의 방법 등에 대한 정보를 얻기가 어렵다.

그러므로 소규모 조직의 지속적 통합 환경 구축 및 프로세스 적용을 위해서는 이를 위한 특정 가이드라인이 필요하다.

따라서 본 논문에서는 다음과 같이 소규모 조직의 특성을 반영하여 가급적 구입 및 유지 보수에 시간이 적게 드는 방안 기준으로 도구를 선정하고, 소규모 조직이 사용하기 쉽도록 프로세스를 제공하고자 한다.

- 오픈소스 도구
- 릴리즈가 1년 기준 3회 이상, 최근 3년 내에 이루어진 도구
- 다른 도구와의 연동을 위한 지원 플러그인이 제공되는 도구

본 논문에서 위의 조건을 기준으로 선정한 3분야, 7개의 도구는 다음과 같다.

- 지속적 통합 빌드를 위한 지원 도구 : Jenkins, Maven
- 정적 분석을 위한 지원 도구 : PMD, JDepend, JavaNCSS
- 단위 테스트를 위한 지원 도구 : Junit, Cobertura

본 논문은 다음과 같은 구성으로 지속적 통합 프로세스 정의를 작성한다.

제 2 장, 관련 연구에서는 지속적 통합을 구성하는 지속적 통합 빌드, 정적 분석, 단위 테스트의 프로세스와 지원 도구에 대해 연구하고, 소규모 조직의 프로젝트의 특성과 지속적 통합 구축이 어려운 이유에 대해 기술한다.

제 3장, 본 논문에서 제안하는 소규모 조직을 위한 오픈소스 기반 지속적 통합 프로세스에 대해 기술한다.

제 4장, 적용 및 검증에서는 제 3장에서 살펴본 프로세스의 적용을 통하여 그 효용성을 검증한다.

제 5장, 결론 및 향후 연구에서는 본 논문에서 제안한 오픈소스 기반 지속적 통합 프로세스의 연구 효과와 향후 연구 내용을 다룬다.

# 2. 관련 연구

## 2.1 지속적 통합

지속적 통합은 개발이 시작되면, 지속적으로 각 개발팀들의 작업들을 통합하여 검증하는 방식으로, 인터페이스 등에서 발생할 수 있는 잠재적 결함을 조기에 발견하기 위한 활동이다[5]. 기술적으로는 여러 개발팀들이 주기적으로 공유 저장소에 넣어둔 소스 코드들을 가져와서 통합하고 빌드하고 검증하는 활동을 말한다. 이 때, 빌드란 소스 코드 파일을 한 곳에 모아 작동할 수 있는 소프트웨어로 변환하는 과정을 말하기도 하고, 변환된 결과물 자체를 일컫기도 한다[6]. 지속적 통합은 크게 빌드와 검증으로 이루어지는데 이를 구체적인 활동으로 나누면 다음과 같은 활동들을 포함한다[7].

- 주기적인 소프트웨어 빌드
- 정적 분석을 통한 빌드된 코드의 품질 검증
- 테스팅
- 빌드 후 절차 자동화

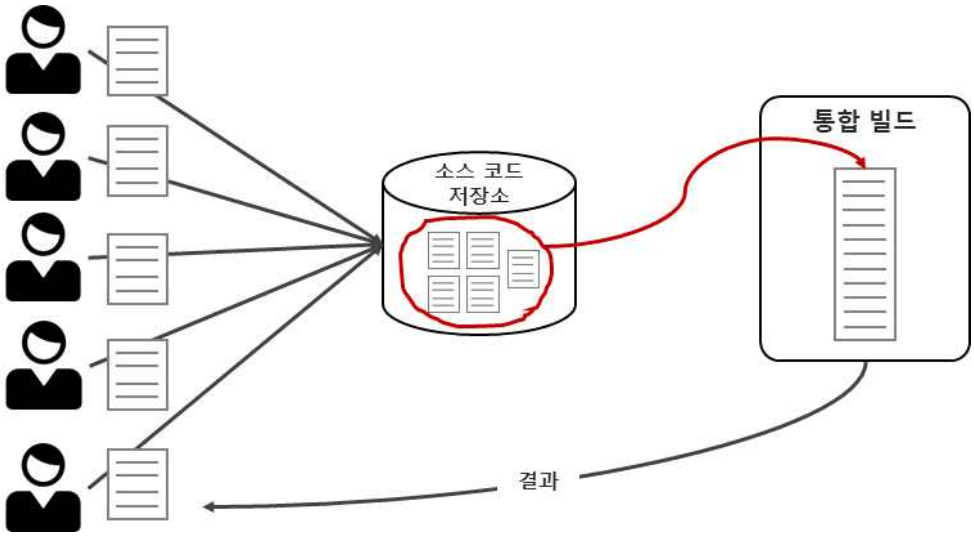
지속적 통합을 프로젝트에 효과적으로 구현하기 위해서는 도구의 지원을 받아 자동화 환경을 구축해야 한다. 일단 도구 기반 환경이 구축되면, 지정된 시간에 자동으로 소프트웨어 빌드를 수행하고, 새로운 코드 혹은 수정된 코드가 있을 경우 지속적으로 테스트를 수행하고 코드 커버리지도 확인한다. 또한 정적 분석을 통하여 코드 자체에 대한 품질도 검증한다.

일반적으로 지속적 통합을 구성하는 활동과 지원도구들을 살펴보면 다음과 같다.

**(1) 지속적 통합 빌드와 지원 도구**

지속적 통합 빌드는 각 개발자들이 작업한 소스 코드를 공용 저장소에 저장하면 이

들을 정기적으로 통합하고, 통합된 소스 코드에 대해 빌드를 수행하는 것이다. 지속적 통합 빌드를 수행하는 원리는 [그림 1]과 같다.




[그림 1] 지속적 통합 빌드 프로세스

지속적 통합 빌드를 지원하는 도구는 CI(지속적 통합) 도구와 빌드 도구가 있다. CI 도구는 지정된 시간에 정기적으로 소스 코드 저장소의 통합 소스 코드를 불러와서, 빌드 도구를 이용하여 빌드를 수행하고 그 결과를 보여주는 작업을 수행한다.

(가) CI 도구

- Bamboo

Bamboo는 Atlassian사에서 개발한 상용 지속적 통합 도구이다[8]. Atlassian사에서 제공하는 이슈 관리 도구 JIRA, 형상 관리 도구 Bitbucket 등과 통합하여 사용이 가능하므로 전체 추적이 가능하다. 하지만, 상용 도구이므로 고가의 비용이 든다는 단점이 있다.

- TeamCity

JetBrains에서 개발한 Java 기반 빌드 관리 및 지속적 통합을 위한 상용 도구이다[9]. 하지만, 상용 도구라 고가의 비용이 든다는 단점이 있다.

- CruiseControl

자바 기반의 오픈소스 지속적 통합 도구이다[10]. 이 도구에서는 빌드 루프(build loop)라는 기능을 제공하여 주요 모듈 빌드를 주기적으로 실행하고 사용자에게 결과를 통보하고 있다. 하지만 최근 릴리즈가 2010년에 이루어지고, 릴리즈 간 간격이 넓어 도구에 대한 신뢰가 떨어진다는 단점이 있다.

- Jenkins

오픈소스 기반 지속적 통합 도구이다[11]. Jenkins는 웹 기반으로 제공되고 있어 사용하기 쉬운 UI가 제공된다. 릴리즈가 1년에 약 50회 이상 이루어지는 개선 및 발전이 빠른 도구이다. 또한, 지원하는 플러그인의 개수가 총 1,763개로, 설치를 통해 다양한 기능을 덧붙여 사용할 수 있으며 도구와의 연동도 쉽게 가능하다.

(나) 빌드 도구

- Ant

Ant는 Apache 소프트웨어 재단에서 개발한 Task 중심의 소프트웨어 빌드 자동화 도구이다[12]. Task를 자바로 작성하고 이를 확장해 XML에서도 사용할 수 있어, 유연하다는 장점이 있다. 정적 분석 도구와 테스트 도구와의 연동을 통해 실행할 수 있다. 하지만, 최신 버전에 제공되지 않아 최근 프로젝트에 적용은 어렵다.

- Gradle

Gradle은 Ant의 Task 개념과 Maven의 특성인 의존성 관리를 혼합한 빌드를 제공

한다[13]. Groovy에 기반한 DSL(Domain Specific Language)을 통해 모든 종류의 빌드 선언적 방식 처리, 초기값 제공 등 편리한 적용이 가능하다. 또한, 안드로이드 앱을 만드는데 필요한 안드로이드 스튜디오의 공식 빌드 시스템이다. 하지만, JavaNCSS 등 정적 분석 도구 플러그인 설치에 대한 지원이 완전하게 이루어지지 않아 정적 분석 도구와의 연동이 어려운 점이 있다.

- Maven

Maven은 오픈소스 기반 빌드 도구이다[14]. 이 도구는 빌드 뿐만 아니라, 프로젝트의 라이센스, 개발자 정보, 의존성 있는 다른 프로젝트에 대한 속성 등을 기술함으로써 프로젝트 정보를 명세할 수 있는 기능을 제공한다. 또한, 원격 저장소를 활용하는 프로젝트에 필요한 라이브러리를 POM(Project Object Model) 파일 만으로 쉽게 설치 및 구성하여 사용하는 장점이 있다. 또한, POM 파일을 통해 Eclipse와 같은 IDE와의 연동을 가능하게 하여 각 개발자들의 프로젝트 파일 구조를 표준화하여 사용할 수 있도록 제공한다.

### 2) 정적 분석 프로세스와 지원 도구

정적 분석(Static analysis)은 소프트웨어를 실행시키지 않고 소스 코드자체를 분석하여 이상 여부를 찾아내는 것이다[15].

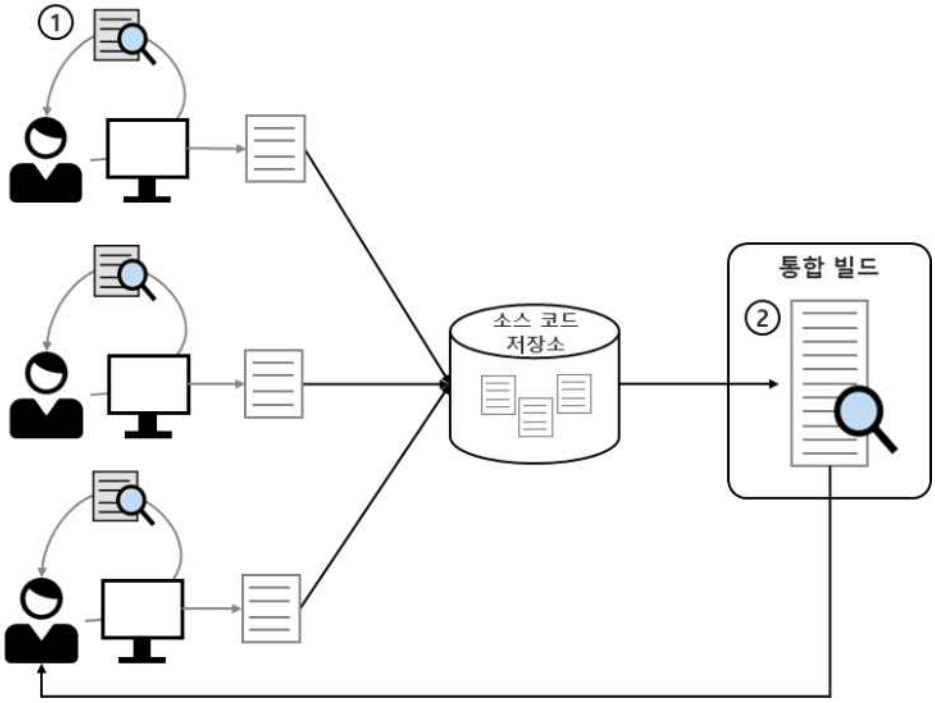
정적 분석을 통해 다음과 같은 효과를 얻을 수 있다.

- 테스트 전 조기 결함 발견
- 복잡도 등 수치를 계산하여 코드 조기 점검
- 동적 분석으로는 발견하기 어려운 결함 발견
- 소프트웨어의 의존도와 불일치성 발견
- 코드와 설계의 유지보수성 향상

또한 정적 분석을 통해 발견하는 전형적인 결함은 다음과 같다.

- 정의되지 않는 값으로 변수 참조
- 사용되지 않은 변수
- 사용되지 않는 코드(Dead Code)
- 지나치게 복잡한 구조
- 코딩 표준 위반
- 보안 취약성

정적 분석은 [그림 2]와 같이 1) 개발 중 소스 코드 규칙 위반 여부나 코딩 표준을 준수하는 지에 대해 확인하고 반영하기 위한 용도와, 2) 각 개발자들의 통합 소스 코드에 대한 정적 분석 결과를 확인하기 위한 용도로 수행한다.




[그림 2] 정적 분석 프로세스

정적 분석은 수작업을 통해 수행하기는 복잡하고 시간이 많이 소요되기 때문에 도구를 사용하여 수행하여야 한다. 정적 분석을 지원하기 위한 도구는 다음과 같이 상용 도구와 오픈소스 도구로 나눌 수 있다.

(가) 상용도구

- Sparrow

파수에서 개발한 규칙 기반 정적 분석 도구이다[16]. 구문 분석과 함께 시큐어 코딩을 위한 소스 코드 보안 약점 분석까지 제공하고 있다. 검출 뿐만 아니라 검출된 결함을 수정하기 위한 수정 가이드를 제공하는 등 차별된 기능들을 제공한다.

- CodeInspector

슈어소프트테크에서 개발한 규칙 기반 정적 분석 도구이다[17]. 자동차, 철도, 항공 등 각 도메인별로 준수해야 하는 코딩 규칙을 자동으로 검사한다. 또한, 소스 코드의 위배 사항을 자동으로 수정함으로써 소프트웨어 품질 개선 작업의 생산성을 20% 향상시켜 준다는 장점이 있다.

- Coverity

Synopsys에서 개발한 Java 소스 코드의 규칙 기반 정적 분석 도구이다[18]. 보안 룰 검증을 위해서도 사용할 수 있다.

- JArchitect

Codergears에서 개발한 정적 분석 도구이다[19]. 의존성과 순환 복잡도에 대해 분석하고 코드 기반 스냅샷 비교와 아키텍처 및 품질 규칙 검증을 수행한다.

(나) 오픈소스 도구

- PMD

PMD는 총 2가지 기능을 제공한다[20]. 첫 번째로 사전에 정의된 약 300개의 규칙을 기반으로 소스 코드를 검증한다. 대표적으로 검출되는 규칙 위반은 선언 후 사용하지 않는 변수, 비어 있는 try, catch, switch문, 너무 많은 메소드를 가진 클래스 등이 있다. 두 번째로 중복 코드를 찾아주는 것이다. 코드 내 특정 수 이상이 동일한 부분을 검출해내는 것이다. 이를 검출하여 수정함으로써 추후 유지보수가 좀 더 쉽게 이루어질 수 있도록 한다.

- Findbugs

Bill Pugh에 의해 개발된 규칙 기반 정적 분석 도구이다[21]. 발견한 결함을

scariest, scary, troubling, of concern의 4가지의 단계로 분류한다.

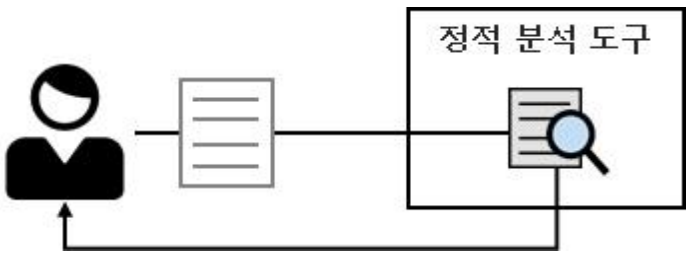
- JavaNCSS

소스 코드의 전체 패키지, 클래스, 메소드 단위를 주석을 제외한 총 코드의 라인 수를 알려준다[19]. 또한, 각 메소드의 순환 복잡도를 알려준다.

- JDepend

자바 패키지의 의존성을 분석하는 도구이다[19]. 이를 통해 패키지의 추상/구체 정도, 균형 정도, 원형 의존성에 대한 파악을 함으로써 개발자가 좀 더 나은 설계를 할 수 있도록 도움을 준다. 제공되는 의존성 지표를 통해 확장성, 재사용성, 유지보수성을 향상시켜 패키지의 품질 향상에 도움을 준다. 이 도구의 주 사용 목적은 패키지 간의 의존성 분석을 하는 것이기 때문에, 전체 소스 코드를 통합 빌드할 때 함께 전체 의존성을 분석하는 방법이 가장 일반적이다.

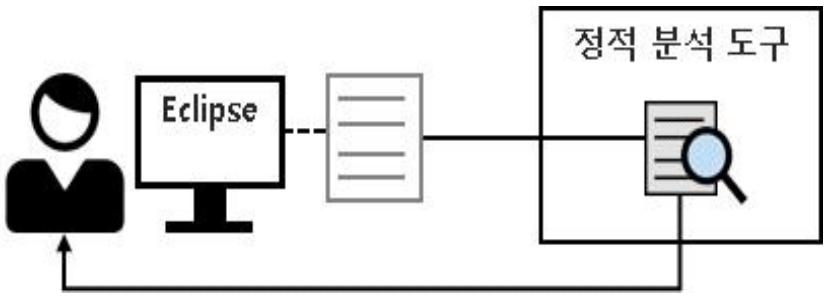
정적 분석 도구를 사용하는 방법은 총 3가지로 이루어져 있다.




[그림 3] 정적 분석 도구 사용 방법 - 1) 독립적 사용

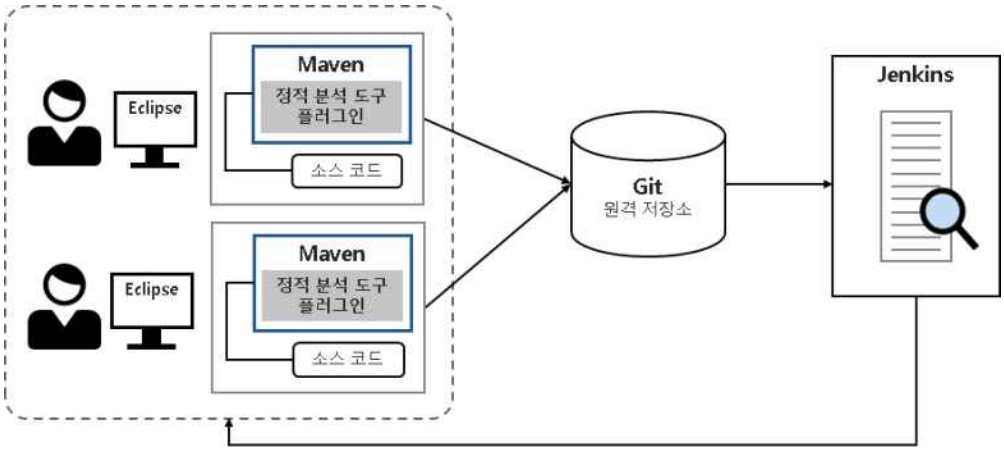
첫 번째로는 [그림 3]과 같이 단순히 정적 분석 도구만을 사용하여 소스 코드를 분석하는 방법이다. 배포되는 정적 분석 도구를 받아 프로그램 실행 혹은 명령어 실행을 통하여 개발자의 소스 코드를 실행하여 도출되는 결과를 확인한다. 정적 분석 도

구를 실행하는 해당 환경의 저장소의 소스 코드에 대하여 분석한 결과만 확인할 수 있다. 이 방법은 별도로 명령어 실행 혹은 프로그램을 실행하고, 생성된 결과물의 위치를 찾아 확인하여야 하기 때문에 권장하지 않는 방법이다.




[그림 4] 정적 분석 도구 사용 방법 - 2) Eclipse 연결해서 사용

두 번째는 [그림 4]와 같이 Eclipse에서 정적 분석 도구 플러그인을 설치하는 방법이다. 개발자들은 Eclipse를 통해 개발 중인 소스 코드의 정적 분석 결과에 대해 확인할 수 있다. 개발 중 소스 코드 규칙 위반 여부나 코딩 표준을 준수하는 지에 대해 확인하고 반영하기 위해 사용한다. 분석 결과를 Eclipse를 통해 바로 확인할 수 있기 때문에 첫 번째 방법에 비해 좀 더 효율적이다.




[그림 5] 정적 분석 도구 사용 방법 - 3) Maven 연결해서 사용

세 번째는 [그림 5]와 같이 Maven에 정적 분석 도구 플러그인을 설치하여 지속적 통합 시 실행하는 것이다. 이 방법을 통해 각 개발자들의 소스 코드들을 통합 빌드할 때 함께 정적 분석을 함으로써 개발팀 전체 혹은 QA가 통합된 소스 코드의 정적 분석 결과에 대해 파악할 수 있다.

정적 분석은 개발자들이 직접 Eclipse를 통해 개발하는 중에 수행하여 자신의 소스 코드에 대한 분석 결과를 확인하고 반영하여 조기 에러를 예방할 수 있는 두 번째 방법이 가장 효과적이다.

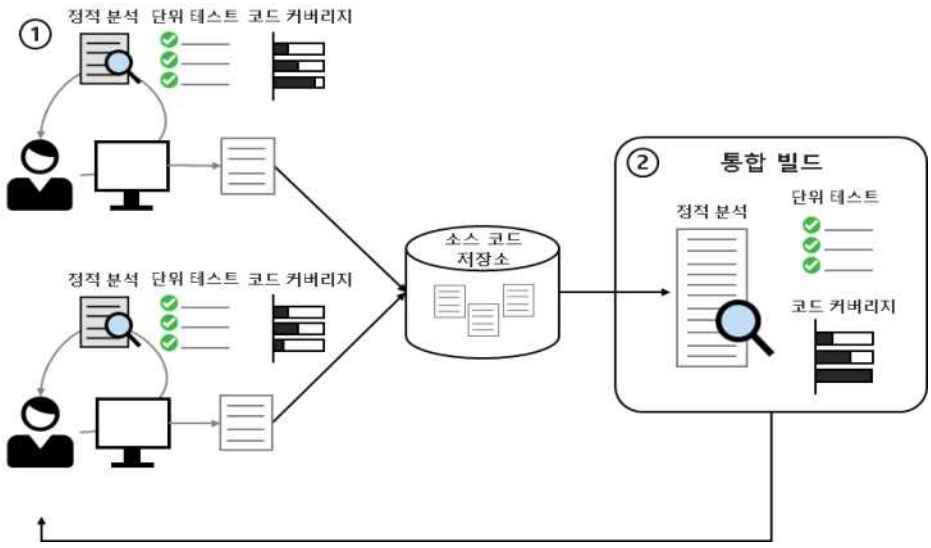
### 3) 단위 테스트 프로세스와 지원 도구

테스트란 개발된 프로그램이 요구하는 기능을 만족하는지의 여부를 확인하기 위해 사용하는 방법이다. 또한 에러가 없다는 것을 보여주는 것이 아니라 결함이 존재한다는 것을 보여줌으로써 제품을 출시하기 전 결함을 수정할 기회를 마련하는 것이 목표이다[22].

본 논문에서 다루는 단위 테스트란 테스트를 위한 코드를 개발자가 직접 작성하여 해당 코드를 실행함으로써 테스트 대상이 되는 코드에 대한 특정 영역을 실행해보는

테스트 방법이다. 개발자는 단위 테스트를 통해 결함을 발견할 때마다 바로 수정하고, 결함에 대한 기록 과정은 생략하는 방식으로 수행한다. 단위 테스트 수행 후 코드 커버리지(Code Coverage)를 확인함으로써 전체 코드의 어느 정도를 테스트했는지 확인할 수 있다. 코드 커버리지란, 소프트웨어의 테스트를 논할 때 얼마나 테스트가 충분한가를 나타내는 지표 중 하나이다. 소프트웨어 테스트를 진행했을 때 코드 자체가 얼마나 실행되었냐는 것이다[22]. 기본적으로 커버리지의 측정 기준은 구문(Statement), 조건(Condition), 결정(Decision)이다. 먼저 구문 커버리지는 프로그램을 구성하는 문장들이 최소한 한 번 이상 실행되면 충족된다. 조건 커버리지는 분기의 각 내부 조건이 참/거짓을 가지면 충족된다. 결정 커버리지는 각 분기의 내부 조건 자체가 아닌 전체 결과가 참/거짓이면 충족된다.

단위 테스트와 코드 커버리지 확인은 다음 그림과 같이 2가지의 용도로 수행된다.




[그림 6] 단위 테스트 프로세스

1) 개발 중 개발자들이 소스 코드에 대한 검증을 하고, 테스트의 충분함 여부를 확인하기 위한 용도와 2) 각 개발자들의 통합된 소스 코드에 대한 테스트 성공률과 코드

커버리지가 얼마인지와 목표를 달성했는지에 대한 여부를 파악하는 용도이다. 단위 테스트와 코드 커버리지 확인을 수행하는 활동은 시간이 많이 소요되기 때문에 보통 지원 도구를 통해 자동화하여 진행한다.

본 논문에서 단위 테스트를 수행하기 위해 단위 테스트 도구와 코드 커버리지 도구를 선정하였다. 단위 테스트 도구란 소스 코드의 메소드를 테스트하는 도구이다[23]. 매개변수와 예상 출력 값을 입력하여 실제 출력 값과 비교함으로써 성공/실패 여부를 도출하는 방식으로 구성되어 있다.

코드 커버리지 도구는 단위 테스트 도구를 통해 소스 코드의 얼마만큼을 테스트 했는지에 대해 계산해주는 도구이다. 이 도구를 실행함으로써 소스 코드가 얼마나 테스트되었고 어느 부분을 더 테스트해야 하는지에 대해 파악할 수 있다. 따라서 단위 테스트 수행이 선행이 되어야 효과적으로 사용할 수 있다.

먼저, 대표적인 단위 테스트 도구는 다음과 같다. 해당 도구에 대한 자세한 정보는 <표 1>에 기재되어 있다. 단위 테스트 도구는 일반적으로 프레임워크 제공의 형태로 진행되므로 실행을 위하여 별도의 설치를 할 필요가 없다.

- Junit

오픈소스 기반 Java 소스 코드 단위 테스트 도구이다. 이 도구는 소스 코드의 '메소드'를 대상으로 테스트한다[23]. 이 도구는 Kent Beck과 Erich Gamma가 개발하였다. Github에서 무료로 제공되며, 라이센스는 Eclipse Public License를 사용한다. 이 도구는 꾸준하게 릴리즈가 업데이트되고 있다. 또한, 사용자를 많이 확보하여 관련 정보를 쉽게 참고할 수 있다는 장점이 있다.

- TestNG

Cedric Beust가 개발한 오픈소스 기반 Java 소스 코드 단위 테스트 도구이다[24]. 이 도구 또한 릴리즈가 최근까지 있어 도구에 대한 신뢰는 높을 수 있지만, 관련 정

보가 JUnit보다 적게 제공되고 있다.

<표 1> 단위 테스트 도구 정보

| | JUnit | TestNG |
|---|---|---|
| 개발자 | Kent Beck, Erich Gamma | Cedric Beust |
| 최근 릴리즈 | 5.0 (2017.09.10) | 6.11 (2017.2.28) |
| 운영 체제 | Cross-platform | Cross-platform |
| 라이센스 | Eclipse Public License | Apache License 2.0 |
| 관련 페이지 | junit.org | testng.org |

대표적인 코드 커버리지 도구는 다음과 같고, 해당 도구에 대한 자세한 정보는 <표 2>에 기재하였다.

- Clover

Atlassian에서 개발한 도구이다[25]. Atlassian에서 제공하는 도구들과 통합하여 사용이 가능하므로 전체 추적이 가능하다. 최근 오픈소스 도구로 제공이 되지만 사용자가 많지 않는 추세라 관련 정보가 많이 제공되지 않는다.

- Cobertura

오픈소스 기반 코드 커버리지 도구이다[23]. Steven에 의해 개발되었으며, 라이센스는 GPL(General Public License) 2.0 이다. 소스 코드를 기준으로 구문과 조건 커버리지를 측정한다. 또한, 부가적으로 순환 복잡도를 측정하여 제공한다. 또한, 플러그인 설치 횟수가 18000가 넘을 정도로 많은 사용자를 확보하고 있다.

- Emma

오픈소스 기반 코드 커버리지 도구이다[26]. Vlad에 의해 개발되었고, 라이센스는

CPL(Common Public License) 1.0이다. 이 도구는 릴리즈가 많지 않고, 관련 정보를 얻기가 어려워 사용에 불편한 점이 있다.

〈표 2〉 코드 커버리지 도구 정보

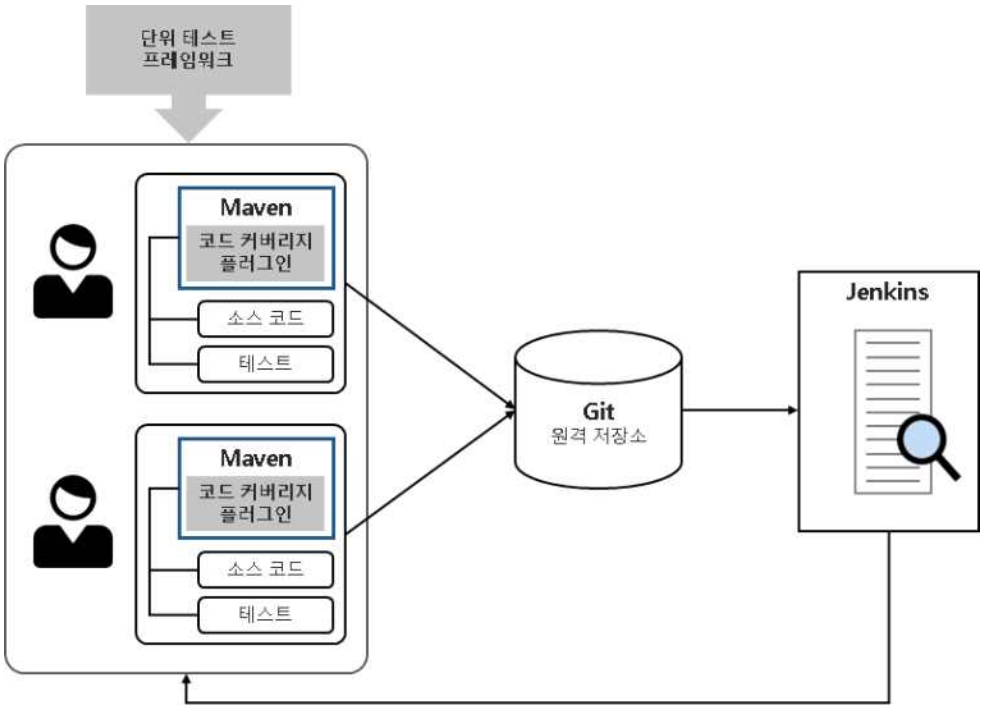
| | Clover | Cobertura | EMMA |
|---|---|---|---|
| 개발자 | Atlassian | Steven Christou | Vlad Roubtsov |
| 최근 릴리즈 | 4.1.2 2016.10.11 | 2.1.1 2015.2.26 | 2.1 2005.5.13 |
| 운영 체제 | Cross-platform | Cross-platform | Cross-platform |
| 라이센스 | Apache 2.0 | GPL 2.0 | CPL 1.0 |
| 관련 페이지 | www.Atlassian.com | github - cobertura | sourceforge - emma |

단위 테스트 도구와 코드 커버리지 도구를 사용하는 방법은 총 2가지로 이루어져 있다.


[그림 7] 단위 테스트 및 코드 커버리지 도구 사용 방법

- 1) Eclipse 연결해서 사용

첫 번째는 Eclipse에 코드 커버리지 도구 플러그인을 설치하는 방법이다. 개발자들은 Eclipse를 통해 개발 중인 소스 코드에 대해 단위 테스트 수행을 함으로써 소스 코드에 대한 결함 발견 및 수정을 통해 검증 작업을 한다. 그 후 단위 테스트가 얼마나 수행되었는지에 대해 파악하기 위해 코드 커버리지를 확인한다.




[그림 8] 단위 테스트 및 코드 커버리지 도구 사용 방법

- 2) Maven 연결해서 사용

두 번째는 Maven에 코드 커버리지 도구 플러그인을 설치하여 지속적 통합 빌드와 함께 실행을 하는 것이다. 이 방법을 통해 각 개발자들의 소스 코드들을 통합 빌드할 때 함께 단위 테스트 및 커버리지 분석을 함으로써 통합 소스 코드의 테스트 성공률 및 커버리지에 대해 파악할 수 있다. 이 수치들을 기반으로 목표를 달성했는가에 대한 여부를 알 수 있다.

## 2.2 소규모 조직의 소프트웨어 개발

### (1) 소규모 조직의 프로젝트 특성

소규모 조직의 프로젝트 특성은 다음과 같다[27].

- 재정(Finance)

소규모 조직은 경제적으로 취약하다. 따라서 수입과 지출 등 현금 유동성에 좌우되고, 프로젝트의 이익에 의존적이다. 따라서 예산 내에서 프로젝트를 수행해야 한다.

- 고객(Customer)

소규모 조직은 고객을 포함한 프로젝트 인원 사이의 관계가 아주 밀접하다. 따라서 프로젝트의 유동성도 매우 높은 편이다. 또한, 소프트웨어 개발은 강력하게 사람 중심의 작업이기 때문에 그들 사이의 의사소통이 아주 중요하다.

- 학습과 성장(Learning and Growth)

소규모 조직은 소프트웨어 프로세스 평가 및 개선에 대한 지식이 상대적으로 부족하다. 또한, 지식 습득과 학습 역량이 제한적인 면이 있어 새로운 지식을 적용하기까지의 시간이 오래 소요된다.

**(2) 소규모 조직에서 지속적 통합 구축의 어려움**

위의 특성들을 기반으로 고려해 볼 때, 소규모 조직이 자체적으로 지속적 통합을 구축하는 것은 쉬운 일이 아니라는 것을 알 수 있다.

일반적으로 소규모 조직에는 지속적 통합에 대한 전문적인 지식을 학습하고 시도할 시간과 인력이 부족한 것이 현실이라, 전문성을 확보하기가 힘들다. 이러한 부분을 보완하기 위해 상용 솔루션을 구입하거나, 전문 컨설팅 기업의 도움을 받을 수 있겠지만, 이 방안은 비용이 많이 들기 때문에 현실적이지 못한 면이 있다.

따라서 소규모 조직의 지속적 통합을 위해서는 오픈소스 도구 기반의 구체적인 지침과 예제를 통한 쉬운 설명이 필요하다. 이러한 점을 반영하여 본 논문에서는 소규모 조직이 쉽게 이해하고 적용하기 위한 지속적 통합 프로세스를 제안하고자 한다.

# 3. 소규모 조직을 위한 오픈소스 기반 지속적 통합 프로세스

본 논문에서 제안하는 지속적 통합 프로세스는 소규모 조직들이 상황에 맞춰서 점진적으로 구축할 수 있도록, <표 3>과 같이 기반 구축 단계와 3개의 단계로 나누어 제공한다.

기반 구축 단계에서 작업 산출물 관리를 위해 필수적으로 갖추어야 할 형상 관리 프로세스를 구축하고, 1단계에서는 지속적 통합 빌드 프로세스를 구축한다. 1단계를 완료하면, 실질적으로 협업 개발 환경에서 코드를 지속적으로 통합 빌드하는 환경이 갖추어진다. 2단계와 3단계는 검증 과정으로, 2단계에서는 정적 분석을, 3단계에서는 단위 테스트 프로세스를 추가로 구축한다.

<표 3> 소규모 조직을 위한 오픈소스 기반 지속적 통합 프로세스 구성

| 단계 | 수행 항목 | 활용 도구 |
|---|---|---|
| 기반 구축 | 소스 코드 형상 관리 프로세스 구축 | Git |
| 1 | 지속적 통합 빌드 프로세스 구축 | Maven, Jenkins |
| 2 | 정적 분석 프로세스 구축 및 지속적 통합 반영 | PMD, JDepend, JavaNCSS |
| 3 | 단위 테스트 프로세스 구축 및 지속적 통합 반영 | JUnit, Cobertura |

본 논문에서 오픈소스 도구를 선정한 기준은 서론에서 언급한 바와 같이 다음과 같다.

- 오픈소스 도구
- 릴리즈가 1년 기준 3회 이상, 최근 3년 내에 이루어진 도구
- 다른 도구와의 연동을 위한 지원 플러그인이 제공되는 도구

이 기준에서 추구하는 도구는 (1) 현재 가장 많이 사용되고 있고, 그러므로 (2) 다른 도구와의 연동 사례를 찾아볼 수 있으며, (3) 잦은 릴리즈로 구축과 유지 보수에 시간이 적게 드는 것들이다.
하지만, 개발 분야와 사용 언어 등에 따라 도구들이 다양하게 적용될 수 있기 때문에, 본 논문에서 제공하는 가이드라인은 다음 조건에 맞는 프로젝트들로 한정하였다.

- 개발 분야 : 소프트웨어 관련 개발 및 서비스
- 참여 인원 : 5명 이상 25명 이하
- 기간 : 대략 3~4개월
- 참여 역할 : 프로젝트 관리자, 아키텍처 담당자, 품질관리 담당자, 개발자
- 사용 언어 : Java
- IDE : Eclipse

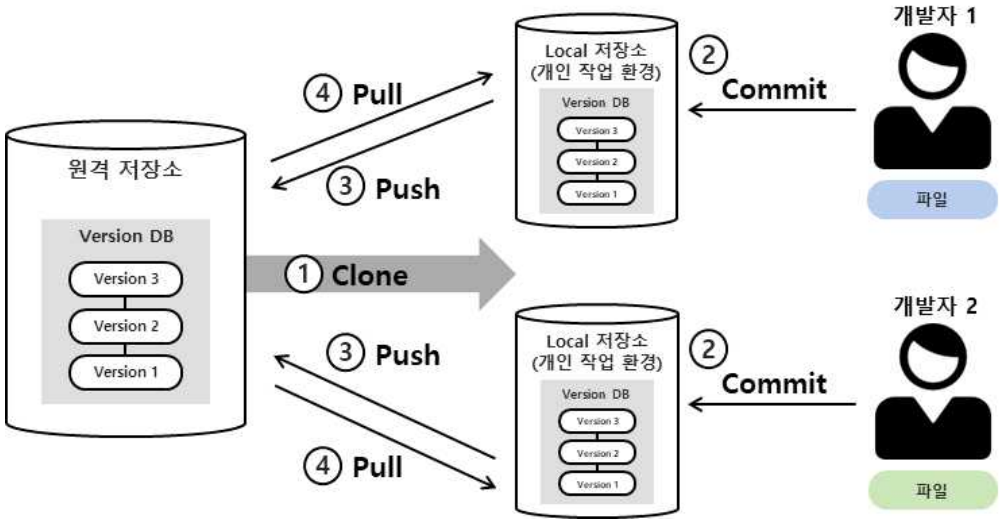
이를 기준으로 <표 3>과 같이 9개의 도구를 선정하였다.

## 3.1. 기반 구축 - 소스 코드 형상관리 프로세스 구축

프로젝트 주요 산출물의 관리를 담당하는 형상 관리 프로세스는 지속적 통합을 위해서도 기반 구축 프로세스로 정립되어 있어야 한다. 형상 관리 프로세스를 통해 개발자들의 소스 코드를 통합 저장소에서 효율적으로 관리하고 있어야 이를 기반으로 지속적 통합을 진행할 수 있기 때문이다.

형상 관리는 주요 산출물을 형상 항목으로 정하고, 산출물들의 변경 과정을 관리한다. 여러 사람이 개발하는 소스 코드에 대해서도 누가, 왜 소스 코드를 변경했는지에 대해 알 수 있도록 변경에 대한 이력을 관리한다. 이를 통해 실수가 있거나, 에러를 발견했을 때에는 언제든지 이전 버전으로 돌아갈 수 있다.

형상 관리 프로세스의 구현 형태는 [그림 9]와 같다.




[그림 9] 형상 관리 프로세스 구축

형상 관리 프로세스는 형상 관리 지원 도구 Git을 통해 쉽게 구현 될 수 있다.

소스 코드를 저장하는 저장소는 2가지로 분류된다. 각 개발자들이 직접 작업하고 관리하는 개인 작업 환경인 Local 저장소와, 각 개발자들의 Local 저장소의 내용을 한꺼번에 모아서 중앙에서 관리하는 원격 저장소가 있다.

1) 처음 각 개발자들은 각자 작업할 환경을 만들기 위해 원격 저장소 자체를 자신의 Local 환경으로 복사하게 된다. 이를 Git에서는 Clone라 칭한다. 그 후, 2) 개발자들은 자신의 저장소에서 개발을 하고, Commit이라는 기능을 사용하여 변경 사항을 Local 저장소에 저장한다. 3) 개인이 작업한 내용을 원격 저장소에 저장하기 위해서는 Push라는 기능을 사용하고, 4) 원격 저장소로부터 다른 개발자의 작업 내용을 반

영하기 위해서는 Pull이라는 기능을 사용한다. 이를 분산 버전 관리 방식이라 한다 [28]. Git을 통해 개발 초반부터 각 개발자들이 개발하는 소스 코드를 체계적으로 관리할 수 있다.

## 3.2. 1단계 - 지속적 통합 빌드 프로세스 구축

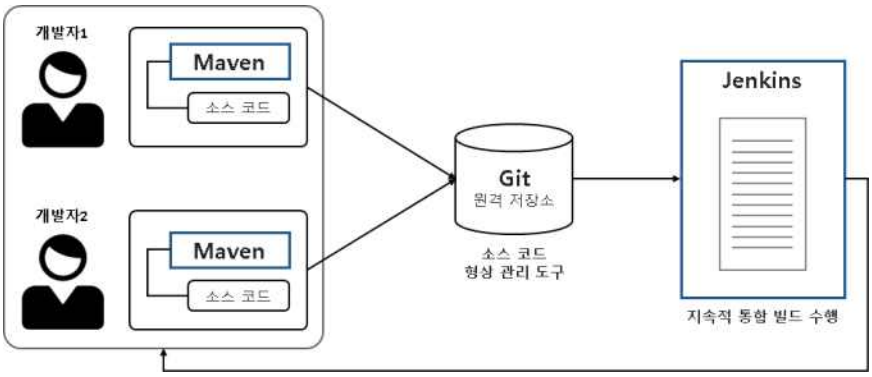
2.1.1에서 언급했듯이, 지속적 통합 빌드는 소프트웨어 통합에서 발생하는 요소를 개발 초기에 해결하기 위해 지속적으로 빌드하는 것이다.

본 논문에서는 지속적 통합 빌드를 효과적으로 수행하기 위해 앞서 서론에서 언급한 3가지의 조건을 기준으로 CI 도구 Jenkins와, 빌드 도구 Maven을 선정하였다.

[그림 10]과 같이 Maven은 정형화된 프로젝트 구조를 제공하여 모든 프로젝트의 정보를 기록할 수 있는 형식을 제공한다. 이 도구를 통해 모든 개발자들은 개발한 소스 코드를 동일한 모습으로 빌드할 수 있다. Maven을 통해 각 개발자는 동일한 프로젝트 구조에서 개발하고, Git을 통해 원격 저장소에 소스 코드를 한꺼번에 저장하고 관리한다.

원격 저장소에 저장된 소스 코드는 Jenkins와 Maven을 통해 매일 혹은 정기적으로 빌드하고, 개발자 혹은 QA에게 결과를 알려준다. 각 프로젝트마다 Jenkins에서는 Item 단위로 통합 소스 코드를 저장하여 빌드 결과를 시각화하여 내타낸다.

정기적으로 빌드가 수행될 때마다 자동화된 절차에 의해 통합 내용은 자동 검증 되며, 이로 인해 통합 시 발생하는 에러를 조기에 발견할 수 있다.




[그림 10] 지속적 통합 빌드 프로세스 구축

### (1) 목적

지속적으로 개발자들의 소스 코드를 통합하여 매일 빌드하여 결과를 알려준다.

### (2) 역할과 책임

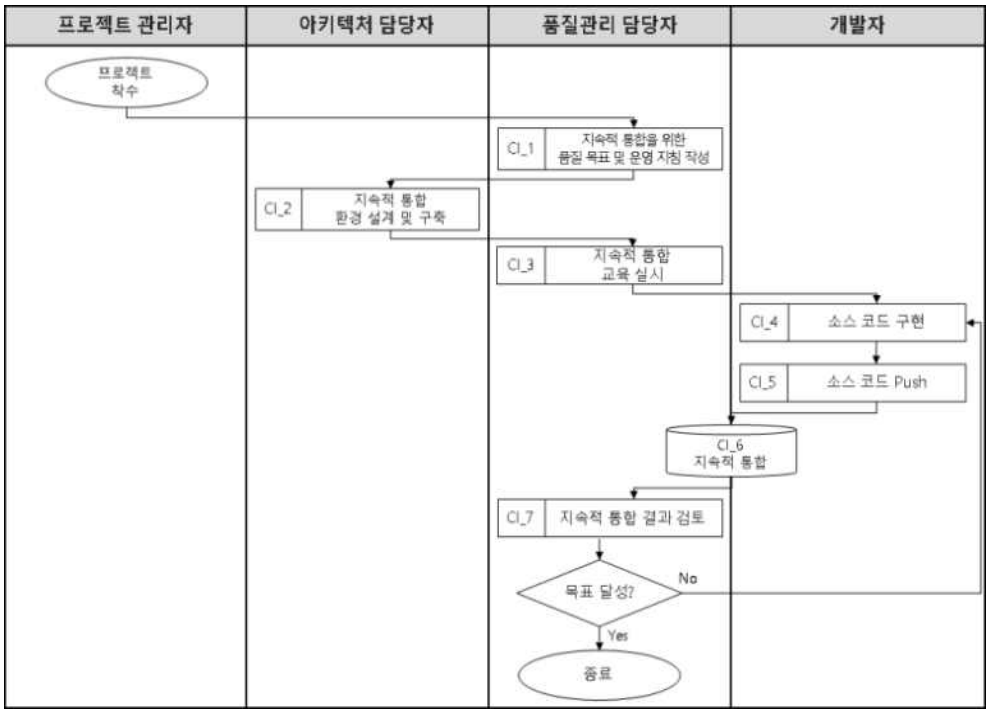
체계적인 지속적 통합 빌드의 진행을 위해서는 각 구성원의 역할(Role)과 책임(Responsibility)이 정의되고 이행되어야 한다. 따라서 현재 프로젝트의 환경에 대해 파악하여 <표 4>와 같이 구성원의 역할을 구성하여 기반 환경을 마련하고 교육을 진행해야 한다.

<표 4> 구성원 역할 및 책임

| 역할(Role) | 책임(Responsibility) |
|---|---|
| 프로젝트 관리자 (Project Manager) | - 프로젝트의 계획과 실행에 대해 총괄책임<br>- 각 역할에 대한 담당자 지정 |
| 아키텍처 담당자 (Application Architect) | - 요구사항을 충족하는 SW 아키텍처 설계 및 검증<br>- 아키텍처에 따른 상세설계 및 구현의 적절성 확인 |
| 품질관리 담당자 (Quality Assurance) | - 비즈니스 목표/요구사항에 따라 품질 목표 수립<br>- 품질보증 계획 수립, 이행을 위한 준비, 교육 제공<br>- 통합 빌드의 성공 여부 모니터링 및 결과 공지 |
| 개발자 (Developer) | - 정의된 요구사항, 설계에 따라 코드 구현 |

### (3) 절차

#### 1) Swim Lane 다이어그램




[그림 11] 지속적 통합 빌드 프로세스 다이어그램

### 2) 활동 정의

지속적 통합 빌드 프로세스에서 수행하는 활동에 대해 간략하게 설명하자면 <표 5>와 같다.

〈표 5〉 지속적 통합 빌드 프로세스 활동 정의

| ID | 활동 명 | 상세 활동 설명 |
|---|---|---|
| CI_1 | 지속적 통합을 위한 품질 목표 및 운영 지침 작성 | - 품질관리 담당자는 지속적 통합의 품질 목표 및 운영 지침에 관해 작성한다. |
| CI_2 | 지속적 통합 환경 설계 및 구축 | - 아키텍처 담당자는 지속적 통합 환경을 설계하고 구축한다. |
| CI_3 | 지속적 통합 교육 실시 | - 품질관리 담당자는 교육 대상을 선정하고, 지속적 통합의 목표와 운영 지침 및 도구 사용 가이드에 대한 교육을 한다. |
| CI_4 | 소스 코드 구현 | - 개발자는 Eclipse를 통해 개발을 진행한다. |
| CI_5 | 소스 코드 Push | - 개발자는 운영 지침의 조건을 만족하는 소스 코드를 Git을 통해 Push한다. |
| CI_6 | 지속적 통합 | - Jenkins는 지정된 시간마다 통합된 소스 코드에 대한 빌드를 수행하고, 결과를 알려준다. |
| CI_7 | 지속적 통합 결과 검토 | - 품질관리 담당자는 매일 빌드 결과에 대해 확인하고, 결과에 대해 공지한다. |

각 활동들에 대해 정의를 하면 다음과 같다.

〈표 6〉 ‘지속적 통합을 위한 품질 목표 및 운영 지침 작성’ 정의

<table>
<tr><th colspan="2">ID</th><th colspan="2">활동 명</th></tr>
<tr><td colspan="2">CI_1</td><td colspan="2">지속적 통합을 위한 품질 목표 및 운영 지침 작성</td></tr>
<tr><th colspan="4">정의</th></tr>
<tr><td colspan="4">품질관리 담당자는 지속적 통합의 품질 목표 및 운영 지침에 관해 작성한다.</td></tr>
<tr><td>착수 기준</td><td colspan="3">1. 프로젝트에 착수하였다.</td></tr>
<tr><td>종료 기준</td><td colspan="3">1. 품질관리 담당자가 지속적 통합을 위한 품질 목표 및 운영 지침을 작성하였다.</td></tr>
<tr><th colspan="3">주요 활동</th><th>작업자</th></tr>
<tr><td colspan="3">1. 품질관리 담당자는 프로젝트의 정보에 대해 파악한다.<br>- 프로젝트 개발 언어<br>- 프로젝트의 규모<br>- 프로젝트 기간<br>- 프로젝트 참여 인원<br><br>2. 품질관리 담당자는 프로젝트의 지속적 통합을 위한 품질 목표 및 운영 지침을 작성한다.<br>- 지속적 통합의 목표<br>- Git 사용 규칙 작성<br>- 통합 빌드 성공 요건 선정<br>- 통합 빌드 실패 시 담당자별 행동 요령</td><td>품 질 관 리 담당자</td></tr>
<tr><th colspan="2">입력물</th><th colspan="2">출력물</th></tr>
<tr><td colspan="2">프로젝트의 정보</td><td colspan="2">지속적 통합을 위한 품질 목표 및 운영 지침</td></tr>
</table>

<표 7> ‘지속적 통합 환경 설계 및 구축’ 정의

<table>
<tr><th>ID</th><th colspan="2">활동 명</th></tr>
<tr><td>CI_2</td><td colspan="2">지속적 통합 환경 설계 및 구축</td></tr>
<tr><th colspan="3">정의</th></tr>
<tr><td colspan="3">아키텍처 담당자는 지속적 통합을 위한 품질 목표 및 운영 지침을 참고하여 프로젝트에 적합하도록 지속적 통합 환경을 설계하고, 그에 맞게 구축한다.</td></tr>
<tr><td>착수 기준</td><td colspan="2">1. 품질관리 담당자가 지속적 통합을 위한 품질 목표 및 운영 지침을 작성하였다.</td></tr>
<tr><td>종료 기준</td><td colspan="2">1. 아키텍처 담당자가 지속적 통합 환경을 구축하였다.</td></tr>
<tr><th colspan="2">주요 활동</th><th>작업자</th></tr>
<tr><td colspan="2">1. 아키텍처 담당자는 프로젝트의 환경과 목표에 걸맞은 지속적 통합 환경을 설계한다.<br>- 빌드 도구 선정<br>- CI 도구 선정<br><br>2. 아키텍처 담당자는 설계를 기반으로 지속적 통합 환경을 구축한다.<br>- Maven 환경 구축<br>- Jenkins 환경 구축<br>- 도구 간 연동</td><td>아키텍처 담당자</td></tr>
<tr><th>입력물</th><th colspan="2">출력물</th></tr>
<tr><td>지속적 통합을 위한 품질 목표 및 운영 지침</td><td colspan="2">지속적 통합 환경 설계도<br>지속적 통합 환경</td></tr>
</table>

〈표 8〉 ‘지속적 통합 교육 실시’ 정의

<table>
<tr><th colspan="2">ID</th><th colspan="2">활동 명</th></tr>
<tr><td colspan="2">CI_3</td><td colspan="2">지속적 통합 교육 실시</td></tr>
<tr><th colspan="4">정의</th></tr>
<tr><td colspan="4">품질관리 담당자는 교육 대상을 선정하고, 지속적 통합의 품질 목표와 운영 지침 및 사용 가이드에 대한 교육을 한다.</td></tr>
<tr><td>착수 기준</td><td colspan="3">1. 품질관리 담당자가 지속적 통합을 위한 품질 목표 및 운영 지침을 작성하였다.<br>2. 아키텍처 담당자가 지속적 통합 환경을 구축하였다.</td></tr>
<tr><td>종료 기준</td><td colspan="3">1. 품질관리 담당자가 지속적 통합 교육을 실시하였다.</td></tr>
<tr><th colspan="3">주요 활동</th><th>작업자</th></tr>
<tr><td colspan="3">1. 품질관리 담당자는 교육 대상을 선정한다.<br>- 개발자<br><br>2. 품질관리 담당자는 교육을 실시한다.<br>- 지속적 통합 도구 사용 가이드<br>- 빌드 도구 사용 가이드<br>- Eclipse 플러그인 설치 가이드<br>- 지속적 통합을 위한 품질 목표 및 운영 지침 내용</td><td>품 질 관 리 담당자</td></tr>
<tr><th colspan="2">입력물</th><th colspan="2">출력물</th></tr>
<tr><td colspan="2">지속적 통합을 위한 품질 목표 및 운영 지침<br>지속적 통합 환경</td><td colspan="2">지속적 통합 교육 자료</td></tr>
</table>

〈표 9〉 ‘소스 코드 구현’ 정의

| ID | | 활동 명 |
|---|---|---|
| CI_4 | | 소스 코드 구현 |
| 정의 | | |
| 개발자는 Eclipse를 통해 소스 코드 구현을 진행한다. | | |
| 착수 기준 | 1. 아키텍처 담당자가 지속적 통합 환경을 구축하였다.<br>2. 품질관리 담당자가 지속적 통합 교육을 실시하였다. | |
| 종료 기준 | 1. 개발자는 구현을 완료하였다. | |
| 주요 활동 | | 작업자 |
| 1. 개발자는 Eclipse를 통해 개발을 진행한다. | | 개발자 |
| 입력물 | | 출력물 |
| 지속적 통합을 위한 품질 목표 및 운영 지침<br>지속적 통합 환경 | | 소스 코드 |

〈표 10〉 ‘소스 코드 Push’ 정의

| ID | | 활동 명 |
|---|---|---|
| CI_5 | | 소스 코드 Push |
| 정의 | | |
| 개발자는 Push 조건에 적합한 소스 코드를 원격 저장소로 Push한다. | | |
| 착수 기준 | 1. 개발자는 Eclipse를 통해 구현을 진행한다. | |
| 종료 기준 | 1. 개발자는 소스 코드를 원격 저장소로 Push한다. | |
| 주요 활동 | | 작업자 |
| 1. 개발자는 운영 지침에 기재된 Push 조건에 적합한 소스 코드를 Push하여 원격 저장소에 저장한다. | | 개발자 |
| 입력물 | | 출력물 |
| 소스 코드 | | - |

〈표 11〉 ‘지속적 통합’ 정의

| ID | 활동 명 |
|---|---|
| CI_6 | 지속적 통합 |
| 정의 | |
| Jenkins는 지정된 시간마다 통합된 소스 코드에 대한 빌드를 수행하고, 결과를 알려준다. | |
| 착수 기준 | 1. 아키텍처 담당자가 지속적 통합 환경을 구축하였다.<br>2. 원격 저장소에 통합된 소스 코드가 저장된다. |
| 종료 기준 | 1. 통합 빌드를 완료하였다. |
| 주요 활동 | 작업자 |
| 1. Jenkins는 매일 지정된 시간마다 빌드 도구 Maven을 통해 통합된 소스 코드에 대해 빌드하고, 결과를 알려준다. | - |
| 입력물 | 출력물 |
| 소스 코드<br>지속적 통합 환경 | 지속적 통합 결과 |

<표 12> ‘지속적 통합 결과 검토’ 정의

| ID | 활동 명 |
|---|---|
| CI_7 | 지속적 통합 결과 검토 |
| 정의 | |
| 품질관리 담당자는 매일 빌드 결과에 대해 확인하고, 결과에 대해 공지한다. | |
| 착수 기준 | 1. 지속적 통합이 완료되었다. |
| 종료 기준 | 1. 품질관리 담당자는 지속적 통합 결과에 대해 공지한다. |
| 주요 활동 | 작업자 |
| 1. 품질관리 담당자는 지속적 통합 결과에 대해 매일 확인한다.<br>- 빌드 결과<br><br>2. 품질관리 담당자는 지속적 통합 결과에 대해 공지한다.<br>- 빌드 실패한 경우, 운영 지침에 기재된 방식에 따라 조치를 취한다.<br><br>3. 품질관리 담당자는 지속적으로 지속적 통합 결과에 대해 프로젝트 관리자에게 보고한다. | 품질관리 담당자 |
| 입력물 | 출력물 |
| 지속적 통합 결과 | - |

## 3.3. 2단계 - 정적 분석 프로세스 구축 및 지속적 통합 반영

정적 분석이란 소프트웨어를 실행시키지 않고 소스 코드 자체를 분석하여 이상 여부를 찾아내는 것이다.

본 논문에서는 PMD, JDepend, JavaNCSS를 정적 분석 지원도구로 선정하였다. 각 도구의 기능은 앞서 2.1.2에서 언급하였지만 간단하게 정리하면 <표 13>과 같다.

PMD는 규칙을 기반으로 소스 코드의 어떤 부분이 위반하였는 지와 중복되는 코드에 대해 분석한다. 주로 개발자들이 작업 중 실행하여 결과에 대해 확인하고 반영하는 목적으로 많이 사용된다.

JavaNCSS는 주석을 제외한 소스 코드의 라인 수와 순환 복잡도에 대해 분석한다. 개발자는 분석을 통해 어느 메소드의 순환 복잡도가 높은 지에 대해 확인하고, 통합된 산출물에 전체 패키지, 클래스, 메소드 개수와 순환 복잡도에 대해 파악할 수 있다.

JDepend는 소스 코드의 패키지의 추상/구체 정도, 균형 정도, 원형 의존성에 관한 정보를 알려줌으로써 개발자가 소스 코드 설계에 대한 판단을 할 수 있도록 도와준다.

<표 13> 정적 분석 도구의 기능

| 도구 | 기능 |
|---|---|
| PMD | - 코드 작성 규칙 기반 위반 분석<br>- 중복 코드 탐색 |
| JDepend | - 소스 코드의 패키지 의존성 분석 |
| JavaNCSS | - 소스 코드 라인 수에 대한 분석<br>- 소스 코드의 순환 복잡도에 대한 분석 |

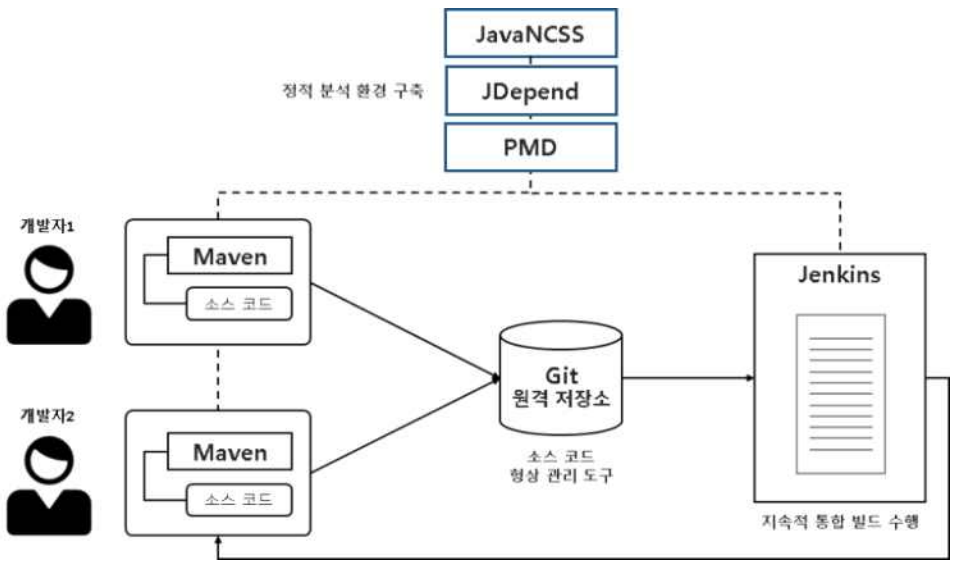
본 논문에서는 정적 분석 도구를 사용하는 방법으로 2.1.2에서 언급한 도구 사용 방

법 3가지 중 2) Eclipse 연결해서 사용하는 방법과 3) Maven 연결해서 사용하는 방법을 선정하였다.

정적 분석 프로세스 구축 및 지속적 통합 적용을 위하여 1단계에서 정의한 지속적 통합 빌드 프로세스를 기반으로 정적 분석 도구 PMD, JDepend, JavaNCSS를 활용한다. 각 개발자가 개발 중 Eclipse를 통해 소스 코드에 대한 정적 분석 결과를 확인할 수 있도록 연동한다. 개발자들은 개발 중 정기적으로 결과를 확인함으로써 소스 코드 설계에 대한 검토를 한다.

또한, 개발 초반부터 통합된 산출물에 대한 정적 분석 결과도 확인할 수 있도록 Maven에 정적 분석 플러그인을 설치하여 Jenkins의 통합 빌드 수행 시 정적 분석 또한 함께 수행하여 결과를 개발 팀 전체와 QA에게 알린다.

이를 통해 개발자들은 개발 중에 소스 코드의 정적 분석 결과에 대해 확인하고 이를 반영할 수 있다. 또한, 통합 소스 코드에 대한 정적 분석 결과를 개발팀 전체 혹은 QA가 파악할 수 있다.




[그림 12] 정적 분석 프로세스 구축 및 지속적 통합 반영

### (1) 목적

지속적으로 개발자들의 소스 코드를 통합하여 매일 빌드하고, 개발자들의 소스 코드에 대해 정적 분석을 수행하여 결함 혹은 결함을 유발할 만한 요소를 조기에 발견하고 그에 대한 조치를 취한다.

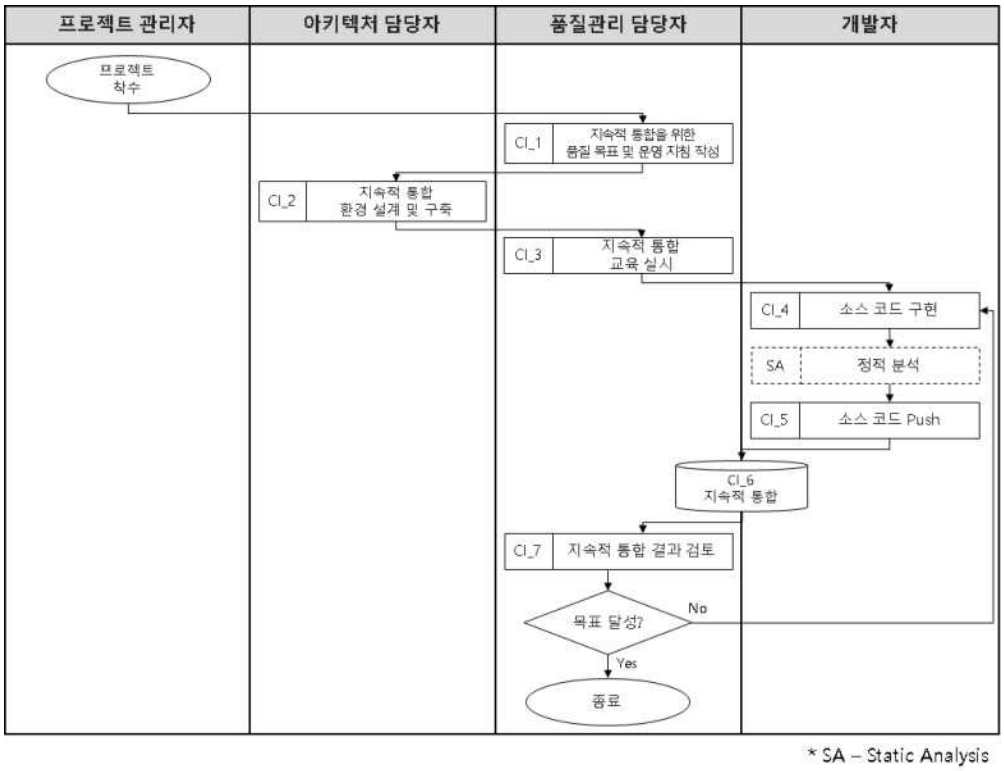
### (2) 역할과 책임

<표 14> 구성원 역할 및 책임

| 역할 | 책임 |
|---|---|
| 프로젝트 관리자 (Project Manager) | - 프로젝트의 계획과 실행에 대해 총괄책임<br>- 각 역할에 대한 담당자 지정 |
| 아키텍처 담당자 (Application Architect) | - 요구사항을 충족하는 SW 아키텍처 설계 및 검증<br>- 아키텍처에 따른 상세설계 및 구현의 적절성 확인 |
| 품질관리 담당자 (Quality Assurance) | - 비즈니스 목표/요구사항에 따라 품질 목표 수립<br>- 품질보증 계획 수립, 이행을 위한 준비, 교육 제공<br>- 통합 빌드의 성공 여부 모니터링 및 결과 공지<br>- 정적 분석 결과 모니터링 |
| 개발자 (Developer) | - 정의된 요구사항, 설계에 따라 코드 구현<br>- 정적 분석을 통한 소스 코드검증 |

### (3) 절차

#### 1) Swim Lane 다이어그램




[그림 13] 정적 분석 프로세스 구축 및 지속적 통합 반영 다이어그램

### 2) 활동 정의

정적 분석 프로세스 및 지속적 통합 반영 단계에서 수행되는 활동에 대해 간략하게 설명하자면 <표 15>와 같다.

〈표 15〉 정적 분석 프로세스 및 지속적 통합 반영 활동 정의

| ID | 활동 명 | 상세 활동 설명 |
|---|---|---|
| CI_1 | 지속적 통합을 위한 품질 목표 및 운영 지침 작성 | - 품질관리 담당자는 지속적 통합의 품질 목표 및 운영 지침에 관해 작성한다.<br>- 품질관리 담당자는 최종 정적 분석 목표 기준을 선정한다. |
| CI_2 | 지속적 통합 환경 설계 및 구축 | - 아키텍처 담당자는 프로젝트의 목표에 적합한 정적 분석 도구를 선정한다.<br>- 아키텍처 담당자는 지속적 통합 환경을 설계하고 구축한다. |
| CI_3 | 지속적 통합 교육 실시 | - 품질관리 담당자는 교육 대상을 선정하고, 지속적 통합의 목표와 운영 지침 및 도구 사용 가이드에 대한 교육을 한다. |
| CI_4 | 소스 코드 구현 | - 개발자는 Eclipse를 통해 개발을 진행한다. |
| SA | 정적 분석 | - 개발자는 개발 중인 소스 코드에 대한 정적 분석을 수행하여 결과를 확인하고 프로젝트의 목표에 부합하는지 확인한다.<br>- 정적 분석 결과가 프로젝트의 정적 분석 목표 기준과 다르다면 해당 소스 코드를 수정한다. |
| CI_5 | 소스 코드 Push | - 개발자는 운영 지침의 조건을 만족하는 소스 코드를 Git을 통해 Push한다. |
| CI_6 | 지속적 통합 | - Jenkins는 지정된 시간마다 통합된 소스 코드에 대한 빌드를 수행하고, 결과를 알려준다.<br>- Jenkins는 지정된 시간마다 통합 소스 코드에 대해 정적 분석을 수행하고 결과를 시각화하여 나타낸다. |
| CI_7 | 지속적 통합 결과 검토 | - 품질관리 담당자는 매일 빌드 결과에 대해 확인하고, 정적 분석 결과가 목표 기준에 적합한지 확인하여 결과에 대해 공지한다.<br>- 빌드 실패 시, 품질관리 담당자는 그에 대해 조치를 취한다.<br>- 정적 분석 결과가 목표 기준에 부합하지 않는다면 그에 대한 조치를 취한다. |

각 활동들에 대해 정의를 하면 다음과 같다.

〈표 16〉 ‘지속적 통합을 위한 품질 목표 및 운영 지침 작성’ 정의

<table>
<tr><th colspan="2">ID</th><th colspan="2">활동 명</th></tr>
<tr><td colspan="2">CI_1</td><td colspan="2">지속적 통합을 위한 품질 목표 및 운영 지침 작성</td></tr>
<tr><th colspan="4">정의</th></tr>
<tr><td colspan="4">품질관리 담당자는 지속적 통합의 품질 목표 및 운영 지침에 관해 작성한다.</td></tr>
<tr><td>착수 기준</td><td colspan="3">1. 프로젝트에 착수하였다.</td></tr>
<tr><td>종료 기준</td><td colspan="3">1. 품질관리 담당자가 지속적 통합을 위한 품질 목표 및 운영 지침을 작성하였다.</td></tr>
<tr><th colspan="3">주요 활동</th><th>작업자</th></tr>
<tr><td colspan="3">1. 품질관리 담당자는 프로젝트의 정보에 대해 파악한다.<br>- 프로젝트 개발 언어<br>- 프로젝트의 규모<br>- 프로젝트 기간<br>- 프로젝트 참여 인원<br><br>2. 품질관리 담당자는 프로젝트의 지속적 통합을 위한 품질 목표 및 운영 지침을 작성한다.<br>- 지속적 통합의 목표<br>- Git 사용 규칙 작성<br>- 통합 빌드 성공 요건 선정<br>- 통합 빌드 실패 시 담당자별 행동 요령<br>- 정적 분석 품질 목표</td><td>품 질 관 리 담당자</td></tr>
<tr><th colspan="2">입력물</th><th colspan="2">출력물</th></tr>
<tr><td colspan="2">프로젝트의 정보</td><td colspan="2">지속적 통합을 위한 품질 목표 및 운영 지침</td></tr>
</table>

〈표 17〉 ‘지속적 통합 환경 설계 및 구축’ 정의

<table>
<tr><th colspan="2">ID</th><th colspan="2">활동 명</th></tr>
<tr><td colspan="2">CI_2</td><td colspan="2">지속적 통합 환경 설계 및 구축</td></tr>
<tr><th colspan="4">정의</th></tr>
<tr><td colspan="4">아키텍처 담당자는 지속적 통합을 위한 품질 목표 및 운영 지침을 참고하여 프로젝트에 적합하도록 지속적 통합 환경을 설계하고, 그에 맞게 구축한다.</td></tr>
<tr><td>착수 기준</td><td colspan="3">1. 품질관리 담당자가 지속적 통합을 위한 품질 목표 및 운영 지침을 작성하였다.</td></tr>
<tr><td>종료 기준</td><td colspan="3">1. 아키텍처 담당자가 지속적 통합 환경을 구축하였다.</td></tr>
<tr><th colspan="3">주요 활동</th><th>작업자</th></tr>
<tr><td colspan="3">1. 아키텍처 담당자는 프로젝트의 환경과 목표에 걸맞은 지속적 통합 환경을 설계한다.<br>- 빌드 도구 선정<br>- CI 도구 선정<br>- 정적 분석 도구 선정<br><br>2. 아키텍처 담당자는 설계를 기반으로 지속적 통합 환경을 구축한다.<br>- Maven 환경 구축<br>- Jenkins 환경 구축<br>- PMD 환경 구축<br>- JavaNCSS 환경 구축<br>- JDepend 환경 구축<br>- 도구 간 연동</td><td>아키텍처 담당자</td></tr>
<tr><th colspan="2">입력물</th><th colspan="2">출력물</th></tr>
<tr><td colspan="2">지속적 통합을 위한 품질 목표 및 운영 지침</td><td colspan="2">지속적 통합 환경 설계도<br>지속적 통합 환경</td></tr>
</table>

〈표 18〉 ‘지속적 통합 교육 실시’ 정의

<table>
<tr><th colspan="2">ID</th><th>활동 명</th></tr>
<tr><td colspan="2">CI_3</td><td>지속적 통합 교육 실시</td></tr>
<tr><th colspan="3">정의</th></tr>
<tr><td colspan="3">품질관리 담당자는 교육 대상을 선정하고, 지속적 통합의 품질 목표와 운영 지침 및 사용 가이드에 대한 교육을 한다.</td></tr>
<tr><td>착수 기준</td><td colspan="2">1. 품질관리 담당자가 지속적 통합을 위한 품질 목표 및 운영 지침을 작성하였다.<br>2. 아키텍처 담당자가 지속적 통합 환경을 구축하였다.</td></tr>
<tr><td>종료 기준</td><td colspan="2">1. 품질관리 담당자가 지속적 통합 교육을 실시하였다.</td></tr>
<tr><th colspan="2">주요 활동</th><th>작업자</th></tr>
<tr><td colspan="2">1. 품질관리 담당자는 교육 대상을 선정한다.<br>- 개발자<br><br>2. 품질관리 담당자는 교육을 실시한다.<br>- 지속적 통합 도구 사용 가이드<br>- 빌드 도구 사용 가이드<br>- 정적 분석 지표 의미<br>- Eclipse 플러그인 설치 가이드<br>- 지속적 통합을 위한 품질 목표 및 운영 지침 내용</td><td>품 질 관 리 담당자</td></tr>
<tr><th colspan="2">입력물</th><th>출력물</th></tr>
<tr><td colspan="2">지속적 통합을 위한 품질 목표 및 운영 지침<br>지속적 통합 환경</td><td>지속적 통합 교육 자료</td></tr>
</table>

〈표 19〉 ‘소스 코드 구현’ 정의

| ID | | 활동 명 |
|---|---|---|
| CI_4 | | 소스 코드 구현 |
| 정의 | | |
| 개발자는 Eclipse를 통해 소스 코드 구현을 진행한다. | | |
| 착수 기준 | 1. 아키텍처 담당자가 지속적 통합 환경을 구축하였다.<br>2. 품질관리 담당자가 지속적 통합 교육을 실시하였다. | |
| 종료 기준 | 1. 개발자는 구현을 완료하였다. | |
| 주요 활동 | | 작업자 |
| 1. 개발자는 Eclipse를 통해 구현을 진행한다. | | 개발자 |
| 입력물 | | 출력물 |
| 지속적 통합을 위한 품질 목표 및 운영 지침<br>지속적 통합 환경 | | 소스 코드 |

〈표 20〉 ‘정적 분석’ 정의

| ID | | 활동 명 |
|---|---|---|
| SA | | 정적 분석 |
| 정의 | | |
| 개발자는 개발 중인 소스 코드에 대한 정적 분석을 수행하여 결과를 확인하고 반영한다. | | |
| 착수 기준 | 1. 개발자는 Eclipse를 통해 구현을 진행한다. | |
| 종료 기준 | 1. 개발자는 정적 분석 결과를 확인하고 반영한다. | |
| 주요 활동 | | 작업자 |
| 1. 개발자는 구현 중 Eclipse를 통해 지속적으로 소스 코드에 대해 정적 분석을 수행하여 결과를 확인하고 반영한다.<br>- PMD(PMD): 규칙 기반 코드 분석<br>- PMD(CPD): 중복 코드 분석<br>- JDepend: 의존성 분석<br>- JavaNCSS: 주석 제외 코드 라인 수 분석 | | 개발자 |
| 입력물 | | 출력물 |
| 소스 코드 | | 정적 분석 결과 |

〈표 21〉 ‘소스 코드 Push’ 정의

| ID | | 활동 명 |
|---|---|---|
| CI_5 | | 소스 코드 Push |
| 정의 | | |
| 개발자는 Push 조건에 적합한 소스 코드를 원격 저장소로 Push한다. | | |
| 착수 기준 | 1. 개발자는 Eclipse를 통해 구현을 진행한다. | |
| 종료 기준 | 1. 개발자는 소스 코드를 원격 저장소로 Push한다. | |
| 주요 활동 | | 작업자 |
| 1. 개발자는 운영 지침에 기재된 Push 조건에 적합한 소스 코드를 Push하여 원격 저장소에 저장한다. | | 개발자 |
| 입력물 | | 출력물 |
| 소스 코드 | | - |

〈표 22〉 ‘지속적 통합’ 정의

| ID | | 활동 명 |
|---|---|---|
| CI_6 | | 지속적 통합 |
| 정의 | | |
| Jenkins는 지정된 시간마다 통합된 소스 코드에 대한 빌드를 수행하고, 결과를 알려준다. | | |
| 착수 기준 | 1. 아키텍처 담당자가 지속적 통합 환경을 구축하였다.<br>2. 원격 저장소에 통합된 소스 코드가 저장된다. | |
| 종료 기준 | 1. 통합 빌드를 완료하였다. | |
| 주요 활동 | | 작업자 |
| 1. Jenkins는 매일 지정된 시간마다 빌드 도구 Maven을 통해 통합된 소스 코드에 대해 빌드하고, 결과를 알려준다.<br><br>2. Jenkins는 매일 지정된 시간마다 빌드와 함께 지정된 정적 분석 도구 PMD, JDepend, JavaNCSS를 통해 정적 분석을 시행하고, 결과를 시각화하여 알려준다. | | - |
| 입력물 | | 출력물 |
| 소스 코드<br>지속적 통합 환경 | | 지속적 통합 결과 |

〈표 23〉 ‘지속적 통합 결과 검토’ 정의

<table>
<tr><th colspan="2">ID</th><th>활동 명</th></tr>
<tr><td colspan="2">CI_7</td><td>지속적 통합 결과 검토</td></tr>
<tr><td colspan="3">정의</td></tr>
<tr><td colspan="3">품질관리 담당자는 매일 빌드 결과에 대해 확인하고, 결과에 대해 공지한다.</td></tr>
<tr><td>착수 기준</td><td colspan="2">1. 지속적 통합이 완료되었다.</td></tr>
<tr><td>종료 기준</td><td colspan="2">1. 품질관리 담당자는 지속적 통합 결과에 대해 공지한다.</td></tr>
<tr><td colspan="2">주요 활동</td><td>작업자</td></tr>
<tr><td colspan="2">1. 품질관리 담당자는 지속적 통합 결과에 대해 매일 확인한다.<br>- 빌드 결과<br>- 정적 분석 결과<br><br>2. 품질관리 담당자는 지속적 통합 결과에 대해 공지한다.<br>- 빌드 실패한 경우, 운영 지침에 기재된 방식에 따라 조치를 취한다.<br>- 정적 분석의 결과가 목표에 부합하지 않을 경우, 운영 지침에 기재된 방식에 따라 조치를 취한다.<br><br>3. 품질관리 담당자는 지속적으로 지속적 통합 결과에 대해 프로젝트 관리자에게 보고한다.</td><td>품 질 관 리 담당자</td></tr>
<tr><td colspan="2">입력물</td><td>출력물</td></tr>
<tr><td colspan="2">지속적 통합 결과</td><td>-</td></tr>
</table>

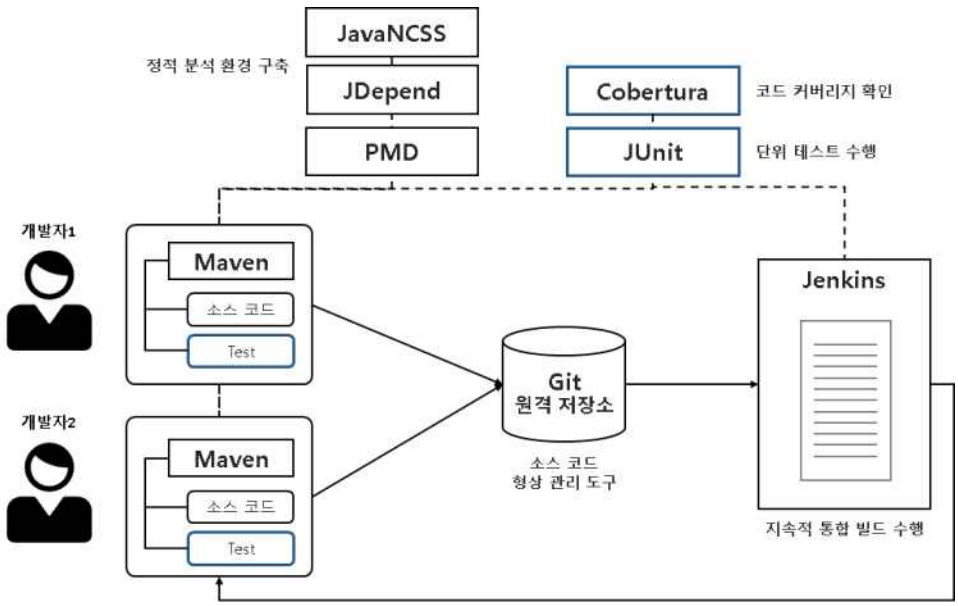
## 3.4. 3단계 - 단위 테스트 프로세스 구축 및 지속적 통합 반영

테스팅 목적으로는 단위 테스트 수행과 그에 대한 코드 커버리지 확인을 하는 활동을 제안한다. 앞서 2.1.3에서 언급하였듯, 단위 테스트를 위한 코드를 개발자가 직접 작성하여 해당 코드를 실행함으로써 테스트 대상이 되는 코드에 대한 특정 영역을 실행해보고 소스 코드에 대한 검증을 한다. 그 후, 코드 커버리지를 확인함으로써 단위 테스트를 통하여 수행된 테스트가 전체 코드에 대해 어느 정도 테스트를 수행했는지 확인할 수 있다. 또한, 통합 소스 코드의 테스트 성공률 및 커버리지 파악을 통

해 목표를 달성했는지에 대해 여부를 알 수 있다. 테스트 프로세스 수행을 위하여 본 논문에서는 단위 테스트 도구로는 JUnit, 코드 커버리지 도구로는 Cobertura를 선정하였다.

[그림 14]와 같이 Maven을 통해 프로젝트를 생성하면 테스트를 위한 폴더 구조가 생성되고, 테스트 수행을 위한 테스트 케이스는 test 폴더 위치에 작성한다.

또한, 코드 커버리지 도구 사용을 위해 Maven에 Cobertura 플러그인을 설치한다. 각 개발자의 Eclipse에서 단위 테스트 수행을 통해 소스 코드 검증을 하고, Maven을 통해 단위 테스트가 얼마나 수행되었는 지에 대해 Cobertura 결과로 확인한다. 추후 각 개발자들의 소스 코드가 Maven과 Jenkins를 통해 통합 빌드될 때 통합 산출물에 대해 테스트가 얼마나 성공하였는 지와 얼마나 테스트했는 지에 대해 코드 커버리지로 파악할 수 있다.




[그림 14] 단위 테스트 프로세스 구축 및 지속적 통합 반영

### (1) 목적

지속적으로 개발자들의 소스 코드를 통합하여 매일 빌드하고, 개발자들의 소스 코드에 대해 정적 분석 및 단위 테스트를 수행하여 조기에 문제를 발견하고 그에 대한 조치를 취한다. 그 후, 코드 커버리지 확인을 통해 단위 테스트 수행 범위에 대해 확인한다.

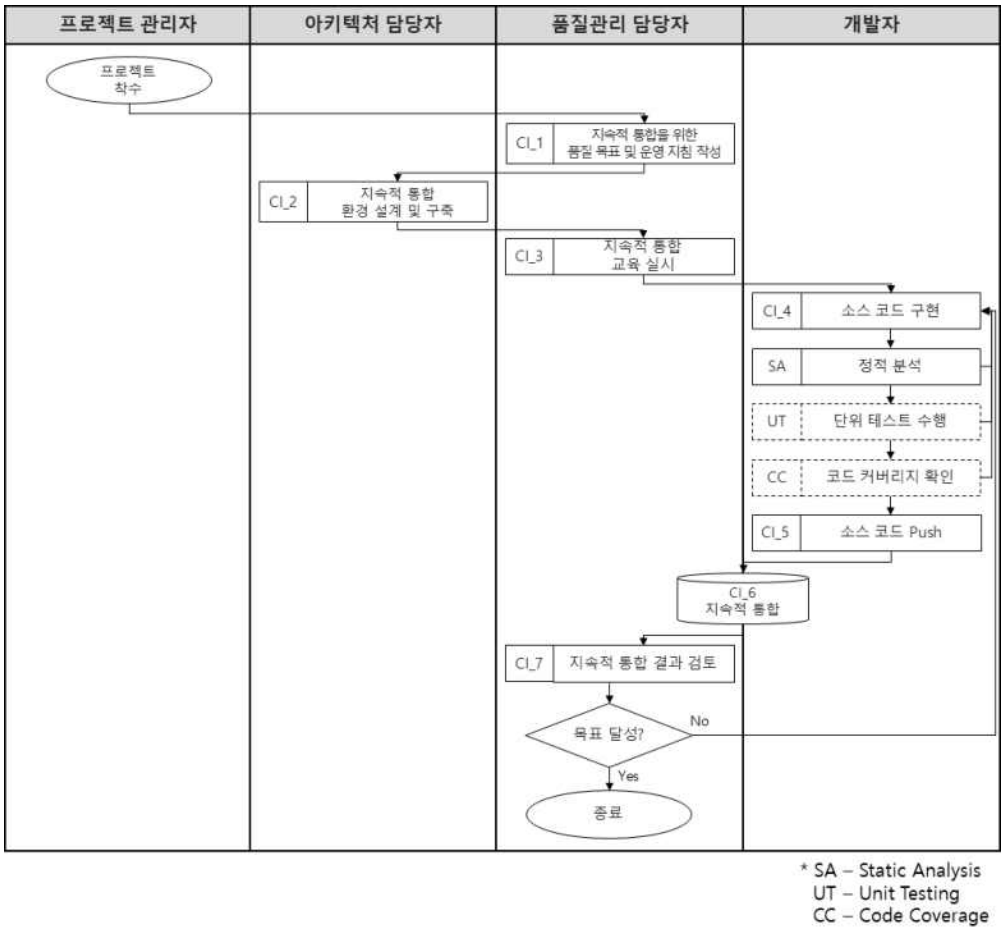
### (2) 역할과 책임

〈표 24〉 구성원 역할 및 책임

| 역할 | 책임 |
|---|---|
| 프로젝트 관리자 (Project Manager) | - 프로젝트의 계획과 실행에 대해 총괄책임<br>- 각 역할에 대한 담당자 지정 |
| 아키텍처 담당자 (Application Architect) | - 요구사항을 충족하는 SW 아키텍처 설계 및 검증<br>- 아키텍처에 따른 상세설계 및 구현의 적절성 확인 |
| 품질관리 담당자 (Quality Assurance) | - 비즈니스 목표/요구사항에 따라 품질 목표 수립<br>- 품질보증 계획 수립, 이행을 위한 준비, 교육 제공<br>- 통합 빌드의 성공 여부 모니터링 및 결과 공지<br>- 정적 분석 결과 및 단위 테스트 성공률과 코드 커버리지 모니터링 |
| 개발자 (Developer) | - 정의된 요구사항, 설계에 따라 코드 구현<br>- 정적 분석, 단위 테스트와 코드 커버리지 확인을 통한 소스 코드 검증 |

### (3) 절차

#### 1) Swim Lane 다이어그램




[그림 15] 단위 테스트 프로세스 구축 및 지속적 통합 반영 다이어그램

### 2) 활동 정의

단위 테스트 프로세스 및 지속적 통합 반영 단계에서 수행되는 활동에 대해 간략하게 설명하자면 <표 25>와 같다.

<표 25> 단위 테스트 프로세스 및 지속적 통합 반영 활동 정의

| ID | 활동 명 | 상세 활동 설명 |
|---|---|---|

| | | |
|---|---|---|
| CI_1 | 지속적 통합을 위한 품질 목표 및 운영 지침 작성 | - 품질관리 담당자는 지속적 통합의 품질 목표 및 운영 지침에 관해 작성한다.<br>- 품질관리 담당자는 최종 정적 분석 목표 기준을 선정한다.<br>- 품질관리 담당자는 최종 단위 테스트 성공률 및 코드 커버리지에 대한 목표 기준을 선정한다. |
| CI_2 | 지속적 통합 환경 설계 및 구축 | - 아키텍처 담당자는 프로젝트의 목표에 적합한 정적 분석 도구와 단위 테스트 및 코드 커버리지 도구를 선정한다.<br>- 아키텍처 담당자는 지속적 통합 환경을 설계하고 구축한다. |
| CI_3 | 지속적 통합 교육 실시 | - 품질관리 담당자는 교육 대상을 선정하고, 지속적 통합의 목표와 운영 지침 및 도구 사용 가이드에 대한 교육을 한다. |
| CI_4 | 소스 코드 구현 | - 개발자는 Eclipse를 통해 개발을 진행한다. |
| SA | 정적 분석 | - 개발자는 개발 중인 소스 코드에 대한 정적 분석을 수행하여 결과를 확인하고 프로젝트의 목표에 부합하는지 확인한다.<br>- 정적 분석 결과가 프로젝트의 정적 분석 목표 기준과 다르다면 해당 소스 코드를 수정한다. |
| UT | 단위 테스트 수행 | - 개발자는 개발 중인 소스 코드에 대해 단위 테스트를 수행하여 검증한다. |
| CC | 코드 커버리지 확인 | - 개발자는 단위 테스트의 수행 범위를 파악하기 위해 코드 커버리지를 확인한다. |
| CI_5 | 소스 코드 Push | - 개발자는 운영 지침의 조건을 만족하는 소스 코드를 Git을 통해 Push한다. |
| CI_6 | 지속적 통합 | - Jenkins는 지정된 시간마다 통합된 소스 코드에 대한 빌드를 수행하고, 결과를 알려준다.<br>- Jenkins는 지정된 시간마다 통합 소스 코드에 대해 정적 분석을 수행하고 결과를 시각화하여 나타낸다.<br>- Jenkins는 지정된 시간마다 각 개발자들이 수행한 단위 테스트 성공률과 코드 커버리지에 대한 정보를 시각화하여 나타낸다. |

| | | |
|---|---|---|
| CI_7 | 지속적 통합 결과 검토 | - 품질관리 담당자는 매일 빌드 결과에 대해 확인하고, 결과에 대해 공지한다.<br>- 품질관리 담당자는 매일 정적 분석 결과가 목표 기준에 적합한지 확인한다.<br>- 품질관리 담당자는 매일 단위 테스트 성공률 및 코드 커버리지가 목표 기준에 적합한지 확인한다.<br>- 빌드 실패 시, 품질관리 담당자는 그에 대해 조치를 취한다.<br>- 정적 분석 결과 혹은 단위 테스트 성공률 및 코드 커버리지가 목표 기준에 부합하지 않는다면 그에 대한 조치를 취한다. |

각 활동들에 대해 정의를 하면 다음과 같다.

〈표 26〉 ‘지속적 통합을 위한 품질 목표 및 운영 지침 작성’ 정의

| ID | | 활동 명 |
|---|---|---|
| CI_1 | | 지속적 통합을 위한 품질 목표 및 운영 지침 작성 |
| 정의 | | |
| 품질관리 담당자는 지속적 통합의 품질 목표 및 운영 지침에 관해 작성한다. | | |
| 착수 기준 | 1. 프로젝트에 착수하였다. | |
| 종료 기준 | 1. 품질관리 담당자가 지속적 통합을 위한 품질 목표 및 운영 지침을 작성하였다. | |
| 주요 활동 | | 작업자 |
| 1. 품질관리 담당자는 프로젝트의 정보에 대해 파악한다.<br>- 프로젝트 개발 언어<br>- 프로젝트의 규모<br>- 프로젝트 기간<br>- 프로젝트 참여 인원<br><br>2. 프로젝트의 지속적 통합을 위한 품질 목표 및 운영 지침을 작성한다.<br>- 지속적 통합의 목표<br>- Git 사용 규칙 작성<br>- 통합 빌드 성공 요건 선정<br>- 통합 빌드 실패 시 담당자별 행동 요령<br>- 정적 분석 품질 목표<br>- 단위 테스트 성공률 및 코드 커버리지 품질 목표 | | 품 질 관 리 담당자 |
| 입력물 | | 출력물 |
| 프로젝트의 정보 | | 지속적 통합을 위한 품질 목표 및 운영 지침 |

〈표 27〉 ‘지속적 통합 환경 설계 및 구축’ 정의

<table>
<tr><th colspan="2">ID</th><th>활동 명</th></tr>
<tr><td colspan="2">CI_2</td><td>지속적 통합 환경 설계 및 구축</td></tr>
<tr><th colspan="3">정의</th></tr>
<tr><td colspan="3">아키텍처 담당자는 지속적 통합을 위한 품질 목표 및 운영 지침을 참고하여 프로젝트에 적합하도록 지속적 통합 환경을 설계하고, 그에 맞게 구축한다.</td></tr>
<tr><td>착수 기준</td><td colspan="2">1. 품질관리 담당자가 지속적 통합을 위한 품질 목표 및 운영 지침을 작성하였다.</td></tr>
<tr><td>종료 기준</td><td colspan="2">1. 아키텍처 담당자가 지속적 통합 환경을 구축하였다.</td></tr>
<tr><th colspan="2">주요 활동</th><th>작업자</th></tr>
<tr><td colspan="2">1. 아키텍처 담당자는 프로젝트의 환경과 목표에 걸맞은 지속적 통합 환경을 설계한다.<br>- 빌드 도구 선정<br>- CI 도구 선정<br>- 정적 분석 도구 선정<br>- 단위 테스트 도구 선정<br>- 코드 커버리지 도구 선정<br><br>2. 아키텍처 담당자는 설계를 기반으로 지속적 통합 환경을 구축한다.<br>- Maven 환경 구축<br>- Jenkins 환경 구축<br>- PMD 환경 구축<br>- JavaNCSS 환경 구축<br>- JDepend 환경 구축<br>- Junit 환경 구축<br>- Cobertura 환경 구축<br>- 도구 간 연동</td><td>아 키 텍 처 담당자</td></tr>
<tr><th>입력물</th><th colspan="2">출력물</th></tr>
<tr><td>지속적 통합을 위한 품질 목표 및 운영 지침</td><td colspan="2">지속적 통합 환경 설계도<br>지속적 통합 환경</td></tr>
</table>

<표 28> ‘지속적 통합 교육 실시’ 정의

<table>
<tr><th colspan="2">ID</th><th>활동 명</th></tr>
<tr><td colspan="2">CI_3</td><td>지속적 통합 교육 실시</td></tr>
<tr><th colspan="3">정의</th></tr>
<tr><td colspan="3">품질관리 담당자는 교육 대상을 선정하고, 지속적 통합의 품질 목표와 운영 지침 및 사용 가이드에 대한 교육을 한다.</td></tr>
<tr><td>착수 기준</td><td colspan="2">1. 품질관리 담당자가 지속적 통합을 위한 품질 목표 및 운영 지침을 작성하였다.<br>2. 아키텍처 담당자가 지속적 통합 환경을 구축하였다.</td></tr>
<tr><td>종료 기준</td><td colspan="2">1. 품질관리 담당자가 지속적 통합 교육을 실시하였다.</td></tr>
<tr><th colspan="2">주요 활동</th><th>작업자</th></tr>
<tr><td colspan="2">1. 품질관리 담당자는 교육 대상을 선정한다.<br>- 개발자<br><br>2. 품질관리 담당자는 교육을 실시한다.<br>- 지속적 통합 도구 사용 가이드<br>- 빌드 도구 사용 가이드<br>- 단위 테스트 도구 사용 가이드<br>- 코드 커버리지 도구 사용 가이드<br>- 단위 테스트 및 코드 커버리지의 의미<br>- 정적 분석 지표 의미<br>- Eclipse 플러그인 설치 가이드<br>- 지속적 통합을 위한 품질 목표 및 운영 지침 내용</td><td>품 질 관 리 담당자</td></tr>
<tr><th colspan="2">입력물</th><th>출력물</th></tr>
<tr><td colspan="2">지속적 통합을 위한 품질 목표 및 운영 지침<br>지속적 통합 환경</td><td>지속적 통합 교육 자료</td></tr>
</table>

〈표 29〉 ‘소스 코드 구현’ 정의

| ID | | 활동 명 |
|---|---|---|
| CI_4 | | 소스 코드 구현 |
| 정의 | | |
| 개발자는 Eclipse를 통해 소스 코드 구현을 진행한다. | | |
| 착수 기준 | 1. 아키텍처 담당자가 지속적 통합 환경을 구축하였다.<br>2. 품질관리 담당자가 지속적 통합 교육을 실시하였다. | |
| 종료 기준 | 1. 개발자는 구현을 완료하였다. | |
| 주요 활동 | | 작업자 |
| 1. 개발자는 Eclipse를 통해 구현을 진행한다. | | 개발자 |
| 입력물 | | 출력물 |
| 지속적 통합을 위한 품질 목표 및 운영 지침<br>지속적 통합 환경 | | 소스 코드 |

〈표 30〉 ‘정적 분석’ 정의

| ID | | 활동 명 |
|---|---|---|
| SA | | 정적 분석 |
| 정의 | | |
| 개발자는 개발 중인 소스 코드에 대한 정적 분석을 수행하여 결과를 확인하고 반영한다. | | |
| 착수 기준 | 1. 개발자는 Eclipse를 통해 구현을 진행한다. | |
| 종료 기준 | 1. 개발자는 정적 분석 결과를 확인하고 반영한다. | |
| 주요 활동 | | 작업자 |
| 1. 개발자는 구현 중 Eclipse를 통해 지속적으로 소스 코드에 대해 정적 분석을 수행하여 결과를 확인하고 구현에 반영한다.<br>- PMD(PMD): 규칙 기반 코드 분석<br>- PMD(CPD): 중복 코드 분석<br>- JDepend: 의존성 분석<br>- JavaNCSS: 주석 제외 코드 라인 수 분석 | | 개발자 |
| 입력물 | | 출력물 |
| 소스 코드 | | 정적 분석 결과 |

〈표 31〉 ‘단위 테스트 수행’ 정의

| ID | 활동 명 |
|---|---|
| UT | 단위 테스트 수행 |
| 정의 | |
| 개발자는 개발 중인 소스 코드에 대해 단위 테스트를 수행하여 검증한다. | |
| 착수 기준 | 1. 개발자는 Eclipse를 통해 구현을 진행한다. |
| 종료 기준 | 1. 개발자는 단위 테스트 결과를 확인하고 반영한다. |
| 주요 활동 | 작업자 |
| 1. 개발자는 구현 중 Junit을 통해 테스트 케이스를 생성한다.<br><br>2. 테스트 케이스를 기반으로 지속적으로 소스 코드에 대해 단위 테스트를 수행하여 결과를 확인하고 구현에 반영한다. | 개발자 |
| 입력물 | 출력물 |
| 소스 코드 | 단위 테스트 결과 |

〈표 32〉 ‘코드 커버리지 확인’ 정의

| ID | 활동 명 |
|---|---|
| CC | 코드 커버리지 확인 |
| 정의 | |
| 개발자는 단위 테스트의 수행 범위를 파악하기 위해 코드 커버리지를 확인한다. | |
| 착수 기준 | 1. 개발자는 Eclipse를 통해 구현을 진행한다.<br>2. 개발자는 단위 테스트 결과를 수행한다. |
| 종료 기준 | 1. 개발자는 코드 커버리지를 확인한다. |
| 주요 활동 | 작업자 |
| 1. 개발자는 단위 테스트를 수행한 후, Cobertura를 통해 소스 코드에 대한 코드 커버리지를 확인한다.<br><br>2. 프로젝트 운영 지침의 코드 커버리지 목표 기준과 비교한다.<br>- 코드 커버리지가 프로젝트의 목표보다 떨어진다면, 해당 소스 코드에 대해 단위 테스트를 추가로 수행한다. | 개발자 |
| 입력물 | 출력물 |
| 소스 코드 | 코드 커버리지 지표 |

〈표 33〉 ‘소스 코드 Push’ 정의

| ID | | 활동 명 |
|---|---|---|
| CI_5 | | 소스 코드 Push |
| 정의 | | |
| 개발자는 Push 조건에 적합한 소스 코드를 원격 저장소로 Push한다. | | |
| 착수 기준 | 1. 개발자는 Eclipse를 통해 구현을 진행한다. | |
| 종료 기준 | 1. 개발자는 소스 코드를 원격 저장소로 Push한다. | |
| 주요 활동 | | 작업자 |
| 1. 개발자는 운영 지침에 기재된 Push 조건에 적합한 소스 코드를 Push하여 원격 저장소에 저장한다. | | 개발자 |
| 입력물 | | 출력물 |
| 소스 코드 | | - |

〈표 34〉 ‘지속적 통합’ 정의

| ID | | 활동 명 |
|---|---|---|
| CI_6 | | 지속적 통합 |
| 정의 | | |
| Jenkins는 지정된 시간마다 통합된 소스 코드에 대한 빌드를 수행하고, 결과를 알려준다. | | |
| 착수 기준 | 1. 아키텍처 담당자가 지속적 통합 환경을 구축하였다.<br>2. 원격 저장소에 통합된 소스 코드가 저장된다. | |
| 종료 기준 | 1. 통합 빌드를 완료하였다. | |
| 주요 활동 | | 작업자 |
| 1. Jenkins는 매일 지정된 시간마다 빌드 도구 Maven을 통해 통합된 소스 코드에 대해 빌드하고, 결과를 알려준다.<br><br>2. Jenkins는 매일 지정된 시간마다 빌드와 함께 지정된 정적 분석 도구 PMD, JDepend, JavaNCSS를 통해 정적 분석을 시행하고, 결과를 시각화하여 알려준다.<br><br>3. Jenkins는 지정된 시간마다 각 개발자들이 Junit을 통해 수행한 단위 테스트 성공률과 Cobertura를 통해 분석한 코드 커버리지에 대한 정보를 시각화하여 나타낸다. | | - |
| 입력물 | | 출력물 |
| 소스 코드<br>지속적 통합 환경 | | 지속적 통합 결과 |

〈표 35〉 ‘지속적 통합 결과 검토’ 정의

<table>
<tr><th colspan="2">ID</th><th>활동 명</th></tr>
<tr><td colspan="2">CI_7</td><td>지속적 통합 결과 검토</td></tr>
<tr><th colspan="3">정의</th></tr>
<tr><td colspan="3">품질관리 담당자는 매일 빌드 결과에 대해 확인하고, 결과에 대해 공지한다.</td></tr>
<tr><td>착수 기준</td><td colspan="2">1. 지속적 통합이 완료되었다.</td></tr>
<tr><td>종료 기준</td><td colspan="2">1. 품질관리 담당자는 지속적 통합 결과에 대해 공지한다.</td></tr>
<tr><th colspan="2">주요 활동</th><th>작업자</th></tr>
<tr><td colspan="2">1. 품질관리 담당자는 지속적 통합 결과에 대해 매일 확인한다.<br>- 빌드 결과<br>- 정적 분석 결과<br>- 단위 테스트 성공률<br>- 코드 커버리지<br><br>2. 품질관리 담당자는 지속적 통합 결과에 대해 공지한다.<br>- 빌드 실패한 경우, 운영 지침에 기재된 방식에 따라 조치를 취한다.<br>- 정적 분석의 결과 혹은 단위 테스트 성공률과 코드 커버리지가 목표에 부합하지 않을 경우, 운영 지침에 기재된 방식에 따라 조치를 취한다.<br><br>3. 품질관리 담당자는 지속적으로 지속적 통합 결과 상태에 대해 프로젝트 관리자에게 보고한다.</td><td>품 질 관 리 담당자</td></tr>
<tr><th>입력물</th><th colspan="2">출력물</th></tr>
<tr><td>지속적 통합 결과</td><td colspan="2">-</td></tr>
</table>

# 4. 적용 및 검증

3장에서 제안한 소규모 조직을 위한 지속적 통합 프로세스의 효용성을 확인하기 위해 학부 수업의 프로젝트 팀들에게 적용하고, 그 과정 및 결과를 분석해 보았다.

## 4.1. 적용 대상 프로젝트 개요

본 논문에서는 2년에 걸쳐 학부 수업의 프로젝트를 수행하는 10개 팀들에게 구현과정에서 지속적 통합 환경을 구축하도록 요구하였다. 각 팀은 소규모 조직의 특징을 반영하기 위해 5명으로 구성하였고, 동일한 소프트웨어 요구사항을 부과하였다. 프로젝트의 팀들의 환경은 다음 표와 같다.

(1) 1차 대상 프로젝트

<표 36> 1차 대상 프로젝트 환경

| 프로젝트 내용 | To do List 관리 프로그램 개발 |
|---|---|
| 참여 인원 | 5명 (프로젝트 관리자 1명, 품질관리 담당자 및 개발자 4명) |
| 프로젝트 기간 | 2016. 05. 01 ~ 2016. 06. 20 (약 50일) |
| 개발 생명주기 | 폭포수 모형 |
| 사용 언어 | Java |
| IDE | Eclipse |
| **지속적 통합 적용 단계** | 1단계 - 지속적 통합 빌드 프로세스 |

(2) 2차 대상 프로젝트

<표 37> 2차 대상 프로젝트 환경

| | |
|---|---|
| 프로젝트 내용 | To do List 관리 프로그램 개발 |
| 참여 인원 | 5명 (프로젝트 관리자 1명, 품질관리 담당자 및 개발자 4명) |
| 프로젝트 기간 | 2017. 04. 20 ~ 2016. 06. 20 (약 60일) |
| 개발 생명주기 | 폭포수 모형 |
| 사용 언어 | Java |
| IDE | Eclipse |
| **지속적 통합 적용 단계** | 2단계 - 정적 분석 프로세스 |

## 4.2. 적용 과정

### (1) 프로젝트 적용 과정

본 논문에서는 1단계 지속적 통합 빌드 프로세스의 활동에 맞추어 프로젝트에 적용하였다. 적용 대상 프로젝트 인원들의 지속적 통합에 대한 인식이 부족하였고, 교육과 숙지를 위한 시간이 충분하지 않아, 1단계로 선정하였다. 2차 프로젝트에는 2단계 정적 분석 프로세스 구축 및 지속적 통합 적용까지 적용해보았다.

1차, 2차 적용 프로젝트의 역할은 본 논문에서 제안한 것과 같이 프로젝트 관리자, 품질관리 담당자, 개발자로 지정하였다. 단, 지속적 통합 환경 설계 및 구축과 전반적인 품질 관리를 위해 아키텍처 담당자와 품질관리 담당자의 역할은 본 논문의 저자가 맡아 수행하였다. [그림 11]과 [그림 13]에서 제시한 프로세스에 따라 진행하였다.

#### 1) CI_1. 지속적 통합을 위한 품질 목표 및 운영 지침 작성

프로젝트의 지속적 통합의 품질 목표와 운영 지침에 대해 다음의 항목에 대해 정의하였다. 2차 프로젝트의 품질 목표 및 운영 지침은 지속적 통합 빌드에 관한 내용 뿐만 아니라 정적 분석 품질 목표와 운영 지침에 대해 작성하였다.

〈표 38〉 1차 프로젝트 – 지속적 통합을 위한 품질 목표 및 운영 지침

| 품질 목표 | - 지속적 통합 빌드 결과는 성공이어야 한다. |
|---|---|
| 운영 지침 | **[공통 지침]**<br>- Git을 통해 Commit 시, Commit Message에는 명시된 형식에 맞추어 작성한다.<br>(예, (개발 진도율) 관련 ID - 세부 내용)<br>- 매일 오전 5시에 Jenkins와 Maven을 통해 지속적 통합 빌드가 수행된다.<br><br>**[개발자 지침]**<br>- 작업하기 전, Pull을 수행하여 Local 저장소를 최신 버전으로 업데이트한다.<br>- Eclipse를 통해 빌드 성공일 시에 Push한다.<br>- 지속적 빌드 실패 시, 해당 개발자는 메일 공지를 받은 당일 디버깅을 완료한다.<br><br>**[품질관리 담당자 지침]**<br>- 지속적 통합 빌드의 상태에 대해 검토하고, 실패 시 프로젝트 관리자와 해당 개발자에게 메일로 공지한다.<br>- 지속적 통합 빌드 상태에 대해 정기적으로 프로젝트 관리자에게 보고한다. |

〈표 39〉 2차 프로젝트 – 지속적 통합을 위한 품질 목표 및 운영 지침

| 품질 목표 | - 지속적 통합 빌드 결과는 성공이어야 한다.<br>- 정적 분석 결과 지표는 다음의 조건을 만족해야 한다.<br>- PMD: 규칙 위반 개수 0<br>- PMD: 중복 토큰(100이상) 수 0<br>- JDepend: 원형 의존성 0 |
|---|---|

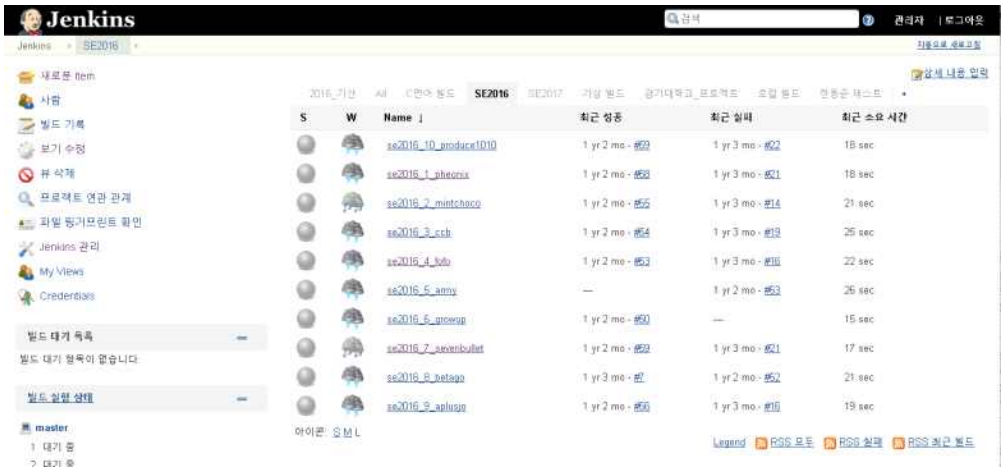
| | |
|---|---|
| 운영 지침 | **[공통 지침]**<br>- Git을 통해 Commit 시, Commit Message에는 명시된 형식에 맞추어 작성한다.<br>(예, (개발 진도율) 관련 ID - 세부 내용)<br>- 매일 오전 5시에 Jenkins와 Maven을 통해 지속적 통합 빌드가 수행된다.<br><br>**[개발자 지침]**<br>- 작업하기 전, Pull을 수행하여 Local 저장소를 최신 버전으로 업데이트한다.<br>- 소스 코드가 아래의 조건을 만족할 시 Push한다.<br>- Eclipse를 통해 빌드 성공하였다.<br>- 소스 코드에 대해 정적 분석을 수행하였을 때, 정적 분석 품질 목표를 만족한다.<br>- 지속적 빌드 실패 시, 해당 개발자는 메일 공지를 받은 당일 디버깅을 완료한다.<br><br>**[품질관리 담당자 지침]**<br>- 지속적 통합 빌드의 상태에 대해 검토하고, 실패 시 프로젝트 관리자와 해당 개발자에게 메일로 공지한다.<br>- 지속적 통합 빌드 상태에 대해 정기적으로 프로젝트 관리자에게 보고한다. |

### 2) CI_2. 지속적 통합 환경 설계 및 구축

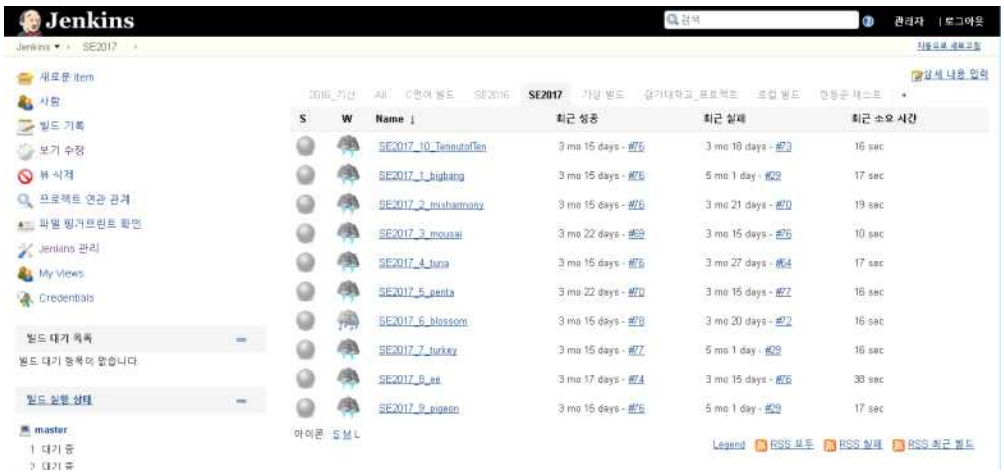
아키텍처 담당자는 지속적 통합의 기반을 마련하기 위해 Git을 설치하여 소스 코드 형상 관리 환경을 구축하였다. 또한, 지속적 통합 도구 Jenkins와 빌드 도구 Maven을 설치하여 형상 관리 도구와의 연동을 완료하였다. [그림 16]과 같이 총 10개 팀의 지속적 통합을 위해 10개의 Item을 생성하여 진행하였다. 이를 통해, Jenkins는 매일 지정된 시간에 Git 원격 저장소의 소스 코드를 불러와 빌드를 수행하였다.

2차 프로젝트에서는 추가적으로 정적 분석 환경 구축을 위해 [그림 17]과 같이

Maven 환경에 정적 분석 도구 플러그인을 설정하였다. 이를 통해, Jenkins는 빌드 뿐만 아니라 정적 분석까지 수행한다. Local 저장소의 정적 분석 환경 구축을 위해서는 개발자들을 대상으로 Eclipse 정적 분석 도구 플러그인을 설치하도록 교육을 수행하였다.




[그림 16] 1차 프로젝트 - 지속적 통합 빌드 프로세스 구축 (Jenkins)




[그림 17] 2차 프로젝트 - 지속적 통합 빌드 프로세스 구축 (Jenkins)

### 3) CI_3. 지속적 통합 교육 실시

1차 프로젝트 내 개발자들을 대상으로 교육을 실시하였다. 교육은 품질 관리 담당자인 본 논문의 저자가 진행하였다. 교육 내용은 다음과 같다.

- 지속적 통합 프로세스 정의
- Git 사용 가이드
- Maven 사용 가이드
- Jenkins 사용 가이드
- 지속적 통합 품질 목표와 운영 지침

2차 프로젝트에서는 기존의 교육 내용에 다음의 항목을 추가하여 진행하였다.

- 정적 분석 도구 사용 가이드 (PMD, JavaNCSS, JDepend)
- Eclipse 환경 정적 분석 도구 플러그인 설치 가이드

### 4) CI_4. 소스 코드 구현

1차, 2차 프로젝트의 개발자들은 Eclipse를 통해 소스 코드 구현을 진행하였다.

### 5) SA. 정적 분석

본 활동은 2단계 프로세스를 적용한 2차 프로젝트에만 해당한다. 개발자들은 소스 코드에 대해 정적 분석을 수행하였다. 만약, 프로젝트의 품질 목표인 다음의 조건을 만족하지 않는다면 만족하도록 소스 코드를 수정하였다.

- PMD: 규칙 위반 개수 0
- PMD: 중복 토큰(100이상) 수 0
- JDepend: 원형 의존성 0

### 6) CI_5. 소스 코드 Push

개발자들은 Eclipse에서 빌드를 성공할 시, 2차 프로젝트에는 추가로 정적 분석 품질 목표 조건까지 만족할 시 소스 코드를 원격 저장소에 Push하도록 하였다.

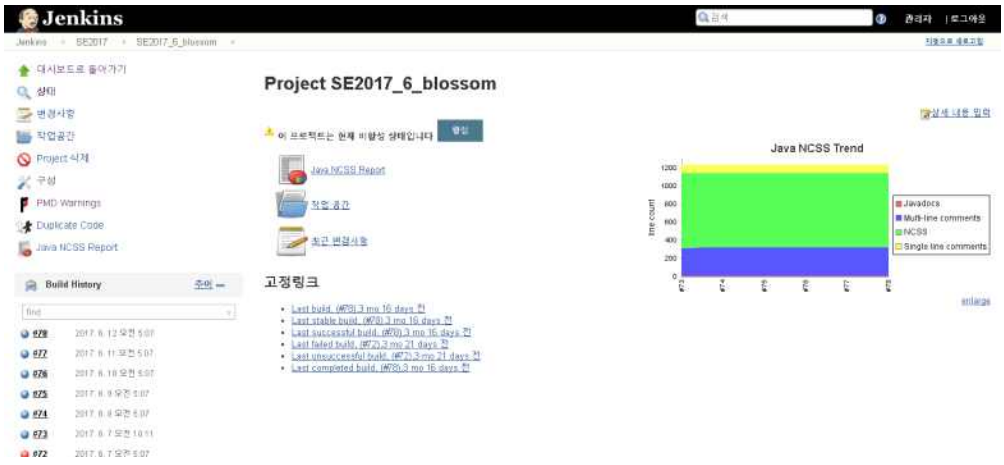
구현 중 Local 저장소에 변경 사항을 저장하는 명령어인 Commit과 원격 저장소의 변경 사항을 Local 저장소에 반영하는 명령어 Pull은 자주 활용하도록 지정하였다.

### 7) CI_6. 지속적 통합

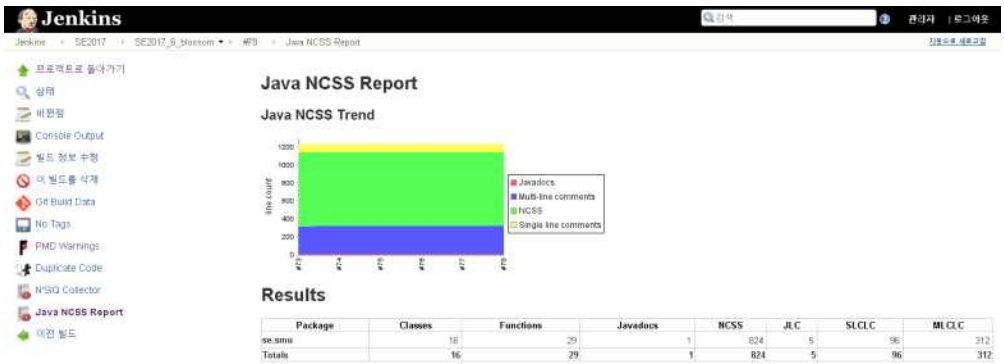
Jenkins는 1차 프로젝트에서 [그림 18]과 같이 매일 지정된 시간인 오전 5시 34분에 Maven을 통해 빌드를 수행하고, 결과를 나타내었다.

2차 프로젝트는 [그림 19]과 같이 오전 5시 7분에 Maven을 통한 빌드 결과와 PMD, JDepend, JavaNCSS를 통해 정적 분석을 수행한 결과를 시각화하여 나타내었다. 정적 분석 결과는 [그림 20], [그림 21], [그림 22]와 같이 Jenkins에 시각화하여 볼 수 있다. 품질 관리 담당자는 이 결과를 통해 소스 코드에 대한 검토를 하였다.

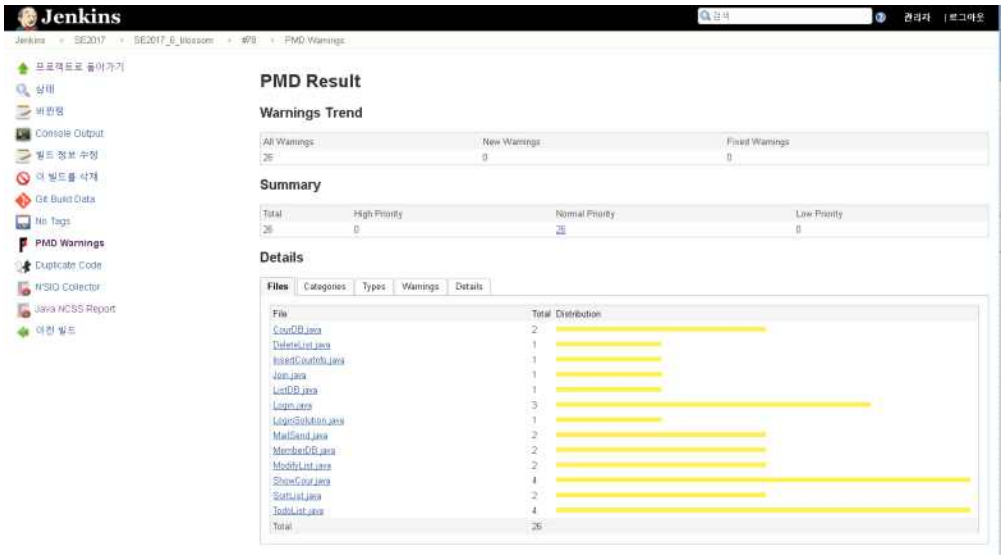

[그림 18] 1차 프로젝트 - 지속적 통합 결과 화면(Jenkins)



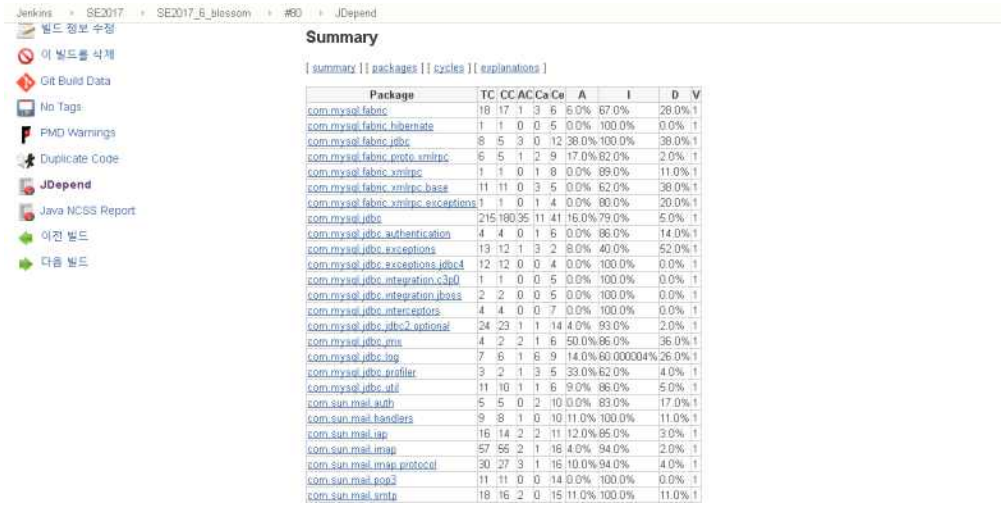

[그림 19] 2차 프로젝트 - 지속적 통합 결과 화면(Jenkins)




[그림 20] 2차 프로젝트 - 정적 분석(JavaNCSS) 결과 화면




[그림 21] 2차 프로젝트 – 정적 분석(PMD) 결과 화면




[그림 22] 2차 프로젝트 – 정적 분석(JDepend) 결과 화면

### 8) CI_7. 지속적 통합 결과 검토

품질관리 담당자는 매일 지속적 통합 빌드 결과에 대해, 2차 프로젝트의 경우에는 추가로 정적 분석 결과에 대해 검토하고, 빌드 실패 혹은 정적 분석 결과가 품질 목

표에 미치지 못할 경우, 프로젝트 관리자와 해당 개발자에게 메일로 실패를 공지하였다.


[그림 23] 실패 공지 메일

문제가 발생한 소스 코드의 해당 개발자는 공지 받은 당일 에러 제거 및 정적 분석 목표에 맞도록 소스 코드를 수정하도록 하였다.

## 4.3. 적용 평가

본 논문에서는 1차, 2차에 걸쳐 학부 수업의 프로젝트를 수행하는 10개 팀들에게 구현과정에서 지속적 통합 환경을 구축하도록 요구하였다. 1차 프로젝트를 대상으로는 지속적 통합 빌드 프로세스를 적용하였고, 2차 프로젝트를 대상으로는 정적 분석 프로세스 구축 및 지속적 통합을 적용하였다. 적용 결과는 다음과 같다.

1차 프로젝트의 경우, 착수 후 4주가 지난 시점(5/28) 기준으로 10개 중 9개의 팀이 빌드 실패임을 볼 수 있었다. 하지만 프로젝트 중반(6/8)부터 종료까지 10개 중 2개의 팀만이 빌드 실패 결과를 보여주었다.

2차 프로젝트는 4주가 지난 시점(5/20)에는 10개 중 9개의 팀이 빌드 실패이었고, 프로젝트 중반(6/8)에는 10개 중 2개의 팀만이 빌드 실패 결과를 보여주었다. 프로젝트 종료 때는 1개의 팀만 빌드 실패로 끝마쳤다.

1차와 2차 프로젝트 모두 초반에는 형상 관리 도구 숙지가 완벽하게 되지 않았고, 지속적 통합 빌드에 대한 개념 파악이 되지 않아 빌드 실패가 빈번하게 일어난 것을 볼 수 있었다. 프로젝트 중반, 그리고 후반으로 접어들면서 개수가 급격하게 줄

어드는 것을 볼 수 있다. 이는 프로젝트 진행 중에도 지속적 통합을 수행하여, 정기적으로 소스 코드에 대한 에러 혹은 결함의 발생에 대해 확인하고 디버깅할 수 있었기 때문이다. 또한, 2차 프로젝트의 경우 지속적으로 소스 코드에 대한 정적 분석 결과를 추가로 확인하고 반영하였기 때문에 더욱 빠른 시정조치가 가능하였고, 1차 프로젝트와 비교하였을 때 빌드 실패 발생 빈도가 확연히 빠른 시간 내 줄어든 것을 볼 수 있었다.

# 5. 결론 및 향후 연구

소프트웨어의 개발 규모가 점차적으로 커지고 있고 여러 사람의 협업이 중요해진 시기에 소프트웨어 프로젝트의 지속적 통합은 성공을 위한 Best Practice 중 하나로 인정받고 있다. 프로젝트의 초기 단계부터 지속적으로 통합 작업을 수행하여 인터페이스 관련 오류 등 통합 관련 에러의 발생을 조기에 발견하여 해결하는 것이 중요하다. 소프트웨어 산업의 많은 규모를 차지하고 있는 소규모 조직의 프로젝트에서도 지속적 통합이 중요하다.

하지만, 지속적 통합 프로세스를 조직 내에 구축하는 것은 많은 전문성을 요구한다. 공용 저장소 내 모듈들의 자동 빌드, 지속적인 정적 분석 수행, 소스 코드에 대한 단위 테스트 수행 및 코드 커버리지 확인 등 프로세스와 도구들의 연계가 중요한 역할을 한다. 그러므로 소규모 조직들에게 프로세스의 정의, 필요한 도구들의 선정, 절차와 도구를 연계하는 프로세스 구축 등을 지원해주는 가이드라인이 필요하다.

본 논문에서는 소규모 조직의 특성을 반영하여 오픈소스 중심의 도구들을 선정하고, 사용하기 쉽도록 프로세스를 제공하였다.

본 논문이 제안하는 프로세스를 활용하는 기업들은 쉽게 도구들에 대하여 이해할 수 있을 것이며, Jenkins, Maven의 환경 구축 과정 수행 후, PMD, JDepend, JavaNCSS, Junit, Cobertura 도구들의 연계 또한 쉽게 구축할 수 있을 것이다.

프로젝트 수행 중 지속적 통합 시기, 담당자와 역할 등 일부는 각 프로젝트의 특성에 맞추어 조정을 할 수 있을 것이다. 효용성을 입증하기 위해 포함시킨 적용 사례를 참고하면 실질적인 구축과 관련하여 이해할 수 있을 것이다.

향후 연구로는 보다 많은 사례들을 통해 프로세스 3단계 적용까지 수행하여 본 논문 연구 내용에 대한 효용성을 입증하고자 한다. 또한, Jenkins에 모인 지속적 통합 데이터를 활용하여, 조직에게 개발과 관련된 정보를 제공할 수 있도록 연구하고자 한다.

# 참 고 문 헌


[1] Stephan Diehl, Software Visualization - Visualizing the Structure, Behaviour, and Evolution of Software, Springer, 2007

[2] Paul M. Duvall, Introducing Continuous Integration as a Way to Improve Software Quality and Reduce Risk, informIT, 2007

[3] 폴 M. 듀발, 스티븐 M. 마티야스, 앤드류 글로버, 지속적인 통합, 위키북스, 최재훈 옮김, 2008

[4] “SW개발 품질관리 매뉴얼”, <SW공학센터>, 2013

[5] https://www.martinfowler.com/articles/continuousIntegration.html

[6] 폴 M. 듀발, 스티븐 M. 마티야스, 앤드류 글로버, 지속적인 통합, 위키북스, 최재훈 옮김, 2008

[7] Nicholas Whitehead, HOW-TO Continuous integration with Hudson - Open source CI server offers easy setup and configuration, JAVA-WORLD, 2008

[8] https://en.wikipedia.org/wiki/Bamboo_(software)

[9] https://en.wikipedia.org/wiki/TeamCity

[10] “CruiseControl 활용 가이드“, <소프트웨어자산뱅크>, <https://www.swbank.kr/helper/tool/toolView.do?name=CruiseControl>

[11] https://jenkins.io/

[12] “Ant 활용 가이드“, <소프트웨어자산뱅크>, <https://www.swbank.kr/helper/tool/toolView.do?name=Ant>

[13] https://en.wikipedia.org/wiki/Gradle

[14] Tim O’Brien, Maven - Sonatype이 만든 Maven 핵심 가이드,

장선진 옮김, 지앤선, 2010, 3-10쪽

[15] 권원일, 이현주, 최승희, 이승호, 박은영, 조현길, 개발자도 알아야 할 소프트웨어 테스팅 실무, STA, 2010, 81-83쪽

[16] http://www.fasoo.com/%EC%8A%A4%ED%8C%A8%EB%A1%9C%EC%9A%B0-sparrow

[17] http://www.suresofttech.com/ko/tool/code_inspector

[18] https://www.synopsys.com/software-integrity/resources/datasheets/coverity.html

[19] https://en.wikipedia.org/wiki/JArchitect

[20] 한동준, 김도균, 오픈소스 파워툴, 지앤선, 2017, 191-192, 225, 235-237쪽

[21] Markus, Sprunck. Findbugs - Static Code Analysis of Java, 2013

[22] 한혁수, 소프트웨어 공학의 소개, 홍릉과학출판사, 2017, 258-259, 268-271쪽

[23] 한동준, 김도균, 오픈소스 파워툴, 지앤선, 2017, 153-155, 169-170쪽

[24] https://en.wikipedia.org/wiki/TestNG

[25] https://confluence.atlassian.com/clover/about-clover-71598399.html

[26] http://emma.sourceforge.net/

[27] Claude Y. Laporte, Simon Alexandre, and Rory V. O'Connor, A Software Engineering Lifecycle Standard for Very Small Enterprises, 2008

[28] 장윤기, Practical 자바 유틸리티, 인사이트, 2016, 2-51쪽

## ABSTRACT

# Open Source Based Continuous Integration Process for Small Organizations

*Oh, Seung won*

*Dept. of Computer Science*

*The Graduate School*

*Sangmyung University*

A software project has a difficulty in recognizing defects that occur during the development process because of software invisibility. These problems have been a major cause of software project failure. In the field of software engineering, project management theory, various methods and process models have been developed as a solution to this problem. Also Tools have been developed and applied to build and maintain. Recently as the proportion of software in the current system, most projects have a collaborative environment in which multiple participants develop software systems together. At some point of implementation stage, modules programmed by each developers have to be integrated. At this time, errors that are not recognized in individual implementation usually come up. The problem of these errors are not enough time and too much codes

to be fixed. This can lead to project failure. In order to overcome these problems, CI(Continuous Integration) has been recognized as a proven practice in the field. CI integrates consistently, builds, and verifies source code in the early stages of the project to detect potential problem early. Therefore, hiring CI in the development is necessary in small organizations, but CI activities require expertise in the process area, and needs a lot of trial and error for customizing to the organization. There are commercial solutions for CI, but it is very expensive for small organization. As an alternative, there are open source tools which are provided free of cost. Taking all of these into account, using open source tools is the best solution. However, it is difficult to select proper tools, and learn how to interwork tools. So, specific CI guidelines are needed for small organizations.

Therefore, in this paper, I proposed an open source based CI process for small organizations. For this, I have studied the CI and characteristics of software development in small organizations. Based on the results of study, I selected the most appropriate open source tools and provided 4 stages CI process that consists of a stage for establishing foundation and 3 stages for build, static analysis and unit test. To demonstrate the effectiveness, I applied the proposed CI process to the undergraduate projects for two years and described the results and lesson-learned.