

# SW 개발·테스트 중심 DevOps 환경구축 및 활용

---

강사 정보

시네틱스 대표 한동준  
[handongjoon@gmail.com](mailto:handongjoon@gmail.com)  
[dongjoon.han@synetics.kr](mailto:dongjoon.han@synetics.kr)

# 교육 소개

---

## □ 목적

- DevOps 운영에 필수인 개발(Development) 환경의 자동화를 위하여 오픈소스 도구를 활용한 ALM(Application Lifecycle Management) 환경 구축 및 운영 역량을 확보

## □ 내용

- DevOps에서 ALM의 필요성에 구성에 대한 이해
- 주요 오픈소스 ALM 도구 이해
  - 단독으로 사용하는 방법
  - Maven, Jenkins, Eclipse 에서 사용하는 방법

## □ 교육에서 강조하는 부분

- 도구 설치, 도구 간 연동 방법
- Jenkins 중심의 활용 방법

## □ 교육 시간 내에 상대적으로 적게 다루는 부분

- 국내에 이미 책과 교육이 활성화 되어 있는 도구의 기본 사용 방법: Maven, Git

# 강의 순서

---

1. DevOps 와 오픈소스 개발 환경 자동화의 이해
2. 빌드: Maven
3. 버전 관리: Git
4. 지속 통합: Jenkins
5. 단위 테스트: Junit / Cobertura
6. 룰 기반 소스코드 분석: PMD
7. 의존성 분석: Jdepend
8. 소스코드 매트릭: JavaNCSS

# 시간표 – 1일차

구분	시간	내용	
1일차	09:00 ~ 09:30	교육 접수 및 안내	
	09:30 ~ 12:30	DevOps와 오픈소스와 개발 환경 자동화의 이해	<ul style="list-style-type: none"><li>- DevOps와 개발 환경 자동화</li><li>- ALM의 필요성과 구축 방법</li><li>- 오픈소스 활용의 장·단점</li></ul>
		Maven 빌드 관리	<ul style="list-style-type: none"><li>- 도구 설치 및 설정</li><li>- 빌드 단계 이해</li><li>- Goal의 활용</li><li>- pom.xml 이해</li></ul>
	12:30 ~ 13:30	점심	
	13:30 ~ 17:30	서버 환경 구성	<ul style="list-style-type: none"><li>- 클라우드 서비스 이해</li><li>- 임시 계정 생성</li><li>- JDK 설정</li><li>- WAS 설정</li></ul>
		Git을 이용한 버전 관리	<ul style="list-style-type: none"><li>- 분산 버전 관리의 개념</li><li>- Git 기본 사용법</li><li>- 서버 설치 및 설정</li><li>- 저장소 생성</li><li>- 클라이언트 도구 설치 및 설정</li></ul>
		Jenkins를 이용한 지속적 통합	<ul style="list-style-type: none"><li>- 지속적 통합 개요</li><li>- Jenkins 동작 방식</li><li>- 서버 도구 설치 및 설정</li><li>- 플러그인 설치 및 설정</li><li>- 기본 Job 생성 및 설정</li><li>- 권한 및 Node 설정</li></ul>

## 시간표 – 2일차

2일차	09:30 ~ 12:30	PMD를 이용한 소스코드 정적 분석	- 정적분석 개요 - 클라이언트 도구 설치 및 설정 - Maven 설정 및 활용 - Jenkins 설정 및 활용
	12:30 ~ 13:30	점심	
	13:30 ~ 17:30	Junit을 이용한 단위 테스트	- 단위 테스트 자동화 개요 - Assert를 이용한 단위 테스트
		Cobertura를 이용한 커버리지 분석	- 테스트 커버리지 이해 - 테스트 커버리지 활용 방안 - Maven 및 Jenkins 설정
		JDepend를 이용한 의존성 분석	- 소스코드 의존성 개요 - Object Oriented Metric 이해 - Maven 및 Jenkins 설정

# 개발자 PC 구성

---

## □ H/W

- 실습 PC 사양

## □ S/W

- JDK 1.8
- Eclipse(최신)
  - Git 플러그인(기본 설치)
  - m2e 플러그인 – Git
  - PMD, Jdepend, JavaNCSS 플러그인
- Maven 3.X
- TortoiseSVN Last Ver.
- Git Last Ver.

# 서버 구성 (1대)

## □ Azure (공통)

- Windows Server 사용

## □ S/W

- JDK 8
- Tomcat 8.0
- Jenkins

The image shows the Microsoft Azure portal interface. On the left, there is a sidebar with various service icons: All resources, Resource groups, App Services, SQL Database, MySQL (Document DB), Virtual machines, Load Balancers, Storage accounts, Virtual networks, Azure Active Directory, Monitor, Azure Advisor, Security Center, Billing, Help + support, and More services. The main area is the 'Dashboard' which displays several cards: 'Resource Group' (MyAzureApp), 'Web Front End' (MyAzureApp, MyAzureApp web service, MyAzureApp web service), 'Database' (MyAzureApp db database, MyAzureApp db database), 'Processes' (All runs, DTU percentage and Database size percentage past week), 'Application map' (MyAzureApp, Client side, Server side, Last 24 hours), and 'Cognitive Services' (Total Calls past week). Above the dashboard, there is a large promotional banner with the text 'Create your Azure free account today' and 'Get started with 12 months of free services'. It features two buttons: 'Start free >' and 'Or buy now >'. The URL 'www.microsoft.com/azuresdksetup' is visible at the bottom right of the banner.

# 오픈소스 파워툴 소개

---

# 오픈소스 파워툴 (지앤선, 2017.05)



오픈소스 ALM을 활용한 효율적이며  
유지 보수가 쉬운 소프트웨어 개발 환경 구축하기

한동준, 김도균 지음

志&蟬  
지앤선

## [참고] 오픈소스 파워툴 책 목차

장	도구명	용도	교육 포함
1	ALM	전체 구성	V
2	Maven	Java 빌드	V
3	Subversion	버전 관리	
4	Git	버전 관리	V
5	SCMManager	버전 관리 통합 서버	
6	Jenkins	지속 통합	V
7	JUnit	단위 테스트	V
8	Cobertura	테스트 커버리지	V
9	PMD	룰기반 정적분석	V
10	JavaNCSS	Java 매트릭	V
11	JDepend	패키지 의존성 분석	V
12	Redmine	이슈 관리	
13	Mylyn	이슈 관리 클라이언트	
14	시나리오	다 같이 해보기	V
15	NSIS	Windows 배포판 생성	

# 1 DevOps 와 오픈소스 개발 환경 자동화의 이해

---

시작은 딱딱하다. 책 전체에서 가장 재미없다.

왜 ALM이 필요하고, 오픈소스를 사용하는 장/단점을 설명한다.

유사한 상용 도구를 소개하지만, 기승전'오픈소스'로 마무리한다.

# DevOps란

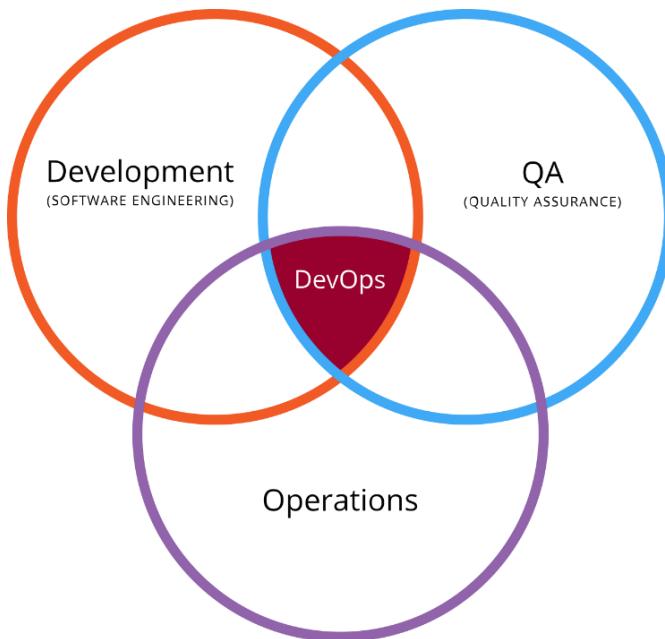
---

## □ 개요

- 소프트웨어의 개발(Development)과 운영(Operations)의 합성어
- 소프트웨어 개발자와 정보기술 전문가 간의 소통, 협업 및 통합을 강조하는 개발 환경이나 문화

## □ 목적

- 제품 출시까지 걸리는 기간(time to market) 단축
- 새로운 판의 더 낮은 실패율
- 픽스 간 짧아진 리드 타임(상품 생산 시작부터 완성까지 걸리는 시간)
- 복구 시 더 빠른 평균 시간 (새로운 릴리스의 충돌 및 시스템을 비활성화하는 상황에서)



출처: 위키피디아

# DevOps 파이프라인

작지만 자연스러운 반복

계획 및 추적

개발

빌드 및 테스트

배포

모니터링 및 운영



DevOps를 운영하기 위해 Dev / Ops 모든 단계의 자동화가 필수

## 자동화

자동화에 투자하면 반복적인 수동 업무를 없애고, 반복 가능한 프로세스와 안정적인 시스템을 만들 수 있습니다.

자동화 빌드, 테스트, 배포 및 프로비저닝은 이러한 단계를 갖추고 있지 않은 팀이 일반적으로 시작해야 하는 과정입니다. 모두에게 도움이 되는 시스템을 구축하는 것보다 개발자, 테스터 및 운영자가 협력해야 하는 것이 더 중요한 이유는 무엇입니까?

자동화를 처음 접하는 팀은 보통 지속적 배포로 시작합니다. 지속적 배포란 대부분 클라우드 기반 인프라로 촉진된 자동화 테스트를 통해 각 코드 변경 사항을 실행한 다음, 성공적인 빌드를 패키징하고, 자동화 배포를 사용하여 생산을 추진하는 과정입니다. 짐작할 수 있듯이 지속적 배포는 쉽고 빠르게 만들어낼 수는 있지만 ROI는 충분한 가치가 있습니다.

출처: Atlassian

**Dev, 즉 개발과 테스트 중심의 자동화를 ALM 이라고도 함**

# ALM: Application Lifecycle Management

## □ 정의 (위키피디아)

- SW 제품의 생명 주기(개발, 운영)를 관리하는 것
  - 운영을 포함한다는 의미에서 SDLC와의 차이가 있음
- 요구사항 관리, 설계, 테스팅, 유지보수, 지속 통합, 프로젝트 관리, 변경 관리, 형상 관리, 릴리즈 관리를 포함

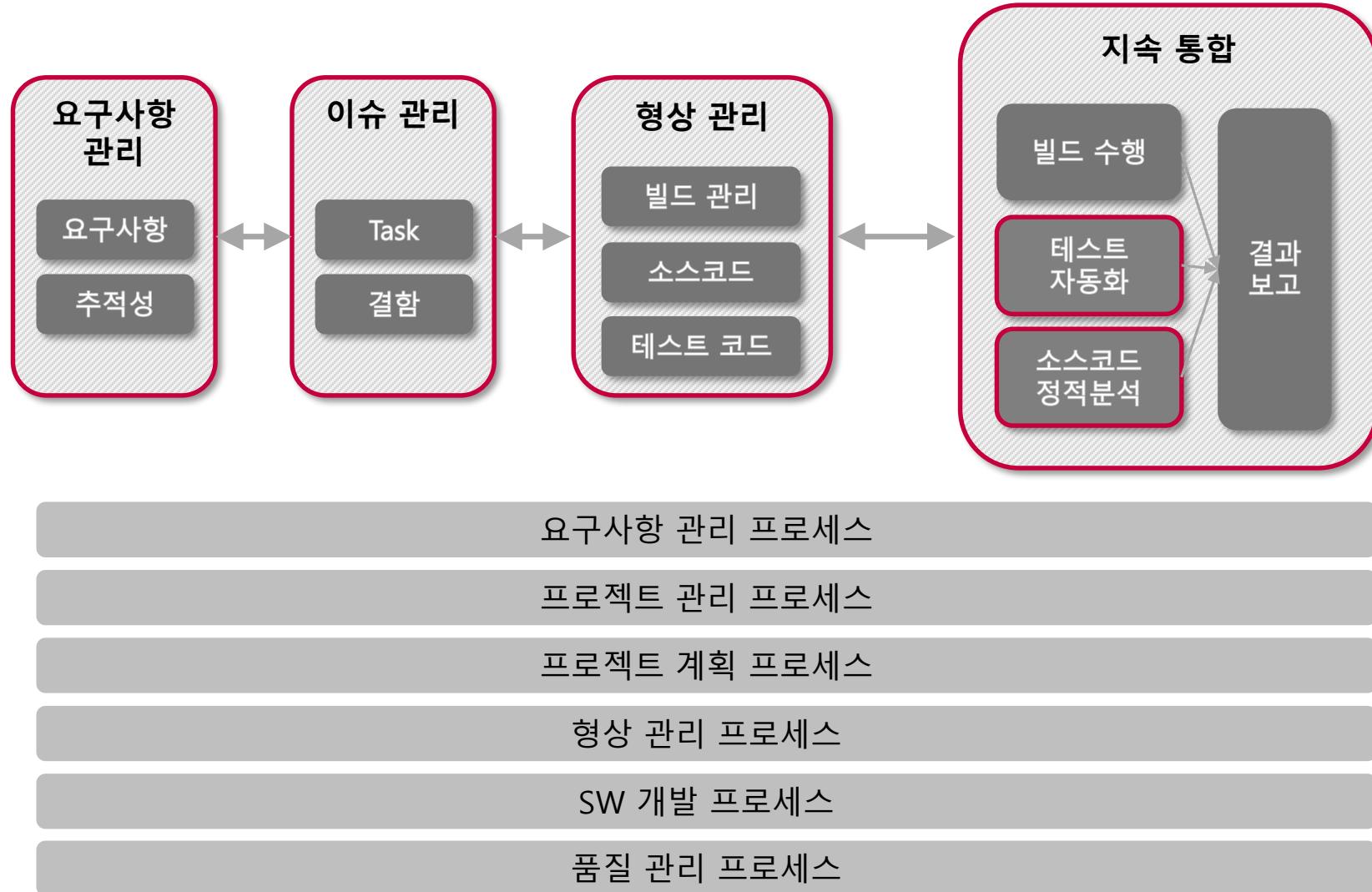


## □ 현대적 정의: Integrated ALM

- 프로세스 뿐만 아니라 분산 팀 환경 및 경량 개발 등 최근의 소프트웨어 개발 추세에 따라 도구 중심으로 구성
    - 도구를 중심으로 실시간 협업, 도구 연동과 가시화, 중앙 데이터 저장소 활용, 프로젝트 관리 및 보고 체계 구축
    - 누가, 언제, 왜, 무엇을 변경했는지 팀원 간 빠른 공유, 개발 및 관리 일관성, 품질 이슈 조기 해결이 목표
  - 한 벤더의 ALM 솔루션을 사용하는 시기는 지났으며, 필요에 따라 적절한 도구를 선택하는 추세로 Open API를 제공하여 도구 간 연동 지원
    - Integrated ALM의 핵심 영역인 형상 관리, 지속 통합 도구는 오픈소스의 점유율이 가장 높음  
(예. Git: 60%, SVN: 30% / Jenkins: 80%)
  - 주요 도구 구성: 요구사항 관리, 형상 관리, 소스코드 품질(테스트/정적분석), 지속 통합
- ※ DevOps의 기본 조건은 (Integrated) ALM의 구축 및 운영

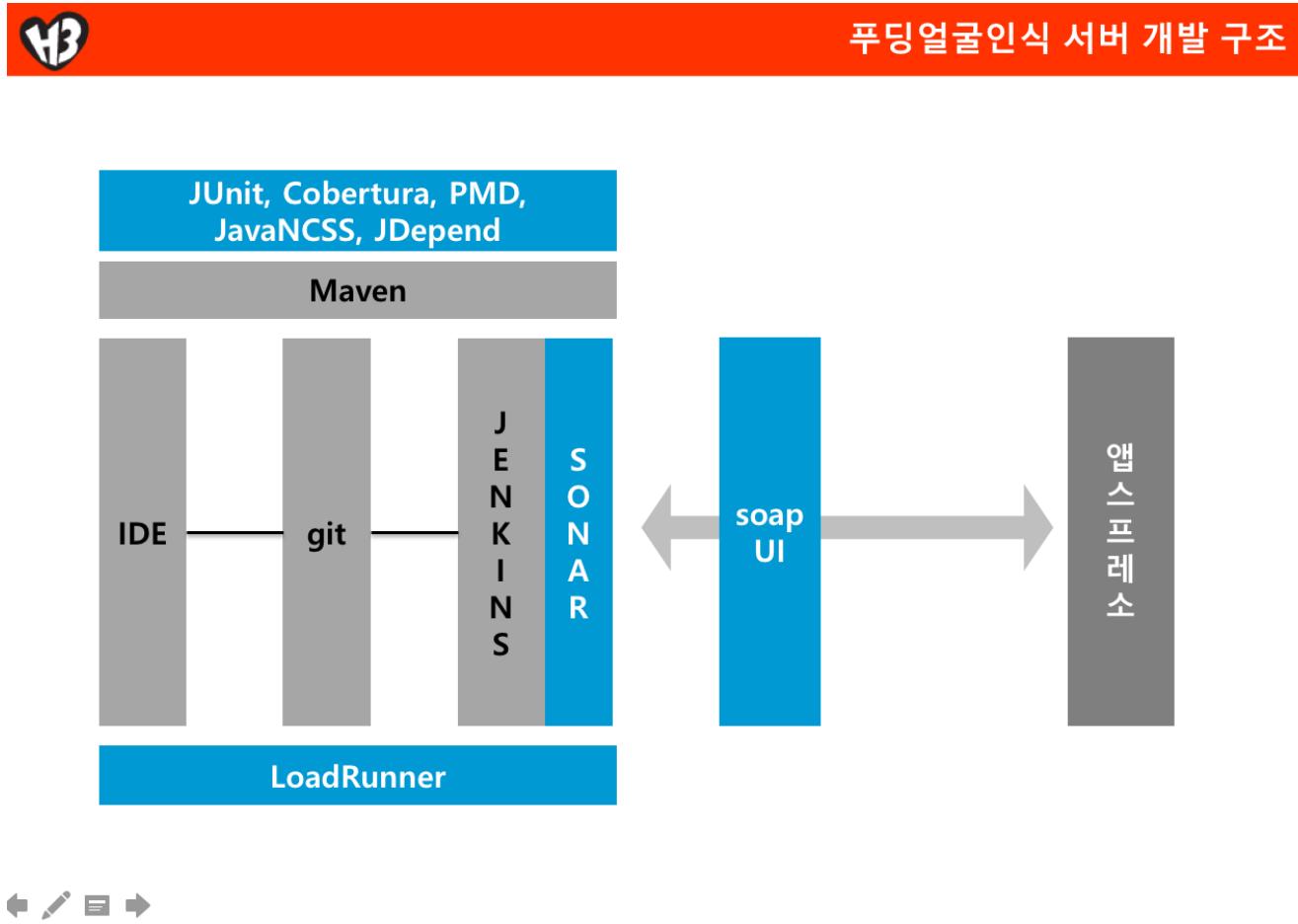
# ALM 구성

- ALM은 수행 절차를 정의한 프로세스와 이를 지원하는 도구로 구성되어야 함



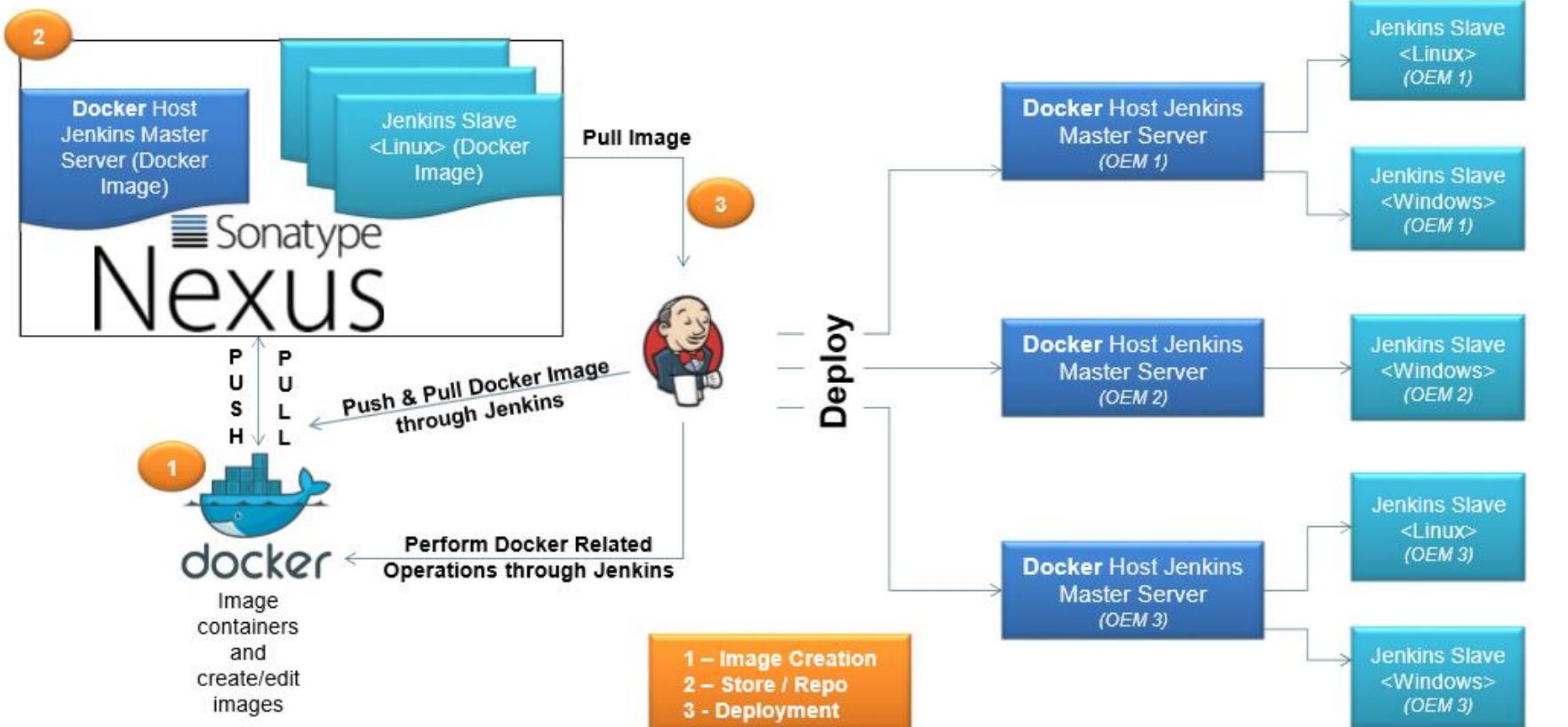
# 이제는 많이 하는 소프트웨어 개발 활동

## 2012 H3 컨퍼런스 발표 – 이미 ALM 사용사례 발표

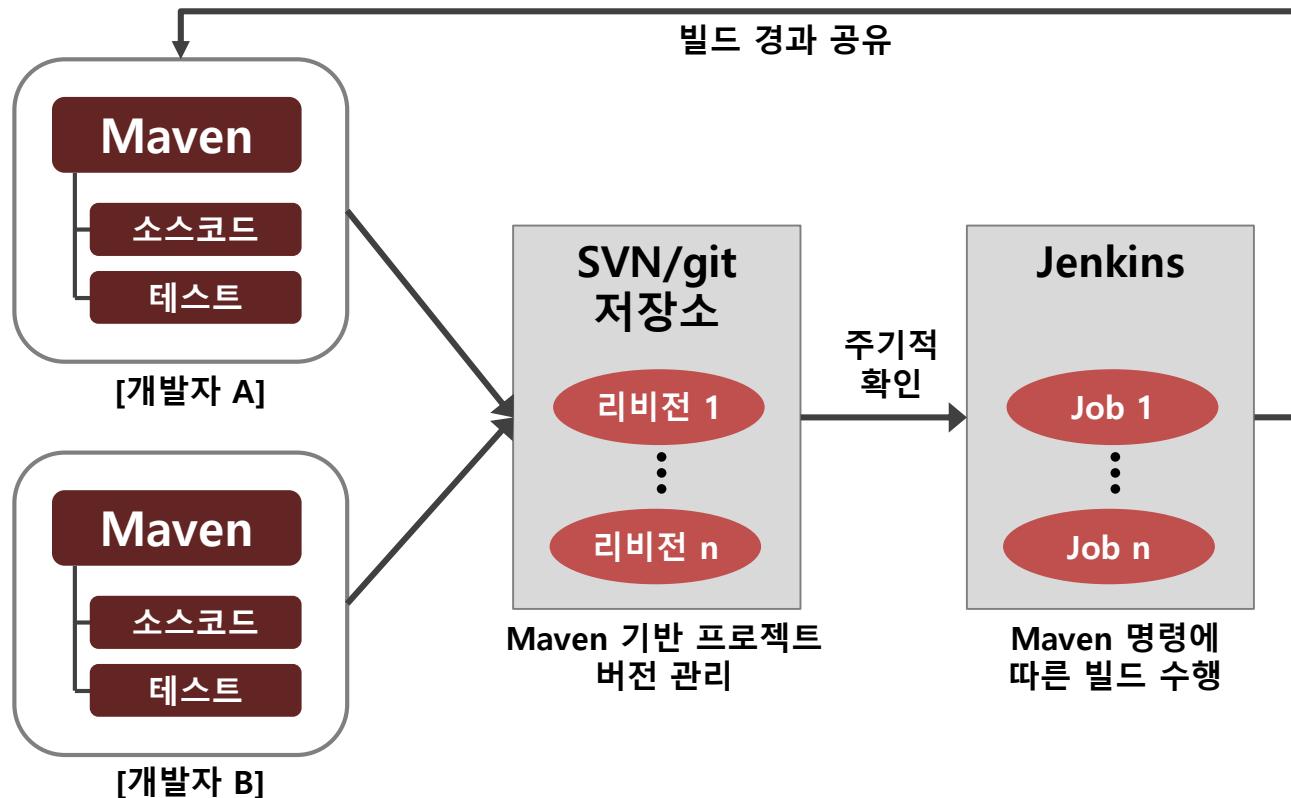


# 심지어 자동차 SW 개발에도 적용

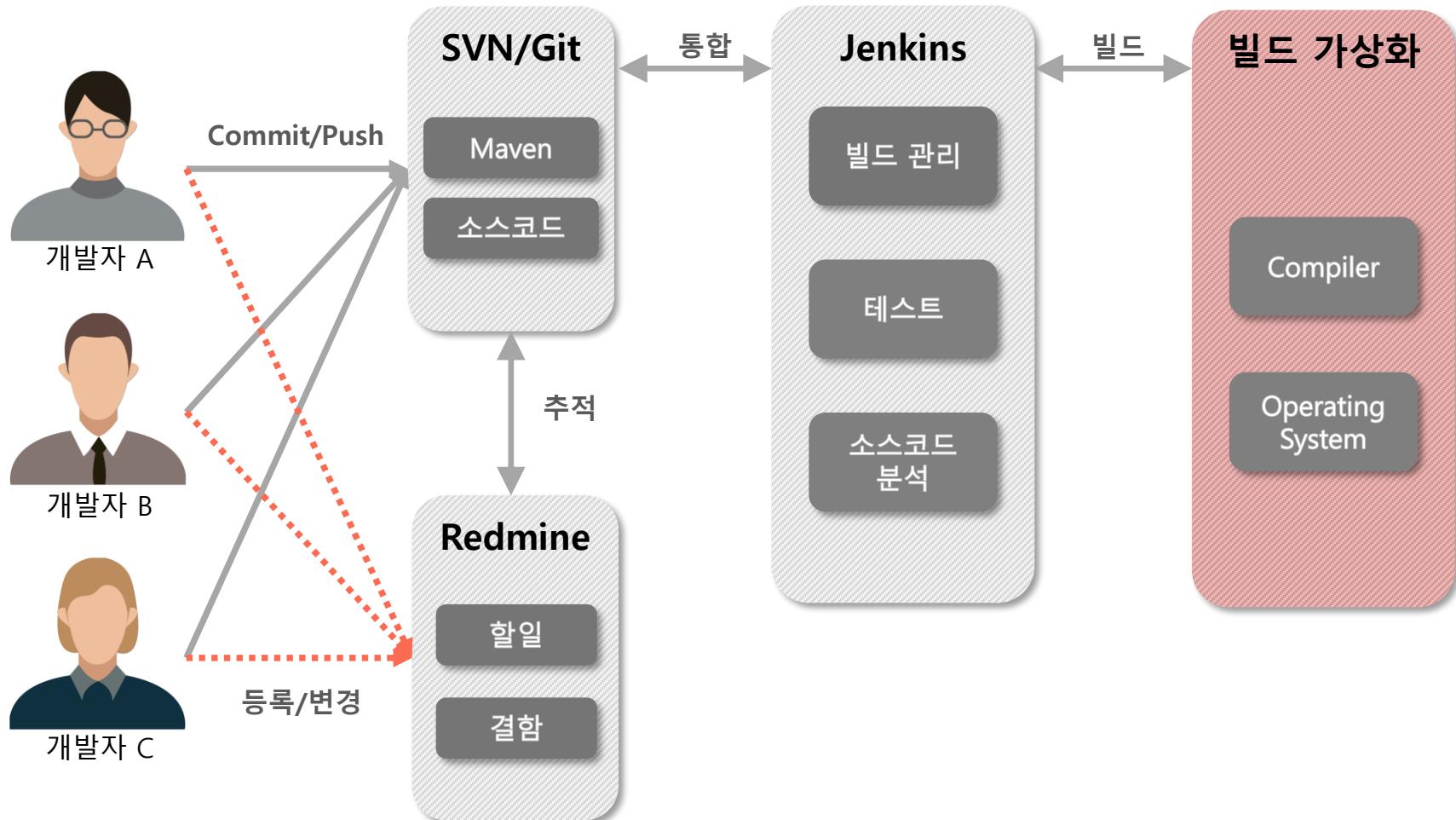
## Docker Deployment Architecture



# 일반적인 오픈소스 기반 ALM 구성



# 보다 확대한 구성



## 2 Maven

---

Java의 빌드 도구다. 유사한 도구로는 Ant와 Gradle이 있다.  
Maven을 개발을 위한 정해진 소스코드 구조를 생성해주고,  
빌드와 관련한 설정과 명령어를 제공한다.

# 도구 소개

---

## □ 도구 개요

- Java 빌드 관리 도구

## □ 사용 목적

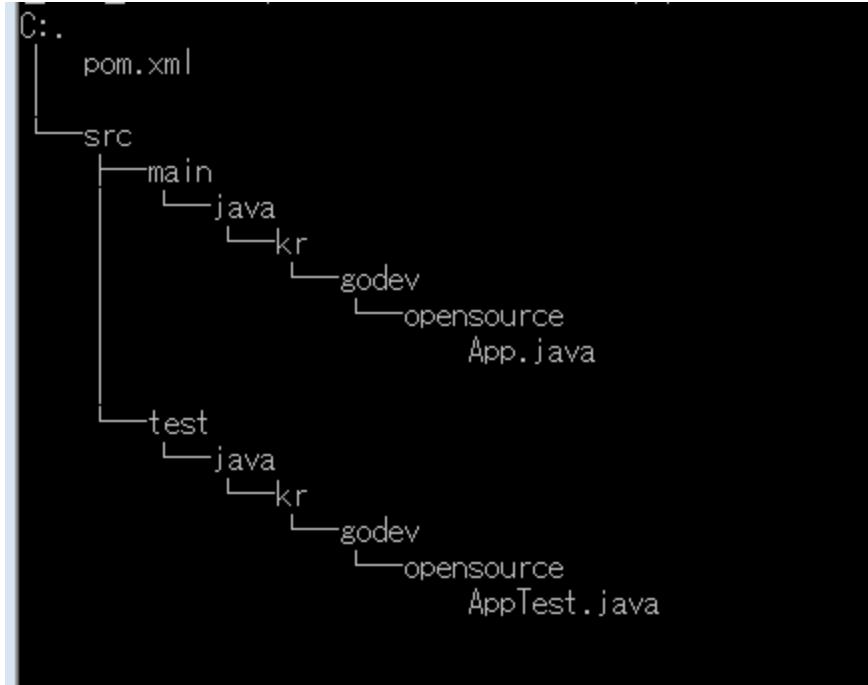
- 모두가 동일한 바이너리를 생성하기 위해서
- 정형화된 소스코드 구조를 사용하기 위해서
- 라이브러리 의존성을 관리하기 위해서
- 다른 도구와 쉽게 연동하기 위해서

## □ 일반적인 사용 방법

- 팀의 AA(Application Architect)가 Maven 구조 생성 및 의존성 관리
- pom.xml 파일에 라이브러리 의존성 및 플러그인 관리
  - QA의 경우, 품질 관련 플러그인 설정
- 개발 팀원은 정해진 패키지에서 소스코드 개발

# 도구 특징

## □ 정형화된 프로젝트 구성



## □ pom.xml

- pom.xml을 살펴보면, 사용하는 라이브러리와 구성 요소를 파악할 수 있음

# 라이프사이클과 단계

## □ 라이프사이클

라이프사이클	설명
clean	이전 빌드에서 생성된 산출물을 삭제하는 기능
default	프로젝트 빌드/배포와 관련된 기능
site	프로젝트의 사이트 문서 생성과 관련된 기능

## □ default의 단계

라이프사이클	설명
validate	정상적인 프로젝트이고 필요한 모든 정보가 준비되었는지 확인
compile	프로젝트의 소스코드를 컴파일
test	컴파일한 소스코드를 단위 테스트 프레임워크를 이용해 테스트
package	JAR와 같이 지정된 포맷으로 패키징
verify	패키지가 품질 수준을 만족하는지 검증
install	패키지를 로컬 저장소에 설치
deploy	최종 패키지를 원격 저장소로 복사

## □ Maven 명령

*mvn clean package site pmd:pmd jdepend:generate*

# 플러그인 Goal과 설정 방법

## □ 플러그인 Goal

### Plugin Documentation

Goals available for this plugin:

Goal	Report?	Description
pmd:check	No	Fail the build if there were any PMD violations in the source code.
pmd:cpd	Yes	Creates a report for PMD's CPD tool. See <a href="http://pmd.sourceforge.net/cpd.html">http://pmd.sourceforge.net/cpd.html</a> for more detail.
pmd:cpd-check	No	Fail the build if there were any CPD violations in the source code.
pmd:help	No	Display help information on maven-pmd-plugin. Call <code>mvn pmd:help -Ddetail=true -Dgoal=&lt;goal-name&gt;</code> to display parameter details.
pmd:pmd	Yes	Creates a PMD report.

## □ 플러그인 설정

### Usage

The PMD plugin generates PMD and CPD reports using the PMD code analysis tool.

To include a report with default rule sets and configuration in your project site, set the following in the <reporting> section of your POM:

```
<project>
  ...
  <reporting>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-pmd-plugin</artifactId>
        <version>3.1</version>
      </plugin>
    </plugins>
  </reporting>
  ...
</project>
```

# 다운로드

---

□ 대상: 개발자 PC, 빌드 서버

□ 사전조건: JDK의 설치

□ 다운로드

- Maven 웹사이트 다운로드: <https://maven.apache.org/download.cgi>
- 최신버전: 3.6. X
- Windows 개발 환경을 위한 'Binary zip archive' 다운로드

## Files

Maven is distributed in several formats for your convenience. Simply pick a ready-made binary distribution archive and follow the [installation instructions](#). Use a source archive if you intend to build Maven yourself.

In order to guard against corrupted downloads/installations, it is highly recommended to verify the signature of the release bundles against the public [KEYS](#) used by the Apache Maven developers.

	Link	Checksum	Signature
Binary tar.gz archive	<a href="#">apache-maven-3.5.2-bin.tar.gz</a>	<a href="#">apache-maven-3.5.2-bin.tar.gz.md5</a>	<a href="#">apache-maven-3.5.2-bin.tar.gz.asc</a>
Binary zip archive	<a href="#">apache-maven-3.5.2-bin.zip</a>	<a href="#">apache-maven-3.5.2-bin.zip.md5</a>	<a href="#">apache-maven-3.5.2-bin.zip.asc</a>
Source tar.gz archive	<a href="#">apache-maven-3.5.2-src.tar.gz</a>	<a href="#">apache-maven-3.5.2-src.tar.gz.md5</a>	<a href="#">apache-maven-3.5.2-src.tar.gz.asc</a>
Source zip archive	<a href="#">apache-maven-3.5.2-src.zip</a>	<a href="#">apache-maven-3.5.2-src.zip.md5</a>	<a href="#">apache-maven-3.5.2-src.zip.asc</a>

# 설치 및 설정 - 1

---

## □ 설치

- 다운로드 받은 파일 압축 풀기
- 적절한 폴더로 이동
  - 본 강의에서는 "C:\Dev\_tools\Maven"에 이동
  - Setup.exe 형식의 설치 파일이 아닌, Portable 형식

## □ 설정

- 시스템 환경변수에 'MAVEN\_HOME' 추가
  - 값은 C:\Dev\_tools\Maven
- 시스템 환경변수의 path에 Maven bin 폴더 추가
  - 값은 C:\Dev\_tools\Maven\bin

## ※ 참고: JDK 설치 및 설정

- 설치: 기본 설치
- 설정
  - 시스템 환경변수에 'JAVA\_HOME' 추가: JDK 설치 폴더(JRE 아님!!)
  - 시스템 환경변수의 path에 JDK bin 폴더 추가: JDK 설치 폴더의 bin 폴더

## 설치 및 설정 - 2

### □ 시스템 환경변수가 제대로 적용되도록 재부팅 후 확인

- CMD를 열고, "mvn -version" 실행



```
C:\>mvn -version
Apache Maven 3.2.5 (12a6b3acb947671f09b81f49094c53f426d8cea1; 2014-12-15T02:29:2
3+09:00)
Maven home: C:\maven\bin..
Java version: 1.7.0_75, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.7.0_75\jre
Default locale: ko_KR, platform encoding: MS949
OS name: "windows 7", version: "6.1", arch: "amd64", family: "windows"
C:\>
```

# 프로젝트 생성 - 1

## □ C:\sourcecode\maven1에서 생성

## □ 명령

- mvn archetype:generate -DgroupId=kr.godev.opensource -DartifactId=my\_project  
-DinteractiveMode=false

프로젝트 명

```
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----
[INFO]
[INFO] ] maven-archetype-plugin:2.2:generate (default-cli) @ standalone-pom ]
[INFO]
[INFO] [ maven-archetype-plugin:2.2:generate (default-cli) @ standalone-pom [
[INFO]
[INFO] --- maven-archetype-plugin:2.2:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Batch mode
[INFO] No archetype defined. Using maven-archetype-quickstart (org.apache.maven.archetypes:maven-archetype-quickstart:1.0)
[INFO] -----
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0
[INFO] -----
[INFO] Parameter: groupId, Value: kr.godev.opensource
[INFO] Parameter: packageName, Value: kr.godev.opensource
[INFO] Parameter: package, Value: kr.godev.opensource
[INFO] Parameter: artifactId, Value: my_project
[INFO] Parameter: basedir, Value: C:\
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\my_project
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 5.529s
[INFO] Finished at: Tue Feb 04 22:16:07 KST 2014
[INFO] Final Memory: 10M/26M
[INFO] -----
```

## 프로젝트 생성 - 2

### □ 폴더 생성 결과

```
C:.
└── pom.xml

└── src
    ├── main
    │   └── java
    │       └── kr
    │           └── godev
    │               └── opensource
    │                   └── App.java
    └── test
        └── java
            └── kr
                └── godev
                    └── opensource
                        └── AppTest.java
```

## pom.xml

---

- Maven 프로젝트의 대부분의 설정 정보를 담고 있는 파일
- 기본 생성된 pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>kr.godev.opensource</groupId>
  <artifactId>my_project</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>my_project</name>
  <url>http://maven.apache.org/</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

# 플러그인 설정 추가

---

## □ PMD 플러그인 추가 예제

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>kr.godev.opensource</groupId>
  <artifactId>my_project</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>my_project</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-pmd-plugin</artifactId>
        <version>3.1</version>
      </plugin>
    </plugins>
  </build>

</project>
```

## 최초 Compile 실습

---

- mvn package 실행

Build Success 결과 확인

# [실습] Junit 4 빌드

## □ Github에서 Junit 4 소스코드 다운로드

- <https://github.com/junit-team/junit4>
- mvn install 실행

A programmer-oriented testing framework for Java. <http://junit.org/junit4/>

2,195 commits 5 branches 20 releases 138 contributors EPL-1.0

Branch: master New pull request Find file Clone or download

stefanbirkner Don't build with Oracle JDK 7 on Travis anymore ...

.mvn/wrapper Build with Maven 3.1.1 (using Maven Wrapper)

.settings Revert on request of kcooney:

doc Build with Maven 3.1.1 (using Maven Wrapper)

lib Add Hamcrest source JAR for easy reference

src Fix dead link to the ant task in FAQ (#1478)

.classpath Add resource source folders to Eclipse project

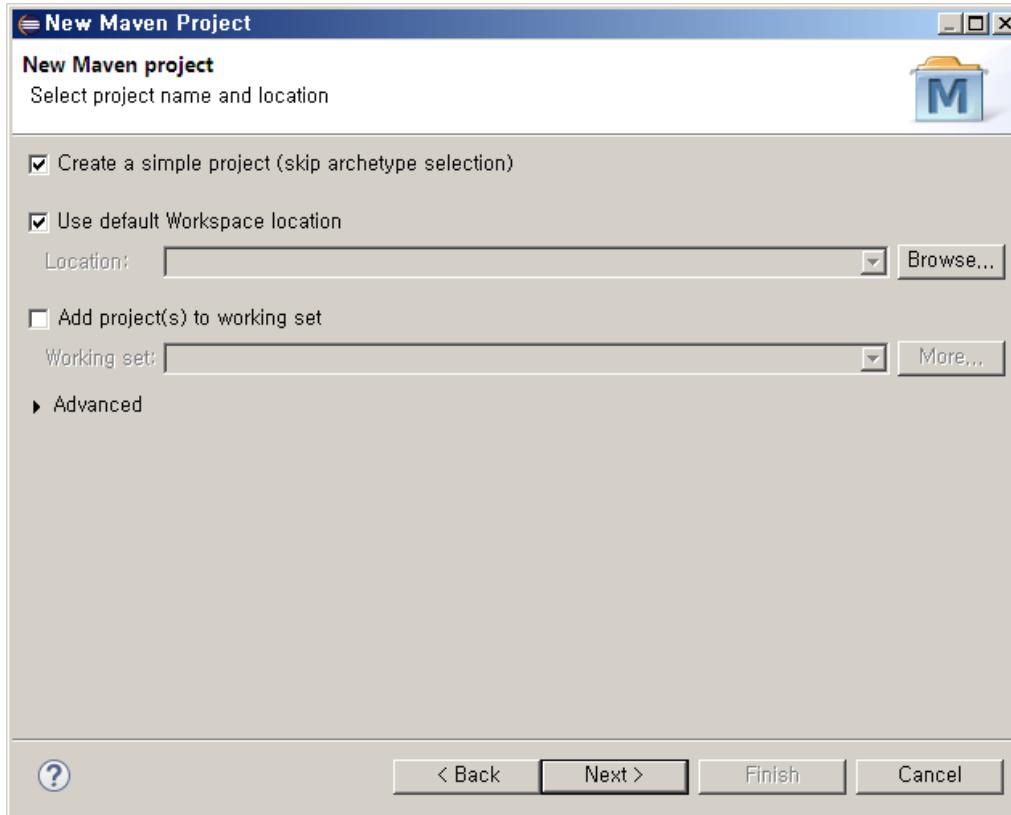
Clone with HTTPS Use Git or checkout with SVN using the web URL.  
https://github.com/junit-team/junit4.git

Open in Desktop Download ZIP

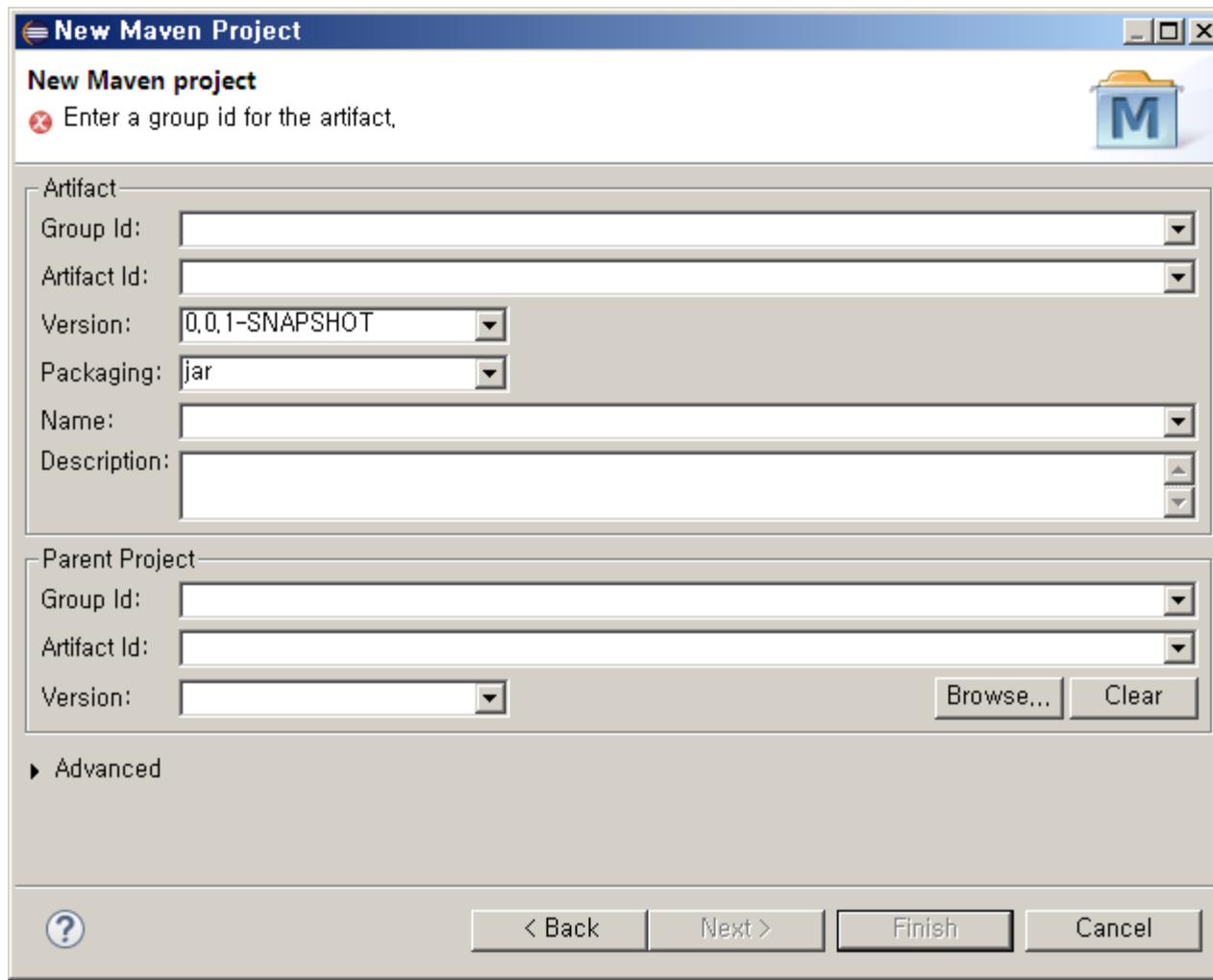
5 years ago  
4 months ago  
3 years ago

## [Eclipse] 사용 개요

- 최근 버전(Indigo 이후 버전)은 Maven 플러그인이 내장
- Maven 프로젝트 생성
  - [File] - [New] - [Other] 를 클릭하여 프로젝트 생성 마법사 실행

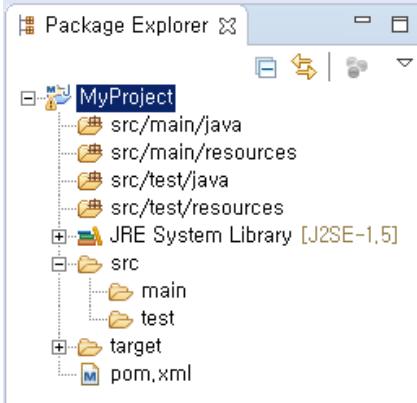


## [Eclipse] 프로젝트 정보 입력



# [Eclipse] 프로젝트 생성 확인 및 pom 변경

## □ 프로젝트 생성 확인



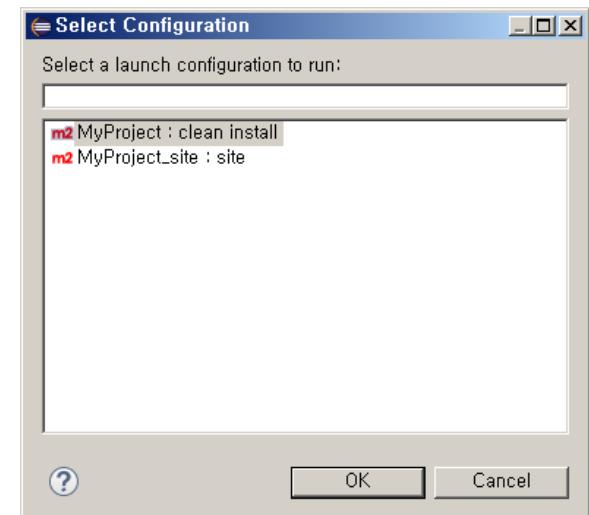
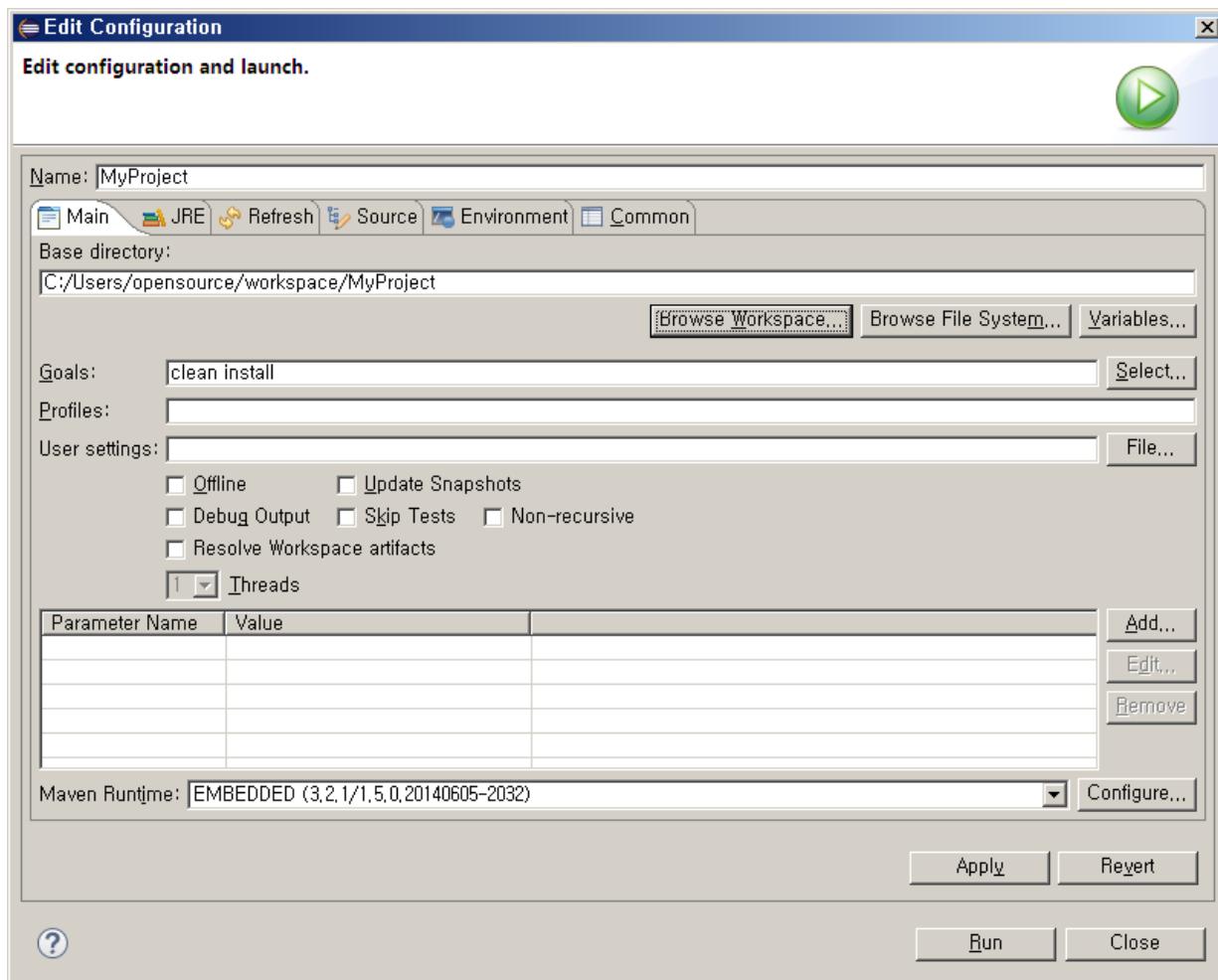
## □ pom 파일 변경

The screenshot shows the Maven Project Properties dialog for the MyProject/pom.xml file. The Overview tab is selected, displaying the following artifact information:

Artifact	Project
Group Id: kr.godev.opensource	Organization
Artifact Id: *MyProject	SCM
Version: 0.0.1-SNAPSHOT	Issue Management
Packaging: jar	Continuous Integration

Below the artifact details, there are sections for Parent, Properties, and Modules. The Modules section contains a "New module element" button. At the bottom, tabs for Overview, Dependencies, Dependency Hierarchy, Effective POM, and pom.xml are visible.

# [Eclipse] Build 설정 및 실행



# 서버 설정



## 가입 후 Azure에 서버 추가

---

### □ Azure에 Windows VM 추가

- 네트워크 보안에서 80, 8080 인바운드 허용

### □ 설치 필요 SW

- JDK 1.8
- Maven 3.6.X
- Tomcat 8

## 3      Git

---

Git은 핫하다. 이미 여러 책과 좋은 설명서가 공개되어 있다.

Git의 기본적인 이해와 Jenkins에서 연동하기 위한 수준 정도로 다룬다.

# 도구 소개

---

## □ 도구 개요

- 버전 관리의 대표 도구

## □ 목적 : 커뮤니케이션

- 표준: 산출물의 무결성 확보
- 소스코드 버전 관리
  - 마음의 평화: 실수해도 이전 버전으로 빠르게 되돌리기
- 왜 이렇게 수정하였지 확인
- 과거의 내가 얼마나 멋진 개발자였는지 확인

## □ 일반적인 사용 방법

- 별도의 저장소 운영 서버 사용
  - Git: GitHub, Bitbucket, Gitlab, **Giblit**
  - 통합 저장소: SCMManger
- 클라이언트는 Core나 별도의 확장 도구 사용
  - Eclipse 플러그인: 기본 포함
  - Tortoise 시리즈
  - Sourcetree

# Git 개요

---

## □ 역사

- 2005년 리누스 토발즈를 중심으로 개발
- 현재 GitHub에서 운영

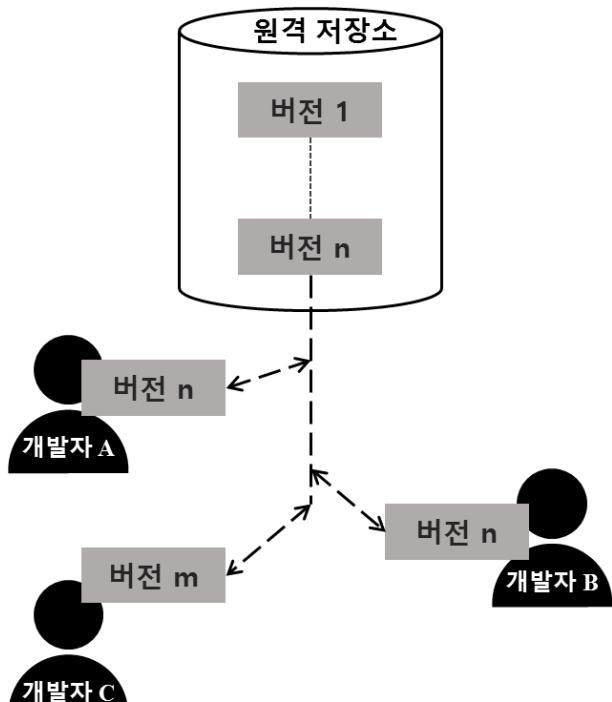
## □ 사용 추세

- 리눅스, Git을 포함하여 주요 오픈소스는 Git(GitHub) 사용
- 회사의 신규 프로젝트는 Git을 많이 사용하는 추세

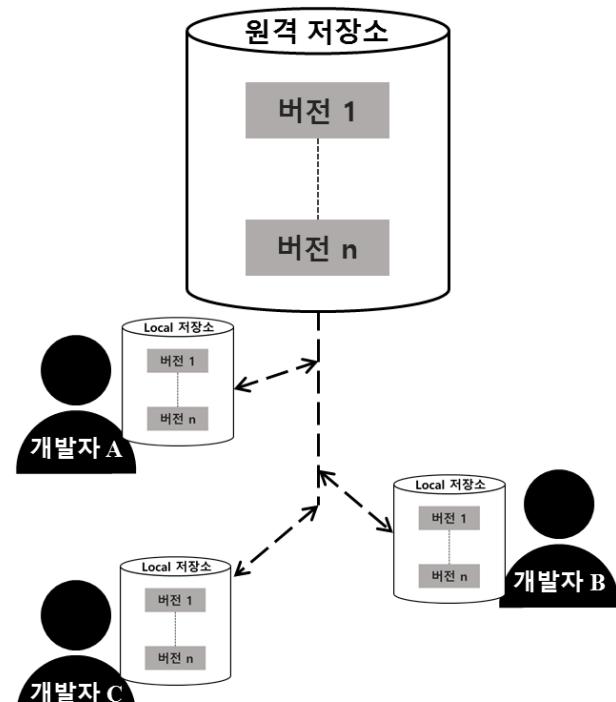
# 버전 관리 방식

## □ 분산 시스템 기반

- 원격 저장소와 동일한 로컬 저장소를 각 개발자가 유지
  - SVN은 최종 리비전만 각 개발자가 유지



Subversion



Git

## 기본 용어

---

영문	설명
Init	기존 폴더에 저장소를 생성
Clone	원격 저장소를 내 로컬에 복사
Push	내 저장소의 변경사항을 원격 저장소에 반영
Pull	원격 저장소의 변경사항을 내 저장소에 반영
Add	수정된 파일을 Staging Area에 등록하여 Staged 상태 만들
Check-out	작업할 브랜치를 지정
Commit	Staged 상태의 파일을 로컬 저장소에 반영
Remote Repository	공유를 위해 사용하는 저장소, 서버의 역할

# Git의 Branch

---

## □ 브랜치(Branch)

- 원본의 복사본
- 대부분의 VCS(Version Control System)에서 지원
- 용도
  - 특정 기능을 원본에 영향을 주지 않고 개발/테스트 하기 위해
  - 베이스라인을 설정하기 위해

## □ Subversion 브랜치 사용의 어려움

- 브랜치 생성 시, 원본을 특정 폴더에 복사함 (용량 \* 2)
  - 예) Visual C++의 3기가 소스코드의 단계 별/릴리즈 별 브랜치 10회 = 3기가 \* 10

## □ Git 브랜치의 용이함

- Git은 변경 사항을 저장하지 않고, 변경을 스냅샷 포인터로 관리
- 서버 자체를 복제하기 때문에, 포인터를 이용한 브랜치 생성 및 이동 가능
  - 예) Visual C++의 3기가 소스코드의 단계 별/릴리즈 별 브랜치 10회 = 40byte \* 10

## □ Git를 보다 잘 사용하고 싶다면, 브랜치를 어떻게 사용하는가에 좌우

## 브랜치 전략

---

### □ 업계에서 3개의 브랜치 전략을 추천

1. Subversion 처럼 사용하는 방식
2. GitFlow를 따르는 방식
3. Pull Request(GitHubFlow)를 따르는 방식

## 1) Subversion 처럼 사용하는 방식

---

### □ 방법

- Master 브랜치(기본 브랜치)를 공유하는 방법

### □ 장점

- Subversion 사용자가 쉽게 Git에 적용 가능

### □ 단점

- 브랜치를 사용하지 않음

### □ 추천하는 조직

- 기존 Subversion 등 중앙 집중식 도구에서 Git으로 넘어오는 조직
- 급격한 변화가 어려운 조직

## 2) GitFlow를 따르는 방식

---

### □ 방법

- GitFlow 도구를 설치하고, GitFlow의 프로세스를 준수
  - Git의 확장으로, Git의 브랜치 사용을 쉽게 적용할 수 있도록 확장 도구로 제공

### □ 장점

- 정형화된 브랜치 전략의 사용
- 참고할 수 있는 자료가 많음

### □ 단점

- 임베디드 SW보다 릴리즈가 잦은 웹 서비스 분야에 보다 적합
  - 임베디드 SW에도 적용이 안되는 것은 아님
  - K사의 사례

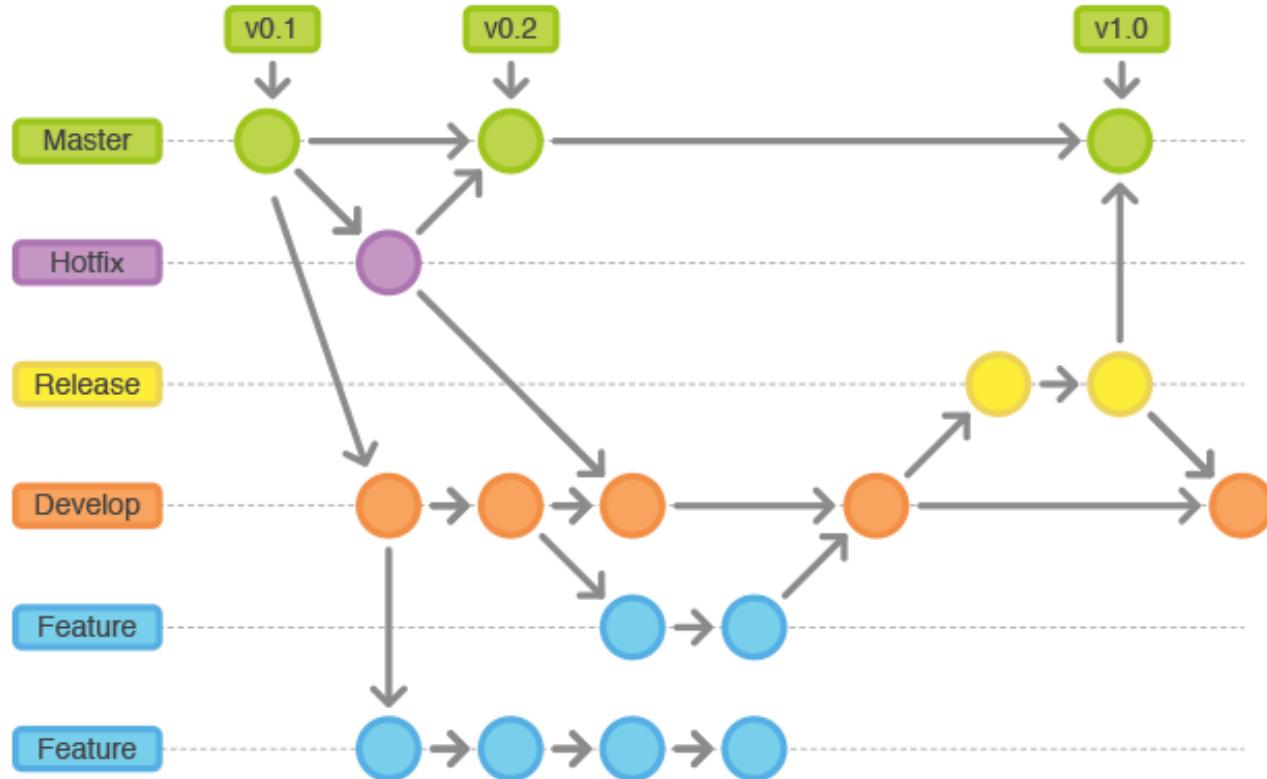
### □ 추천하는 조직

- 브랜치를 처음 적용하는 조직
- 중규모(협업 인원이 20명) 이하인 경우

## 2) GitFlow를 따르는 방식 - GitFlow

### ❑ Vincent Driessen가 개발한 브랜치 전략 및 지원 도구

- Sourcetree와 같은 Git 도구에서 기본 지원
- Master: 운영 환경과 동일한 소스코드가 있는 브랜치



### 3) Pull Request(GitHubFlow)를 따르는 방식

---

#### □ 방법

- 개발자가 저장소를 Fork(복제) 또는 Branch하여 수정하고, 변경 사항의 적용을 요청(Pull Request)
  - GitHub에서 오픈소스에 Contribute 하는 대표적인 방식

#### □ 장점

- 관리자의 검토 후, 개발자의 소스코드 반영 여부를 결정 가능

#### □ 단점

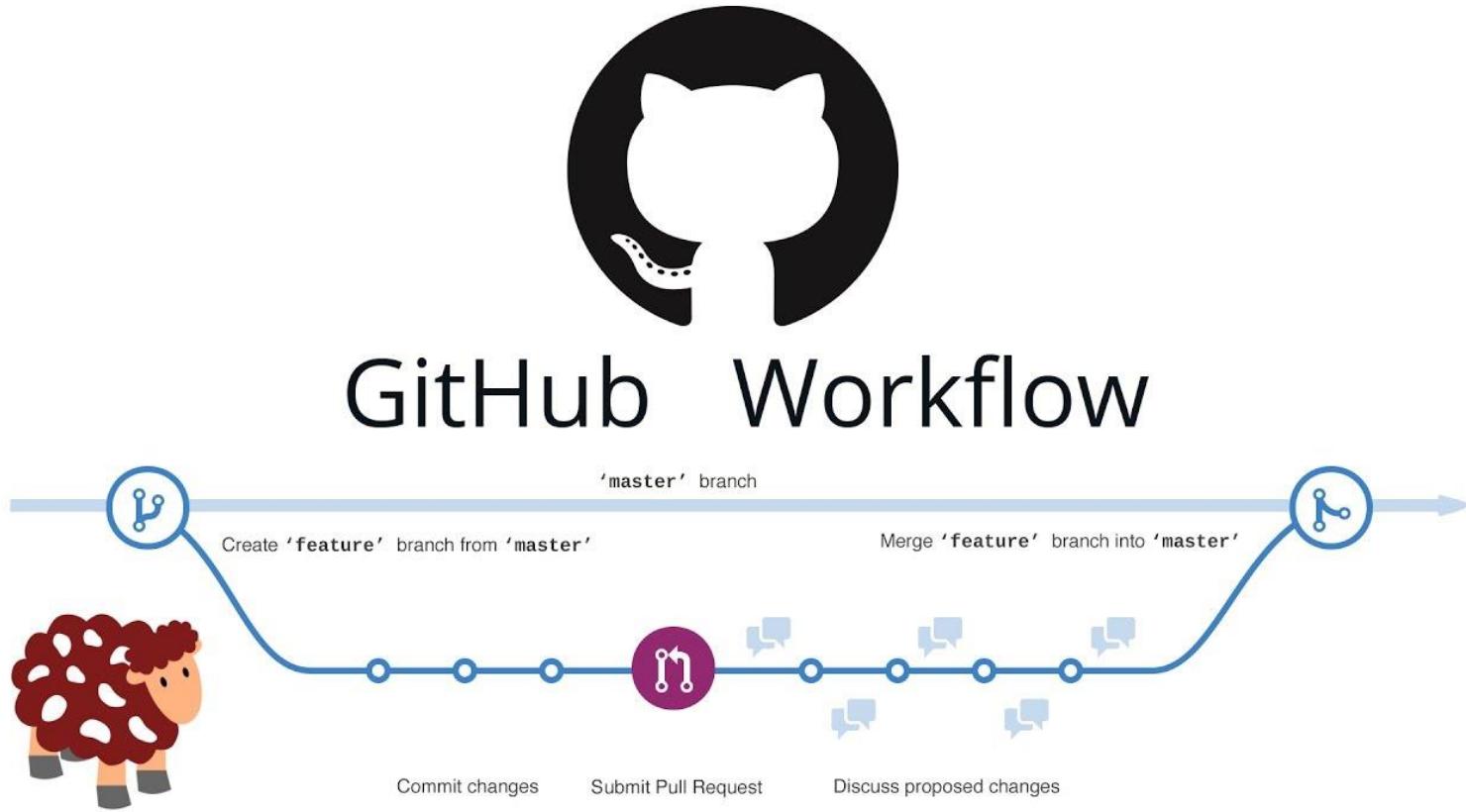
- Pull Request를 처리하기 위한 관리자 필요
- 원활한 사용을 위해서는 Pull Request를 지원하는 서버 도구 사용 필요

#### □ 추천하는 조직

- 협력 업체와 동일한 저장소로 공유하는 경우
- 대규모 개발인 경우

### 3) Pull Request(GitHubFlow)를 따르는 방식

- GitHub에서 운영되는 대부분의 오픈소스 개발에 적용



# [서버 설정] GitHub을 원격 저장소로 이용하기

## □ <https://github.com/>

- 회원가입하면, public 저장소 무제한 생성 가능 (private도 3개 가능)
  - 다른 사람에게도 소스코드가 보이지만, 커밋하는 사람을 지정할 수 있음
  - Private 저장소는 최소 월 \$7에서 시작

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner  /

Great repository names are short and memorable. Need inspiration? How about [fictional-barnacle](#).

Description (optional)

**Public**  
Anyone can see this repository. You choose who can commit.

**Private**  
You choose who can see and commit to this repository.

Initialize this repository with a README  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** | Add a license: **None** | ⓘ

**Create repository**

# [저장소 Fork] JUnit4 저장소 Fork

## □ Fork: 다른 사람의 저장소를 수정하기 위해, 내 저장소로 복사해 오는 기능

- 향후 Pull Request를 통해 원래 저장소에 병합
- 오픈소스에서 Contribute에 많이 사용하는 방식

A programmer-oriented testing framework for Java. <http://junit.org/junit4/>

2,195 commits	5 branches	20 releases	138 contributors	EPL-1.0	
Branch: master ▾	New pull request	Create new file	Upload files	Find file	Clone or download ▾
stefanbirkner Don't build with Oracle JDK 7 on Travis anymore ...			Latest commit a832c5a on 16 Oct		
.mvn/wrapper	Build with Maven 3.1.1 (using Maven Wrapper)	2 months ago			
.settings	Revert on request of kcooney:	4 years ago			
doc	Build with Maven 3.1.1 (using Maven Wrapper)	2 months ago			
lib	Add Hamcrest source JAR for easy reference	6 years ago			
src	Fix dead link to the ant task in FAQ (#1478)	4 months ago			
.classpath	Add resource source folders to Eclipse project	3 years ago			
.gitattributes	Use Unix-style line endings. (#1405)	11 months ago			
.gitignore	Use Unix-style line endings. (#1405)	11 months ago			

## [저장소 Fork] 내 저장소에 Fork 확인

This screenshot shows a GitHub repository page for the forked version of JUnit 4. The repository is named `handongjoon/junit4`, which is a fork from `junit-team/junit4`. The page includes navigation links for `Pull requests`, `Issues`, `Marketplace`, and `Explore`. It also features a bell icon for notifications, a plus sign for creating new repositories, and a grid icon for switching between list and grid view.

The repository summary section displays the following metrics:

- 2,196 commits
- 5 branches
- 20 releases
- 138 contributors

Branch selection dropdown: `master` (selected). Buttons for `New pull request`, `Create new file`, `Upload files`, `Find file`, and `Clone or download`.

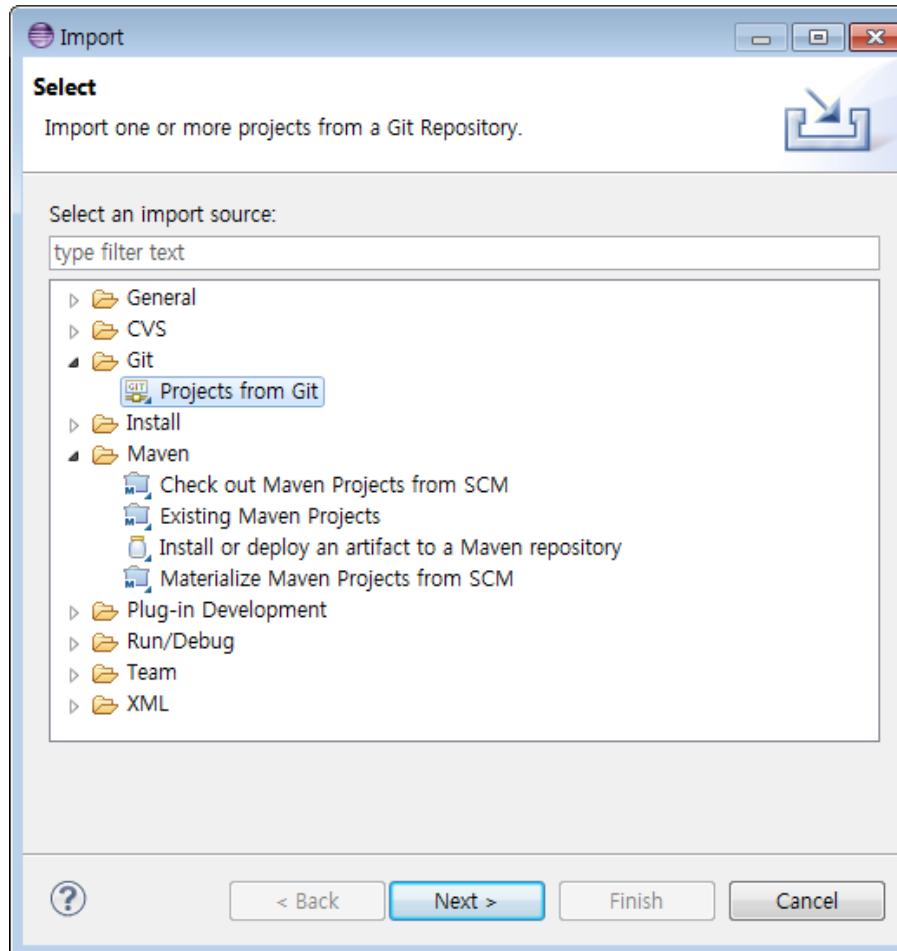
A message indicates that the current branch is 1 commit ahead of the upstream master branch. The commit history table lists the following recent commits:

Author	Commit Message	Time
lexia q123	Build with Maven 3.1.1 (using Maven Wrapper)	Latest commit db13b1a an hour ago
.mvn/wrapper	Revert on request of kcooney:	2 months ago
.settings	Build with Maven 3.1.1 (using Maven Wrapper)	4 years ago
doc		2 months ago

# [Eclipse] 내 로컬로 Clone

## □ SVN과 동일하게, Import 기능 사용

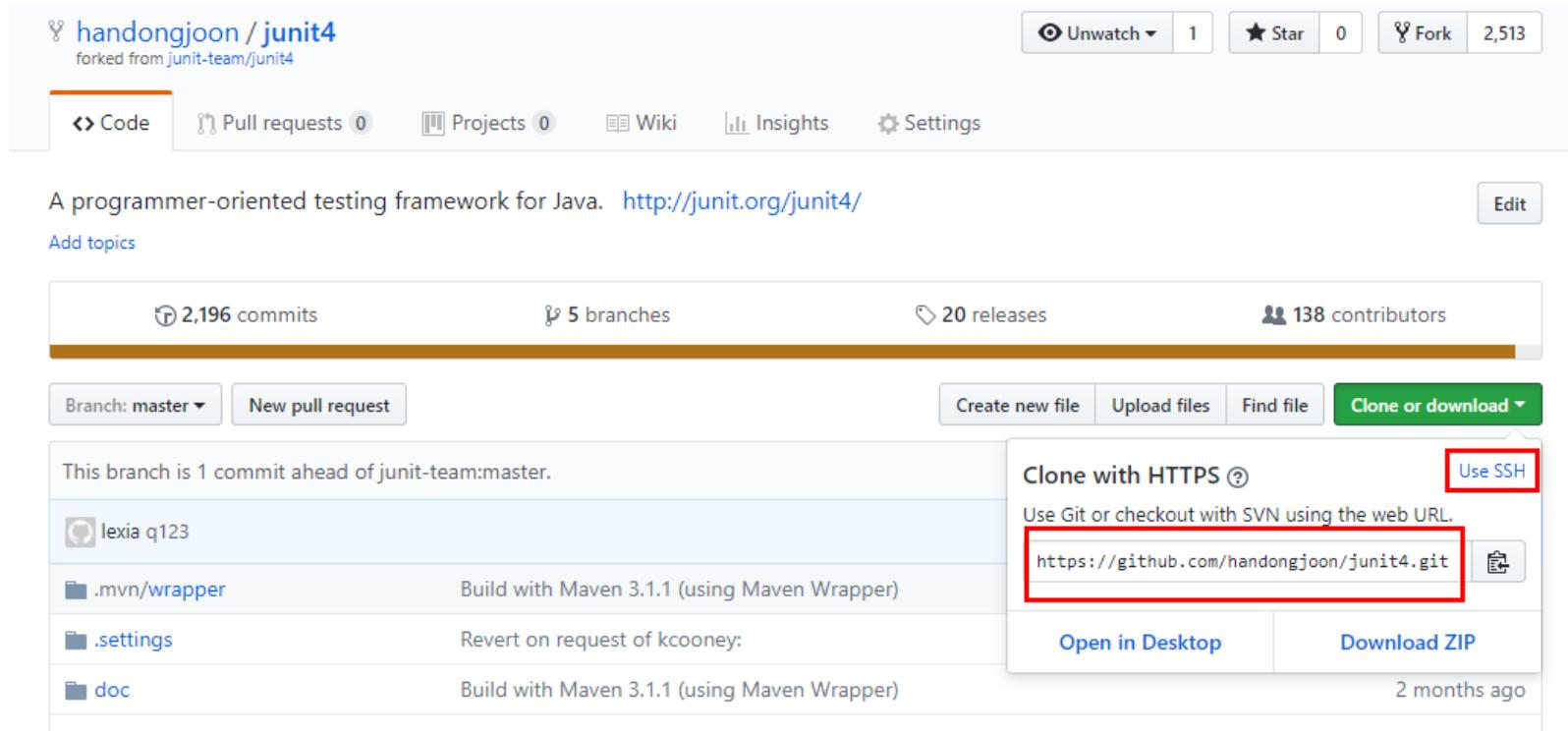
- Maven 프로젝트 이므로, Maven 하위의 체크아웃 메뉴 이용



# [Eclipse] 내 로컬로 Clone

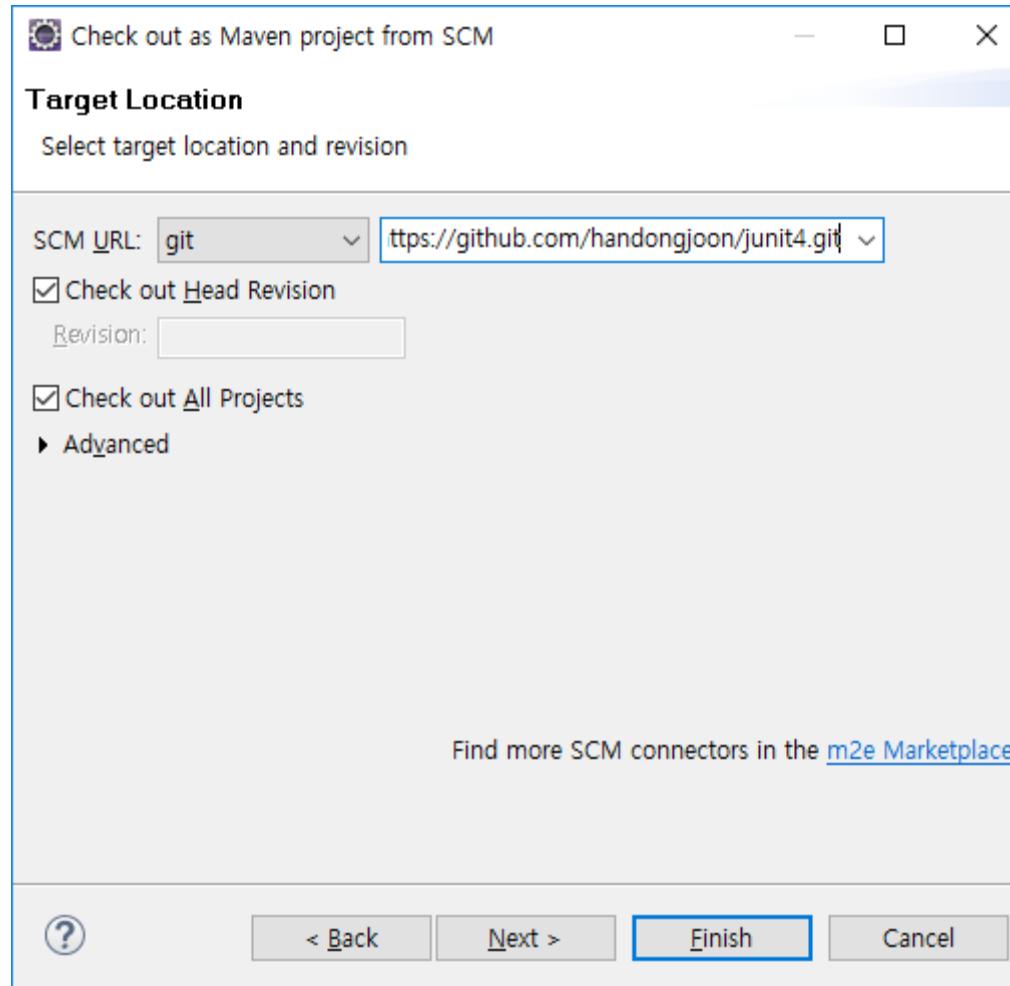
## □ Fork 한 저장소의 Clone을 위한 주소 받아오기

- ssh 설정이 되어있지 않다면, Github 권한으로 사용하기 위해 'Use HTTPS' 클릭



# [Eclipse] 내 로컬로 Clone

## □ Eclipse에 Clone 대상 저장소 위치 지정

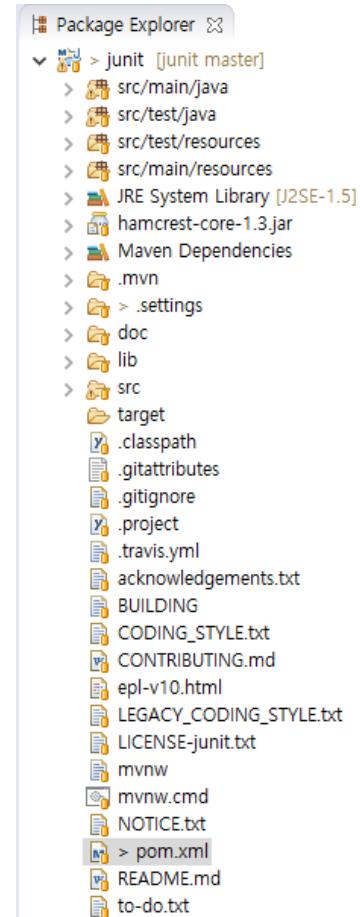


# [Eclipse] 소스코드 수정

## □ 에러가 나는 pom.xml의 플러그인 주석처리

- 저장 후, pom.xml 파일과 상위 폴더에 '>' 표시

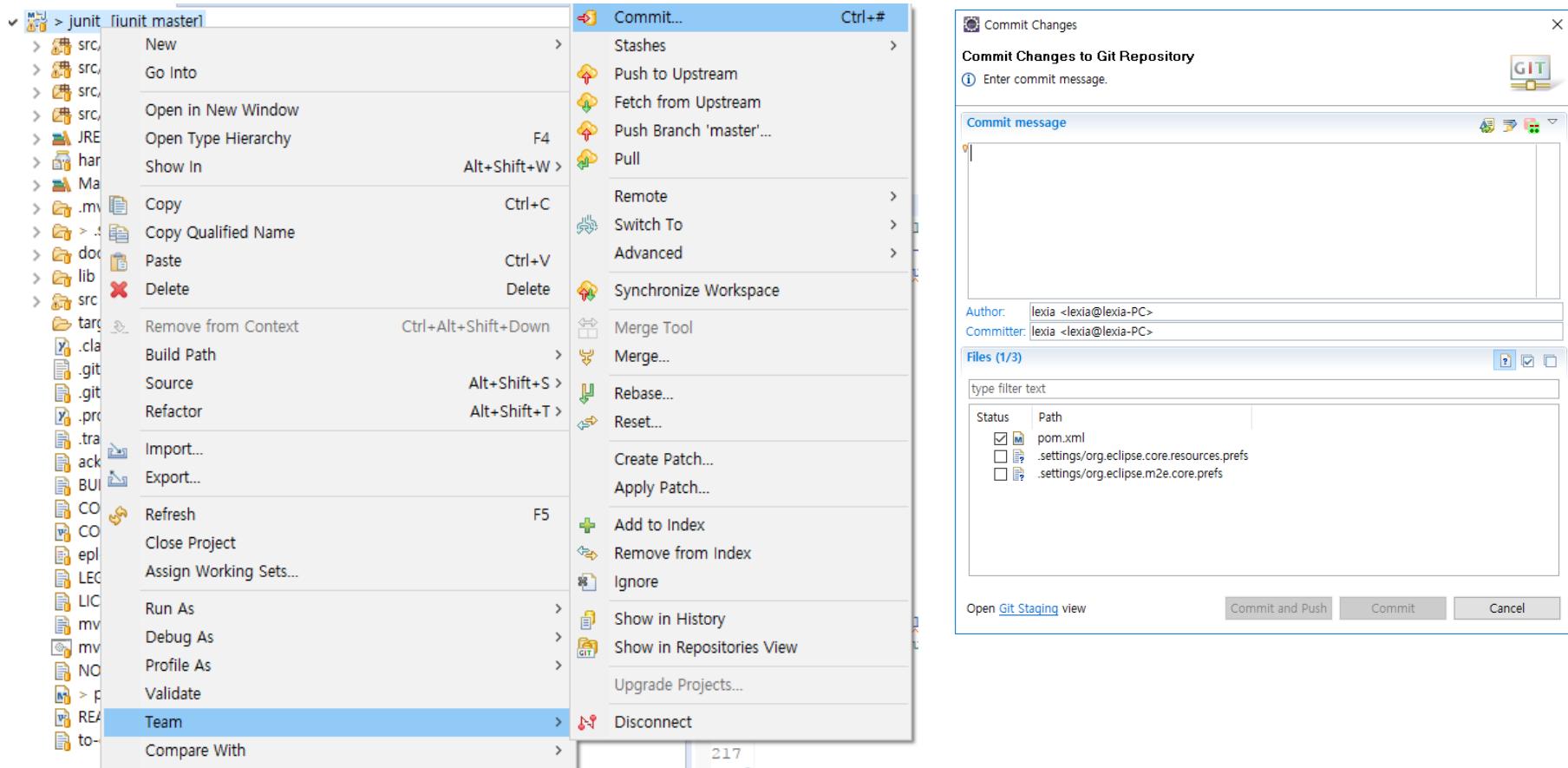
```
193             </plugin>
194             <!--
195             <plugin>
196
197                 <groupId>com.google.code.maven-replacer-plugin</groupId>
198                 <artifactId>replacer</artifactId>
199                 <version>1.5.3</version>
200                 <executions>
201
202                     </executions>
203                     <configuration>
204                         <ignoreMissingFile>false</ignoreMissingFile>
205                         <file>${project.build.sourceDirectory}/junit
206                         <outputFile>${project.build.sourceDirectory}
207                         <regex>false</regex>
208                         <token>@version@</token>
209                         <value>${project.version}</value>
210                     </configuration>
211             </plugin>-->
212             <plugin><!-- Using jdk 1.5.0_22, package-info.java
213             <!--
214                 java compiler plugin forked in extra process
```



# [Eclipse] 변경내용 Commit

## □ Team – Commit 실행

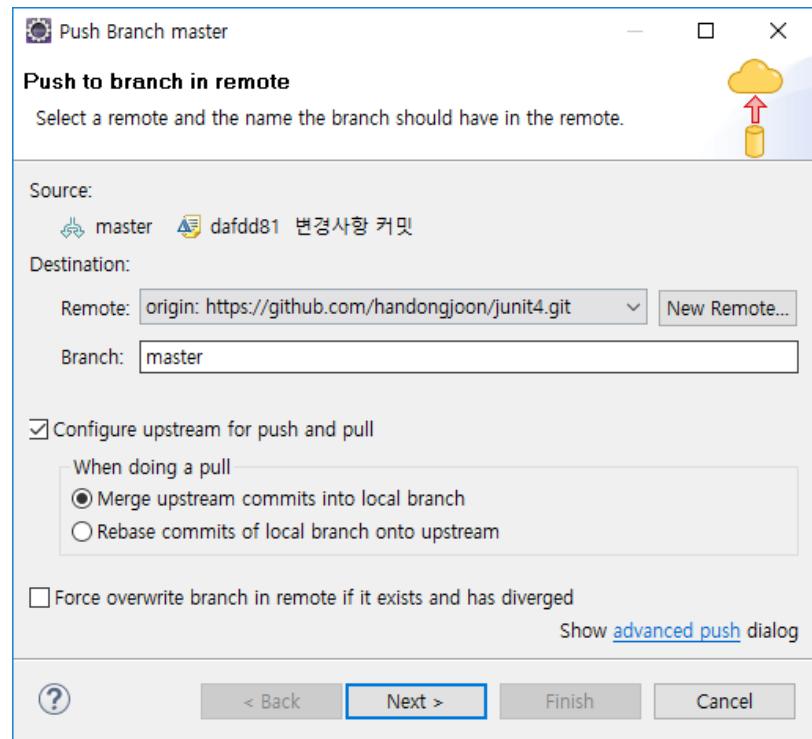
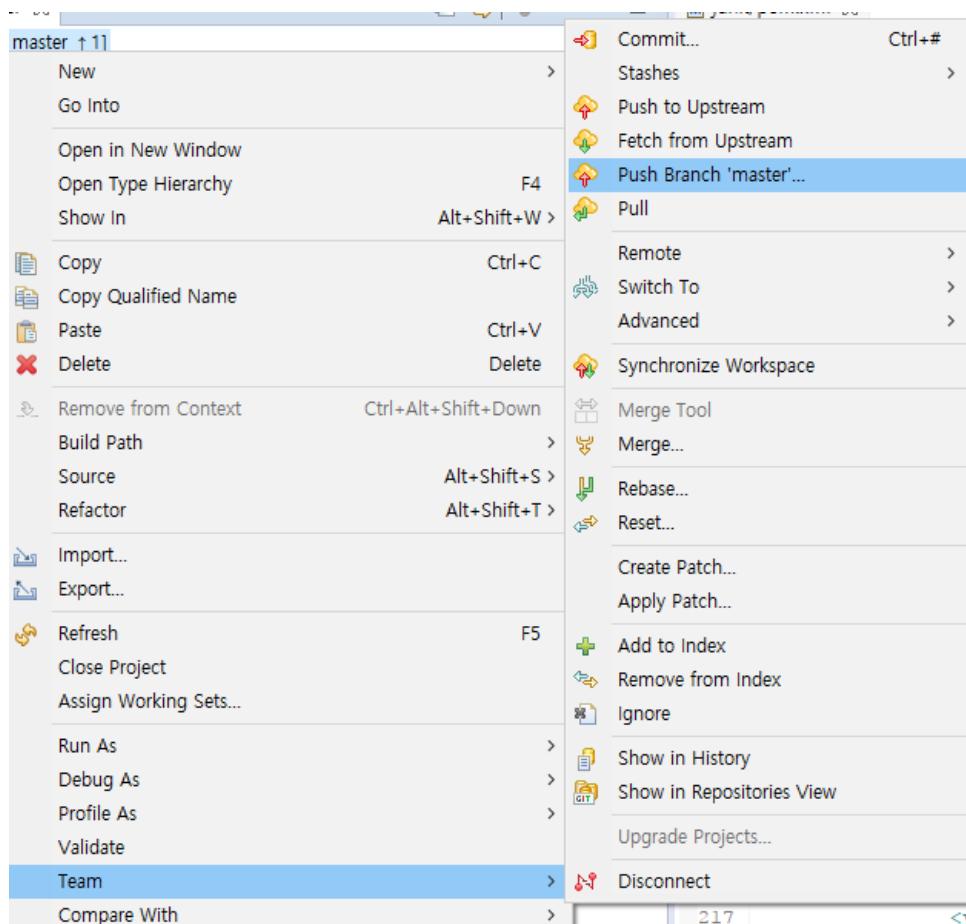
- SVN과 유사하게 커밋 메시지 작성 부분 표시
- 커밋 대상 선정 가능



# [Eclipse] 원격 저장소로 Push

## □ Commit 내용 원격 저장소로 Push 하기

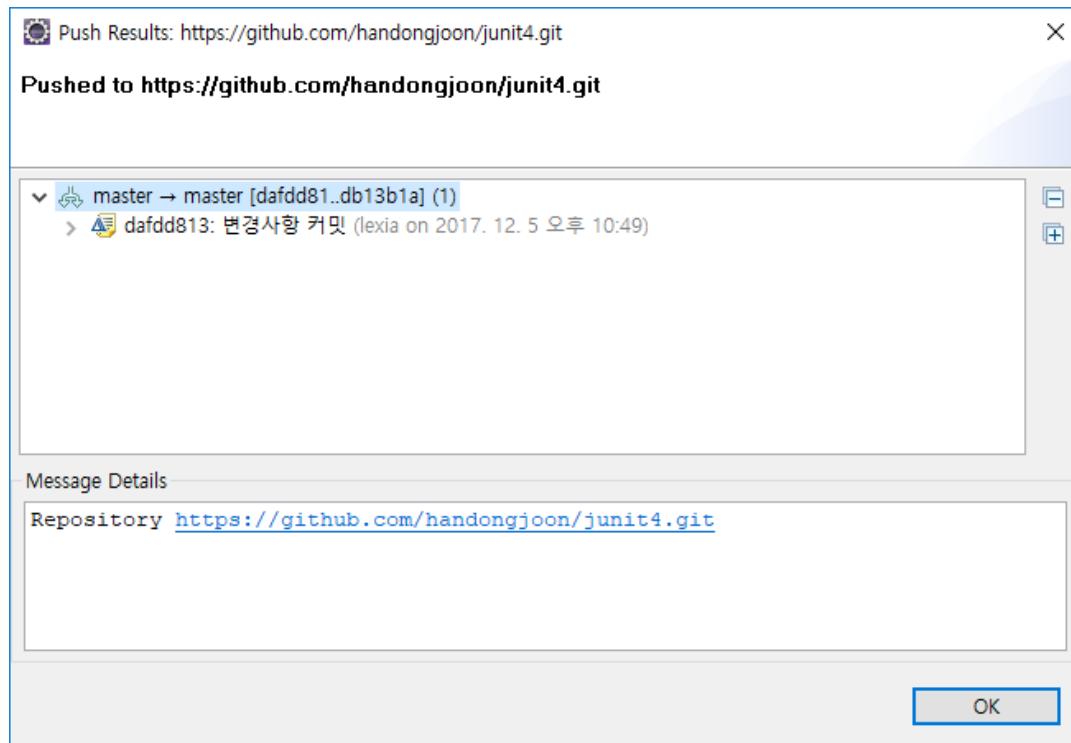
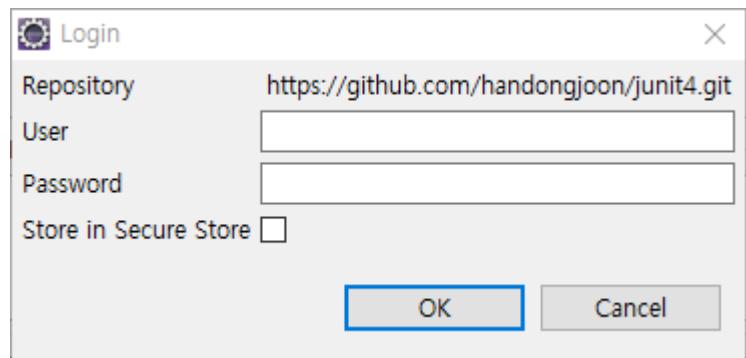
- 현재 작업 중인 master 브랜치 Push



# [Eclipse] 원격 저장소로 Push

## □ Github 계정으로 권한 확인

- Github의 내 저장소에서 변경내용 적용 확인



## 4 Jenkins

---

이 책에서 가장 핵심적인 부분이다.  
대부분의 도구를 Jenkins와 연결한다.  
지속 통합, 즉 자주 빌드하는 것은 중요하다.  
아마 하루에 한 번 이상은 빌드할 것이고, 추세를 볼 수 있다.

## 지속 통합(CI: Continuous Integration)

---

- ⑤ 팀 구성원들이 자신이 한 일을 **자주 통합**하는 소프트웨어 개발 실천 방법
- ⑤ 각 통합은 자동화된 빌드를 검증하여 **최대한 빨리 통합 오류**를 탐지
- ⑤ 하루에 여러 번 통합 빌드를 수행하는 것

- 마틴 파울러

소프트웨어 **통합 오류를 개발 초기부터 예방**하는 것

소프트웨어 통합을 위해 현존하는 가장 훌륭한 전략

---

통합 지옥

Integration HELL



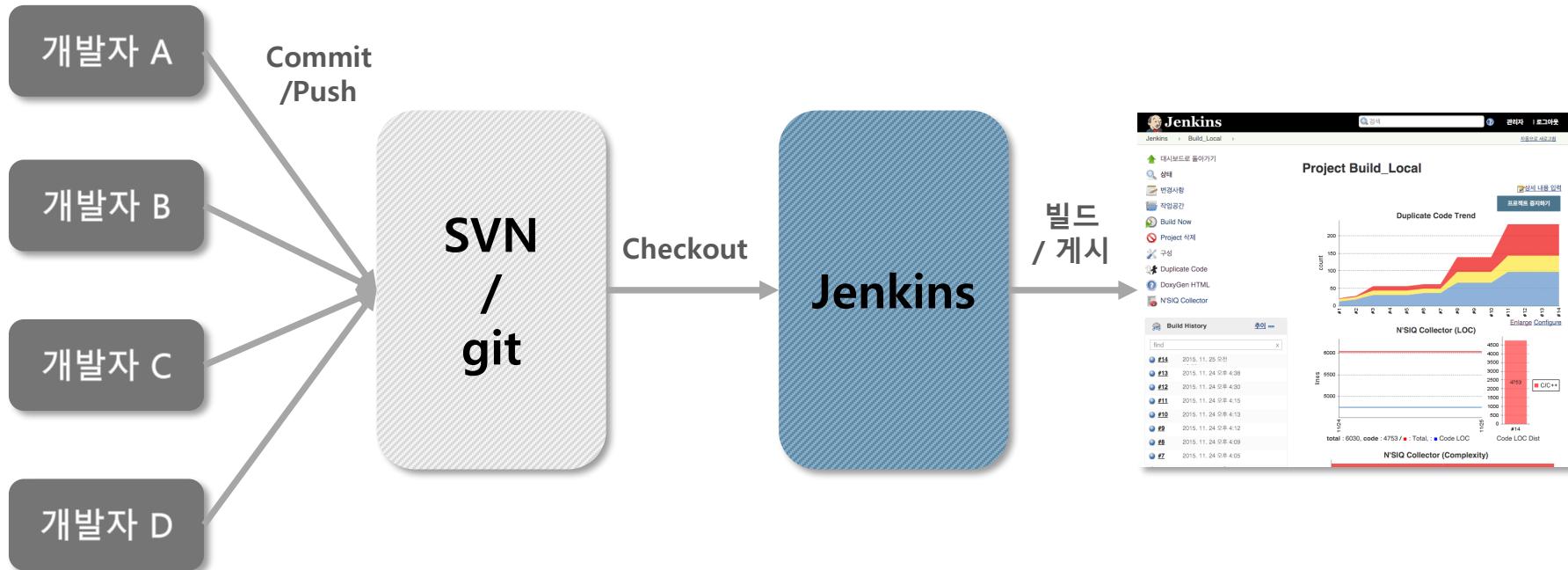
# Jenkins

<http://www.jenkins-ci.org>

- ⑤ 웹 기반 오픈소스 CI 도구
- ⑤ 점유율 약 70%
- ⑤ 1,300 개의 플러그인

# 지속적 통합 - 동작 방식

- ① 버전관리 도구에서 최신 리비전을 체크아웃 받아
- ② 주어진 명령대로 빌드하여
- ③ 결과를 게시/전달함



# 도구 모음

Jenkins      검색      관리자 | 로그아웃

Jenkins > Build\_Local >

[자동으로 새로고침](#)

[대시보드로 돌아가기](#)

[상태](#)

[변경사항](#)

[작업공간](#)

[Build Now](#)

[Project 삭제](#)

[구성](#)

[Duplicate Code](#)

[Doxygen HTML](#)

[N'SIQ Collector](#)

## Project Build\_Local

[상세 내용 입력](#)

[프로젝트 증지하기](#)

### Duplicate Code Trend

count

#1 #2 #3 #4 #5 #6 #7 #8 #9 #10 #11 #12 #13 #14

[Enlarge](#) [Configure](#)

### N'SIQ Collector (LOC)

lines

total : 6030, code : 4753 / ■ : Total, □ : Code LOC

Code LOC Dist

11/24 11/25

#14

4753

■ C/C++

### N'SIQ Collector (Complexity)

Code LOC Dist

Build History

추이

find

번호	날짜
#14	2015. 11. 25 오전
#13	2015. 11. 24 오후 4:38
#12	2015. 11. 24 오후 4:30
#11	2015. 11. 24 오후 4:15
#10	2015. 11. 24 오후 4:13
#9	2015. 11. 24 오후 4:12
#8	2015. 11. 24 오후 4:09
#7	2015. 11. 24 오후 4:05

## [설치]

### □ Tomcat(WAS) 상에 Jenkins WAR 파일을 실행하는 방식

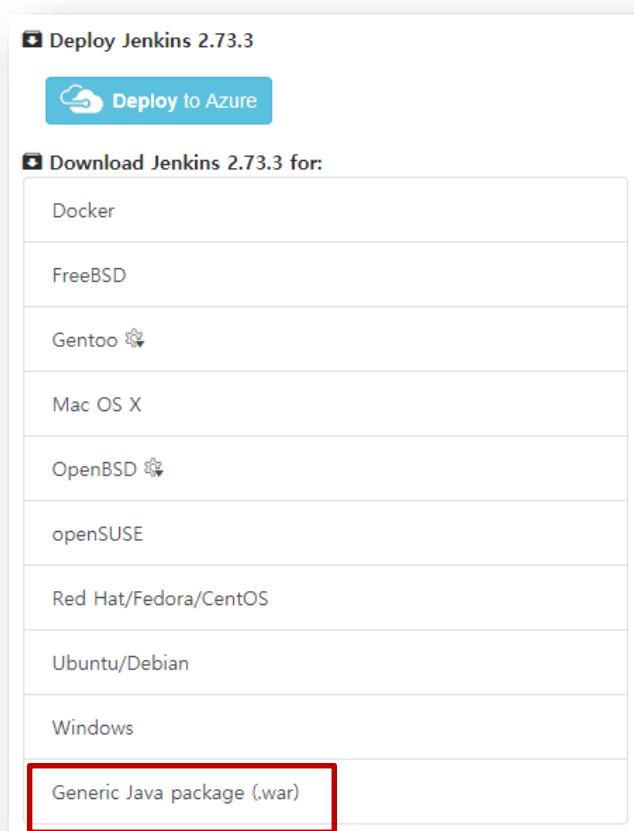
- 단독 실행(standalone)이 가능한 패키지도 제공
- Docker 이미지 및 Azure 배포 제공

### □ 다운로드

- <https://jenkins.io/download/>

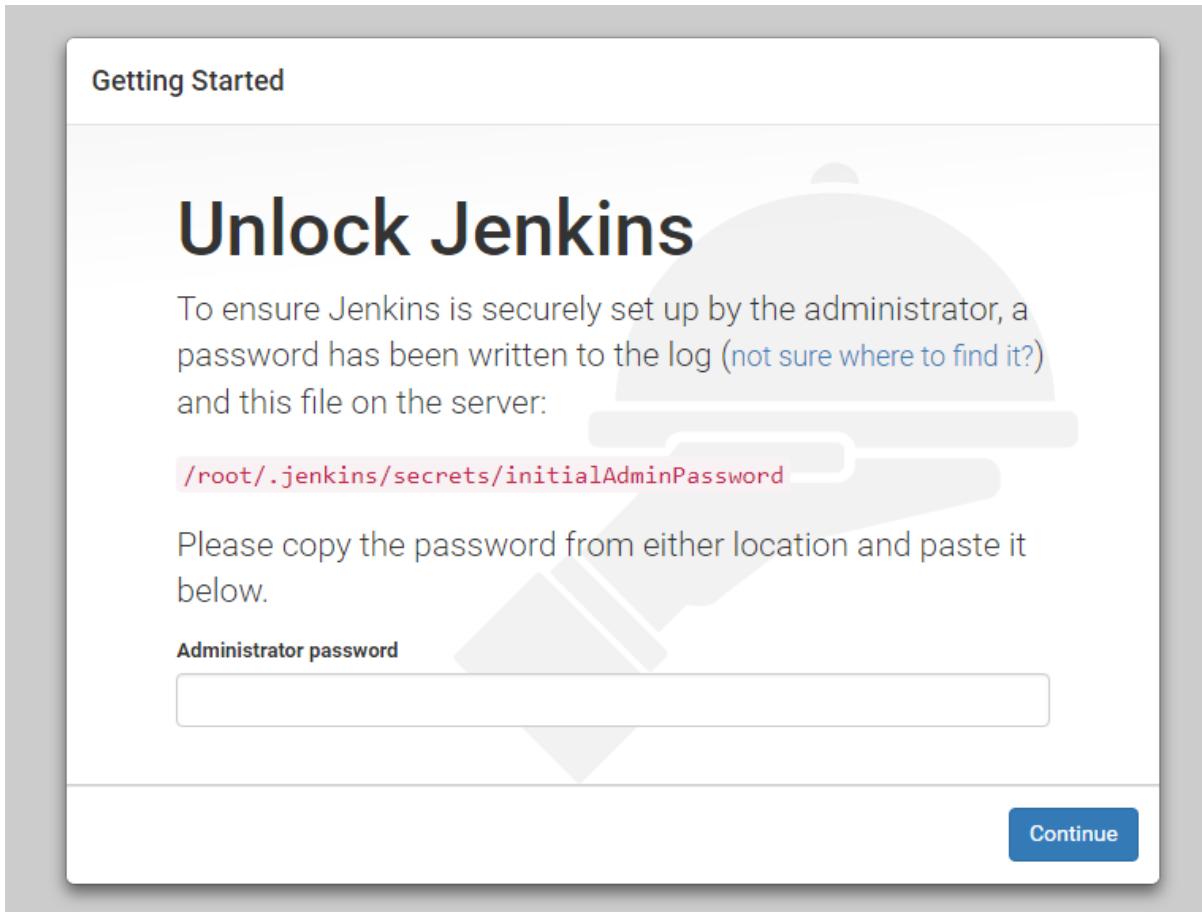
### □ WAR 복사

- 이름을 jenkins.war 로 변경
- jenkins.war 를 Tomcat의 webapp 폴더로 이동
- Tomcat 재시작
- 최초 시작을 위한 특별 ID 입력 필요



## [설정] Initial Password 입력

- Docker의 영구 데이터(Volume)에서 확인



The screenshot shows the Jenkins 'Getting Started' interface. At the top left is the title 'Getting Started'. At the top right is a small 'X' icon. Below the title, the heading 'Customize Jenkins' is displayed in large, bold, dark font. A subtext below it reads: 'Plugins extend Jenkins with additional features to support many different needs.' Two main options are presented in rounded rectangular boxes: 'Install suggested plugins' (left) and 'Select plugins to install' (right). Both boxes contain descriptive text: 'Install plugins the Jenkins community finds most useful.' for the first and 'Select and install plugins most suitable for your needs.' for the second. In the bottom left corner of the main area, the text 'Jenkins 2.46.3' is visible. The background features a faint circular watermark of the Jenkins logo.

Getting Started

Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

**Install suggested plugins**

Install plugins the Jenkins community finds most useful.

**Select plugins to install**

Select and install plugins most suitable for your needs.

Jenkins 2.46.3

# [설정] 초기화면

The screenshot shows the Jenkins initial setup page. At the top, there is a navigation bar with the Jenkins logo, a search bar, and links for '자동으로 새로고침' (Auto-refresh) and '상세 내용 입력' (Detailed input). On the left, there is a sidebar with icons for '새로운 Item', '사람', '빌드 기록', 'Jenkins 관리', and 'Credentials'. Below the sidebar, there are two collapsed sections: '빌드 대기 목록' (Build Queue List) which says '빌드 대기 항목이 없습니다.' (No build items in queue), and '빌드 실행 상태' (Build Execution Status) which shows '1 대기 중' (1 pending) and '2 대기 중' (2 pending). The main content area displays a message: 'Jenkins에 오신 것을 환영합니다.' (Welcome to Jenkins!) and '시작하려면 새 작업을 만들어 주시기 바랍니다.' (Please create a new job to get started). At the bottom, there is a link to '이 페이지 한글화 도와주기' (Help localize this page to Korean) and footer information: '페이지 생성일시: 2016. 2. 11 오후 5시 46분 07초 REST API Jenkins ver. 1.647'.

## [설정] Jenkins 관리

#	메뉴	설명
1	시스템 설정	Jenkins 실행에 대한 전반적인 환경을 설정한다. 초기 화면에 보여질 시스템 메시지, 전역 변수, Maven, JDK 설정 등이 포함된다.
2	Configure Global Security	Jenkins 전반적인 보안 항목을 설정한다. 기본 값은 보안을 사용하지 않는 것이다. LDAP 사용 여부, ID 사용 여부 및 권한이 포함된다.
3	플러그인 관리	약 840여개의 플러그인을 설치하고 업데이트 할 수 있다.
4	노드 관리	빌드 하기 위한 Master(Jenkins가 설치된 서버)와 Slave(빌드만 전용으로 하는 머신)를 설정한다.
5	사용자 관리	Jenkins 사용자를 설정하는 기능으로, 'Configure Global Security'의 설정 결과에 따라 이 메뉴가 표시된다. 최초 설치 후에는 표시되지 않는다.

# [설정] 시스템 기본 설정

---

## □ 홈 디렉토리

- 빌드 Job을 설정하고 빌드할 때 소스 코드 저장소에서 체크아웃한 파일이 저장되고, 빌드되는 로컬 디렉터리의 위치
- 특정 위치로(예를 들면 D:\W 등) 설정하고 싶다면, Jenkins 설치 전에 설정 필요
  - web.xml의 HUDSON\_HOME 변경

## □ # of executors

- Master에서 동시에 빌드하는 Job의 개수

## □ SCM checkout retry count

- 소스 코드 저장소에서 체크아웃을 실패했을 때 재시도 하는 횟수
- 기본값인 0은 체크아웃이 실패해도 빌드가 실패한 것으로 간주

## □ Help make Jenkins better by sending anonymous usage statistics and crash reports to the Jenkins project.

- Jenkins 개선을 위한 정보 보내기
- 오픈소스 프로젝트에 기여하는 하나의 방법

## [설정] Jenkins Location 설정

### □ Jenkins URL

- Jenkins 접속 주소. 반드시 IP나 URL로 변경 필요

### □ System Admin e-mail address

- Jenkins에서 사용자에게 알림 메일을 보낼 때 사용할 관리자의 e-mail 주소

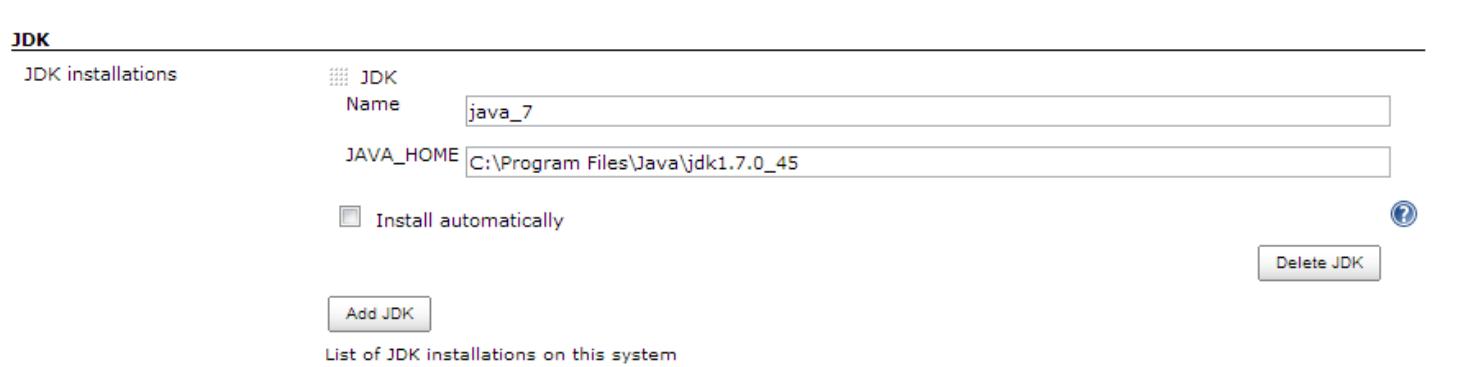
#### Jenkins Location

Jenkins URL	<input type="text" value="http://localhost:8080/jenkins/"/> <span>?</span>
<span style="color: yellow;">⚠ Please set a valid host name, instead of localhost</span>	
System Admin e-mail address	<input type="text" value="address not configured yet &lt;nobody@nowhere&gt;"/> <span>?</span>

# [설정] 빌드 도구 설정

## □ JDK 설정

- 'Add JDK'를 클릭해 JDK 추가
- 자동으로 특정 버전을 추가하거나 JDK 설치 후 JAVA\_HOME의 위치를 지정



## □ Maven 설정

- JDK와 동일한 방식으로 추가

## [설정] Configure Global Security (권한) 설정

- 권한 없어도 접근 가능한 수준부터, 프로젝트 별 권한까지 설정 가능
- 사용 권한을 설정하기 위해서는 'Enable security'를 선택

Enable security ?

TCP port for JNLP slave agents  Fixed :   Random  Disable ?

Disable remember me  ?

Access Control

**Security Realm**

- Delegate to servlet container ?
- Jenkins' own user database ?
- LDAP ?

**Authorization**

- Anyone can do anything ?
- Legacy mode ?
- Logged-in users can do anything ?
- Matrix-based security ?
- Project-based Matrix Authorization Strategy ?

# [설정] Configure Global Security (권한) 설정

---

## □ Security Realm

- 권한이 어디에 위치하는지 지정하는 것으로 Jenkins 내부의 DB에 저장할 수도 있고, 다른 곳의 권한을 사용할 수도 있음
  - Delegate to servlet container: 컨테이너, 즉 Tomcat과 같은 WAS의 권한을 위임받아 사용
  - Jenkins' own user database: Jenkins의 내부 데이터 베이스에 아이디 정보 저장
  - LDAP: 사용자 계정 서버와 연결하여 사용
- 사용자 계정 서버가 없는 경우는 두 번째 선택값인 Jenkins 내부의 DB를 사용

# [설정] Configure Global Security (권한) 설정

## □ Authorization

- Anyone can do anything: 로그인 여부와 상관없이, Jenkins에 접속하는 누구나 Jenkins의 설정을 포함한 모든 기능을 사용할 수 있게 한다. 기본값이지만, 기본 설정 후 보안을 위해 반드시 다른 항목으로 변경해야 한다.
- Legacy mode: Tomcat과 같은 WAS의 admin 권한을 이용하여, admin만 모든 기능을 사용할 수 있고 그 외에는 읽기 전용 권한을 부여한다.
- Logged-in users can do anything: 1번 항목과 비슷하나, 로그인한 사용자가 모든 기능을 사용할 수 있게 한다.
- Matrix-based security: 역할이나 사용자에 따라 권한을 설정한다. 기본값으로 'Anonymous'가 포함되나, 다음 그림과 같이 'Add' 버튼을 이용해서 사용자나 그룹을 추가할 수 있다.

### Authorization

- Anyone can do anything
- Legacy mode
- Logged-in users can do anything
- Matrix-based security

User/group	Overall						Credentials						Slave					
	Administer	Configure	UpdateCenter	ReadRunScripts	UploadPlugins	Create	Delete	ManageDomains	UpdateView	Build	ConfigureConnect	CreateSlave	DeleteSlave	DisconnectSlave	ConnectSlave	DisconnectSlave		
Anonymous	<input type="checkbox"/>																	
admin	<input checked="" type="checkbox"/>																	

User/group to add:

Add

- Project-based Matrix Authorization Strategy

# [설정] Configure Global Security (권한) 설정

## □ Authorization (계속)

- Project-based Matrix Authorization Strategy: Matrix-based security와 유사하나, 각 Job 설정에서 다시 한 번 Job에 대한 권한을 설정할 수 있다. 예를 들어, A 프로젝트에서 빌드하는 Job과 B 프로젝트에서 빌드하는 Job이 다르고, A와 B 프로젝트 팀원은 다른 팀의 Job은 볼 수 없도록 설정할 때 사용한다. 첫 번째 그림과 같이 사용자나 그룹 별 권한을 설정하고, 두 번째 그림과 같이 Job 설정 화면에서 권한을 다시 한 번 설정할 수 있다.

Authorization

- Anyone can do anything
- Legacy mode
- Logged-in users can do anything
- Matrix-based security
- Project-based Matrix Authorization Strategy

User/group	Overall									Credentials										
	Administer	Configure	UpdateCenter	Read	RunScripts	UploadPlugins	Create	Delete	ManageDomains	UpdateView	View	Run	SCM	Job	Discover	Read	WorkSpace	Delete	Update	Tag
admin	<input checked="" type="checkbox"/>																			
Anonymous	<input type="checkbox"/>																			

User/group to add:

Enable project-based security

Block inheritance of global authorization matrix

User/group	Credentials									Job									Run	SCM	
	Create	Delete	ManageDomains	UpdateView	Build	Cancel	Configure	Delete	Discover	Read	WorkSpace	Delete	Update	Tag	Job	Discover	Read	WorkSpace	Delete	Update	Tag
Anonymous	<input type="checkbox"/>																				
admin	<input checked="" type="checkbox"/>																				

User/group to add: admin

# [설정] Configure Global Security (권한) 설정

---

## □ 주의할 점

- Jenkins를 설치한 후 'Jenkins's own user database'을 이용하여 사용 권한을 처음 설정할 때에는, 다음의 순서로 진행하자. 설치 후에는 Jenkins 사용자 DB에 따로 사용자가 없기 때문에, 만약 'Anyone can do anything' 외에 다른 항목으로 설정한다면 아이디가 없기 때문에 로그인을 못해서 아무 작업도 못할 수 있다.

1. 'Security Realm'에서 'Jenkins' own user database'를 선택한다
2. 'Authorization'에서 'Anyone can do anything'를 선택하고 적용한다.
3. 'Jenkins 관리'의 'Manage Users'에서 'admin' 또는 원하는 관리자 계정을 생성한다.
4. 다시 'Configure Global Security' 메뉴로 돌아와서, 'Authorization' 항목의 다른 권한 설정을 선택한다.
5. 단, Matrix-based security 등을 선택 시에는, 3번에서 생성한 관리자 계정을 'Add'로 추가하고 필요한 관리자 권한을 부여한다.

## [설정] 플러그인 관리

- 플러그인의 추가/삭제/업데이트 설정
- 플러그인 추가 후, Jenkins 재시작이 필요할 수 있음

The screenshot shows the Jenkins Manage Plugins interface. At the top, there are tabs: '업데이트된 플러그인 목록' (Updated Plugins List), '설치 가능' (Available), '설치된 플러그인 목록' (Installed Plugins List), and '고급' (Advanced). Below these tabs, there is a table with columns: '설치' (Install), '이름 ↓' (Name ↓), '버전' (Version), and '설치됨' (Installed). A single plugin is listed: 'SCM API Plugin'. The '설치' column has a checkbox next to it. The '이름' column shows 'SCM API Plugin'. The '버전' column shows '1.1'. The '설치됨' column shows '1.0'. Below the table, there is a note: 'This plugin provides a new enhanced API for interacting with SCM systems.' At the bottom left, there is a button labeled '지금 다운로드하고 재시작 후 설치하기' (Download now and restart after installation). In the center, there is a message: 'Update information obtained: 5 sec ago'. At the bottom right, there is a button labeled '지금 확인' (Check now).

# [Job 설정] Job 추가

## □ 메인화면의 '새로운 Item' 클릭

- 최신 버전은 Maven 프로젝트가 Freestyle 프로젝트에 포함

**Enter an item name**

|

» This field cannot be empty, please enter a valid name

**Freestyle project**  
 이것은 Jenkins의 주요 기능입니다. Jenkins은 어느 빌드 시스템과 어떤 SCM(형상관리)으로 묶인 당신의 프로젝트를 빌드할 것이고, 소프트웨어 빌드보다 다른 어떤 것에 자주 사용될 수 있습니다.

**Maven project**  
 Maven 프로젝트를 빌드합니다. Jenkins은 POM 파일의 이점을 가지고 있고 급격히 설정을 줄입니다.

**Pipeline**  
 Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

**External Job**  
 이 유형의 작업은 원격 장비처럼 Jenkins 외부에서 동작하는 프로세스의 실행을 기록하는 것을 허용합니다. 그렇게 설계되어서, 기존의 자동 시스템의 대시보드로서 Jenkins을 사용할 수 있습니다.

**Multi-configuration project**  
 다양한 환경에서의 테스트, 플랫폼 특성 빌드, 기타 등등 처럼 다수의 서로다른 환경설정이 필요한 프로젝트에 적합함.

if you want to create a new item from other existing, you can use this option:

 Copy from

 OK

## [Job 설정] 설명 설정

---

### □ Job에 대해 알고 싶은 내용을 작성

이름

Maven Project

설명

[Plain text] [미리보기](#)

## [Job 설정] 기본 설정

- 오래된 빌드 삭제 ?
- 이 빌드는 매개변수가 있습니다 ?
- 빌드 안함 (프로젝트가 다시 빌드를 할 때까지 새로운 빌드가 실행되지 않습니다.) ?
- 필요한 경우 concurrent 빌드 실행 ?

설정 항목	설명
오래된 빌드 삭제	이 항목을 설정하지 않으면 Jenkins는 빌드 결과를 계속 유지한다. 그러나 하드디스크 용량 등을 고려했을 때, 지난 빌드 결과는 삭제하는 것이 필요할 수도 있다. 이 항목을 선택하면 특정 기간이나 빌드 수만큼 빌드 결과를 보관할 수 있다.
이 빌드는 매개변수가 있습니다.	빌드 시 매개변수를 정의하고 사용할 수 있다. 매개변수는 단순히 Text도 가능하지만, Subversion의 tag 목록도 가능하다.
빌드 안함	이 항목을 선택하면 빌드가 수행되지 않는다. 모든 빌드 설정은 유지하면서, 잠시 빌드를 사용하지 않을 때 활용한다.
필요한 경우 concurrent 빌드 실행	Jenkins는 하나의 Job을 빌드 중 또 다시 해당 빌드를 시작하면 Queue에 쌓이게 된다. 그러나 Queue에는 1개까지만 유지되고, 나머지 빌드는 무시된다. 이 항목을 선택하면 빌드 머신이 가능한 경우 동시 빌드를 수행한다.

## [Job 설정] 소스코드 관리

### □ 빌드를 위해 소스코드 저장소에서 소스코드를 가져오는 설정

소스 코드 관리

None  
CVS  
CVS Projectset  
Git  
 Subversion

Modules Repository URL  ⓘ  
**Repository URL is required.**

Local module directory (optional)  ⓘ

Repository depth  ⓘ

Ignore externals  ⓘ

Add more locations...

Check-out Strategy  ⓘ  
Use 'svn update' whenever possible, making the build faster. But this causes the artifacts from the previous build to remain when a new build starts.

Repository browser  ⓘ

고급...

## [Job 설정] 소스코드 관리

### □ URL에 저장소 주소 입력

- 권한이 필요한 저장소라면, 계정 정보 추가 입력

### □ 체크아웃 전략 설정(SVN의 경우)

전략	설명
Use 'svn update' as much as possible	최초 빌드 또는 최초 빌드가 아니더라도 체크아웃한 기록이 없을 경우 체크아웃을 하고, 그 이후 빌드에는 업데이트 명령만 수행한다. 빌드를 빠르게 진행할 수 있지만, 과거 빌드의 결과가 남아있을 수 있다. 이 메뉴의 기본 값이다.
Always check out a fresh copy	빌드 시작 시 기존에 체크아웃 받은 내용과 빌드 결과를 삭제하고, 새롭게 체크아웃한다. 매번 새롭게 체크아웃하므로 빌드 시간은 오래 걸리나, 클린 빌드가 필요한 경우 사용한다.
Emulate clean checkout by first deleting unversioned/ignored files, then 'svn update'	Always check out a fresh copy과 유사한 결과를 빠르게 만들기 위한 방법으로, 먼저 Subversion 관리를 받지 않는 파일을 삭제하고, 업데이트를 수행한다.
Use 'svn update' as much as possible, with 'svn revert' before update	Use 'svn update' as much as possible과 유사하지만, 업데이트 전 Subversion의 revert 명령을 수행한다. revert 명령은 체크아웃 후 수정이 있을 경우, 모든 수정을 체크아웃 받은 상태로 '되돌아 가도록' 만든다.

## [Job 설정] 빌드 유발

### □ 빌드를 언제 수행할 것인지 지정

#### 빌드 유발

- Build after other projects are built
- Build periodically
- Poll SCM



항목	설명
Build after other projects are built	다른 Job의 빌드가 완료되면 이 빌드를 시작한다. 예를 들어 Library란 Job의 빌드가 완료된 후 이 빌드를 시작하려면, 'Projects to watch' 항목에 Job 이름인 Library를 입력하면 된다.
Build periodically	정해진 일정에 따라 빌드를 진행한다. 일정 지정 형식은 cron 표기법과 유사하다.
Poll SCM	정해진 일정에 따라 소스 코드 저장소에서 변경 사항이 있는지 확인한다. 변경이 있을 경우 빌드를 진행한다. 일정 지정 형식은 cron 표기법과 유사하다.

## [Job 설정] 빌드 유발

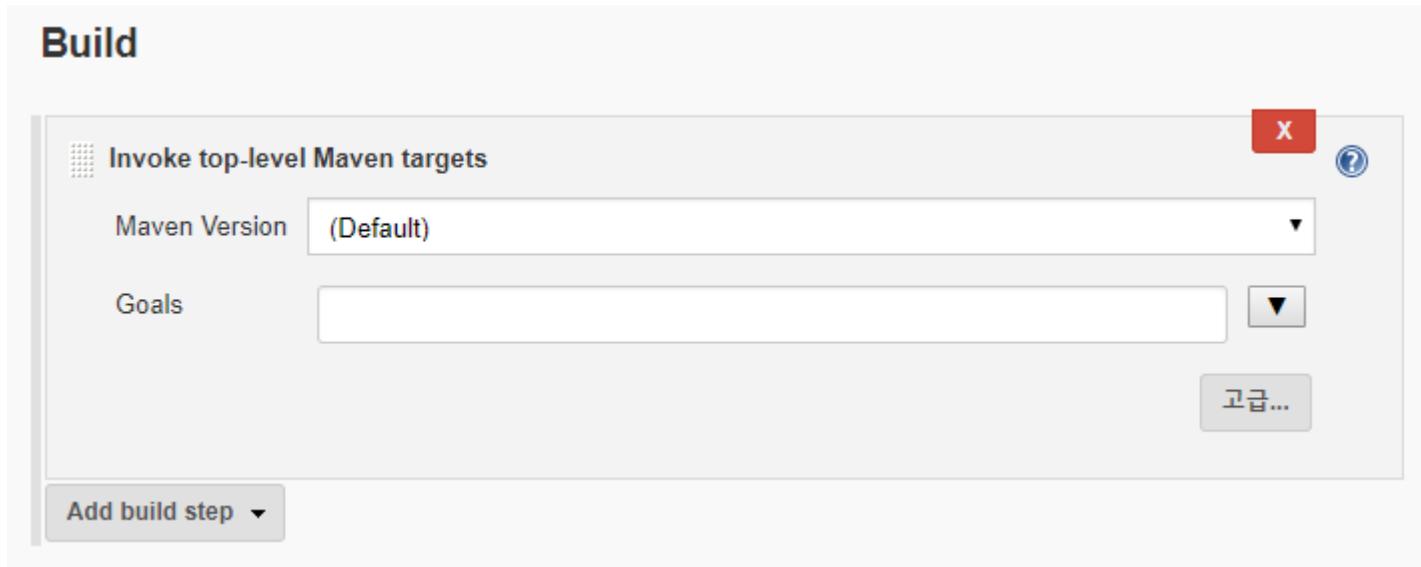
---

### □ Build periodically과 Poll SCM의 일정 지정은 cron 표기법을 사용

- TAB이나 공백으로 구분된 5자리
  - 분 시 일 월 요일순
- 예) 매일 새벽 5시에 빌드를 진행
  - 0 5 \* \* \*
- 예) 15분 단위로 소스 코드 저장소의 변경 사항을 확인
  - H/15 \* \* \* \*
- cron 표기법 참고
  - <https://en.wikipedia.org/wiki/Cron>

## [Job 설정] Maven 프로젝트

- Build 에서 'Invoke top-level Maven Target' 선택



## [Job 실행] 빌드 결과 확인

- Jenkins 메인화면에서 각 Job을 클릭하면 Job 화면으로 이동
- Job이 정상적으로 빌드되었는지 결과만 확인하려면, 왼쪽 하단의 Build History를 확인
  - #1, #2 순으로 빌드가 표시
  - 파란색 공일 경우 빌드 성공
  - 빨간색 공일 경우 빌드 실패

The screenshot shows the Jenkins interface for the 'Maven project MVN\_Test' job. On the left, there's a sidebar with various navigation links: 대시보드로 돌아가기, 상태, 변경사항, 작업공간, Build Now, Maven project 삭제, 구성, and Modules. Below this is a 'Build History' section with a search bar labeled 'find' and a single entry for build #1 from March 29, 2016, at 1:16. At the bottom of this section are RSS feed links for '전체' and '실패'. The main content area is titled 'Maven project MVN\_Test' and contains links for 'Workspace' and 'Recent Changes'. A '고정링크' (Pinned Links) section lists several recent builds. On the right side, there are buttons for '상세 내용 입력' (Input detailed information) and '프로젝트 중지하기' (Stop project).

대시보드로 돌아가기  
상태  
변경사항  
작업공간  
Build Now  
Maven project 삭제  
구성  
Modules

Build History 초이 ←  
find X  
#1 2016. 3. 29 오전 1:16  
[RSS \(전체\)](#) [RSS \(실패\)](#)

Maven project MVN\_Test

Workspace  
Recent Changes

고정링크

- [Last build\\_ \(#1\).19 sec 전](#)
- [Last stable build\\_ \(#1\).19 sec 전](#)
- [Last successful build\\_ \(#1\).19 sec 전](#)
- [Last completed build\\_ \(#1\).19 sec 전](#)

상세 내용 입력  
프로젝트 중지하기

## [Job 실행] 빌드 실패 원인 확인

- 각 빌드 #를 클릭하고, Console Output 확인



**빌드 #1 (2016. 3. 29 오전 1:16:57)**

Revision: 3  
No changes.

사용자 관리자에 의해 시작됨

**Module Builds**

my\_project 1.6 sec

## 5 PMD

---

PMD는 소스코드의 기초 체력을 탄탄하게 만들어 주는 도구다.

## □ 도구 개요

- 업계에서 “안좋은” 방식이라 알려진 코드의 발견
- 사전에 정해진 약 300개의 Rule 을 기반으로 탐지
  - Java는 200개 정도
- 컴파일 에러는 아니지만, 나중에 문제를 발생시킬 수 있는 코드 탐지

## □ Rule 예

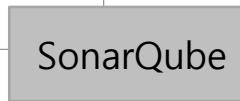
- Connection 열었으면 명시적으로 닫았는가?
- int i; 사용
- 매개변수 15개 사용
- case마다 brake; 있는가?
- final에서 return 금지
- 빈 if 문장은 피해야 함
- ...

## □ 사용 방법

- 코딩하며, 컴파일 전에 사용

# 룰 기반 정적분석 도구

## □ 대표 도구

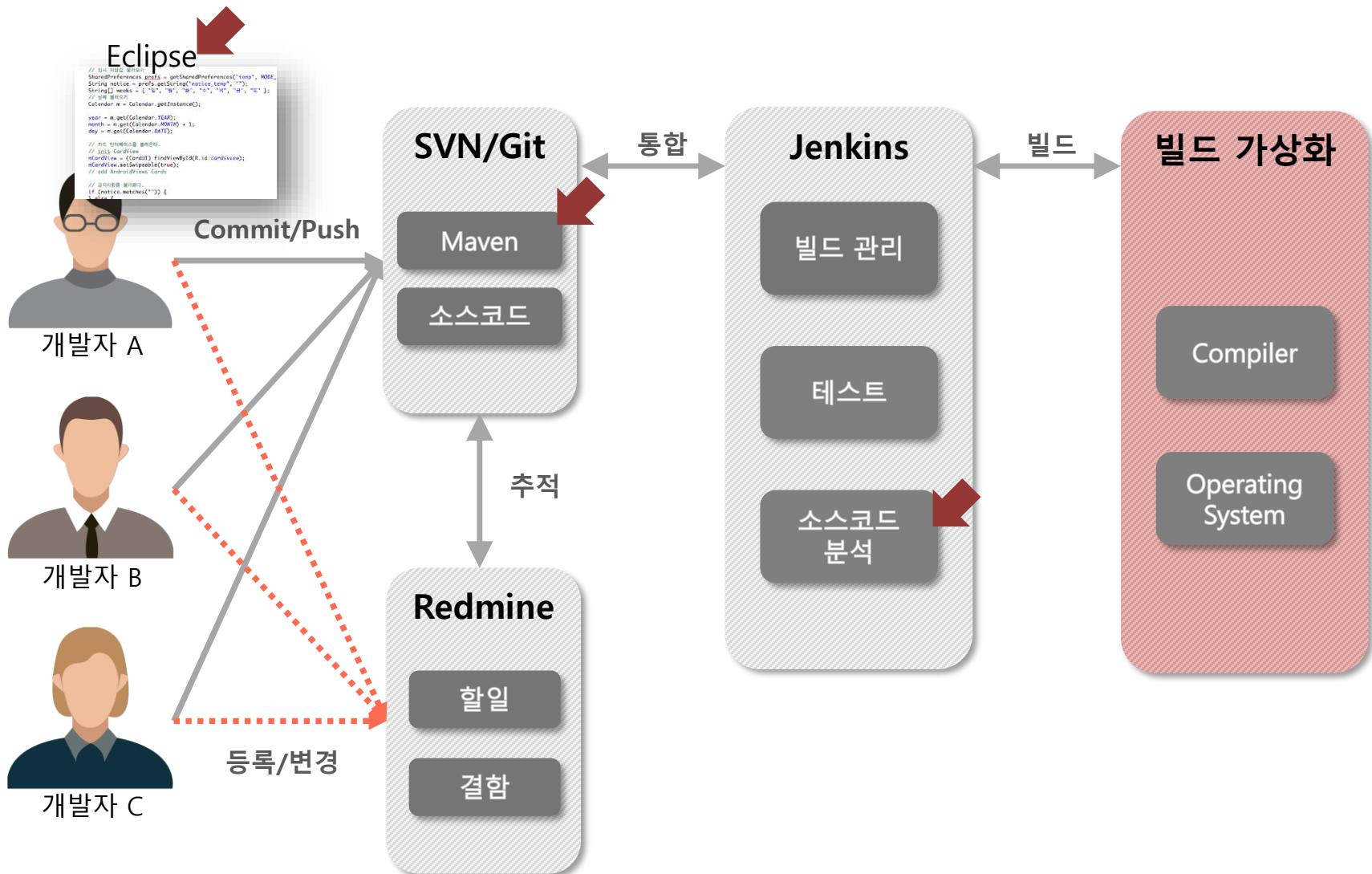
언어	국산	외산	오픈소스	특징
C/C++	Sparrow(파수) CodeInspector(Suresoft) Resort(Soft4Soft)	<a href="#">QAC/QAF(PRQA)</a> <a href="#">Coverity(Synopsys)</a> <a href="#">Polyspace-Bug Finder(MathWorks)</a> <a href="#">Klockwork(Rogue Wave)</a>	CppCheck  	<ul style="list-style-type: none"><li>국산 도구는 행자부 개발 보안 검증</li><li>ISO 26262 인증 (오픈소스 제외)</li></ul>
Java	Sparrow(파수) CodeInspector(Suresoft) Resort(Soft4Soft)	QAC/QAF(PRQA) Coverity(Synopsys)	PMD FindBug	<ul style="list-style-type: none"><li>국산 도구는 행자부 개발 보안 검증</li><li>보안의 경우 제외하고, 오픈소스 위주 사용</li></ul>

※ **빨간색** 표시: 보안 룰 검증 지원

## □ 도구 선택 시 확인 사항

- 분석 대상 언어(C/C++ or JAVA)
- 1종, 2종 오류 발생율**
  - 1종: 결함인데 못잡는 것**
  - 2종: 결함이 아닌데 결함이라 하는 것**

# 툴체인에서의 정적분석 도구 역할



## [참고] 최적의 룰 기반 정적분석 사용

---

### □ IDE와 혼연일체가 되어야함

- Java 계열은 Eclipse, IntelliJ로 대동단결
  - 각 정적분석 도구는 이 2개 IDE 플러그인만 제공하면 됨
  - IDE가 아니더라도, Maven, Gradle 등 빌드 도구에 적용 가능(플러그인)
- C 계열은 컴파일러 자체 IDE 사용
  - 각 정적분석 도구가 각 IDE 별 플러그인 개발 필요 -> 플러그인이 거의 없음
  - 코딩을 위한 IDE 따로, 정적분석 도구 따로 사용

### □ 적용 룰을 최적화해야 함

- PMD의 모든 룰을 사용할 것인가? 필요한 룰만 사용할 것인가?
  - 필요한 룰은 어떻게 선정할 것인가?

### □ 적용 룰은 구현 전 배포 및 교육

- 최악: 코딩 후반부에 정적분석 도구 적용 및 강제화
- MISRA 적용 도구는 고가여서, 협력사에서 사용하기 어려우나 강제하는 경우 발생
  - CppCheck의 MISRA 룰셋 또는 SonarQube 사용 권장

### □ CI(Jenkins)에 적용은 일반화됨

- 거의 대부분의 유명 도구(상용/오픈소스 모두)는 Jenkins 플러그인 제공

## [Maven] Goal 설명

---

### □ 2개의 Goal

Goal	설명
pmd:pmd	PMD를 실행하고, 보고서를 생성한다.
pmd:check	PMD를 실행하고, 위반 발견 시 빌드 실패를 유발한다.

## [Maven] Usage – pmd:pmd

---

```
<project>
...
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-pmd-plugin</artifactId>
      <version>3.4</version>
    </plugin>
  </plugins>
</reporting>
...
</project>
```

## [Maven] Usage – pmd:pmd (원하는 룰셋 지정)

### □ <build> 내에 작성 필요

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-pmd-plugin</artifactId>
    <version>3.4</version>
    <configuration>
        <targetJdk>1.5</targetJdk>
        <sourceEncoding>utf-8</sourceEncoding>
        <rulesets>
            <!-- PMD에 포함된 Braces와 Naming 룰셋 사용 -->
            <ruleset>/rulesets/java/braces.xml</ruleset>
            <ruleset>/rulesets/java/naming.xml</ruleset>
            <!-- 내 컴퓨터에 있는 사용자 수정 룰셋 파일 사용(절대경로) -->
            <ruleset>d:\rulesets\strings.xml</ruleset>
            <!-- 내 컴퓨터에 있는 사용자 수정 룰셋 파일 사용(상대경로) -->
            <ruleset>.\rulesets\strings.xml</ruleset>
            <!-- 웹에 있는 사용자 수정 룰셋 파일 사용 -->
            <ruleset>http://localhost/design.xml</ruleset>
        </rulesets>
        <excludes>
            <exclude>**/*Bean.java</exclude>
            <exclude>**/generated/*.java</exclude>
        </excludes>
    </configuration>
</plugin>
```

## [참고] 원하는 룰셋 파일 만들기

---

- PMD는 사전에 정의된 룰셋을 기반으로 검증한다. 각각의 룰은 java 파일로 작성되어 있고, 이러한 룰의 전반적인 정보는 각 룰셋에 대한 xml 파일에 작성되어 있다. 예를 들어 JAVA Basic 룰셋에 대한 정보는 basic.xml 이란 파일에 있다.
- 이러한 룰셋 xml 파일은 PMD 소스코드를 다운받으면 확인할 수 있다. PMD 웹사이트에서 압축된 소스코드를 다운로드 받고, 압축을 풀고 '/pmd-java/src/main/resources/rulesets/java' 를 확인하면 PMD 각 룰셋에 대한 xml 파일이 있다. 이 xml 파일을 이용하면 여러분에게 필요한 자체 룰만 골라 새로운 룰셋 xml 파일을 만들고 활용할 수 있다.

# [Jenkins] 플러그인 설치

## □ 플러그인 설정에서 Warning Next Generation 검색

- 설치 시 필요한 추가 플러그인을 자동 선택하여 설치함

The screenshot shows the Jenkins plugin manager interface. A search bar at the top contains the text "warning next". Below the search bar, there are four tabs: "업데이트된 플러그인 목록" (Updated Plugins List), "설치 가능" (Installable), "설치된 플러그인 목록" (Installed Plugins List), and "고급" (Advanced). The "설치 가능" tab is selected. The results are listed in a table with columns for the plugin name, description, version, and last updated time.

설치 ↑ 이름	버전	Released
<b>Warnings Next Generation</b>	8.2.0	8 days 12 hr ago
This plugin collects compiler warnings or issues reported by static analysis tools and visualizes the results. It has built-in support for numerous static analysis tools (including several compilers), see the list of <a href="#">supported report formats</a> .		
<b>Checkstyle</b>	4.0.0	1 yr 6 mo ago
deprecated Maven Build Reports		
The CheckStyle plug-in reached end-of-life. All functionality has been integrated into the <a href="#">Warnings Next Generation Plugin</a> .		
<b>Warnings</b>	5.0.1	1 yr 5 mo ago
deprecated Maven Build Reports		
The Warnings plug-in reached end-of-life. All functionality has been integrated into the <a href="#">Warnings Next Generation Plugin</a> .		
<b>FindBugs</b>	5.0.0	1 yr 6 mo ago
deprecated Maven Build Reports		
The FindBugs plug-in reached end-of-life. All functionality has been integrated into the <a href="#">Warnings Next Generation Plugin</a> .		
<b>PMD</b>	4.0.0	1 yr 6 mo ago
deprecated Maven Build Reports		
The PMD plug-in reached end-of-life. All functionality has been integrated into the <a href="#">Warnings Next Generation Plugin</a> .		

# [Jenkins] Job 설정

## □ Goal 설정

Build

Invoke top-level Maven targets

Maven Version: Maven363

Goals: pmd:pmd

고급...

ADD BUILD STEP ▾

# [Jenkins] Job 설정

## □ 보고서 생성 설정

Record compiler warnings and static analysis results X

Static Analysis Tools X

Tool **PMD** ?

Report File Pattern `**/target/pmd.xml` ?

Specifies the report files to scan for issues, such as `**/target/checkstyle-results.xml`. If you leave this field empty, then the default file pattern `**/pmd.xml` will be used.

Skip Symbolic Links  ?

Toggle whether the file scanner will skip symbolic links.

Report Encoding ?

Encoding of your report files.

Custom ID ?

Optional custom ID (URL) of this tool, overwrites the built-in ID 'pmd'.

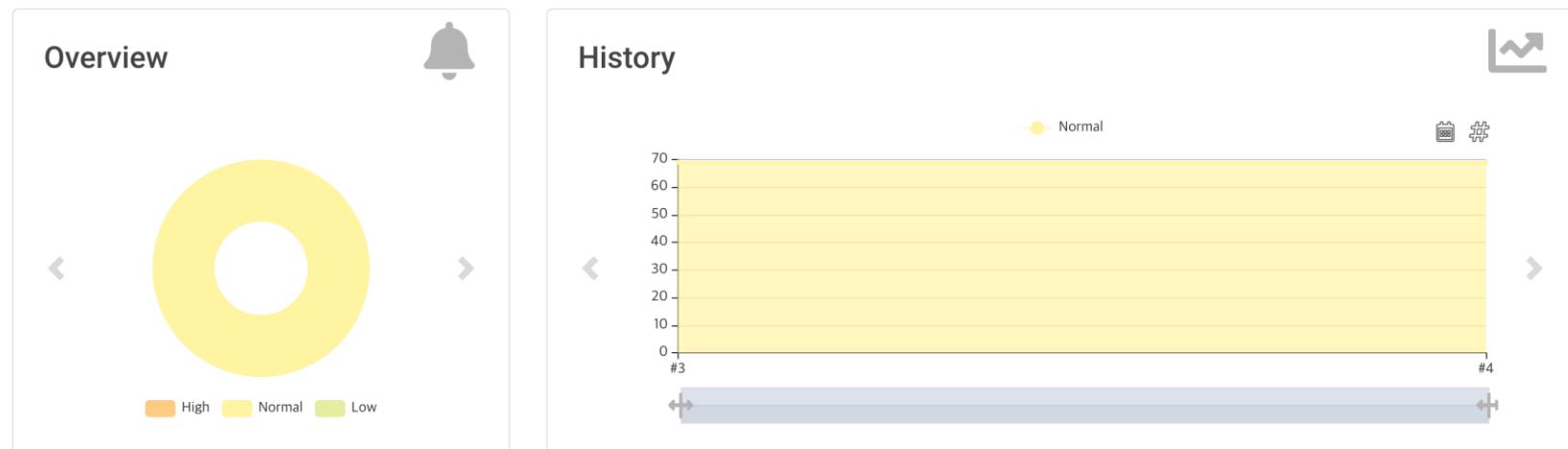
Custom Name ?

Optional custom display name of the tool, overwrites the built-in name 'PMD'.

**ADD TOOL** 고급...

# [Jenkins] PMD 결과

## PMD Warnings



### Details

Packages	Files	Categories	Types	Issues
Show 10 entries				
Search: <input type="text"/>				
Package	Total	Distribution		
junit.framework	11			
junit.runner	2			

# [Jenkins] 위반 상세 화면

## Details

Packages Files Categories Types Issues

Show 10 entries Search:

Package	Total	Distribution
junit.framework	11	
junit.runner	2	
junit.textui	1	
org.junit	4	
org.junit.experimental	1	
org.junit.experimental.results	1	
org.junit.experimental.theories.internal	1	
org.junit.internal	1	
org.junit.internal.management	22	
org.junit.internal.runners	5	
Total	69	

Showing 1 to 10 of 17 entries

1 2

## [실습] Jenkins에서 PMD 분석

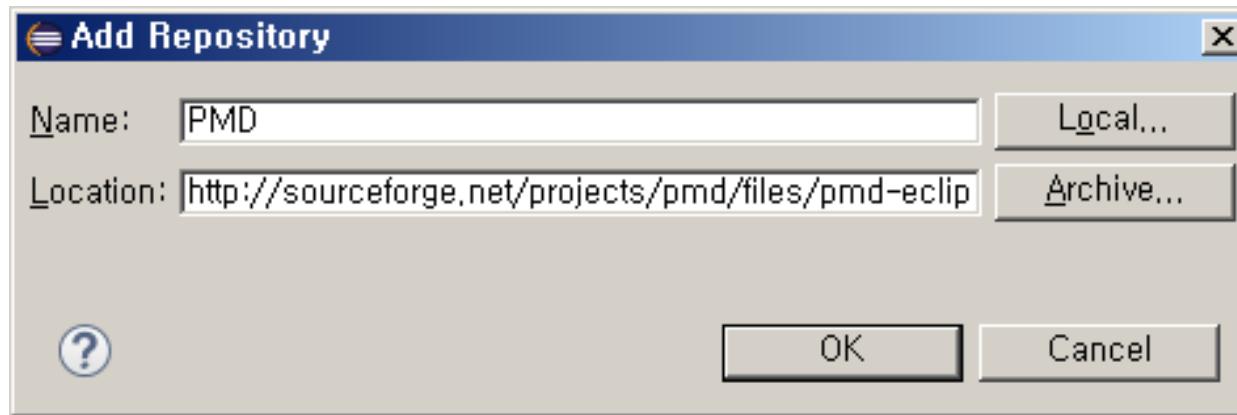
---

- JUnit pom 파일 수정
- 커밋
- Jenkins Job 설정
- 빌드
- 결과 확인

## [Eclipse] 플러그인 설치

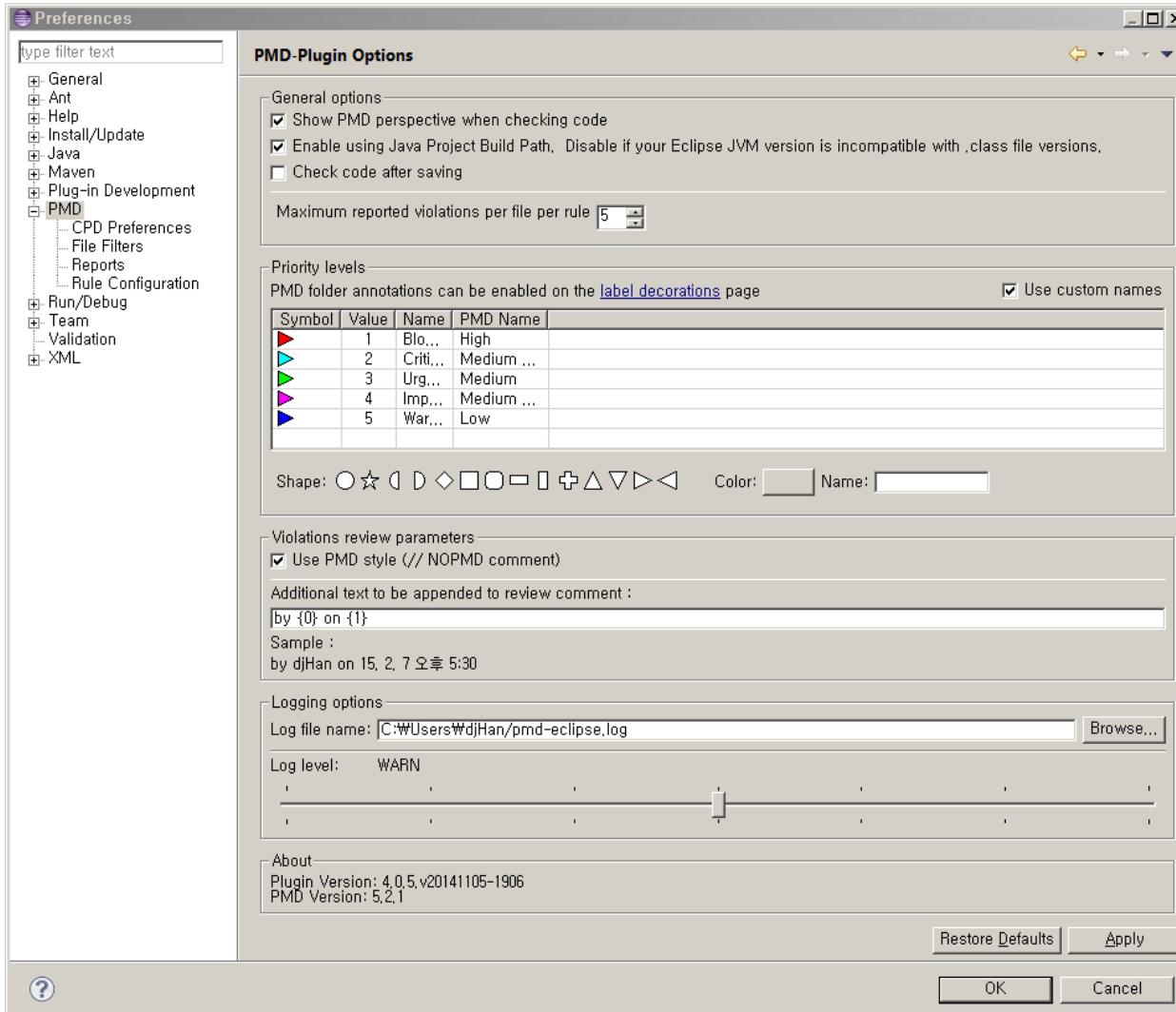
### □ Eclipse Marketplace 가 아닌, Install New Software 방식으로 추가

1. "Help"->"Install New Software" 선택
2. "Add"를 선택
3. Add Repository 창에서 다음 정보를 그림과 같이 입력
  - Name에 "PMD"를 입력
  - Location에 "http://sourceforge.net/projects/pmd/files/pmd-eclipse/update-site/" 입력



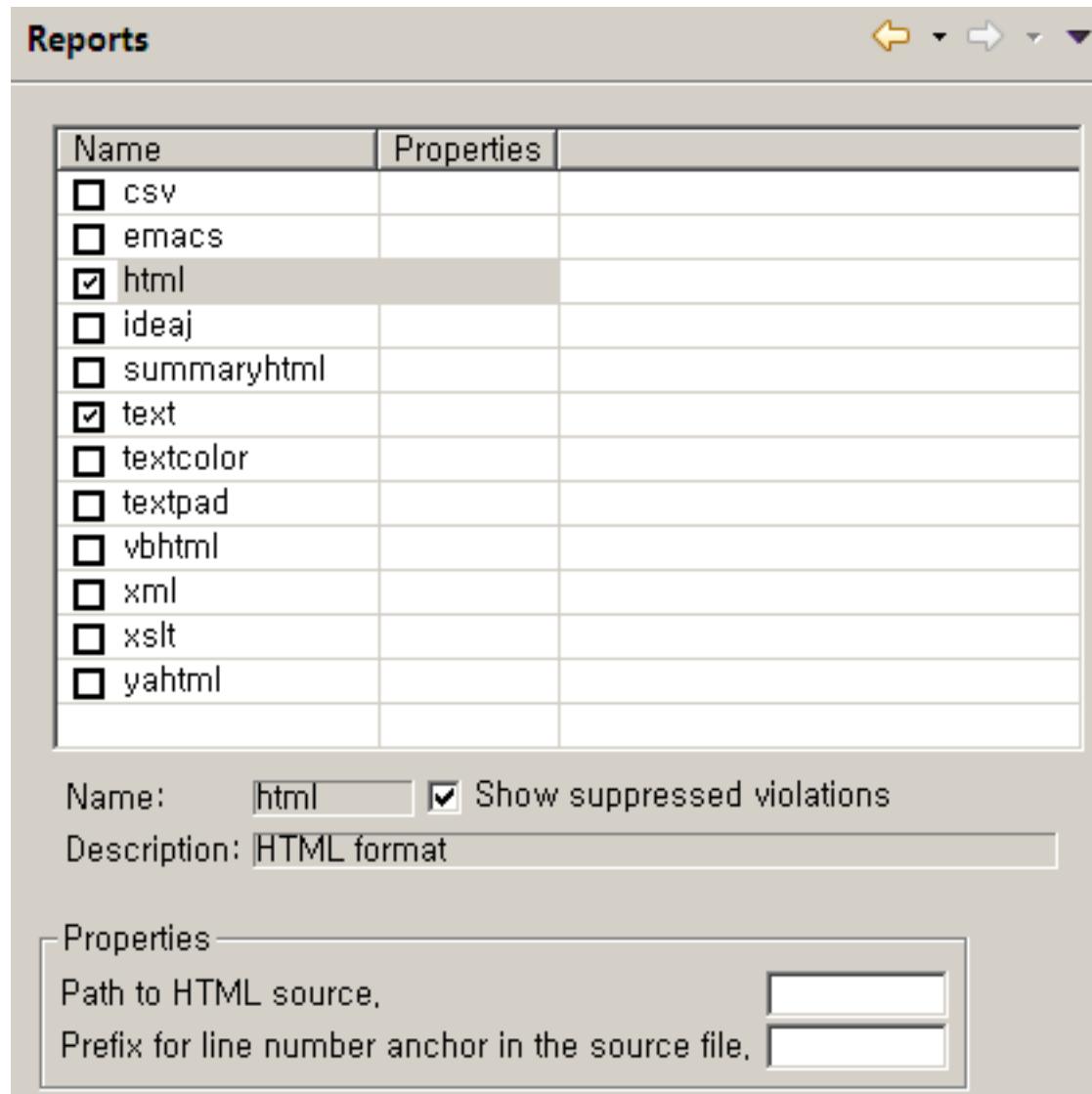
# [Eclipse] PMD 전역 설정

- 여러 프로젝트에 공통적으로 적용하는 설정
- Windows -> Preferences -> PMD를 선택



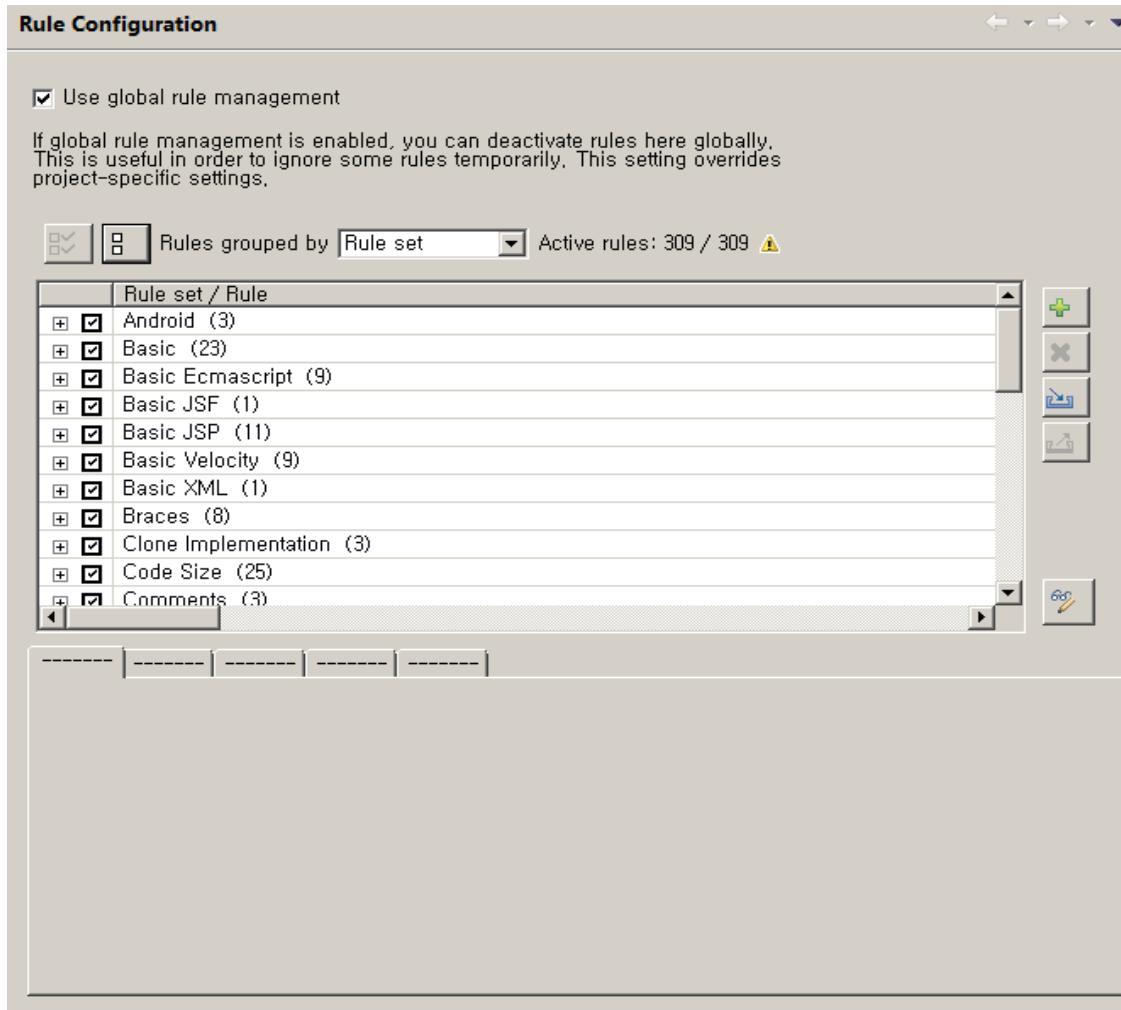
## [Eclipse] PMD 전역 설정

### □ 보고서 생성 방식 설정



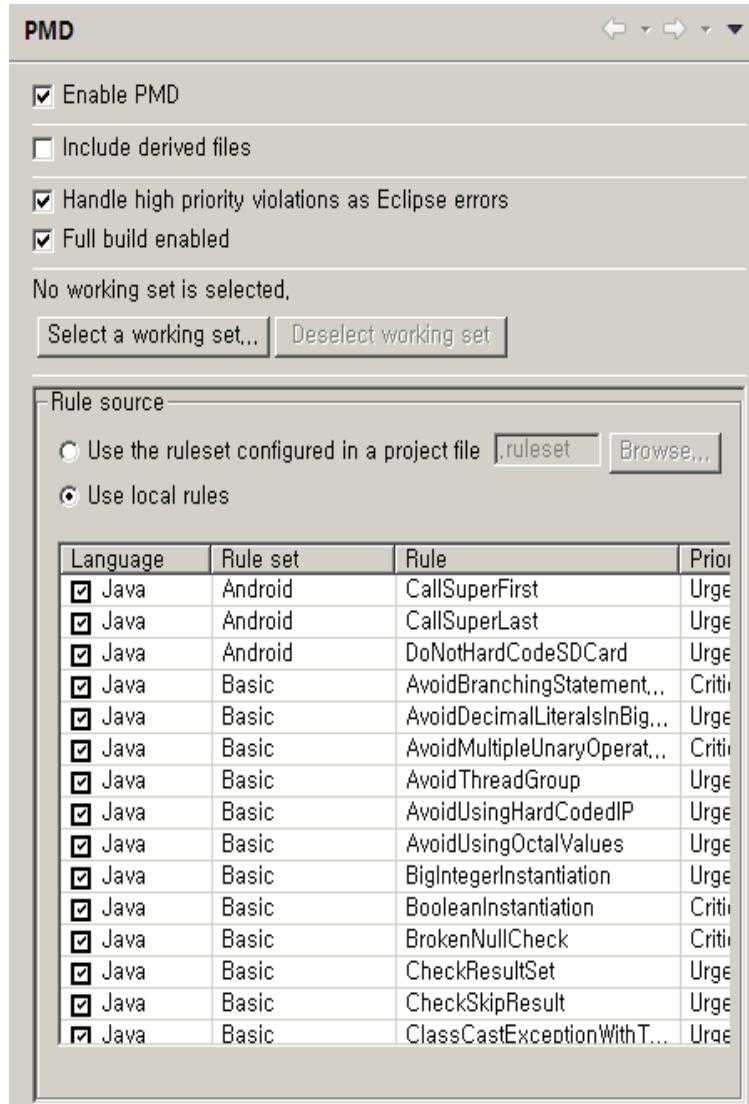
# [Eclipse] PMD 전역 설정

## □ 사용 룰셋 설정



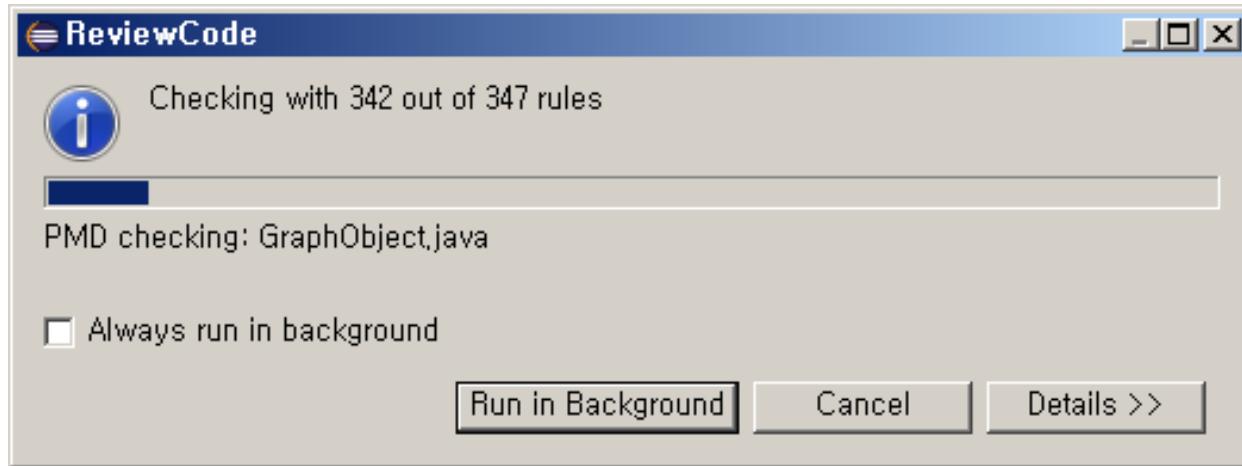
# [Eclipse] PMD 프로젝트 별 설정

## □ 해당 프로젝트에만 적용되는 항목 설정



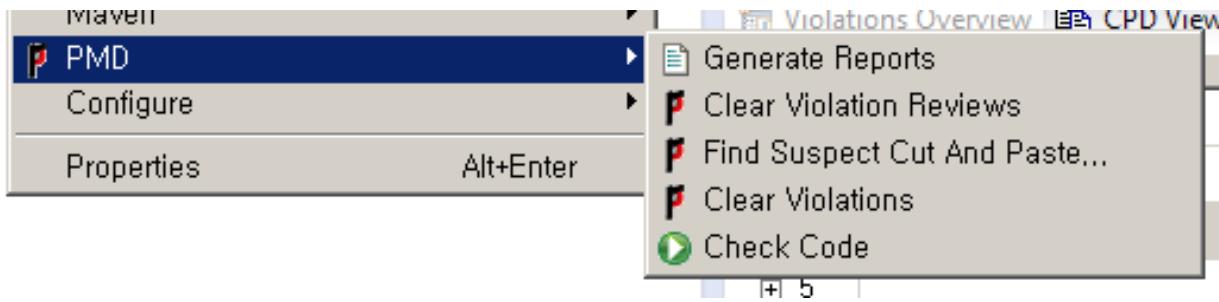
## [Eclipse] 전역 룰과 프로젝트 룰

- PMD 전역설정에서 룰 선택을 제외해도, 프로젝트 별 설정에는 선택되어 있다. 당황하지 말자. 전역설정에서 제외한 룰은 프로젝트 별 설정에 선택되어 있어도 검증하지 않는다.
- 예를 들어 전역설정에서 5개의 룰을 제외했다면, 프로젝트 별 설정에 해당 룰이 선택되어 있어도 다음 그림과 같이 347개 룰에서 5개를 제외한 342개를 검증한다.



# [Eclipse] PMD 실행

## □ 실행 방법: Check Code 실행



## □ 결과 확인

The screenshot shows the Eclipse IDE interface with two views open: 'Violations Outline' and 'Violations Overview'.

**Violations Outline View:** This view lists individual violations with columns for P (Priority), Line, created, Rule, and Error Message. Some entries have red or green icons next to them.

P	Line	created	Rule	Error Message
▶	46	Sat M...	Variabl...	Variables that are fir...
▶	44	Sat M...	Variabl...	Variables that are fir...
▶	30	Sat M...	Comm...	fieldCommentRequir...
▶	35	Sat M...	Comm...	fieldCommentRequir...
▶	42	Sat M...	LongV...	Avoid excessively lo...
▶	32	Sat M...	Comm...	fieldCommentRequir...
▶	19	Sat M...	ImportF...	No need to import a...
▶	34	Sat M...	LongV...	Avoid excessively lo...
▶	24	Sat M...	Comm...	Comment is too larg...
▶	44	Sat M...	Comm...	fieldCommentRequir...
▶	29	Sat M...	Default...	Use explicit scoping...
▶	32	Sat M...	LongV...	Avoid excessively lo...

**Violations Overview View:** This view provides a summary of violations across different elements. It includes columns for Element, # Violations, # Violations/..., and Project.

Element	# Violations	# Violations/...	Project
SessionAuthorizationType.java	3	1500,0	N/A
package-info.java	2	N/A	N/A
CommentSize	2	N/A	N/A
ServerProtocol.java	29	1812,5	N/A
AtLeastOneConstructor	1	62,5	N/A
CommentSize	3	187,5	N/A
DefaultPackage	1	62,5	N/A
LongVariable	7	437,5	N/A
ImportFromSamePackage	1	62,5	N/A
VariableNamingConventions	2	125,0	N/A
CommentRequired	14	875,0	N/A

## [실습] Eclipse에서 JUnit4 분석

---

# PMD 적용 사례

## ◦ PMD 제공 전체 룰

#	Rule Set	Rule 명	선정후보 (QA)	선정후보 (개발자)	룰반영	Sonar 반영	한글 Rule 설명	
48	Design (java)	UnnecessaryLocalBeforeReturn	○		○	○	불필요한 지역 변수 생성은 피해야 함	Avoid the creation o
50	Design (java)	UncommentedEmptyMethod	○		○	○	주석 없는 빈 메소드는 주석 필요	Uncommented Empt
51	Design (java)	UncommentedEmptyConstructor	○		○	○	주석 없는 빈 생성자는 주석 필요	Uncommented Empt
75	Basic (java)	ForLoopShouldBeWhileLoop		○	○	○	for를 while로 간략화 할 수 있는 경우,	Some for loops can
78	Basic (java)	ReturnFromFinallyBlock	○		○	○	finally 블록에서 반환 하는것은 피해야	Avoid returning fro
79	Basic (java)	UnconditionalIfStatement	○		○	○	항상 true 이거나 false인 조건에서는 i	Do not use "if" state
80	Basic (java)	BooleanInstantiation		○	○	○	Boolean 객체는 인스턴스화를 지양함.	Avoid instantiating B
88	Basic (java)	AvoidUsingOctalValues	○		○	○	integer는 0으로 시작하면 안됨. 8진수	Integer literals shou
89	Basic (java)	AvoidUsingHardCodedIP	○		○	○	IP주소를 코드에 하드코딩 하면 안됨	Application with har
101	Strict Exceptions (java)	AvoidThrowingRawExceptionType	○		○	○	가공되지 않은 Exception을 throw하는	Avoid throwing cert
102	Strict Exceptions (java)	AvoidThrowingNullPointerException	○		○	○	NullPointerException을 throw하는 것	Avoid throwing Null
113	Android (java)	DoNotHardCodeSDCard		○	○		"/sdcard"를 사용하는 대신 Environment.get	Use Environment.get
116	Java Logging (java)	SystemPrintln	○		○	○	System.out.println은 보통 디버그 목적	References to Syste
120	Controversial (java)	OnlyOneReturn		○	○	○	메소드는 1개의 return만을 사용해야	A method should ha
121	Controversial (java)	AssignmentInOperand	○		○	○	피연산자내에 할당문이 사용됨. 해당	Avoid assignments i
138	Controversial (java)	AvoidLiteralsInIfCondition		○	○	○	조건문에서 하드 코딩된 literal의 사용	Avoid using hard-co
144	Type Resolution (java)	LooseCoupling		○	○	○	implementation 타입(예. HashSet) 사	Avoid using implem
148	Empty Code (java)	EmptyCatchBlock	○		○	○	빈 catch 블록은 피해야 함	Empty Catch Block f
149	Empty Code (java)	EmptyIfStmt	○		○	○	빈 if 문장은 피해야 함	Empty If Statement f
150	Empty Code (java)	EmptyWhileStmt	○		○	○	빈 while 문장은 피해야 함	Empty While Stateme
151	Empty Code (java)	EmptyTryBlock	○		○	○	빈 try 블록은 피해야 함	Avoid empty try blo
152	Empty Code (java)	EmptyFinallyBlock	○		○	○	빈 finally 블록은 피해야 함	Empty finally blocks
153	Empty Code (java)	EmptySwitchStatements	○		○	○	빈 switch 문장은 피해야 함	Empty switch statem
154	Empty Code (java)	EmptySynchronizedBlock		○	○	○	빈 Synchronized 블록은 피해야 함	Empty synchronized
155	Empty Code (java)	EmptyStatementNotInLoop	○		○	○	for나 while 내의 빈 문장(또는 ;만 있는	An empty statement
156	Empty Code (java)	EmptyInitializer		○	○		빈 초기화는 피해야 함	Empty initializers se
158	Empty Code (java)	EmptyStaticInitializer		○	○	○	빈 static 초기화는 피해야 함	An empty static initi
159	String and StringBuffer (jav	AvoidDuplicateLiterals		○	○	○	반복되는 Stirng literal은 constant로	Code containing du

## CPD 소개

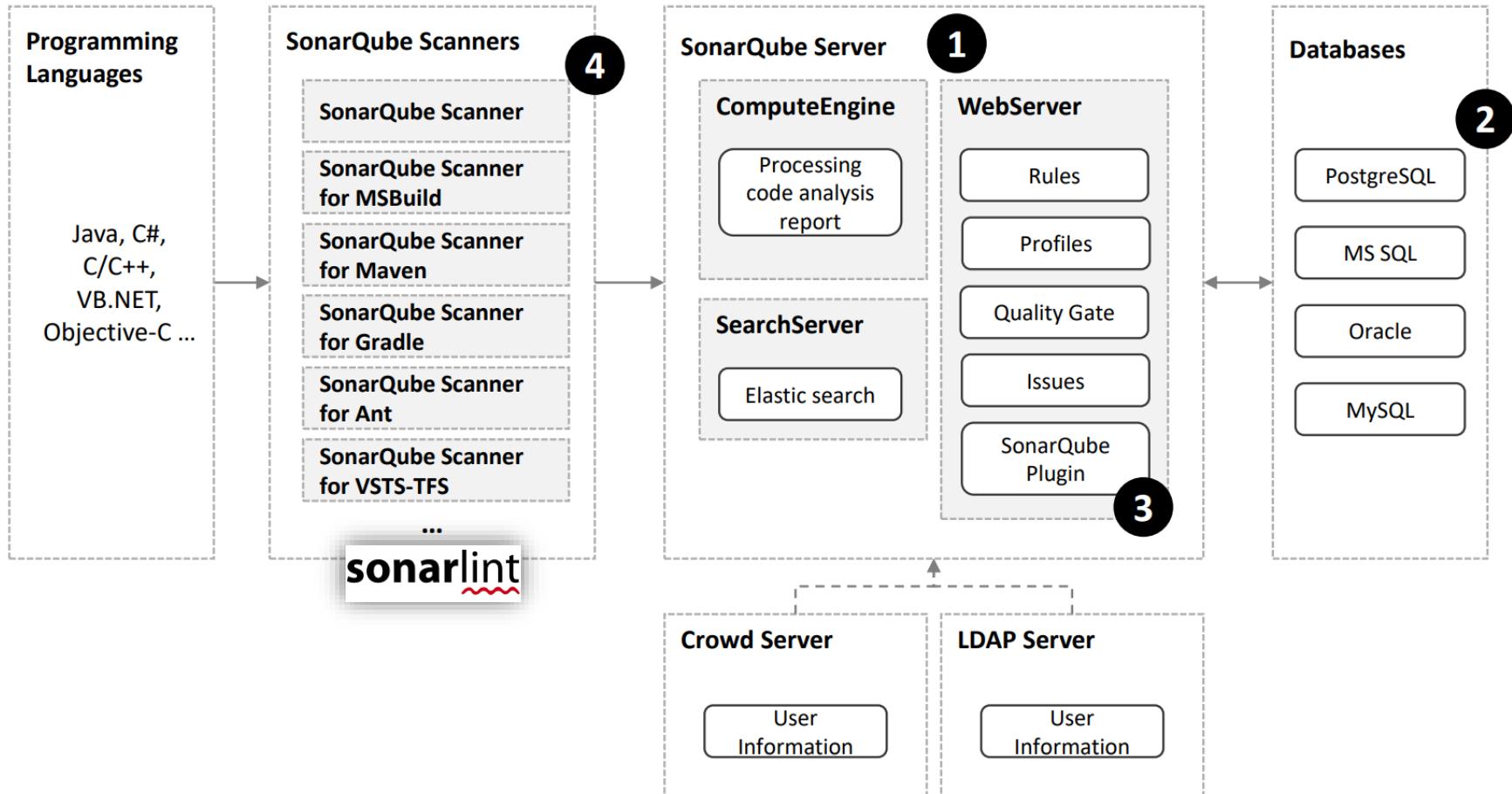
---

- PMD의 부가 기능
- 복사-붙여넣기 코드 탐색
- 설정한 토큰 이상인 경우 탐색

# [참고] SonarQube 소개

## □ 개요

- 27개 이상 언어의 버그, 취약점, 코드악취 발견을 위한 정적분석 및 웹을 통한 결과 게시를 지원
- Server, DB, Scanner, Plugin의 4개 서브 시스템으로 구성



## [참고] SonarQube Ruleset

SonarQube는 Java, Object C, Python을 포함한 27개 이상의 개발 언어를 지원하며, 버그, 코드악취, 취약점으로 코딩 룰을 분류함



**버그:** 실행 중 기대하지 않은 행동을 할 수 있는 코드나 잠재적 버그



**코드악취:** 유지보수성을 저하시키는 기술 부채를 의미



**취약점:** 설계와 다른 방향으로 프로그램을 실행시킬 수 있는 약점

구분	Java	C/C++	C#	Python	Objective C
버그	118	73	74	13	47
코드악취	332	253	256	40	177
취약점	48	2	21	1	2
Security Hotspot	30	0	22	0	0
<b>총합</b>	<b>528</b>	<b>328</b>	<b>373</b>	<b>54</b>	<b>226</b>

[Ruleset 구분]

## 6 JUnit / Cobertura

---

Eclipse에서 JUnit을 이용해 테스트를 작성하고 실행하는 방법을 다룬다.  
Maven과 Jenkins에서 테스트를 수행하고 성공/실패 여부를 확인하자.

JUnit으로 테스트를 하면, 소스코드를 얼마나 테스트 했는지 궁금하다.  
테스트가 부족한 부분이 어디인지 알아야, 보완할 수 있기 때문이다.  
Cobertura는 테스트한 소스코드의 부분, 즉 테스트 커버리지를 알려준다.

# JUnit - 1

---

## □ 단위 테스트

- 보통 메소드를 대상으로 함
- “이런 값을 입력하고 실행하면 아마 이런 출력값이 나올텐데, 정말 예상대로 출력되었는가?” 확인
- 리팩토링의 기본 조건

## □ Assertion

- 메소드의 매개변수(parameter)와 반환값(return value)을 이용하여 테스트
- 특정 메소드에 매개변수를 지정하고 실행하였을 때,  
반환값이 개발자가 예상한 값과 같은지 확인하는 방법

Assertion 메소드	내용
assertEquals([message], expected, actual)	기대값과 결과값이 동일한지 비교
assertNotNull([message], expceted)	기대값이 null이 아닌지 비교
assertNotSame([message], expceted, actual)	기대값과 결과값이 동일한 객체가 아닌지 비교
assertNull([message], expceted)	기대값이 null 인지 비교
assertSame([message], expceted, actual)	기대값과 결과값이 동일한 객체인지 비교

# JUnit 예제

## 더하기 클래스

```
1 package calculate;
2
3 public class Calculate {
4     public int plus(int i, int j){
5         return i+j;
6     }
7 }
8
```

## 테스트 클래스

```
1 package calculate;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5
6 public class CalculateTest {
7
8     @Test
9     public void testPlus() {
10        Calculate calc = new Calculate();
11        int result = calc.plus(1, 2);
12
13        assertEquals(3, result);
14    }
15 }
```

## 테스트 결과

```
Finished after 0.014 seconds
Runs: 1/1 ✘ Errors: 0 ✘ Failures: 0
calculate.CalculateTest [Runner: JUnit 4] (0.000 s)
  testPlus (0.000 s)
```

## 기능이 추가된 계산기 클래스

---

```
package calculate;
```

```
public class Calculate {
```

```
    public int plus(int i, int j){
```

```
        return i+j;
```

```
}
```

```
    public int multiple(int i, int j){
```

```
        return i*j;
```

```
}
```

```
    public int minus(int i, int j){
```

```
        return i-j;
```

```
}
```

```
}
```

## 기능이 추가된 계산기 테스트 클래스

```
package calculate;

import static org.junit.Assert.*;
import org.junit.Test;

public class CalculateTest {

    @Test
    public void testPlus() {
        Calculate calc = new Calculate();
        int result = calc.plus(1, 2);

        assertEquals(3, result);
    }

    @Test
    public void testPlus3_4() {
        Calculate calc = new Calculate();
        int result = calc.plus(3, 4);

        assertEquals(7, result);
    }

    @Test
    public void testMultiple() {
        Calculate calc = new Calculate();
        int result = calc.multiple(2, 3);

        assertEquals(6, result);
    }

    @Test
    public void testMinus() {
        Calculate calc = new Calculate();
        int result = calc.minus(7, 2);

        assertEquals(5, result);
    }
}
```

## 테스트 케이스 이름 만들기

### □ 테스트 이름 만드는 것에 제한은 없으나, 관행적으로 test메소드명() 형식으로 작성

- JUnit3 는 이런 방법이어야 테스트로 인식
- JUnit4 는 @Test를 사용해서 제한이 없음

### □ Java의 특성 상 한글로 만드는 것도 가능

- 해외 진출, 협업 등이 아니라면, 한글 테스트도 좋은 방법

```
@Test  
public void 더하기테스트3과4(){  
    Calculate calc = new Calculate();  
    int result = calc.plus(3, 4);  
  
    assertEquals(7, result);  
}
```

## JUnit - 2

---

### □ Annotation

Annotation	내용
@Test	다음에 나오는 메소드가 JUnit 테스트 메소드임을 지정
@Before	각각의 테스트를 실행하기 전에 한 번씩 실행하는 메소드 예를 들어, 모든 테스트 실행 전에 초기화된 객체를 생성해야 할 때 사용 <code>import org.junit.Before;</code> 필요
@After	각각의 테스트를 실행한 후에 한 번씩 실행하는 메소드 예를 들어, 모든 테스트 실행 후에 자원 해제 등을 해야할 때 사용 <code>import org.junit.After;</code> 필요

## @Before Annotation 적용 코드

```
package calculate;

import static org.junit.Assert.*;

import org.junit.Before; // org.junit.Before를 Import 해야 함
import org.junit.Test;

public class CalculateTest {

    Calculate calc;

    @Before
    public void setUp(){
        calc = new Calculate();
    }

    @Test
    public void testPlus() {
        int result = calc.plus(1, 2);

        assertEquals(3, result);
    }

    @Test
    public void testPlus3_4() {
        int result = calc.plus(3, 4);

        assertEquals(7, result);
    }
}
```

```
@Test
public void testMultiple() {
    int result = calc.multiple(2, 3);

    assertEquals(6, result);
}

@Test
public void testMinus() {
    int result = calc.minus(7, 2);

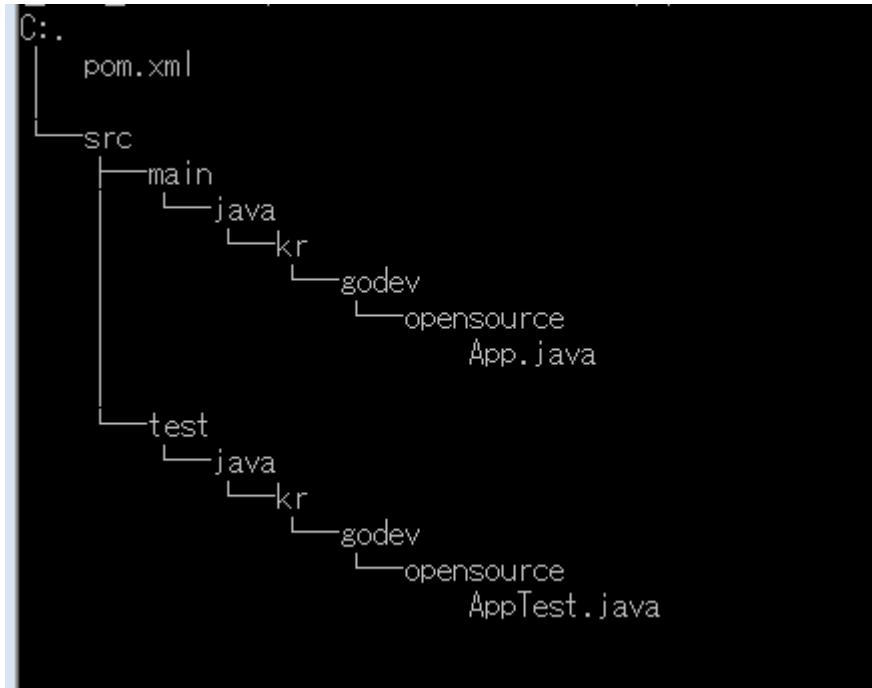
    assertEquals(5, result);
}

@Test
public void 더하기테스트3과4(){
    int result = calc.plus(3, 4);

    assertEquals(7, result);
}
}
```

## [Eclipse] Maven 에서 JUnit 테스트를 위한 구조

- 테스트를 위한 구조를 소스코드 구조와 동일하게 생성



## [Eclipse] Maven default 라이프사이클 다시 보기

### □ 3번째 test 단계부터 테스트 수행

라이프사이클	설명
validate	정상적인 프로젝트이고 필요한 모든 정보가 준비되었는지 확인
compile	프로젝트의 소스코드를 컴파일
test	컴파일된 소스코드를 단위 테스트 프레임워크를 이용해 테스트
package	JAR와 같이 지정된 포맷으로 패키징
verify	패키지가 품질 수준을 만족하는지 검증
install	패키지를 로컬 저장소에 설치
deploy	최종 패키지를 원격 저장소로 복사

# [Eclipse] 테스트 실행

## □ maven 명령 실행

**mvn test**

```
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building cal 0.0.1-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ cal ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
.....
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ cal ---
[INFO] Surefire report directory: C:\Users\DJHan\workspace\cal\target\surefire-reports
```

-----  
T E S T S

```
-----  
Running calculate.CalculateTest  
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.056 sec
```

Results :

```
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
```

```
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.978 s
[INFO] Finished at: 2016-05-10T21:52:58+09:00
[INFO] Final Memory: 9M/245M
[INFO] -----
```

# Cobertura

---

## □ 소스코드의 얼마만큼 테스트 했는가?

- 라인(Line) 커버리지 (Statement 커버리지 라고도 함)
- 브랜치(Branch) 커버리지 (Decision 커버리지 라고도 함)

## □ 왜 소스코드 테스트 커버리지가 중요한가?

- 소스코드 품질을 위한 테스트 목표 (QA가 닥달하기 딱 좋은 지표)
  - 안전성 분야에서는 100%가 기준
  - Cobertura를 사용하면, **커버리지 미달 시 Maven 빌드 실패하도록 설정 가능**
- 테스트가 부족한 부분은 확인
  - 불필요한 코드인가?
  - 테스트가 충분한가?

# Line 커버리지

---

## □ Line(Statement) 커버리지 산출 공식

- 테스트한 구문 / 전체 구문

※ 구문(statement): ';'로 끝나는 명령 라인

## □ 의미

- 프로그래밍 언어의 기본 구성 단위에 대한 검증
- 개발자가 구현한 전체 구문 및 블록에 대하여 최소한의 검증 수행
- 낮은 단계의 커버리지이나 시스템이 복잡할 수록 달성하기 어려움

# Line 커버리지

## □ 예제

```
#include <stdio.h>

void api1(int i){

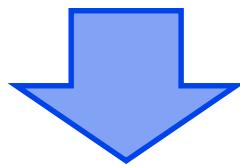
    printf("...\\n");
    if(i<0){
        printf("...\\n");
        printf("...\\n");
    }
}
```

### Statement Coverage

Q1) i를 0으로 테스트 한 경우:

Q2) i를 -1로 테스트 한 경우:

기혼자이고  
나이가 35세를 초과하면  
100만원의 소득 공제를 받음



Condition      Condition  
if(married && (age > 35))  
                      ----- Decision  
then amount = amount + 100;

# Branch(Decision) 커버리지

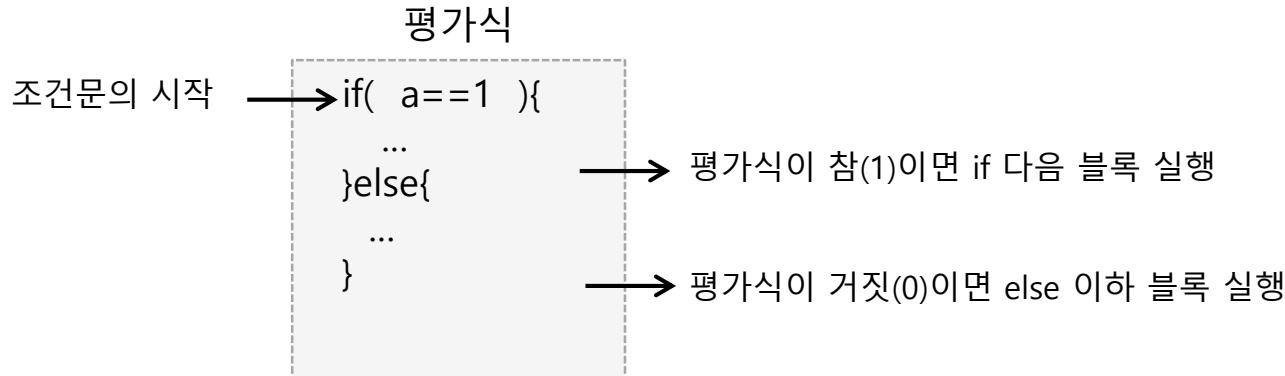
## ✓ Decision Coverage 산출 공식

- 테스트한 Decision 수 / 전체 Decision 수
- Decision Coverage 100% 달성하기 위한 최대 테스트 케이스 수
  - 전체 조건문 수 \* 2

## ✓ 의미

- 결정의 참/거짓을 한 번씩 테스트 해보는 것

## ✓ 코드 분석 팁(Code Reading Tip)



- 각 조건문은 2개의 분기를 가진다.

## Branch(Decision) 커버리지

```
void util(int a){  
    if( a>0 ){  
        .....  
    }  
  
    if( a==1 ){  
        ...  
    }else{  
        ...  
    }  
}
```

Q1) a를 0으로 테스트 한 경우:

Q2) a를 1으로 테스트 한 경우:

Q3) Decision Coverage 100%를 달성하기 위한 최소개의 테스트 케이스는?

# 도구 모음

## Packages

All  
[junit.extensions](#)  
[junit.framework](#)  
[junit.runner](#)  
[junit.textui](#)  
[org.junit](#)  
[org.junit.experimental](#)  
[org.junit.experimental.categories](#)  
[org.junit.experimental.max](#)

## All Packages

### Classes

[ActiveTestSuite \(72%\)](#)  
[After \(N/A\)](#)  
[AfterClass \(N/A\)](#)  
[AllDefaultPossibilitiesBuilder \(](#)  
[AllMembersSupplier \(81%\)](#)  
[AllTests \(100%\)](#)  
[Annotatable \(N/A\)](#)  
[AnnotatedBuilder \(100%\)](#)  
[AnnotationValidator \(60%\)](#)  
[AnnotationValidatorFactory \(7](#)  
[AnnotationsValidator \(97%\)](#)  
[ArrayComparisonFailure \(95%](#)  
[Assert \(69%\)](#)  
[Assert \(89%\)](#)  
[AssertionFailedError \(100%\)](#)  
[Assignments \(97%\)](#)  
[Assume \(57%\)](#)  
[AssumptionViolatedException](#)  
[BaseTestRunner \(47%\)](#)  
[Before \(N/A\)](#)  
[BeforeClass \(N/A\)](#)  
[BlockJUnit4ClassRunner \(100%](#)

## Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	257	86% 3521/4091	84% 1095/1302	1.775
<a href="#">junit.extensions</a>	6	82% 52/63	87% 7/8	1.25
<a href="#">junit.framework</a>	17	76% 399/525	90% 139/154	1.605
<a href="#">junit.runner</a>	3	46% 67/144	41% 20/48	2.158
<a href="#">junit.textui</a>	2	76% 99/130	76% 23/30	1.686
<a href="#">org.junit</a>	14	86% 211/244	86% 85/98	1.674
<a href="#">org.junit.experimental</a>	2	91% 21/23	83% 5/6	1.5
<a href="#">org.junit.experimental.categories</a>	11	93% 149/159	94% 83/88	2.571
<a href="#">org.junit.experimental.max</a>	8	85% 92/108	86% 26/30	1.969
<a href="#">org.junit.experimental.results</a>	6	92% 37/40	87% 7/8	1.222
<a href="#">org.junit.experimental.runners</a>	1	100% 7/7	100% 4/4	2
<a href="#">org.junit.experimental.theories</a>	15	93% 167/178	83% 72/86	1.961
<a href="#">org.junit.experimental.theories.internal</a>	8	91% 189/207	85% 77/90	2.413
<a href="#">org.junit.experimental.theories.suppliers</a>	2	100% 7/7	100% 2/2	2
<a href="#">org.junit.internal</a>	13	92% 152/164	94% 53/56	1.881
<a href="#">org.junit.internal.builders</a>	8	98% 57/58	92% 13/14	2
<a href="#">org.junit.internal.matchers</a>	4	75% 40/53	0% 0/18	1.391
<a href="#">org.junit.internal.requests</a>	3	100% 29/29	75% 3/4	1.571
<a href="#">org.junit.internal.runners</a>	18	73% 312/423	63% 84/132	2.194
<a href="#">org.junit.internal.runners.model</a>	3	100% 26/26	100% 4/4	1.5
<a href="#">org.junit.internal.runners.rules</a>	1	100% 40/40	100% 24/24	2.091
<a href="#">org.junit.internal.runners.statements</a>	8	92% 128/139	73% 28/38	3.136
<a href="#">org.junit.matchers</a>	1	9% 1/11	N/A	N/A
<a href="#">org.junit.rules</a>	21	87% 233/265	100% 30/30	1.311
<a href="#">org.junit.runner</a>	19	94% 210/222	90% 49/54	1.516
<a href="#">org.junit.runner.manipulation</a>	9	90% 38/42	100% 18/18	1.632
<a href="#">org.junit.runner.notification</a>	14	97% 140/144	80% 16/20	1.347
<a href="#">org.junit.runners</a>	16	97% 346/354	96% 110/114	1.798
<a href="#">org.junit.runners.model</a>	15	96% 225/233	91% 102/112	1.872
<a href="#">org.junit.validator</a>	9	88% 47/53	91% 11/12	1.733

Report generated by Cobertura 2.0.3 on 14. 4. 6 오후 2:03.

## [Maven] Goal 설명

---

Goal	설명
cobertura:check	설정한 커버리지를 달성하지 못한 경우 빌드를 실패로 처리한다.
cobertura:clean	Cobertura가 사용한 파일을 정리한다.
cobertura:cobertura	테스트 커버리지 보고서를 생성한다.

## [Maven] Usage – cobertura:cobertura

- 테스트 커버리지 보고서는 '`target\site\cobertura`' 위치에 생성

```
<project>
  ...
  <reporting>
    <plugins>
      ...
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>cobertura-maven-plugin</artifactId>
        <version>2.6</version>
      </plugin>
      ...
    </plugins>
  </reporting>
  ...
</project>
```

## [Maven] Usage – cobertura:check

```
<project>
  <build>
    ...
    <plugins>
      ...
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>cobertura-maven-plugin</artifactId>
        <version>2.6</version>
        <configuration>
          <check>
            <branchRate>80</branchRate>
            <lineRate>80</lineRate>
            <totalBranchRate>85</totalBranchRate>
            <totalLineRate>85</totalLineRate>
            <packageLineRate>85</packageLineRate>
            <packageBranchRate>85</packageBranchRate>
            <haltOnFailure>true</haltOnFailure>
          </check>
        </configuration>
      </plugin>
      ...
    </plugins>
    ...
  </build>
</project>
```

## [Maven] <configuration> 설정 값의 의미

---

Tag	내용
<branchRate>80</branchRate>	클래스 분기 커버리지 기준 80%
<lineRate>80</lineRate>	클래스 구문 커버리지 기준 80%
<totalBranchRate>85</totalBranchRate>	전체 테스트의 분기 커버리지 기준 85%
<totalLineRate>85</totalLineRate>	전체 테스트의 구문 커버리지 기준 85%
<packageLineRate>85</packageLineRate>	패키지의 구문 커버리지 85%
<packageBranchRate>85</packageBranchRate>	패키지의 분기 커버리지 85%

## [실습] JUnit 4의 커버리지 확인하기

---

□ mvn clean install cobertura:cobertura

## [Eclipse] Cobertura 플러그인 설정

---

- Cobertura 플러그인이 있으나, 최신 버전에서는 정상 작동하지 않음
- 유사한 기능을 수행하는 eclEmma 플러그인 사용 추천
  
- 개발자의 입장에서는, Eclipse에서 코딩/테스트/커버리지 확인이 가장 생산적임

# [Jenkins] 플러그인 설치

## □ 플러그인 설정에서 Cobertura 검색

필터:

업데이트된 플러그인 목록    설치 가능    설치된 플러그인 목록    고급

설치	이름 ↓	버전
<input checked="" type="checkbox"/>	<a href="#">Cobertura Plugin</a> This plugin allows you to capture code coverage report from <a href="#">Cobertura</a> . Jenkins will generate the trend report of coverage.	1.9.7
<input type="checkbox"/>	<a href="#">Coverage/Complexity Scatter Plot Plugin</a> This plugin shows the coverage/complexity scatter plot from Clover or Cobertura plugin results.	1.1.1
<input type="checkbox"/>	<a href="#">Graphite-plugin</a> This plugin allows you to send these metrics : number of tests, tests skipped, tests failed, build duration, cobertura total line coverage and cobertura total branch coverage to one or more graphite servers. If you don't have a graphite server you can use : [https://www.hostedgraphite.com] to test. For cobertura metrics you need to install cobertura plugin and run cobertura:cobertura in goals section. Be sure to run jenkins your jobs with a Jdk 7 (go to the manage jenkins + configure system + Jdk installations ), because the plugin only works with this version of jdk.	1.2

[재시작 없이 설치하기](#)    [지금 다운로드하고 재시작 후 설치하기](#)    Update information obtained

# [Jenkins] Job 설정

## □ Goal에 cobertura 실행 추가

### Build

Root POM

pom.xml



Goals and options

clean cobertura:cobertura



## □ publish cobertura coverage report에 커버리지 xml 결과 값 위치 추가

- 일반적으로, 기본 값을 그대로 이용하면 됨
- Maven pom 파일에서 xml 출력을 위한 추가 설정 필요 (다음 페이지 참고)

### Publish Cobertura Coverage Report

Cobertura xml report pattern

\*\*/target/site/cobertura/coverage.xml

This is a file name pattern that can be used to locate the cobertura xml report files (for example with Maven2 use \*\*/target/site/cobertura/coverage.xml). The path is relative to the module root unless you have configured your SCM with multiple modules, in which case it is relative to the workspace root. Note that the module root is SCM-specific, and may not be the same as the workspace root.  
Cobertura must be configured to generate XML reports for this plugin to function.

고급...

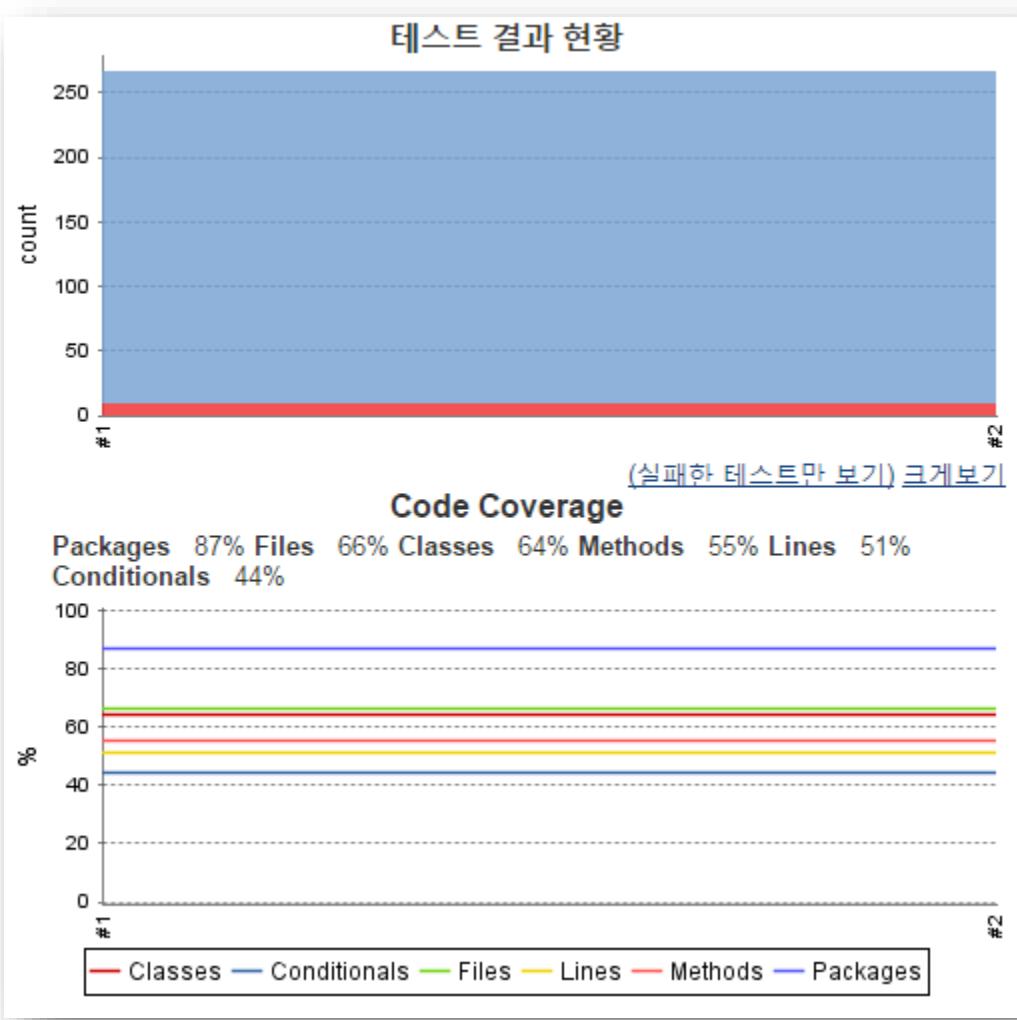
## (참고) Jenkins를 위한 xml 출력 설정

```
<project>
  <build>
    ...
    <plugins>
      ...
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>cobertura-maven-plugin</artifactId>
        <version>2.6</version>
        <configuration>
          <formats>
            <format>xml</format>
          </formats>
        </configuration>
      </plugin>
      ...
    </plugins>
    ...
  </build>
</project>
```

# [Jenkins] 커버리지 결과 확인

Jenkins > Junit\_Cobertura > #1

- 프로젝트로 돌아가기
- 상태
- 바뀐점
- Console Output
- 빌드 정보 수정
- 이 빌드를 삭제
- Tag this build
- Test Result
- Redeploy Artifacts
- Coverage Report
- See Fingerprints



## [실습] Jenkins에서 JUnit 4 빌드

---

- JUnit 4 의 pom 수정
- SVN 커밋
- Jenkins Job 설정
- 빌드
- 결과 확인

## 7 JDepend

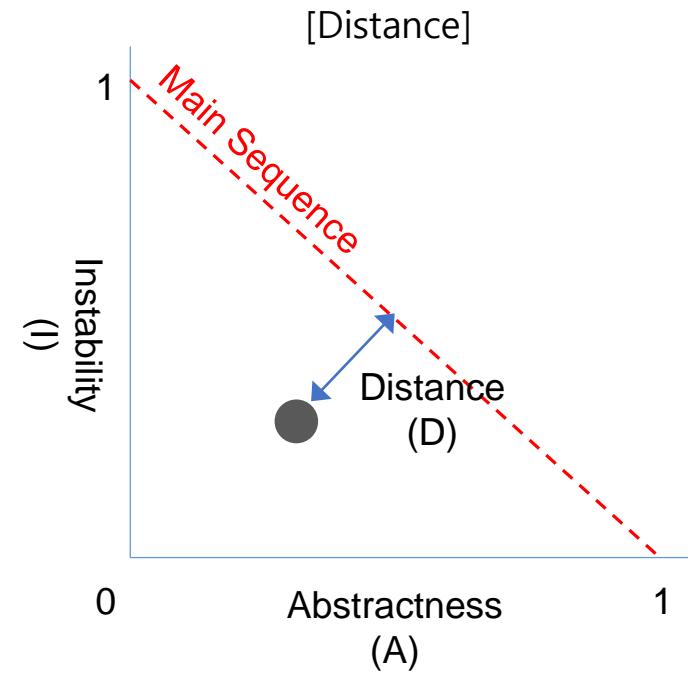
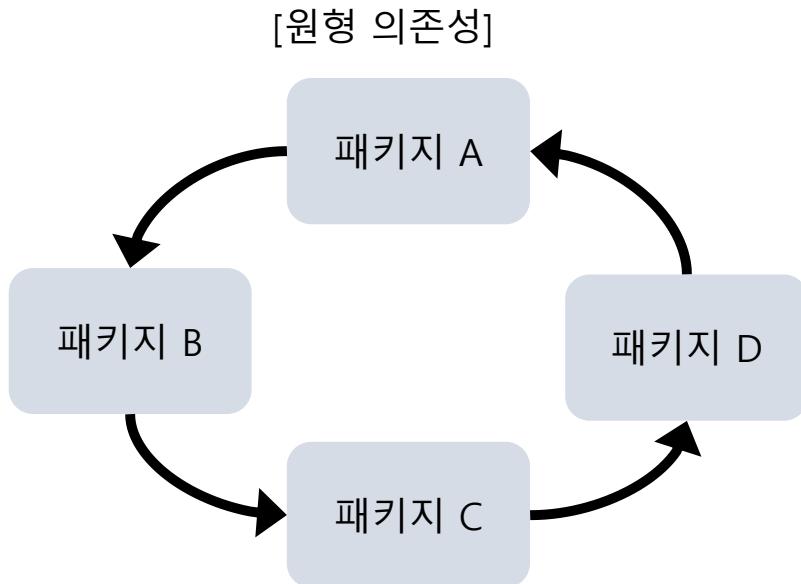
---

JDepend는 밥 아저씨, Robert C. Martin의 설계 품질 측정값을 알려준다.  
내가 너를 호출하는데, 너도 나를 호출하는 원형 의존성을 보게 될 것이다.  
그럼 나를 수정하면, 영향 받는건 누가 될까?

# JDepend

## ▣ 도구 개요

- 패키지 간 의존성 분석
  - 나를 호출한 패키지
  - 내가 호출한 패키지
- 특히, 원형 의존성 존재 여부 확인 가능
- Distance 분석



# 의존성 분석

## □ 개요

- 함수, 변수의 **호출관계를 분석**하는 도구
- 도구가 추구하는 방향에 따라 패키지/클래스(파일)/함수 단위로 표현

## □ 목적

- 아키텍처 구조에서 서브 시스템 간의 의존이 적절한지 확인
  - 서브 시스템 레이어에서 각 레이어 간 호출 관계 (예. Autosar)
  - 서브 시스템 레벨에서 각 레벨 간 호출 관계 (자식과 부모 패키지)
  - 원형 의존성(상호 참조) 관계
- 객체지향에 특화되어, 추상화와 구체화의 정도를 확인 (Part 2 내용)

## □ 대표 도구

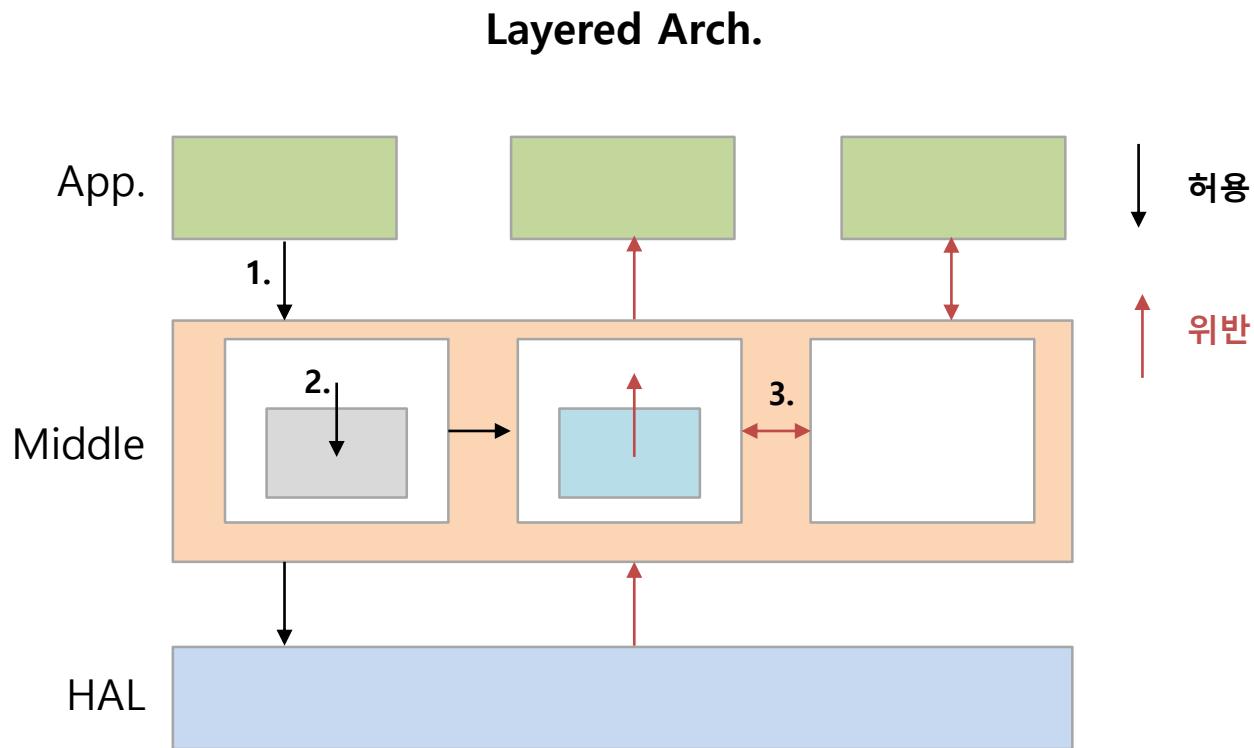
언어	외산		오픈소스
C/C++	<a href="#">Lattix</a>	CppDepend	Dxygen
Java	<a href="#">Imagix4D</a>	JArchitect	<a href="#">JDepend</a> Dxygen

## □ 의존성 분석 표현 종류

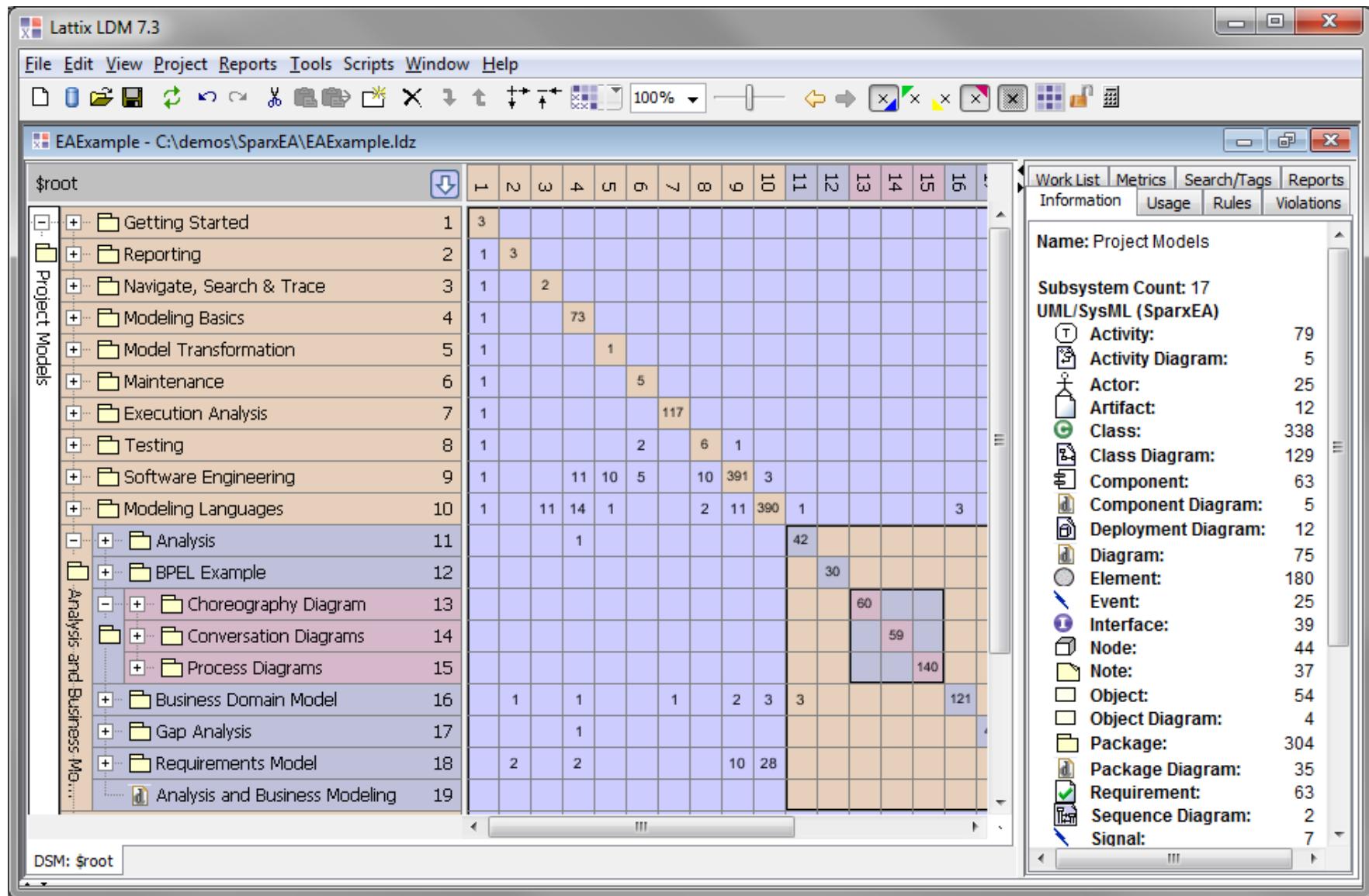
- DSM: Dependency Structure Matrix로, X/Y축의 형태로 서브 시스템 간의 의존성 확인에 유리
- 지표: 호출 대상을 지표로 표현
  - 특히, 객체지향 관련 도구는 **Robert C. Martin**의 논문을 근거로, 의존성 관련 지표를 정의하고 숫자로 표현
- 다이어그램: 호출 관계를 다이어그램으로 표현

## Layered Architecture에서 서브 시스템간 의존성

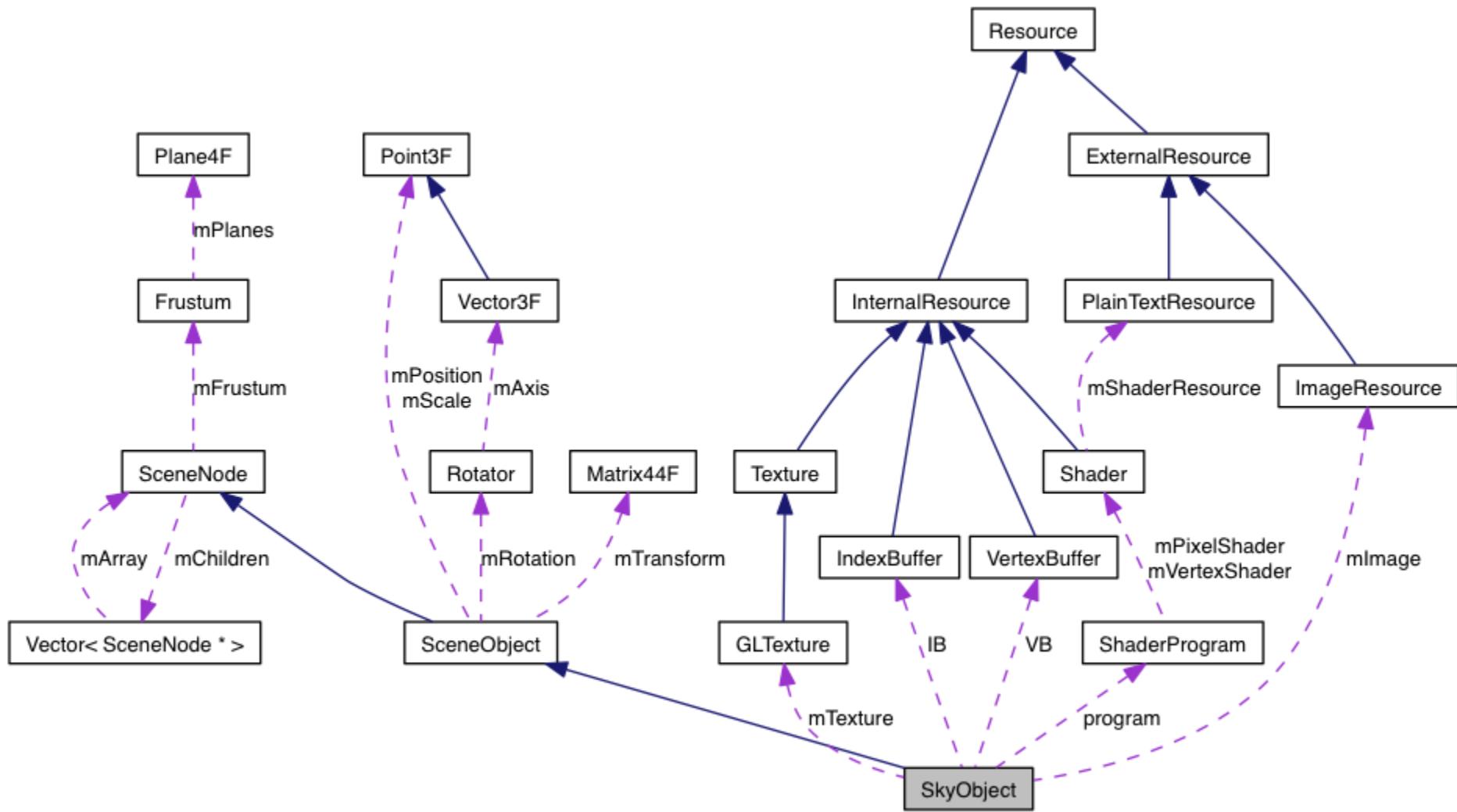
1. 각 레이어 간 호출 관계 (예. Autosar)
2. 동일 레벨 간 호출 관계 (자식과 부모 패키지)
3. 원형 의존성(상호 참조) 관계



# 예제) DSM - Lattix

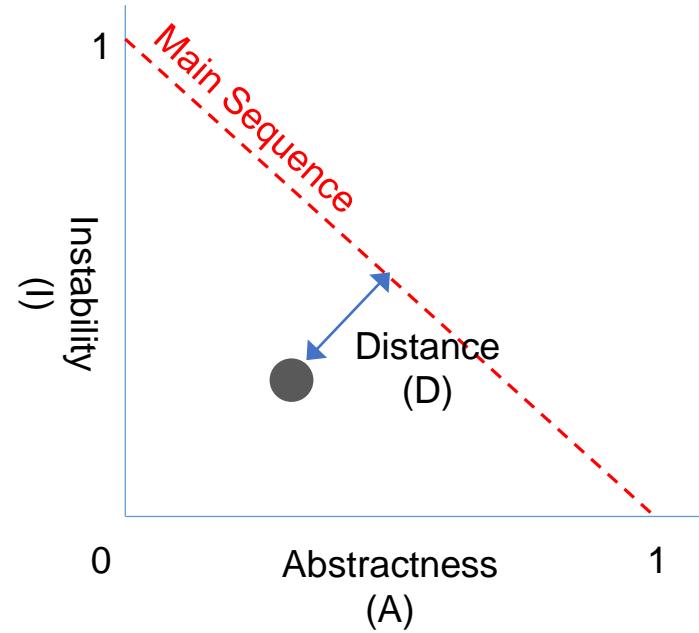


# 예제) 다이어그램 - Doxygen



# Robert C. Martin – OO Metrics

- 기준은 패키지 단위
- **CC(Concrete Class)**
  - 인터페이스나 추상 클래스가 아닌 구체 클래스의 수
- **AC(Abstract Class)**
  - 추상 클래스나 인터페이스의 수를 나타내며 확장성의 척도
- **Ca(Afferent Couplings)**
  - 나에게 의존하는 패키지 수를 나타내며 책임의 척도
- **Ce(Efferent Couplings)**
  - 내가 호출하는 패키지의 수를 나타내며 독립성의 척도
- **A(Abstractness)**
  - $A = AC / (AC+AC)$  추상화 정도를 나타내며, 0 은 구체적인 패키지이며, 1 은 추상적인 패키지
- **I(Instability)**
  - $I = Ce / (Ce+Ca)$  변화에 대한 안정성을 나타내며 0 부터 1 사이의 값
  - 0 은 외부 변화에 영향 없는 패키지이며, 1 은 작은 변화에도 영향 받는 패키지
- **D(Distance to Main Sequence)**
  - Main Sequence 로부터의 거리
  - Main Sequence란 이상적인 패키지로 완전 추상적이면서 안정적이거나 완전 구체적이면서 불안정한 패키지
- **Cycle(Package dependency cycles)**
  - 패키지들 상호 간에 의존성을 가지고 있을 때 발생



# JDepend 모음

JDepend - Eclipse

File Edit Navigate Search Project Run Window Help

Quick Access Java JDepend

Packages 1

- > junit.extensions
- > junit.framework
- > junit.runner
- > junit.textui
- > org.junit
- > org.junit.experimental
- > org.junit.experimental.categories
- > org.junit.experimental.max
- > org.junit.experimental.results
- > org.junit.experimental.runners
- > org.junit.experimental.theories
- > org.junit.experimental.theories.internal
- > org.junit.experimental.theories.suppliers
- > org.junit.function
- > org.junit.internal
- > org.junit.internal.builders
- > org.junit.internal.matchers
- > org.junit.internal.requests
- > org.junit.internal.runners
- > org.junit.internal.runners.model
- > org.junit.internal.runners.rules
- > org.junit.internal.runners.statements
- > org.junit.matchers
- > org.junit.rules

Metrics 2

Instability ->

Selected object(s) 3

Package	CC(co...)	AC(a...)	Ca(aff.)	Ce(eff.)	A	I	D	Cycle
junit.extensions	6	1	2	1	0.14	0.33	0.52	<span style="color: red;">!</span>

Packages with cycle 4

Package	CC(co...)	AC(a...)	Ca(aff.)	Ce(eff.)	A	I	D	Cycle
junit.extensions	6	1	2	1	0.14	0.33	0.52	<span style="color: red;">!</span>

Depends upon - efferent dependencies 5

Package	CC(co...)	AC(a...)	Ca(aff.)	Ce(eff.)	A	I	D	Cycle
junit.extensions	6	1	2	1	0.14	0.33	0.52	<span style="color: red;">!</span>
junit.framework	13	5	17	5	0.27	0.22	0.49	<span style="color: red;">!</span>
junit.runner	1	3	2	2	0.75	0.50	0.25	<span style="color: red;">!</span>
junit.samples.money	0	0	1	0	0.00	0.00	1.00	
junit.tests	0	0	1	0	0.00	0.00	1.00	
junit.textui	2	1	2	2	0.33	0.50	0.16	<span style="color: red;">!</span>
org.hamcrest	0	0	27	0	0.00	0.00	1.00	

Used by - afferent dependencies 6

Package	CC(co...)	AC(a...)	Ca(aff.)	Ce(eff.)	A	I	D	Cycle
junit.extensions	6	1	2	1	0.14	0.33	0.52	<span style="color: red;">!</span>
junit.framework	13	5	17	5	0.27	0.22	0.49	<span style="color: red;">!</span>
junit.runner	1	3	2	2	0.75	0.50	0.25	<span style="color: red;">!</span>
junit.textui	2	1	2	2	0.33	0.50	0.16	<span style="color: red;">!</span>
org.junit	8	10	39	6	0.55	0.13	0.31	<span style="color: red;">!</span>
org.junit.experimental	2	0	1	3	0.00	0.75	0.25	<span style="color: red;">!</span>
org.junit.experimental.categories	83	19	3	11	0.18	0.78	0.02	<span style="color: red;">!</span>
org.junit.experimental.max	7	0	1	7	0.00	0.87	0.12	<span style="color: red;">!</span>

## Metric Results

[ [summary](#) ] [ [packages](#) ] [ [cycles](#) ] [ [explanations](#) ]

The following document contains the results of a JDepend metric analysis. The various metrics are defined at the bottom of this document.

## Summary

[ [summary](#) ] [ [packages](#) ] [ [cycles](#) ] [ [explanations](#) ]

Package	T	C	C	A	C	C	A	I	D	V
junit.extensions	6	6	0	1	2	0.0%		67.0%	33.0%	1
junit.framework	17	13	4	7	9	24.0%		56.0%	20.0%	1
junit.runner	3	1	2	2	6	67.0%		75.0%	42.0%	1
junit.textui	2	2	0	0	6	0.0%		100.0%	0.0%	1
org.junit	18	8	10	11	6	56.0%		35.0%	9.0%	1
org.junit.experimental	2	2	0	0	5	0.0%		100.0%	0.0%	1
org.junit.experimental.categories	11	7	4	0	10	36.0%		100.0%	36.0%	1
org.junit.experimental.max	8	8	0	0	11	0.0%		100.0%	0.0%	1
org.junit.experimental.results	6	6	0	0	7	0.0%		100.0%	0.0%	1
org.junit.experimental.runners	1	1	0	0	5	0.0%		100.0%	0.0%	1
org.junit.experimental.theories	15	8	7	2	9	47.0%		82.0%	28.0%	1
org.junit.experimental.theories.internal	8	8	0	1	6	0.0%		86.0%	14.0%	1
org.junit.experimental.theories.suppliers	2	1	1	0	4	50.0%		100.0%	50.0%	1
org.junit.internal	13	11	2	13	10	15.000001%		43.0%	41.0%	1
org.junit.internal.builders	8	8	0	3	10	0.0%		77.0%	23.0%	1
org.junit.internal.matchers	4	3	1	2	5	25.0%		71.0%	4.0%	1
org.junit.internal.requests	3	3	0	2	6	0.0%		75.0%	25.0%	1

## [Maven] Goal 및 Usage

### □ Goal

Goal	설명
jdepend:generate	JDepend 의존성 분석 보고서를 생성한다.

### □ Usage

```
<project>
...
<reporting>
  <plugins>
    ...
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>jdepend-maven-plugin</artifactId>
      <version>2.0</version>
    </plugin>
    ...
  </plugins>
</reporting>
...
</project>
```

# [Maven] 보고서 화면

---

## Metric Results

[\[ summary \]](#) [\[ packages \]](#) [\[ cycles \]](#) [\[ explanations \]](#)

The following document contains the results of a JDepend metric analysis. The various metrics are defined at the bottom of this document.

## Summary

[\[ summary \]](#) [\[ packages \]](#) [\[ cycles \]](#) [\[ explanations \]](#)

Package	TC	CC	AC	Ca	Ce	A	I	D	V
<a href="#">junit.extensions</a>	6	6	0	1	2	0.0%	67.0%	33.0%	1
<a href="#">junit.framework</a>	17	13	4	7	9	24.0%	56.0%	20.0%	1
<a href="#">junit.runner</a>	3	1	2	2	6	67.0%	75.0%	42.0%	1
<a href="#">junit.textui</a>	2	2	0	0	6	0.0%	100.0%	0.0%	1
<a href="#">org.junit</a>	18	8	10	11	6	56.0%	35.0%	9.0%	1
<a href="#">org.junit.experimental</a>	2	2	0	0	5	0.0%	100.0%	0.0%	1
<a href="#">org.junit.experimental.categories</a>	11	7	4	0	10	36.0%	100.0%	36.0%	1
<a href="#">org.junit.experimental.max</a>	8	8	0	0	11	0.0%	100.0%	0.0%	1
<a href="#">org.junit.experimental.results</a>	6	6	0	0	7	0.0%	100.0%	0.0%	1
<a href="#">org.junit.experimental.runners</a>	1	1	0	0	5	0.0%	100.0%	0.0%	1
<a href="#">org.junit.experimental.theories</a>	15	8	7	2	9	47.0%	82.0%	28.0%	1
<a href="#">org.junit.experimental.theories.internal</a>	8	8	0	1	6	0.0%	86.0%	14.0%	1
<a href="#">org.junit.experimental.theories.suppliers</a>	2	1	1	0	4	50.0%	100.0%	50.0%	1
<a href="#">org.junit.internal</a>	13	11	2	13	10	15.000001%	43.0%	41.0%	1
<a href="#">org.junit.internal.builders</a>	8	8	0	3	10	0.0%	77.0%	23.0%	1
...	...	...	...	...	...	...	...	...	...

## Cycles

[ [summary](#) ] [ [packages](#) ] [ [cycles](#) ] [ [explanations](#) ]

	Package	Package Dependencies
junit.extensions		junit.framework org.junit.runner.manipulation org.junit.runner org.junit.runners.model org.junit.internal.runners.model org.junit.internal org.junit org.junit.internal org.junit.runner.manipulation org.junit.runner org.junit.runners.model org.junit.internal.runners.model org.junit.internal junit.framework org.junit.runner.manipulation org.junit.runner
junit.framework		org.junit.runners.model org.junit.internal.runners.model org.junit.internal org.junit org.junit.internal junit.framework org.junit.runner.manipulation org.junit.runner
junit.runner		org.junit.runners.model org.junit.internal.runners.model org.junit.internal org.junit org.junit.internal

# [Jenkins] 플러그인 설치

## □ 플러그인 관리에서 jdepend 검색

The screenshot shows the Jenkins Plugin Manager interface. At the top, there is a search bar with the text "필터: jdepend". Below the search bar, there are several tabs: "업데이트된 플러그인 목록", "설치 가능" (which is selected), "설치된 플러그인 목록", and "고급". A table lists the results for the search:

설치 ↓	이름	버전
<input type="checkbox"/> <a href="#">JDepend Plugin</a>	The JDepend Plugin is a plugin to generate JDepend reports for builds.	1.2.4

At the bottom of the table, there are two buttons: "재시작 없이 설치하기" and "지금 다운로드하고 재시작 후 설치하기". To the right of these buttons, the text "Update information obtained: 8 mir" is displayed.

# [Jenkins] Job 설정

## □ Goal 설정

### Build

Root POM

pom.xml



Goals and options

jdepend:generate



## □ 보고서 생성 설정

### 빌드 후 조치

#### Report JDepend

Pre-generated JDepend File

target/jdepend-report.xml

*Provide a path to a JDepend file created during the build.  
Use a preceding "/" to specify an absolute path, leave off the "/" to specify  
a path within the workspace.  
Leave blank to have the plugin generate its own file.*

삭제

# [Jenkins] 보고서 생성

---

## Metric Results

[ [summary](#) ] [ [packages](#) ] [ [cycles](#) ] [ [explanations](#) ]

The following document contains the results of a JDepend metric analysis. The various metrics are defined at the bottom of this document.

## Summary

[ [summary](#) ] [ [packages](#) ] [ [cycles](#) ] [ [explanations](#) ]

Package	TC	CC	AC	Ca	Ce	A	I	D	V
junit.extensions	6	6	0	1	2	0.0%	67.0%	33.0%	1
junit.framework	17	13	4	7	9	24.0%	56.0%	20.0%	1
junit.runner	3	1	2	2	6	67.0%	75.0%	42.0%	1
junit.textui	2	2	0	0	6	0.0%	100.0%	0.0%	1
org.junit	18	8	10	11	6	56.0%	35.0%	9.0%	1
org.junit.experimental	2	2	0	0	5	0.0%	100.0%	0.0%	1
org.junit.experimental.categories	11	7	4	0	10	36.0%	100.0%	36.0%	1
org.junit.experimental.max	8	8	0	0	11	0.0%	100.0%	0.0%	1
org.junit.experimental.results	6	6	0	0	7	0.0%	100.0%	0.0%	1
org.junit.experimental.runners	1	1	0	0	5	0.0%	100.0%	0.0%	1
org.junit.experimental.theories	15	8	7	2	9	47.0%	82.0%	28.0%	1
org.junit.experimental.theories.internal	8	8	0	1	6	0.0%	86.0%	14.0%	1
org.junit.experimental.theories.suppliers	2	1	1	0	4	50.0%	100.0%	50.0%	1
org.junit.internal	13	11	2	13	10	15.000001%	43.0%	41.0%	1
org.junit.internal.builders	8	8	0	3	10	0.0%	77.0%	23.0%	1
org.junit.internal.matchers	4	3	1	2	5	25.0%	71.0%	4.0%	1
org.junit.internal.requests	3	3	0	2	6	0.0%	75.0%	25.0%	1

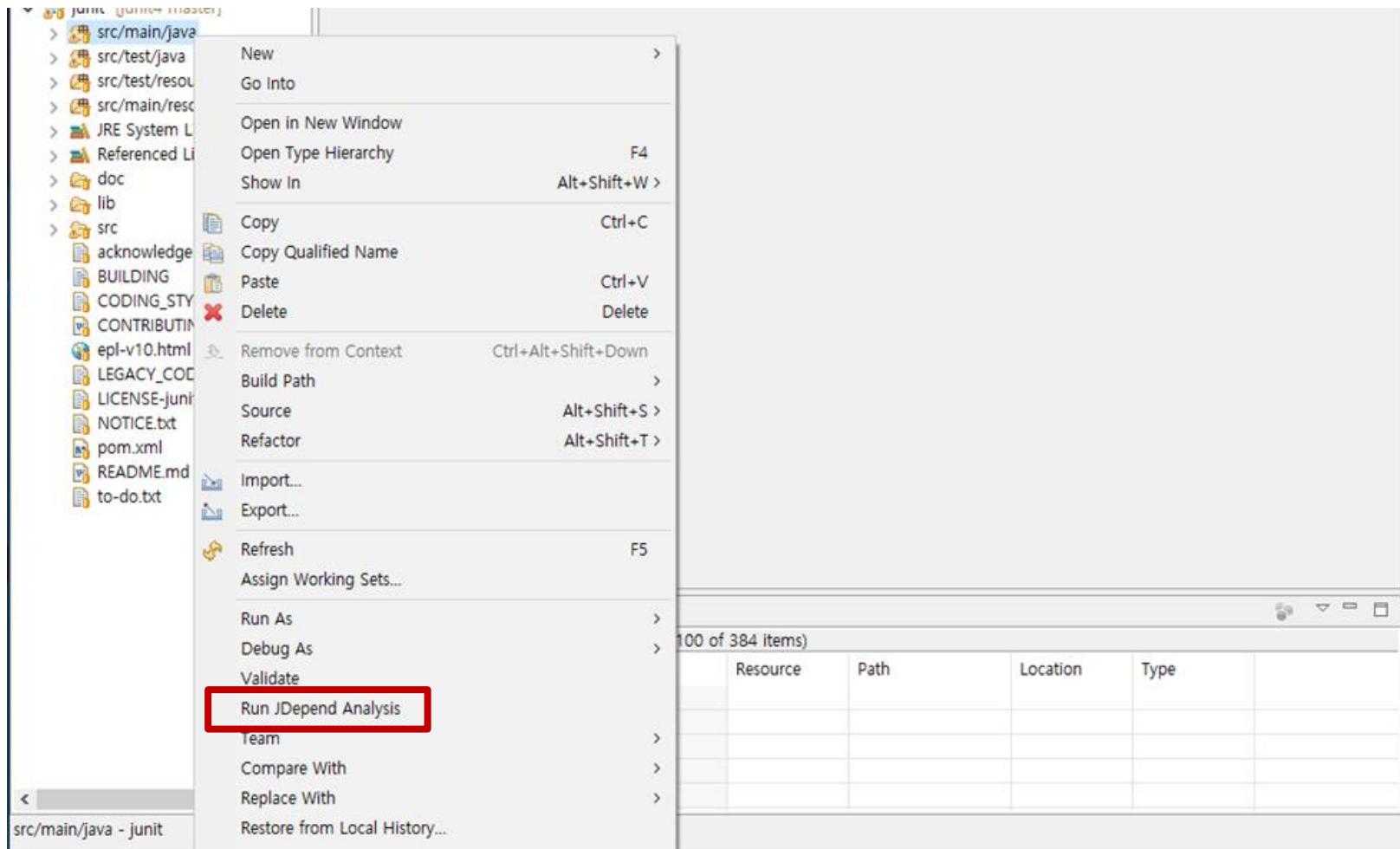
## [Eclipse] 플러그인 설치

---

1. Eclipse 상단의 Help -> Eclipse Marketplace 선택
2. Find에 'JDepend' 입력 후 검색
3. 'JDepend4Eclipse'의 'Install' 클릭
4. 설치 확인 화면에서 'Confirm' 클릭
5. 라이선스에 동의하면 설치 진행
6. 설치 완료 후 Eclipse 재시작

## [Eclipse] 분석 실행

- 프로젝트의 분석 대상 소스코드 패키지를 선택하고 오른쪽 클릭
- 'Run JDepend Analysis'를 선택



# [Eclipse] 분석 결과 확인

JDepend - Eclipse

File Edit Navigate Search Project Run Window Help

Packages X Dependencies X Metrics X

Selected object(s)

Package	CC(co...)	AC(a...)	Ca(aff.)	Ce(eff.)	A	I	D	Cycle
junit.extensions	6	1	2	1	0.14	0.33	0.52	⚠️

Packages with cycle

Package	CC(co...)	AC(a...)	Ca(aff.)	Ce(eff.)	A	I	D	Cycle
junit.extensions	6	1	2	1	0.14	0.33	0.52	⚠️

Depends upon - efferent dependencies

Package	CC(co...)	AC(a...)	Ca(aff.)	Ce(eff.)	A	I	D	Cycle
junit.extensions	6	1	2	1	0.14	0.33	0.52	⚠️
junit.framework	13	5	17	5	0.27	0.22	0.49	⚠️
junit.runner	1	3	2	2	0.75	0.50	0.25	⚠️
junit.samples.money	0	0	1	0	0.00	0.00	1.00	
junit.tests	0	0	1	0	0.00	0.00	1.00	
junit.textui	2	1	2	2	0.33	0.50	0.16	⚠️
org.hamcrest	0	0	27	0	0.00	0.00	1.00	

Used by - afferent dependencies

Package	CC(co...)	AC(a...)	Ca(aff.)	Ce(eff.)	A	I	D	Cycle
junit.extensions	6	1	2	1	0.14	0.33	0.52	⚠️
junit.framework	13	5	17	5	0.27	0.22	0.49	⚠️
junit.runner	1	3	2	2	0.75	0.50	0.25	⚠️
junit.textui	2	1	2	2	0.33	0.50	0.16	⚠️
org.junit	8	10	39	6	0.55	0.13	0.31	⚠️
org.junit.experimental	2	0	1	3	0.00	0.75	0.25	⚠️
org.junit.experimental.categories	83	19	3	11	0.18	0.78	0.02	⚠️
org.junit.experimental.max	7	0	1	7	0.00	0.87	0.12	⚠️

Instability ->

31 elements selected

1 2 3 4 5 6

## 8 JavaNCSS

---

JavaNCSS는 주석을 제외한 소스코드의 라인 수를 계산해 알려준다.

# JavaNCSS

## □ 도구 개요

- NCSS 분석 (NCSS: Non Commenting Source Statements, 주석 제외 코드)
- 패키지, 클래스, 메소드 단위로 분석

## □ 도구 모습

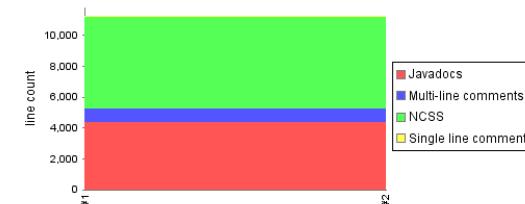
[Maven]

Methods			
[ package ] [ object ] [ method ] [ explanation ]			
TOP 30 Methods containing the most NCSS.			
Methods	NCSS	CCN	Javadocs
junit.runner.BaseTestRunner.getTest(String)	35	18	1
junit.textui.TestRunner.start(String[])	29	13	1
org.junit.runner.JUnitCommandLineParseResult.parseOptions(String[])	26	11	0
org.junit.internal.runners.JUnit38ClassRunner.makeDescription(Test)	24	11	0
org.junit.experimental.theories.Theories.TheoryAnchor.runWithCompleteAssignment(Assignments)	20	1	0
junit.framework.TestSuite.createTest(Class, String)	19	11	1
junit.runner.BaseTestRunner.processArguments(String[])	18	6	1
org.junit.experimental.categories.Categories.CategoryFilter.hasCorrectCategoryAnnotation(Description)	18	13	0
org.junit.internal.runners.MethodRoadie.runWithTimeout(long)	18	1	0
org.junit.runners.Parameterized.TestClassRunnerForParameters.validateFields(Throwable)	18	8	0
junit.framework.TestSuite.addTestsFromTestCase(Class)	17	8	0
org.junit.internal.ComparisonCriteria.arrayEquals(String, Object, Object)	17	11	1
org.junit.internal.runners.MethodRoadie.runTestMethod()	17	8	0
org.junit.internal.runners.JUnit38ClassRunner.filter(Filter)	16	7	0
org.junit.internal.runners.MethodRoadie.Runnable.run()	16	4	0
org.junit.rules.TestWatcher.apply(Statement, Description)	16	1	0
junit.framework.TestCase.runTest()	15	7	1
org.junit.experimental.theories.PotentialAssignment.forName(String, Object)	15	1	0
org.junit.internal.runners.statements.FailOnTimeout.getStuckThread(Thread)	15	10	1
org.junit.internal.runners.statements.FailOnTimeout.getThreadArray(ThreadGroup)	15	5	1
junit.runner.BaseTestRunner.getFilteredTrace(String)	14	7	1

[Jenkins]

Java NCSS Report

Java NCSS Trend



Results

Package	Classes	Functions	Javadocs	NCSS	JLC	SLCLC	MLCLC
junit.extensions	6	24	7	87	29	3	3
junit.framework	17	190	144	664	644	11	3
junit.runner	3	38	14	195	49	7	7
junit.textui	2	35	14	168	66	3	5
org.junit	4	88	73	350	721	0	384
org.junit.experimental	2	8	0	38	0	0	0
org.junit.experimental.categories	8	42	14	250	165	7	50
org.junit.experimental.max	7	32	14	166	74	11	0
org.junit.experimental.results	6	18	10	68	42	0	0
org.junit.experimental.runners	1	2	2	15	19	0	0
org.junit.experimental.theories	10	51	0	281	0	2	160

## [Maven] Goal 설명

---

Goal	설명
javancss:report	라인 수와 순환 복잡도 보고서를 생성한다.
javancss:check	설정한 값보다 라인 수나 순환 복잡도가 높을 경우 빌드 실패로 처리한다.

## [Maven] Usage – javancss:report

---

```
<reporting>
  <plugins>
    ...
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>javancss-maven-plugin</artifactId>
        <version>2.0</version>
      </plugin>
    ...
  </plugins>
</reporting>
```

## [Maven] Usage – javancss:check

### □ <build> 안에 설정

- ccn: 순환복잡도

```
<build>
  <plugins>
    ...
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>javancss-maven-plugin</artifactId>
        <version>2.0</version>
        <configuration>
          <ccnLimit>15</ccnLimit>
          <ncssLimit>80</ncssLimit>
        </configuration>
      </plugin>
    ...
  </plugins>
</build>
```

## [Maven] javancss:check 빌드 실패 예제

---

```
[INFO] -----  
[INFO] BUILD FAILURE  
[INFO] -----  
[INFO] Total time: 1.389s  
[INFO] Finished at: Tue Jul 15 22:37:36 KST 2014  
[INFO] Final Memory: 7M/25M  
[INFO] -----  
[ERROR] Failed to execute goal org.codehaus.mojo:javancss-maven-plugin:2.0:check (default-cli)  
on project junit: Your code has 1 method(s) with a ccn greater than 15 -> [Help 1]
```

# [Jenkins] 플러그인 설치

## □ 플러그인 관리에서 javancss 검색



The screenshot shows the Jenkins plugin management interface. At the top, there is a search bar with the text "필터: javancss". Below the search bar, there is a navigation bar with tabs: "업데이트된 플러그인 목록", "설치 가능" (which is selected), "설치된 플러그인 목록", and "고급". The main area displays a table with the following columns: "설치" (Installation status), "이름" (Name), and "버전" (Version). One plugin is listed:

설치	이름	버전
<input type="checkbox"/>	<a href="#">JavaNCSS Plugin</a> This plugin allows you to use <a href="#">JavaNCSS</a> build reporting tool.	1.1

At the bottom of the table, there are two buttons: "재시작 없이 설치하기" (Install without restart) and "지금 다운로드하고 재시작 후 설치하기" (Download now and install after restart).

## [Jenkins] Job 설정

### □ Goal 설정

#### Build

Root POM

pom.xml



Goals and options

javancss:report



고급...

### □ 보고서 생성 설정

#### Build Settings

E-mail Notification

Publish Java NCSS Report

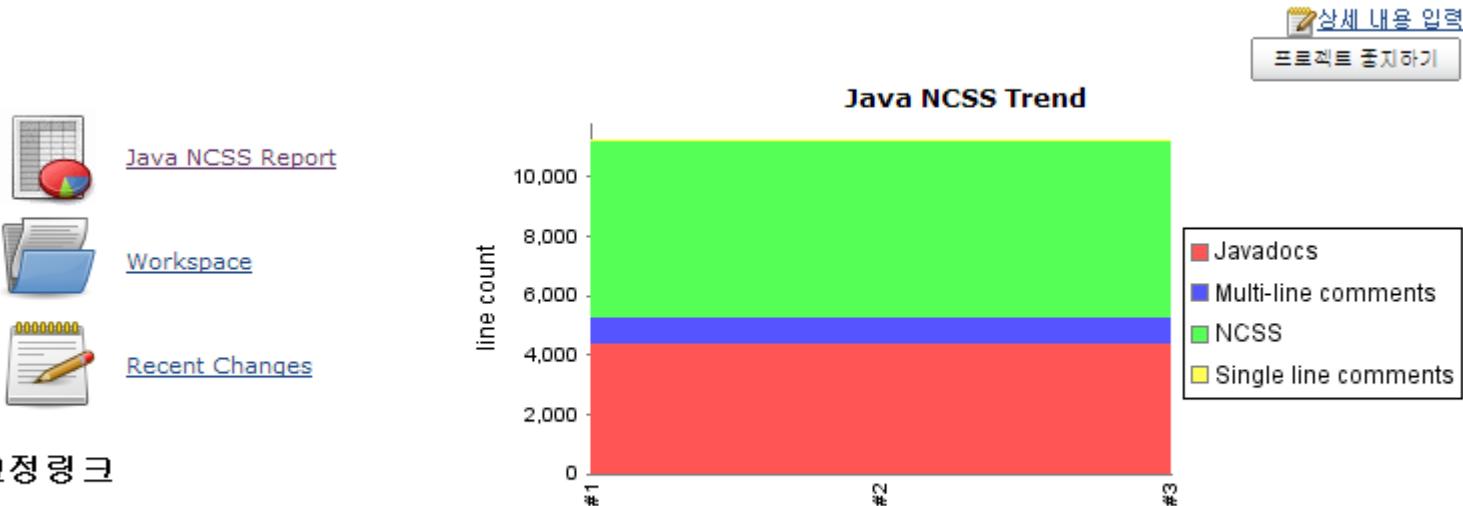
Health Reports

추가

# [Jenkins] 분석 결과

## □ 라인수 추세

### Maven project javancss

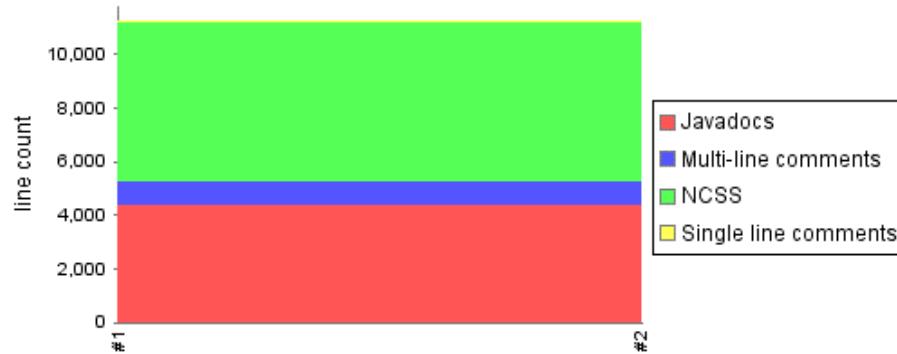


# [Jenkins] 분석 결과

## □ 상세 결과

### Java NCSS Report

#### Java NCSS Trend



### Results

Package ↓	Classes	Functions	Javadocs	NCSS	JLC	SLCLC	MLCLC
junit.extensions	6	24	7	87	29	3	3
junit.framework	17	190	144	664	644	11	3
junit.runner	3	38	14	195	49	7	7
junit.textui	2	35	14	168	66	3	5
org.junit	4	88	73	350	721	0	384
org.junit.experimental	2	8	0	38	0	0	0
org.junit.experimental.categories	8	42	14	250	165	7	50
org.junit.experimental.max	7	32	14	166	74	11	0
org.junit.experimental.results	6	18	10	68	42	0	0
org.junit.experimental.runners	1	2	2	15	19	0	0
org.junit.experimental.theories	10	51	0	281	0	2	160

# 오픈소스 ALM

## 2. 실습 시나리오

---

한동준

[handongjoon@gmail.com](mailto:handongjoon@gmail.com)

# 시나리오

---

## □ 새로운 프로젝트 관리 서비스 개발!

### □ 서버 운영 관점

- SVN 저장소 생성 및 권한 설정
- Redmine 프로젝트 생성 및 권한 설정
- Jenkins Job 생성 및 빌드 설정 (개발 후 설정)

### □ 개발 관점

- Redmine 할일 생성
- Maven 프로젝트 생성
- SVN에 프로젝트 최초 등록
- pom.xml 파일 설정
- Eclipse 플러그인 설정
- Eclipse에서 개발 (코딩, 분석, 테스트...)
- SVN 커밋
- Redmine 변경 추적
- Jenkins 빌드

## 몇 가지 주의사항(책에 실지 않은...)

---

### □ Redmine에서 Git 연결 방법

- Git의 Bare 저장소를 연결해야 함
  - Redmine과 SCManager를 한 서버에서 운영하는 것을 추천
  - Redmine에서 SCManager의 저장소 위치를 지정
- Git의 경로 뒤에 ".git" 추가
  - 예) c:\git\_repo\opensource\.git
- Git 저장소를 Clone한 경우, 사실 잘 연결이 안됨
  - 여러 방법들이 구울링 하면 나오지만, 깔끔하게 실행 안됨

### □ QA가 개발 중간에 PMD, 순환복잡도 준수를 강제로 요구하는건 싸우자는 얘기!

- 테스트 코드가 없고, 시간도 없다면
  - 싸우세요.
  - 배째라고 버티세요.
- 테스트 코드가 있다면
  - 리팩토링 관점에서 고려할 만...
  - 그런데 시간. 인력. 시간. 인력
- 현 수준 유지도 쉽지 않은 요구
  - 보통 이 수준으로 협상
  - 이미 안된 것

**End of Document**

---

**Any Question?**