CS350 Individual Assignment 1

Investigation for CI and some tools

20170660
DaEun Choi

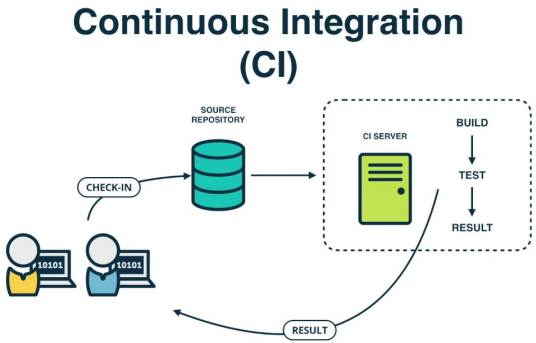
Contents

CI(Continuous Integration)

What is Continuous Integrtion(CI)?

Automation process for merging all developers' working copies to a shared mainline several times a day.

- source management(Git, SVN, ⋯), automated build/test



Why do we need CI?[1]

- Using CI, new application code changes can be **regularly** built and tested and integrated into a shared repository.
- Helping address potential **conflicts** when multiple developers work on the application code at the same time.
- By automated testing, developers can discovers a conflict between new and existing code, easily and more often.

[1] https://www.redhat.com/ko/topics/devops/what-is-ci-cd

Image from "https://geekflare.com/understanding-ci-cd/"

CI(Continuous Integration)

What kind of tests?[1]

- Integration test: combines individual units of the software and tests as a group
- Unit test: tests individual components of the software (e.g. test single function)
- UI test: test if software works well from the user's perspective
- Acceptance test: tests that the software meets up to business requirements

What kind of tools for CI?

Jenkins, Travis CI, Circle CI, Buddy …

- Jenkins is the most widely used CI tool.

[1] https://geekflare.com/understanding-ci-cd/

IntelliJ

What is IntelliJ?

Integrated development environment (IDE) written in Java

- Integrated development environment (IDE): A software application with facilities for software development

Java IDEs

IntelliJ

+ Several Plugins
  High compatibility with Git

– Ultimate version is paid

Eclipse

+ Free to use
  Widely used in Korea
  High compatibility with SVN

– Slow compared to other IDE

NetBeans

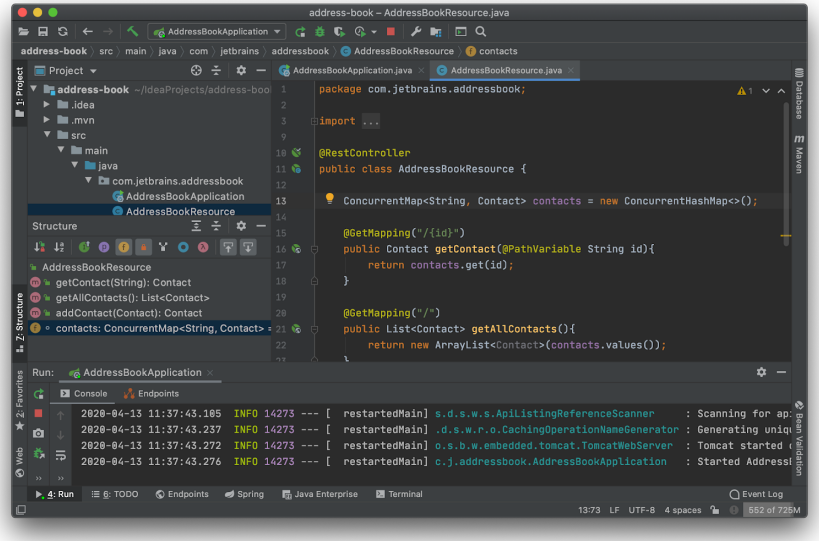
+ Easy installation
  Good for developing platforms
  for desktop environments

– Not widely used

IntelliJ

Main features of IntelliJ[1]

- **Automatic code completion**
  : lists applicable symbols/static methods or constants
- **Data flow analysis**
  : guess the possible runtime symbol type
- **Code inspection & fix bugs**
  : detects mistakes in code and display a little lightbulb pop-up in the editor
- **Version management / Build management**
  : unified interface for major version control systems including Git, SVN, and etc.



[1] https://www.jetbrains.com/idea/features/

JUnit

What is Junit?

Open source framework for Unit testing for the Java programming language[1]
: Test that **certain modules** in the code operate as intended

Features of Junit[2]

- Annotations to identify test methods
- Assertions for testing expected results
- Test runners for running tests
- Run automatically, check results and provide immediate feedback
  (No need for manual operation)
- Simple GUI

[1] https://en.wikipedia.org/wiki/JUnit
[2] https://www.tutorialspoint.com/junit/junit_overview.htm

JUnit

How to use JUnit?[1]

1. Add via Maven / Add the dependency to pom.xml / Add JUnit Library (in Eclipse)
2. Make new JUnit test case: select Class to test (Fig1) → automatically generate test code
3. Customize generated test code
4. Run test (Right-click on Class to test -> RunAs -> JUnit Test)
5. Check test result (Fig2)

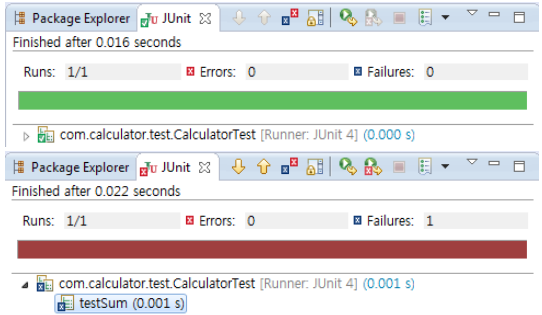


Fig2. Result of JUnit test. (Top: Success, Bottom: Fail)
(Image from [1])

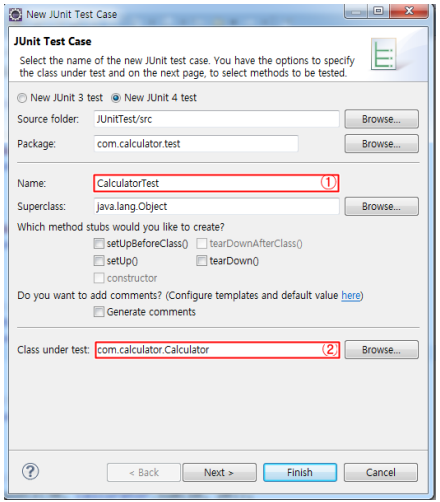


Fig1. Creating new JUnit test case
(Image from [1])

[1] http://www.nextree.co.kr/p11104/

Maven

What is Maven?

Open source project management tool for Java

Build Tool + Project management tool

- Alternative for Apache Ant(Only Build Tool)

Main Features of Maven

- **Library management**
  If you write down the library you want to use for Maven Project in a file called pom.xml, Maven automatically downloads, installs, and routes it automatically.
- **Build**
  Using a Maven project, it automatically builds in the designated directory
- **Manage deployments** with some settings

Maven

Maven Lifecycle

Maven **lifecycle** consists of several **phases.** (Maven works by executing each phase)
Phase is consist of several **goals**, which refers to task.

→ The process of executing the goal linked to each phase is called **build**

Maven Setup[1]

- POM(Project Object Model)
  : Maven records information such as **project structure, environment setting** and **dependencies** required for project management and build through POM file
- settings.xml
  : Settings file related to the Maven build tool

Fig1. Example of pom.xml

[1] https://goddaehee.tistory.com/199

Jenkins

What is Jenkins?

Open source automation server for CI(Continuous Integration)

Developed in JAVA (can be installed any system with JVM)

Jenkins

Main Features of Jenkins

- Automation server
  If developer build a separate CI server, install Jenkins, and register the **Job**, the Job will run automatically using schedular
  → Can be used for regular testing/build/deploy/…
- **Plugins**
  Additional functions can be added by installing various plug-ins
  e.g. Source management(SVN, Ant plugin), Source code analysis(Sonar)

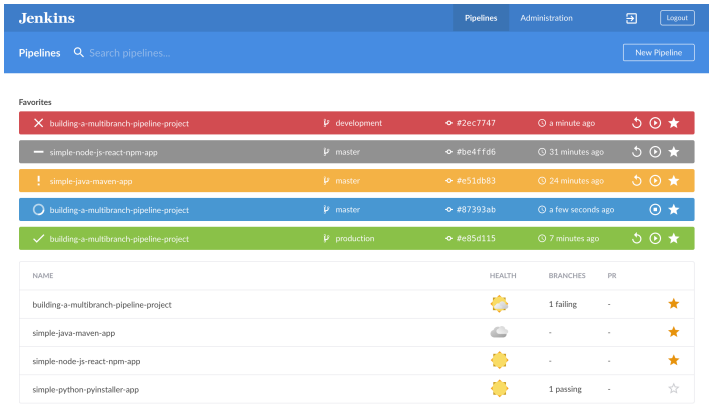
Jenkins

How to use Jenkins?

1. Install CI server with Jenkins
2. Install Plugins
   e.g. SSH, Version control tools(SVN, Git), Ant, Maven, Build Tools, …
3. Set tool configuration
   : set directory of Java, Maven, …
4. Create new Job
   - Name, description, code repository of Job
   - Build setting (schedule, environment setting, configuration setting, …)
5. Run Job

Jenkins Dashboard

Web UI for setting Jenkins



Display list of Jobs, status of each job, …

Git

What is Git?[1]

Distributed **version control system** for tracking changes in source code during software development

- easier management of large projects, avoid overwriting, enable pull the entire code and history
- What is **Version control**? : system that records changes to a files over time
  → able to recall specific versions later
  - Local Version control e.g. GNU RCS
  - Centralized Version Control e.g. CVS, Subversion, and Perforce
    : a single server contains all the versioned files, and clients check out files from central place
  - Distributed Version Control e.g. Git, Mercurial, Bazaar or Darcs
    : each clients mirror the repository and full history → full backup of all the data

[1] https://git-scm.com/book/en/v2/

Git

Advantage of Git[1]

- Feature Branch Workflow
  : Support branch features which are cheap and easy to merge
- Distributed Development
  → fast, easier to scale team, reliable environment (one mistake do not affect whole)
- Pull Requests : ask another developer to merge branches into their repository
  → easy to keep track of changes, enable discussions around their work before integrating

→ Faster Release Cycle

Git Hosting Services

- GitHub, GitLab, Bitbucket, …


GitHub


Bitbucket


GitLab

[1] https://www.atlassian.com/git/tutorials/why-git

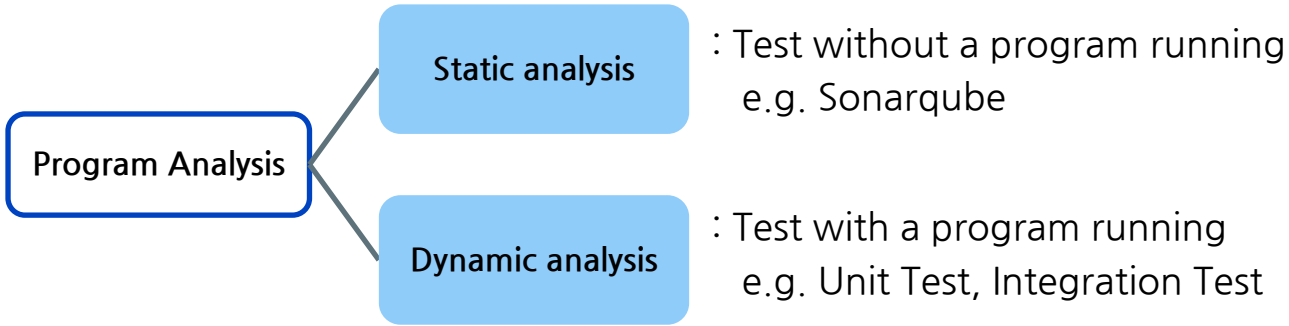
Static analysis

What is static analysis?

Test the software **without running it.**

Help users to write code **obeying certain rules** of programming language

Mainly focus on finding errors (e.g. divide by zero, wrong indexing, …)



[1] https://git-scm.com/book/en/v2/

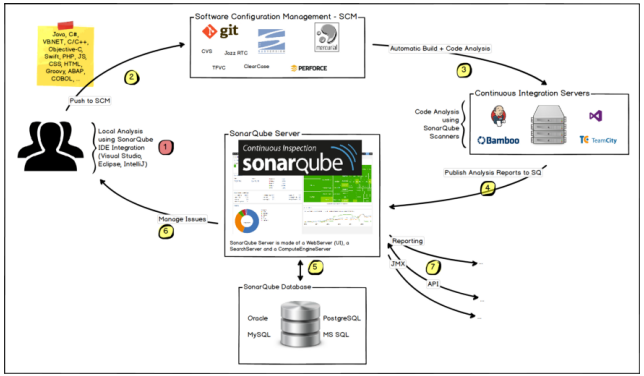
SonarQube

What is SonarQube?[1]

Open source platform for continuous test of code quality

Run static analysis of code to detect bugs, code smells, and security vulnerabilities on 20+ programming languages

How to use SonarQube in CI?[2]



1. Local analysis using SonarLint plugin in IntelliJ
2,3. At CI server, run static analysis using SonarQube scanner while build
4,5. Publish analysis result back to the CI server/SonarQube DB
6. Check result using SonarQube UI at Web server

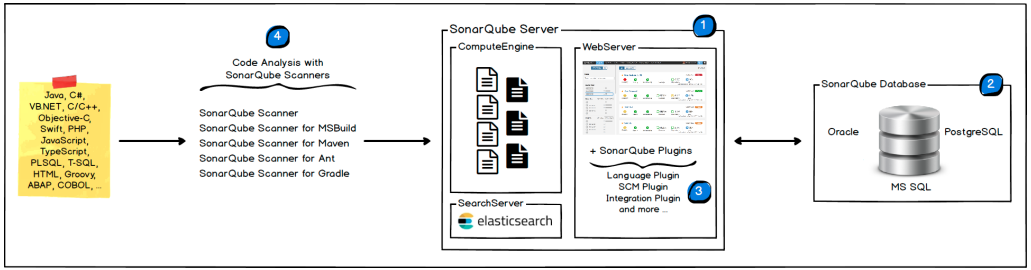
[1] https://en.wikipedia.org/wiki/SonarQube

[2] https://docs.sonarqube.org/latest/architecture/architecture-integration/

SonarQube

Architecture of SonarQube – 4 components of SonarQube[1]

1. SonarQube **server**: Web server(UI for showing result), Search server(Elasticsearch), **Compute engine**(process code analysis and save in DB)
2. SonarQube **DB**: configuration of SonarQube, snapshots
3. SonarQube **plugins** e.g. languages, integration
4. SonarQube **scanner**: run on CI server



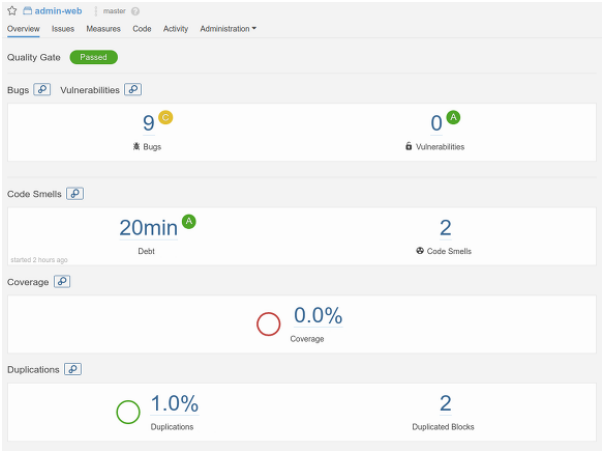


Fig1. SonarQube UI

[1] https://docs.sonarqube.org/latest/architecture/architecture-integration/

PMD

What is PMD?[1]

Open source static source code analyzer that reports on issues found within application code

Features of PMD

- Check Compliance with coding standards
  - Name rule (variable name), Class and method length, …
- Check coding antipatterns
  e.g. Empty try/catch/finally/switch blocks, Too complex expression, …
- CPD (Cut and Paste Detector)
  - Tool for finding questionable code copies
- Customize rules by adjusting property values or parameter values
- Provide plugins for several IDEs

[1] https://en.wikipedia.org/wiki/SonarQube

JaCoCo

What is JaCoCo?[1]

Free code coverage library for Java

- **Code coverage**: measure of how much code was executed when the software was tested.

Features of JaCoCo

- Run test code and generate visible coverage reports
- Ensure that the test results meet the coverage criteria

How to use JaCoCo?

- IntelliJ: select to use JaCoCo as the coverage tool in the run configuration
- Jenkins: Add JaCoCo plugin

[1] https://www.jacoco.org/jacoco/


End of Document