L3. Agile Software Development (II)

Introduction to Gradle

Dr A Boronat

Sprint 1:

- Goal: test next week - Tuesday 17 October 16:00-18:00 (CW 301)
- Schedule: calendar on Blackboard
- Exercises: solutions

Problem 1

Goal

- Software release: software that is developed and tested, i.e. our goal
- Build: software that is compiled and assembled (a jar file), intermediate goal to achieve a release

Problems in release management

- Does the code compile?
- Does the code pass the tests? (unit tests)
- Does the code meet the business requirements? (functionality)
- Does the code meet the quality criteria? (performance, security, etc.)

Solution:

Build automation

HOW? Tooling

Groovy

gradle

compile > test > build > release

spring by Pivotal

JPA Java Persistence API

HIBERNATE

MySQL

GitHub

SPOCK FRAMEWORK

Problem 2

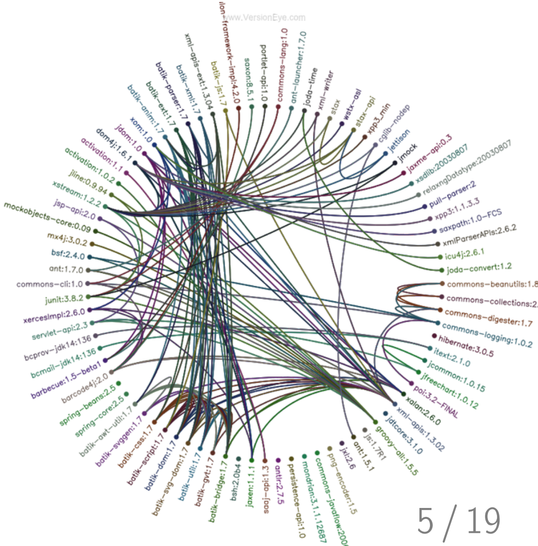
JasperReports

Dependency Hell

- Many dependencies
- Long chains of dependencies
- Conflicting dependencies: when different versions cannot be simultaneously installed
- Circular dependencies

Solution

Dependency management



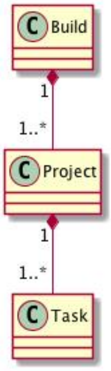
Gradle: a DSL for Build Automation

Features

- Provides a DSL to define a continuous delivery pipeline based on Groovy
- Build automation
    - Declarative builds and build-by-convention
    - Scalable
- Dependency Management
    - Dependencies between projects, to local libraries, to remote repositories

Basic Terminology

- A build consists of one or more projects
- A project is a product to be built or a process to be carried out
  Ex: a library JAR or a web application
  Ex: deploying your application to staging or production environments
- A task is an atomic piece of work which a build performs
  Ex: compiling some classes, creating a JAR, generating Javadoc, or publishing some archives to a repository



Gradle: Basic Tasks

- a task

```
task TaskA
TaskA.description = "task A"
```

- writing actions

```
TaskA.doLast { println "task A" }
TaskA << { println "task A" }
TaskA.doFirst { println "at the start of task A" }
```

- writing tasks as closures

```
task TaskA {
  description "task A"
  doLast { println "taskA" }
}
```

- to execute a task

```
./gradlew TaskA
```

- to show all tasks available

```
./gradlew tasks (--all)
```

Gradle: Tasks Dependencies

- TaskA can execute only if TaskB is executed:

```
task TaskA
task TaskB
TaskA.dependsOn TaskB
```

- equivalently (enforces predecessor):

```
task TaskA {
  dependsOn TaskB
}
task TaskB
```

- similarly (enforces successor):

```
task TaskA
task TaskB
TaskB.finalizedBy TaskA
```

Gradle: Properties

- local variables

```
def version = "1.0"
task TaskA {
  description = "task A - version $version"
}
```

- global variables

```
project.ext.version = "1.0"
task TaskA {
  description = "task A - version $version"
}
```

Build Lifecycle

Initialization

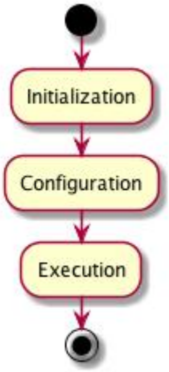
- Determines the projects that will be involved in the build
- Creates a Project instance for each of these projects

Configuration

- Project objects are configured: properties
- Actions are not executed

Execution

- Determines the subset of the tasks to be executed (by the task name arguments passed to the Gradle command and the current directory)
- Gradle then executes the actions in each of the selected tasks.



Gradle: Typed Tasks

- Tasks that are predefined and can be reused: https://docs.gradle.org/current/dsl
- copying files

```
task copyFiles(type: Copy) {
  from 'source'
  into 'target'
}
```

- excluding some files

```
task copyFiles(type: Copy) {
  from 'source'
  into 'target'
  exclude 'pattern'
}

task nestedSpecs(type: Copy) {
    from('src/dist') {
        include '**/*.html'
    }
    into 'build/explodedWar'
    exclude '**/*staging*'
}
```

Plugins

A Gradle plugin is an extension to Gradle which configures your project in some way, typically by adding some pre-configured tasks which together do something useful.

Plugins

- Java plugin
- Eclipse plugin
- Application plugin

Java Plugin

Java plugin

- tasks: compile, unit test, bundle into a JAR file
- source set: group of source files which are compiled and executed together
  E.g. main, test

Example:

```
apply plugin: 'java'
```

How to use it:

```
./gradlew build
./gradle test
```

Eclipse Plugin

Eclipse plugin: to generate files that are used by the Eclipse IDE

Example:

```
apply plugin: 'eclipse'
```

How to use it:

```
./gradlew cleanEclipse
./gradlew eclipse
```

Application Plugin

Application plugin: to create an executable JVM application
E.g. java console applications

Example:

```
apply plugin: 'application'
mainClassName = 'package.ClassName'
```

How to use it:

```
./gradlew run
```

Project dependencies

Dependencies

- other projects
- external libraries
- internal libraries

Repositories

- where libraries are stored

```
repositories {
  mavenCentral()
}
```

- using dependencies from Maven Central

```
dependencies {
  compile 'group:artifactId:version'
}
```

Goals for this week

TODO list (sprint backlog) on Blackboard:

- ☐ Gradle: video and exercises
  1. exercise 1
  2. exercise 2
- ☐ Revise for the test (Tuesday 17 October 16:00-18:00 in CW 301)

Feedback

Exercises

- solutions to be released next Monday
- surgery session on Monday 14:00-15:00 in KE LT2
- laboratory session next Monday to discuss any questions you may have about the exercises
  - you do not need to attend the whole session if you don't have questions

I'm stuck...

- **DO NOT** wait until Monday
- **ASK** in the discussion forum on Blackboard