Adapting Gradle for the CERN Accelerator Control System

Endre Fejes

BE-CO-APS Development Tools team
Vito Baggiolini, Niall Stapley, Natalia Yastrebova, Lajos Cseppentő, Zsolt Kővári

Overview

- CERN Environment
- Migration to Gradle
- How Gradle helped
- How Gradle could have helped more

CERN Beams Controls environment

CERN environment

https://cds.cern.ch/record/2020780

Our users

- ~100 Professional software developers
  - Working on frameworks, libraries, and servers
- ~80 Domain experts: physicists, operators
  - Irreplaceable domain knowledge
    - Know what is needed – want to contribute
  - Limited software experience
  - “Just has to work”

Our systems - scale

- 10+ MLOC of Java
- 1000+ projects
- 20+ level deep dependency hierarchy

Image by Olivier Da Silva Alves

Our systems - limitations

- We’re not Facebook / Google
- All component has to work: no degraded operation
- Limited testing – there is no test LHC, no canaries
- Stability, reliability over cutting edge
  - not ignoring new technologies though

Enable physicists to do physics

Development tools team

- Provide tooling for Java community
- Make development simple and convenient
- Make everything traceable

Migration overview

Old build-system

- ~15 years old, ant-based build system
- Did not scale – designed for couple dozen projects
- Dependency resolution by Apache JJar
- Patched over and over again
- Lack of support: plugins, community

Migration requirements

- Smooth transition - seamless migration
- Without restructuring / refactoring all projects
- Intuitive – with minimal training
- With minimum support overhead

Changing the build system, not the users

Build tool timeline

1998 – Work on new Java Control System started

2002 – Old, ant-based build tool introduced

2011 – Maven pilot

2013 – Gradle adaptation pilot

2015 – Few projects migrated as test

2017 Q1 – Core projects migrated (~50%)

2017 Q2 – 95% of projects migrated

How Gradle helped the migration

Flexible versioning

- Dynamic, programmatic version

```
def suffix = new Date().format("...", tz)
project.version += suffix

> test-1.0.0-20170527-093000.jar
```

- Can be on-demand

```
if (project.hasProperty('timestamp')) { ... }
```

Task generation

Example: creating multiple jar variants per project

```
app
├── build.gradle
├── config-2tier/config.properties
├── config-3tier/config.properties
└── src/main/java/com/example/app/Hello.java
```

Tasks example – expectation

Jars to generate:

```
app/build/libs
├── app-2tier-1.0.0.jar
└── app-3tier-1.0.0.jar
```

META-INF/MANIFEST.MF

```
Manifest-Version: 1.0
Variant: 2tier
```

config.properties

```
variant=2tier
```

config.properties

```
variant=3tier
```

META-INF/MANIFEST.MF

```
Manifest-Version: 1.0
Variant: 3tier
```

Example: generated jar tasks

```
apply plugin: 'java'

['2tier', '3tier'].each { variant ->
  def jarTask = task("jar_${variant}", type: Jar) {
    from fileTree("config-${variant}")
    baseName = "${project.name}-${variant}"
    manifest.attributes(Variant: variant)
  }
}
```

Task generation in general

- Same task type for a dynamic collection of items

  (Create multiple jars)
- Up-to-date status per item

  (Only generate outdated / missing jars)
- Task type that can not operate on 1 item per task

  (Jar task type: one jar per task)

POM customisation

```
task upload(type: Upload) {
  repositories.mavenDeployer {
    pom.project {
      groupId 'com.example'
      artifactId 'app'
      version '1.0.0'
      scm {}
      properties {}
      dependencies {}
      // ...
```

```
<?xml ...>
<project ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>app</artifactId>
  <version>1.0.0</version>
  <scm><url>...</url></scm>
  <properties>...</properties>
  <dependencies>...</dependencies>
```

POM – extra properties

For audit and support, i.e. traceability

```
def propertyMap = [
  'buildBy': System.getProperty('user.name'),
  'buildJdk': System.getProperty('java.version')
  'buildTime': new Date().format("yyyy-MM-dd'T'HH:mm:ssZ")
  'buildAgent': "gradle-${gradle.gradleVersion}".toString(),
  'buildHost': InetAddress.localHost.hostName,
  'buildAddress': InetAddress.localHost.hostAddress,
]
pom.project { properties propertyMap }
```

POM – extra dependencies

Custom dependencies, versions

```
pom.project {
    dependencies {
        dependency {
            groupId "com.example"
            artifactId "corporate-log4-config"
            version "PRO"
        }
    }
}
```

Any custom version

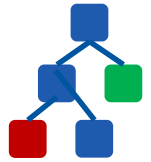
POM customisation – profiles

```
pom.project {
  profiles {
    profile {
      id 'dependency.set.flat'
      activation {
        property { name 'dependencies.set.all' }
        activeByDefault = 'true'
      }
      dependencies { ... }
```

POM customisation – summary

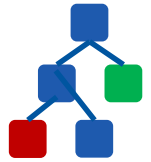
- Extra properties – for audit and support
- Custom dependencies, versions
- Dependency profiles

Dependency resolution



- Improved relocation – fix version-only relocation
  - Needed for old third-parties still in use
    - eg. `xml-apis:xml-apis:2.0.2` → `1.0.b2`
- Dependency deprecation
  - Phasing out old libraries with replacements
- Version aliases ~ Maven labels
  - Latest and “PRO” aliases – legacy requirement

Dep. resolution augmentation



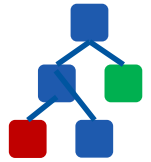
```
configurations.all { config ->
    if (config.state != Configuration.State.UNRESOLVED) {
        return
    }
    config.resolutionStrategy.eachDependency { rule ->
        def original = rule.requested
        def current = rule.target
        rule.useTarget("...")
    }
}
```

Always the original

The original or as mapped by last rule

Any custom logic

Dep. resolution augmentation – example

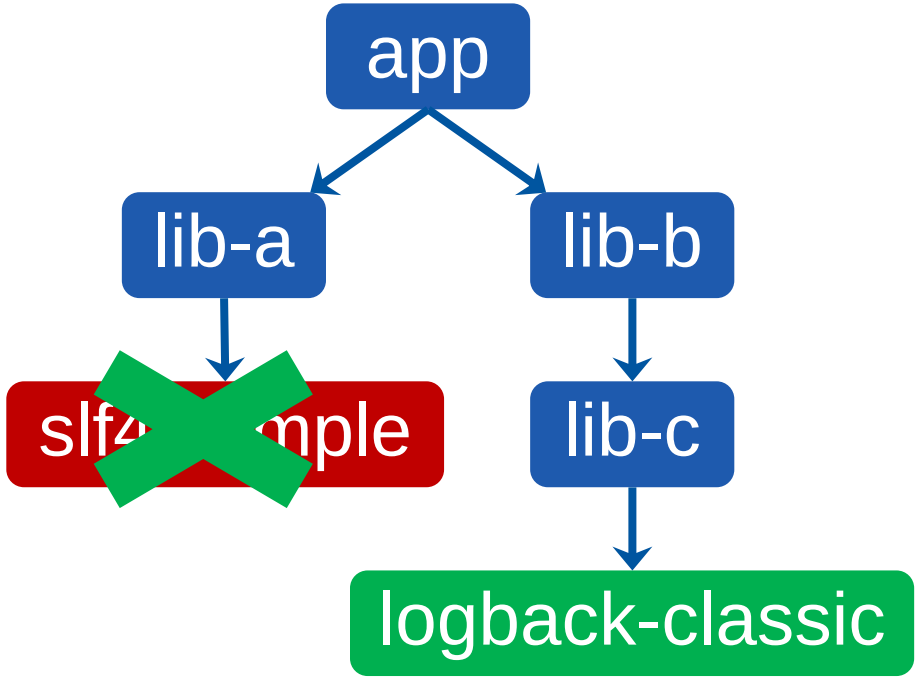


```
configurations.all { config ->
    if (config.state != Configuration.State.UNRESOLVED) {
        return
    }
    config.resolutionStrategy.eachDependency { rule ->
        if (rule.target.version == 'PRO') {
            rule.useVersion(lookupPRO(rule.target))
        }
    }
}
```

Dependency configurations

Example: conflicting transitive dependencies

Goal: remove slf4j-simple
if logback-classic is
present



Dependency configurations

```
$ gradle dependencies --configuration=compile
compile - Dependencies for source set 'main'.
+-- project :lib-a
|    \-- org.slf4j:slf4j-simple:+ -> 1.7.25
|         \-- org.slf4j:slf4j-api:1.7.25
\-- project :lib-b
     \-- project :lib-c
          \-- ch.qos.logback:logback-classic:+ -> 1.2.3
               +-- ch.qos.logback:logback-core:1.2.3
               \-- org.slf4j:slf4j-api:1.7.25
```

Dependency configurations

```
def cloneConfig = configurations.compile.copyRecursive()
def ids = cloneConfig.resolvedConfiguration
        .resolvedArtifacts.collect { it.moduleVersion.id }

def logback = ids.find { it.name == "logback-classic" }
def slf4js = ids.find { it.name == "slf4j-simple" }

if (logback && slf4js) {
  configurations.compile.exclude(
     group: slf4js.group, module: slf4js.name
  )
}
```

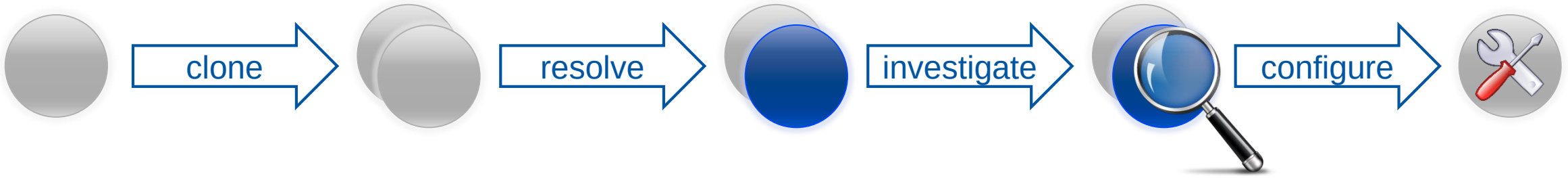
Triggers resolution!

Dependency configurations

```
$ gradle dependencies --configuration=compile
compile - Dependencies for source set 'main'.
+-- project :lib-a
\-- project :lib-b
    \-- project :lib-c
        \-- ch.qos.logback:logback-classic:+ -> 1.2.3
            +-- ch.qos.logback:logback-core:1.2.3
            \-- org.slf4j:slf4j-api:1.7.25
```

Dependency configurations - summary

- Dependency configuration can be cloned
- Resolving clone does not affect the original
  - To investigate dependencies – transitive as well
  - Additional rules can be applied
- Imitate multiple resolution per configuration



build.gradle problem

- Flexible and powerful, but …
- Build logic duplicated for 95% of the projects
- Duplication diverges
- Needs training and refactoring
- Limited tooling
- Vendor lock-in

build.gradle alternative

- Custom build descriptor
- XML based
- Declarative
- Easy maintenance
- Legacy

```
<product name="test" version="1.0" group="com.example">
  <dependencies>
    <dep product="mylib" version="1.0" />
  </dependencies>
</product>
```

Init scripts

- Gradle files in `$GRADLE_HOME/init.d`
- Evaluated for each Gradle execution
- Parsing of custom build descriptor
  - Generic build logic in init scripts
  - Project configuration in XML files
- `build.gradle` can still be used for customisation

- **Most important enabler of seamless migration**

Init scripts – example

$GRADLE_HOME/init.d/corporate.gradle

```
initscript {
    repositories { ... }
    dependencies { ... }
}
allprojects {
  apply plugin: 'java'
  apply from: "$gradle.gradleHomeDir/corp-log.gradle"
}
```

like `buildscript` block in other gradle files

Init scripts – summary

- Avoid boilerplate `build.gradle` - only for the 5%
  - Default plugins applied
  - Default configurations applied
- Hidden in the distribution
- Have to be bundled

or specified on the command line with `-I`

Enabled seamless migration

General flexibility

- Dynamic task generation
- Customisable POM generation
- Customisable dependency resolution
- Init scripts

Work-around for any migration issue

How Gradle could have helped more

Gradle / Groovy

- Groovy code base can grow out of control
  - Difficult to maintain
- We converted to Java
- Gradle Java API improved a lot
  - We started with Gradle 1.x … now on 3.5
- Gradle Kotlin DSL – looking forward for maturity

Version “+”

Can be used in Gradle instead of version ranges

```
dependencies {
    compile "log4j:log4j-api:1.+"
}
```

Gets into the POM as is: breaks Maven compatibility

Logging

- Difficult to customise
- Only internal API to configure or hook into
- No nice way for logging into file (as well)
  - Especially with different log levels

Logging alternative

- Custom logging plugin
- Uses internal API ☹
- Logging into file as well
  - With at least info log level
  - Ease support
  - Avoid re-runs

Logging alternative – log summary

- Cleaner error feedback
- Error and warning messages summarised

```
------------------------------------------------------------
Summary of warnings and errors in lsa-ext-ad
------------------------------------------------------------
[WARN] - Test execution is disabled for this project. To turn it back,
remove the junit.enabled=false property

BUILD SUCCESSFUL
```

Logging alternative – “What’s next” hint

What to do next in case of an error

```
------------------------------------------------------------
Summary of warnings and errors in accsoft-timing-app-log-analyzer
------------------------------------------------------------
:test
- Try turning off the parallel execution with the maxParallelForks=1 property
in case the tests are not completely isolated


BUILD FAILED
```

Logging alternative – “What’s next” hint

… and in case of success

```
------------------------------------------------------------
Hints for lsa-ext-ad
------------------------------------------------------------
:release
  - Artifacts: http://repo/[...]/cern/lsa/lsa-ext-ad/1.2.15
  - PRO of lsa-ext-ad was updated to 1.2.15

BUILD SUCCESSFUL
```

Logging alternative – conclusion

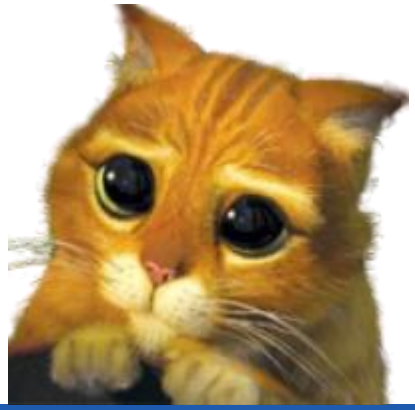
- Filtering out irrelevant noise
- Collecting relevant information

- Open-sourcing is planned …

Building Eclipse plugins

- Lack of support for building Eclipse plug-ins
  - Wuff! Project – abandoned 2 years ago
  - bnd-platform – aimed at OSGi: lots of configuration

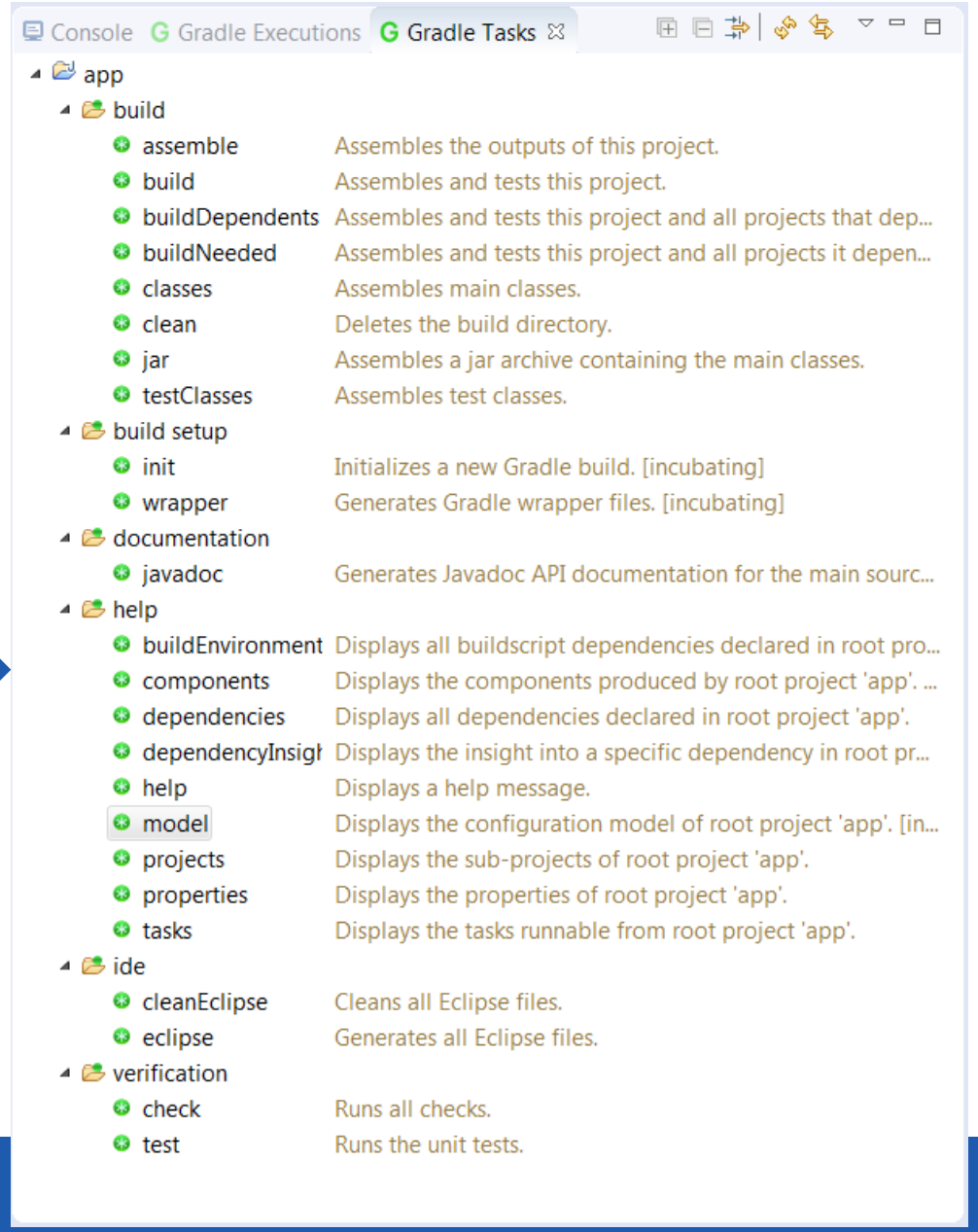
- Maybe the solution in Buildship's buildSrc will be published?



Buildship

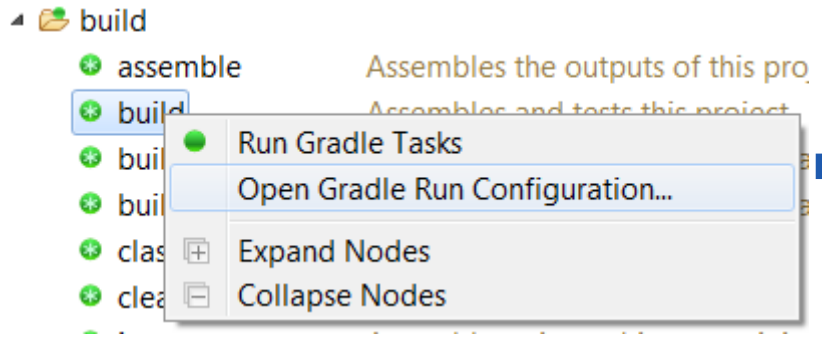
- Lacks simplicity
  - Dozens of unused tasks

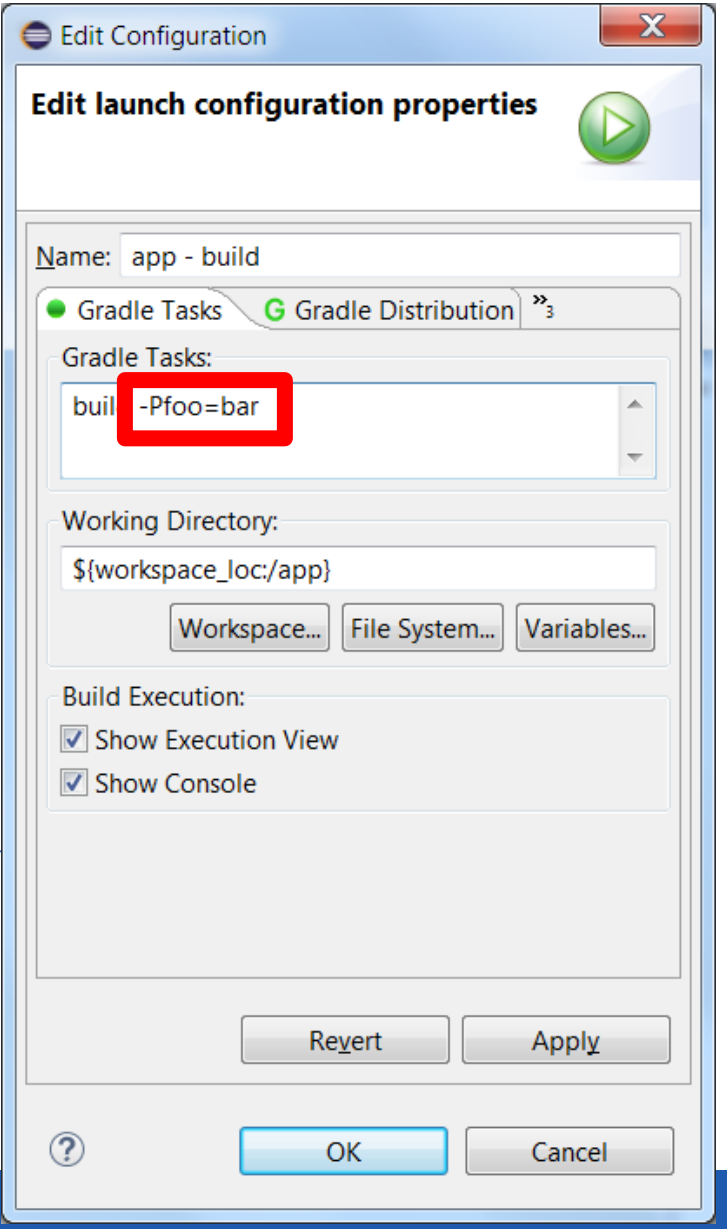
```
1 apply plugin: 'java'
2 apply plugin: 'maven'
3 apply plugin: 'eclipse'
```



Buildship

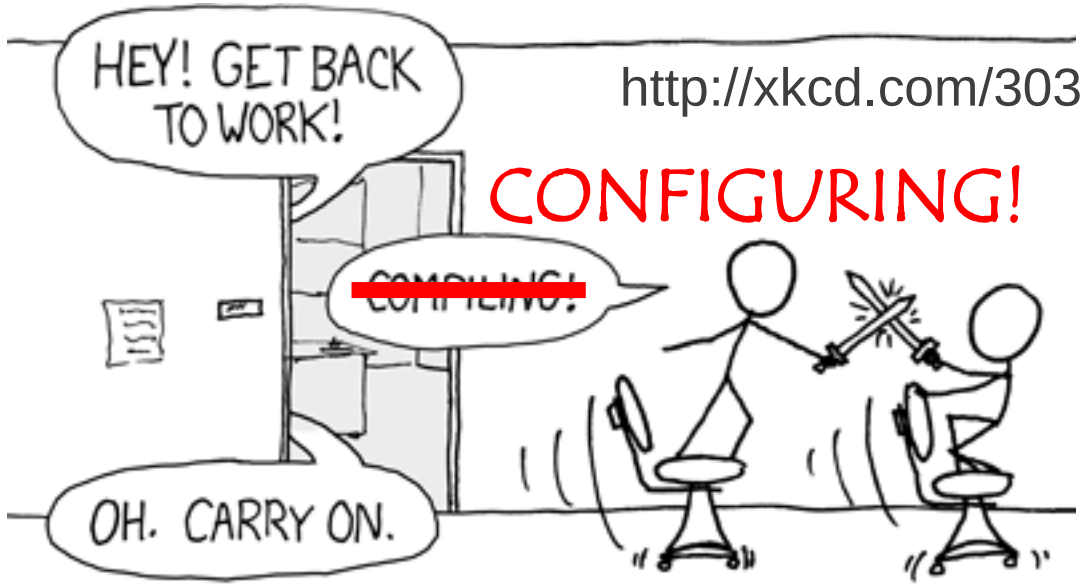
- Lacks simplicity
  - Dozens of unused tasks
  - Arguments in run configurations





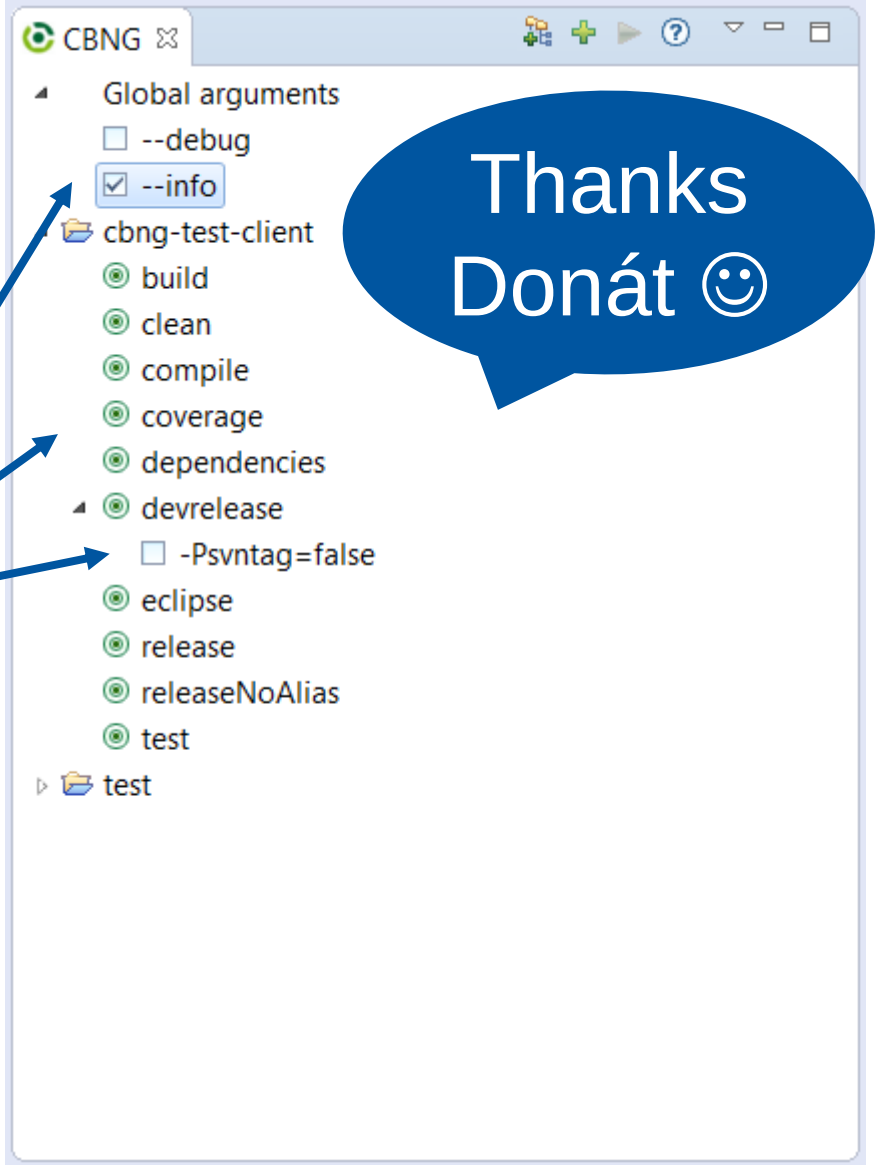
Buildship

- Lacks simplicity
  - Dozens of unused tasks
  - Arguments in run configurations
- Scalability issues
  - Having to re-configure after workspace cleaning



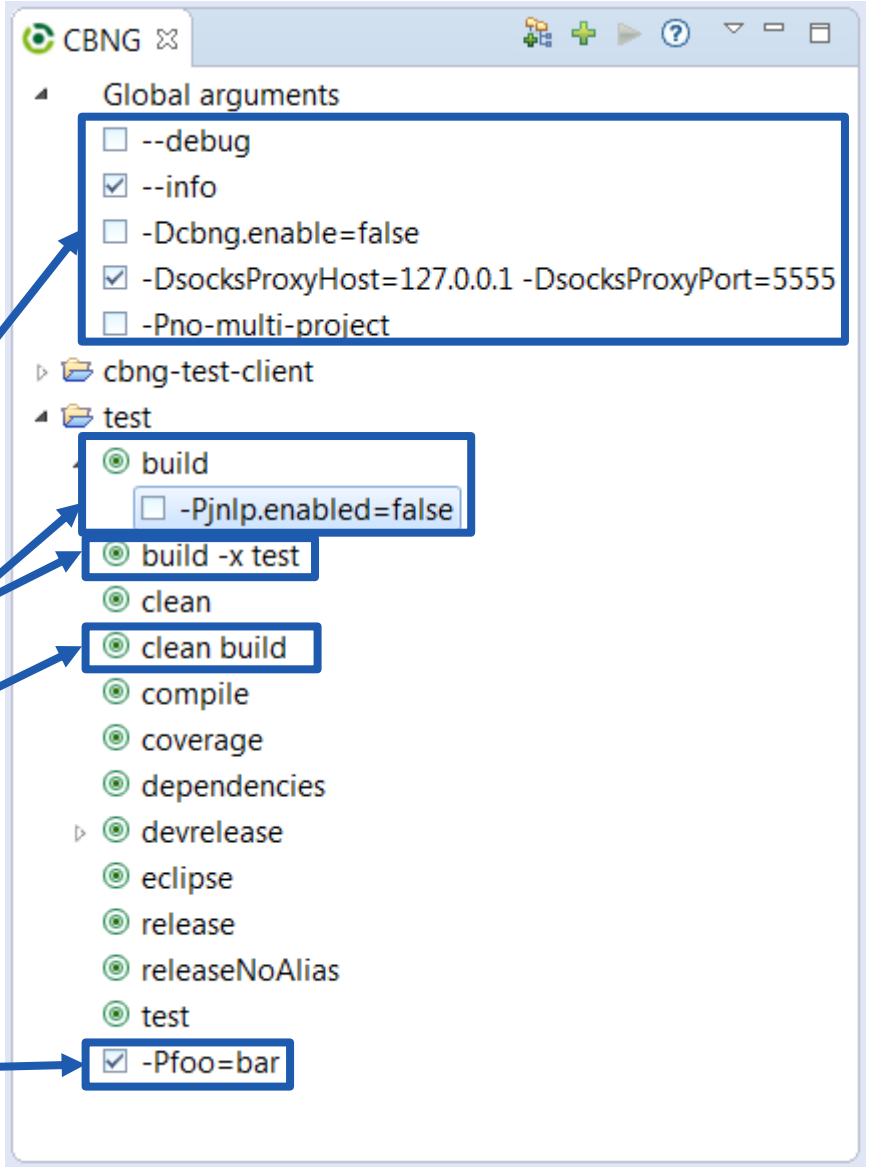
Buildship alternative

- Replacement using Gradle Tooling
  - Inspired by Ant plugin
- Simplicity – static defaults
  - Common tasks and arguments



Buildship alternative

- Replacement using Gradle Tooling
  - Inspired by Ant plugin
- Simplicity – static defaults
  - Common tasks and arguments
- Customisable
  - Parameterised tasks
  - Task combinations
  - Custom arguments



Smooth migration to Gradle

- Gradle features that helped:
  - Dynamic task generation
  - POM customisation
  - Special dependency handling
  - Init scripts

Gradle can replace any build tool?

Yes, it can