

BUILDING A CONTINUOUS DELIVERY PIPELINE WITH GRADLE AND JENKINS



Jenkins

Gary Hale



Java/Groovy developer
Gradleware employee
Compulsive Automator

Gary Hale



Presentation will be available on
<https://speakerdeck.com/ghale>



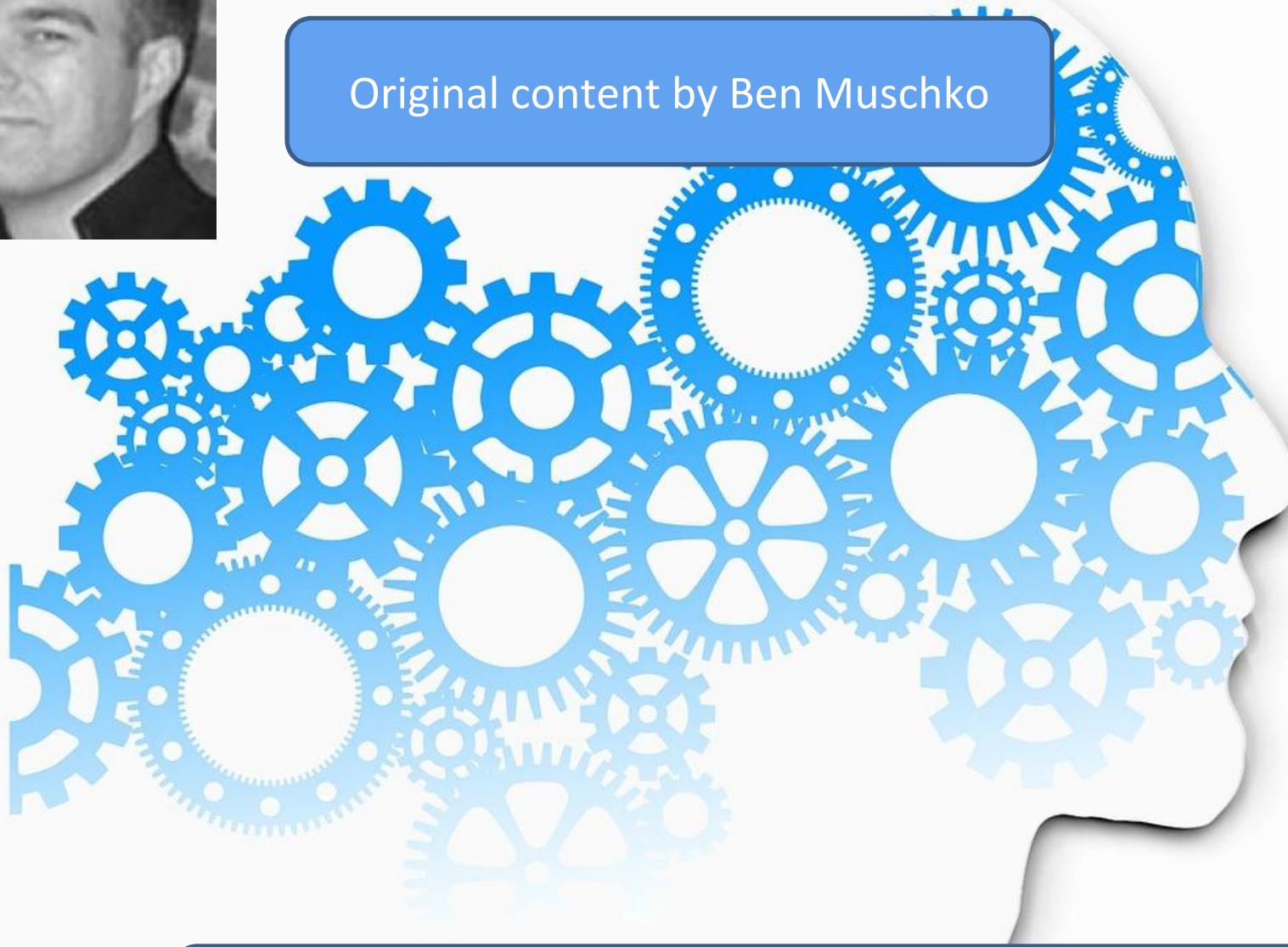
gary.hale@gradleware.com



ghale



Original content by Ben Muschko

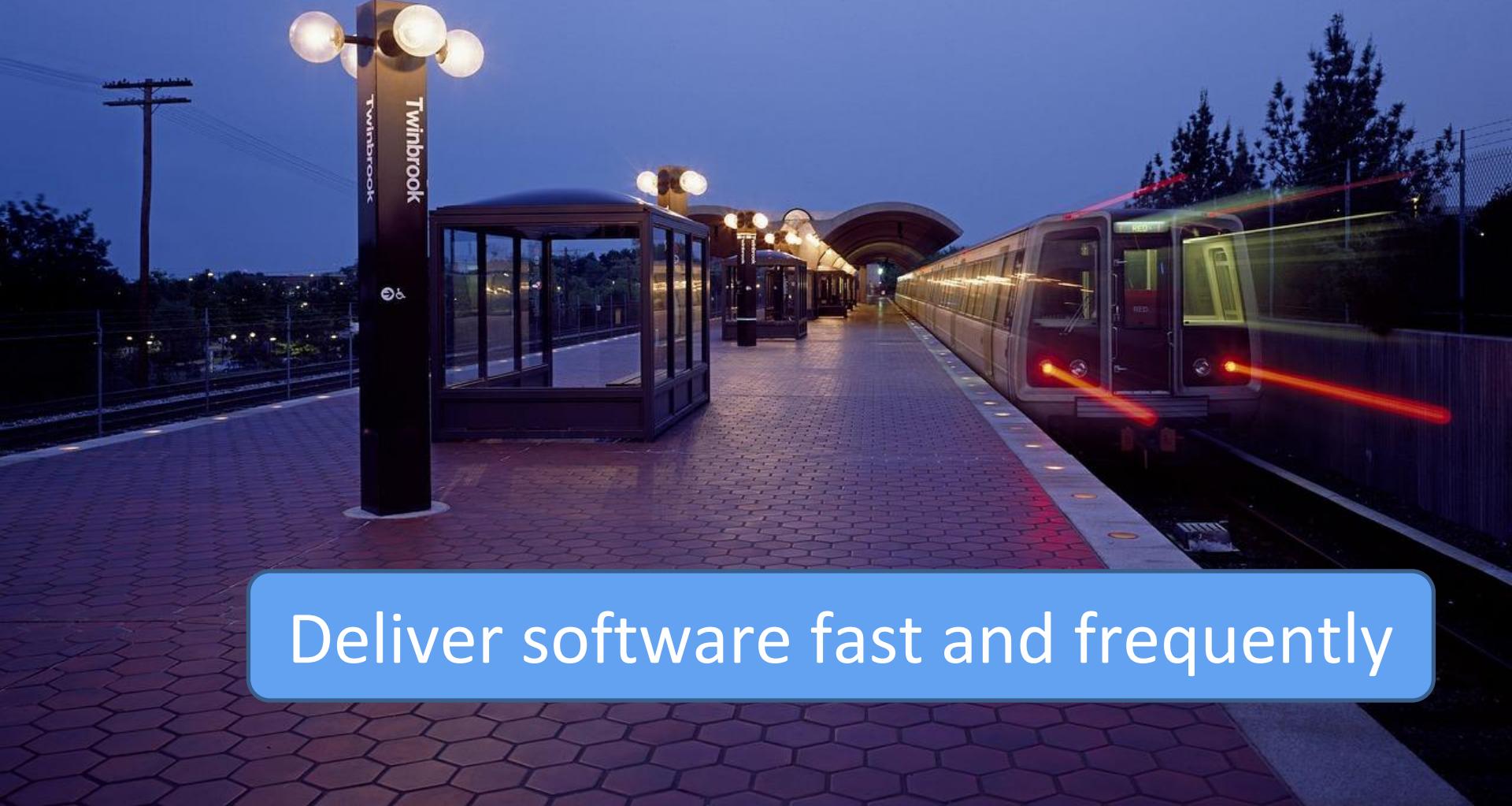


MODIFIED AND USED WITH PERMISSION



Releases don't have to be painful

Continuous Delivery



Deliver software fast and frequently

#1 Every commit can result in a release

#2 Automate everything!

#3 Automated tests are essential

#4 Done means released

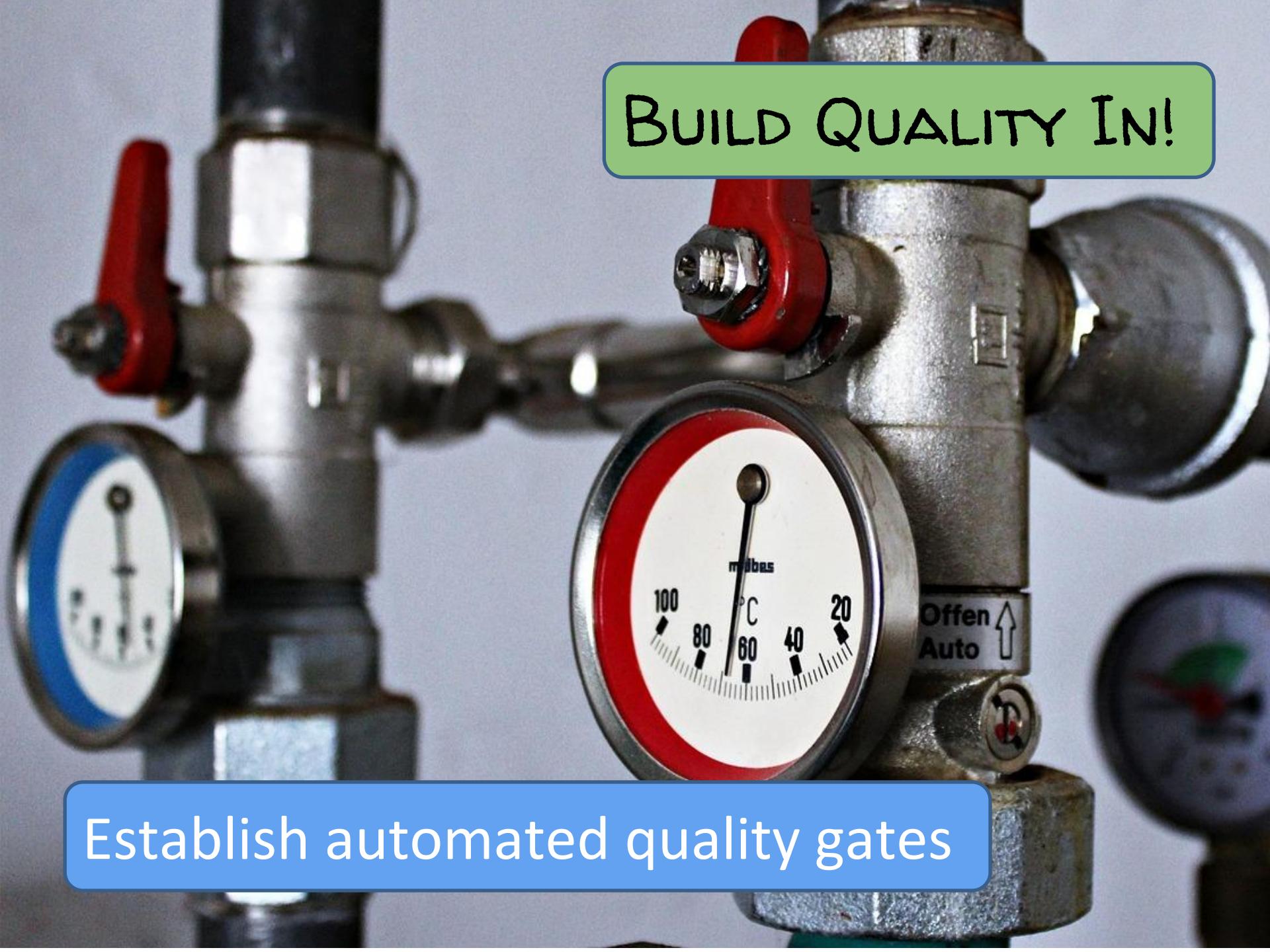
PRINCIPLES



THE BUILD AS A PIPELINE

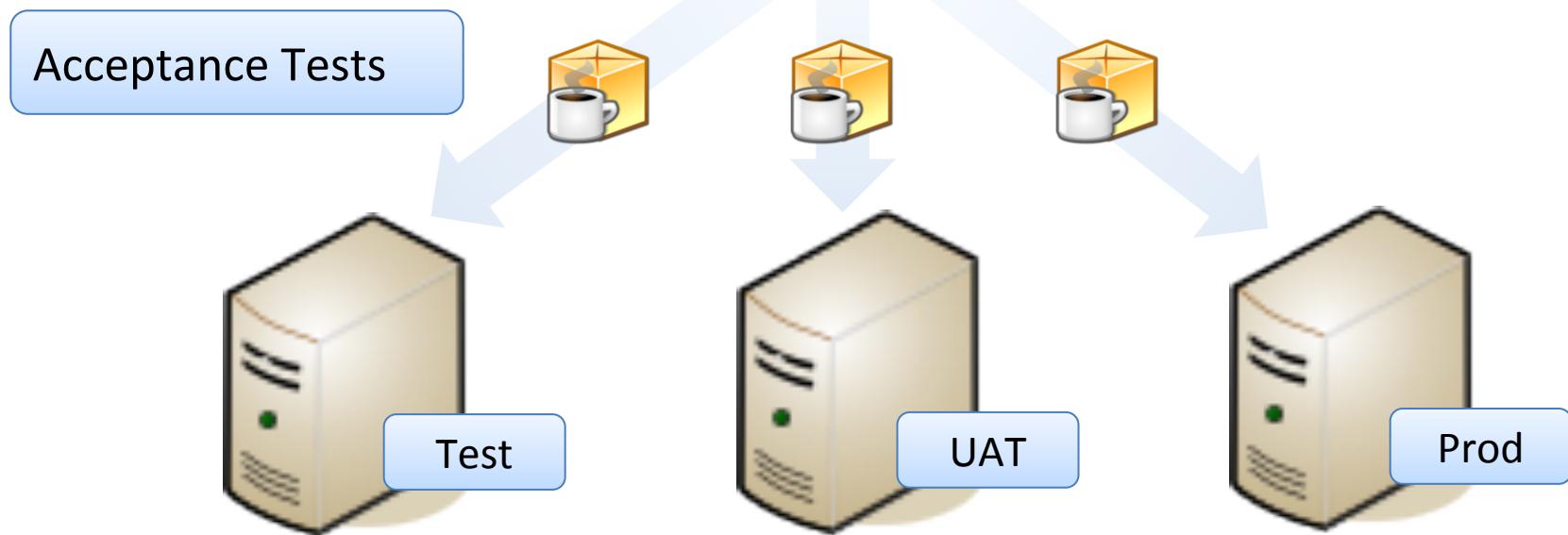
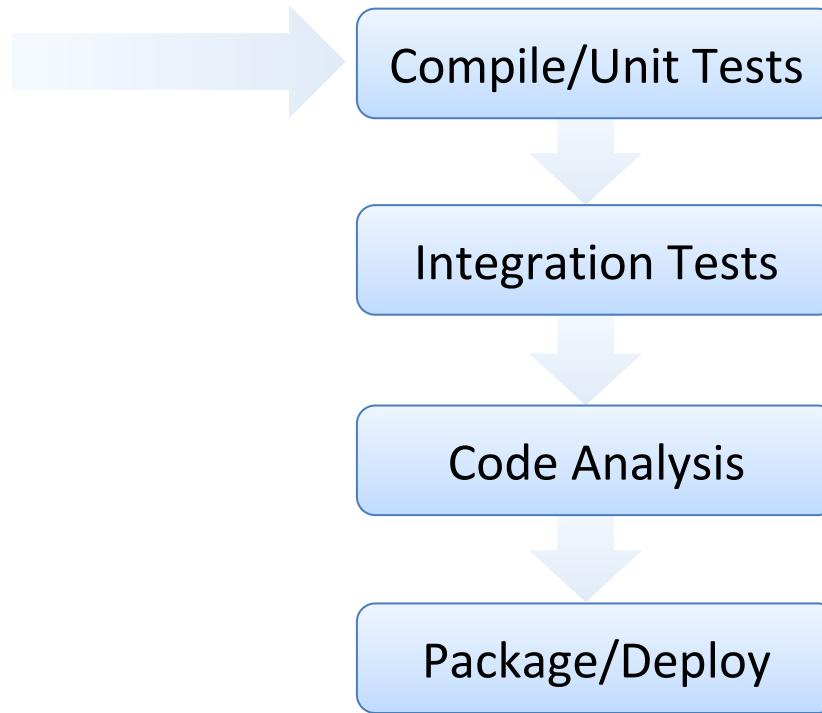


Automated manifestation of delivery process

A close-up photograph of industrial piping and valves. In the center, there is a circular temperature gauge with a red border. The scale ranges from 20 to 100 degrees Celsius, with major markings every 20 units and minor markings every 10 units. The word "mbar" is visible above the 100 mark. To the right of the gauge, a valve handle has the German words "Offen Auto" with an upward arrow next to it. On the far left, another valve handle is partially visible. The background is a plain, light-colored wall.

BUILD QUALITY IN!

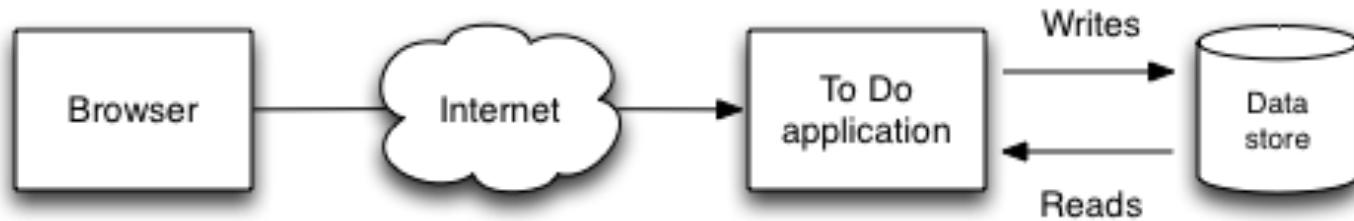
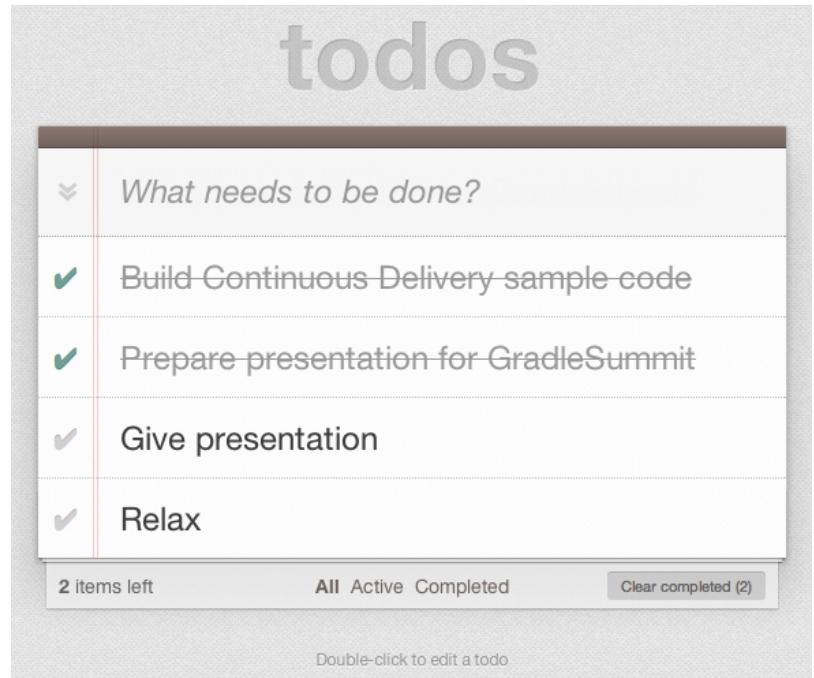
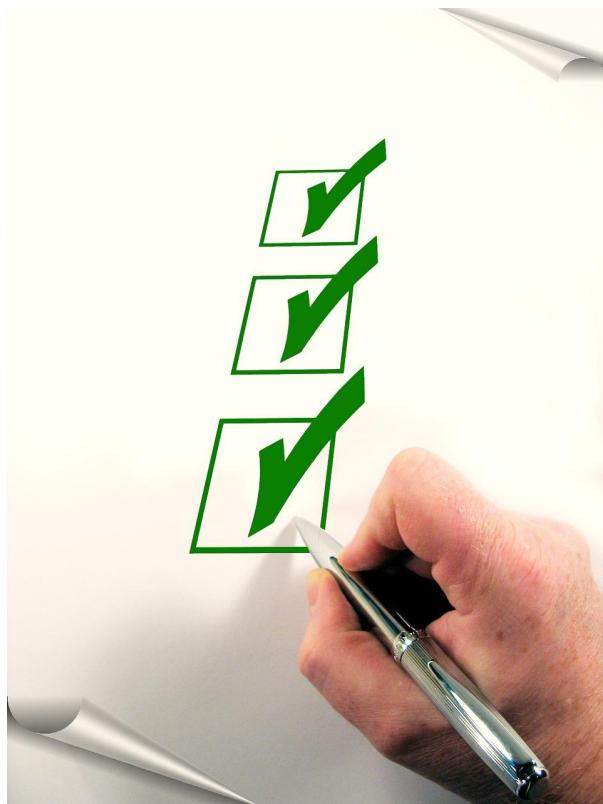
Establish automated quality gates



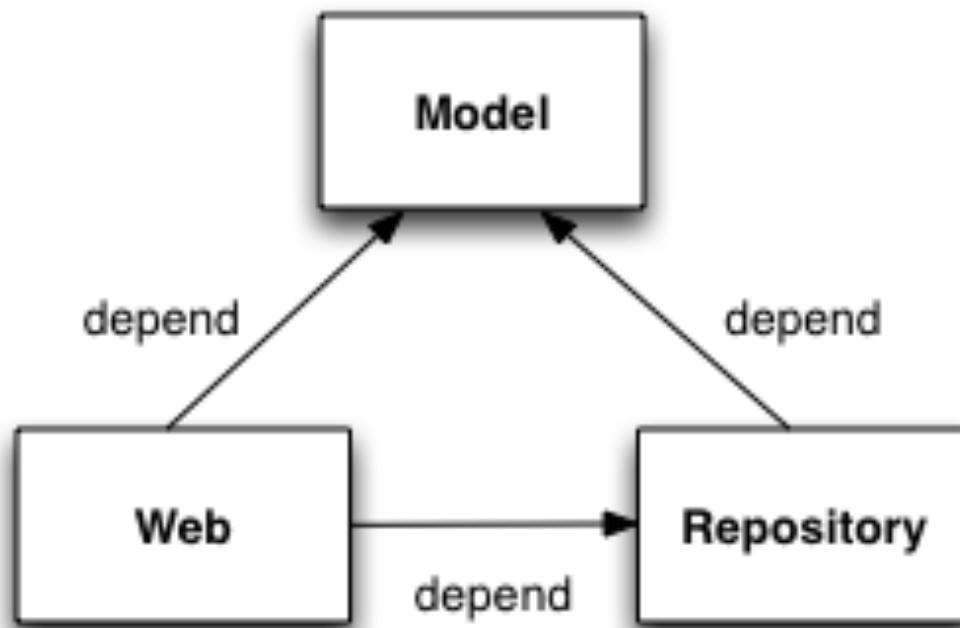


OK, SO WHAT NOW?

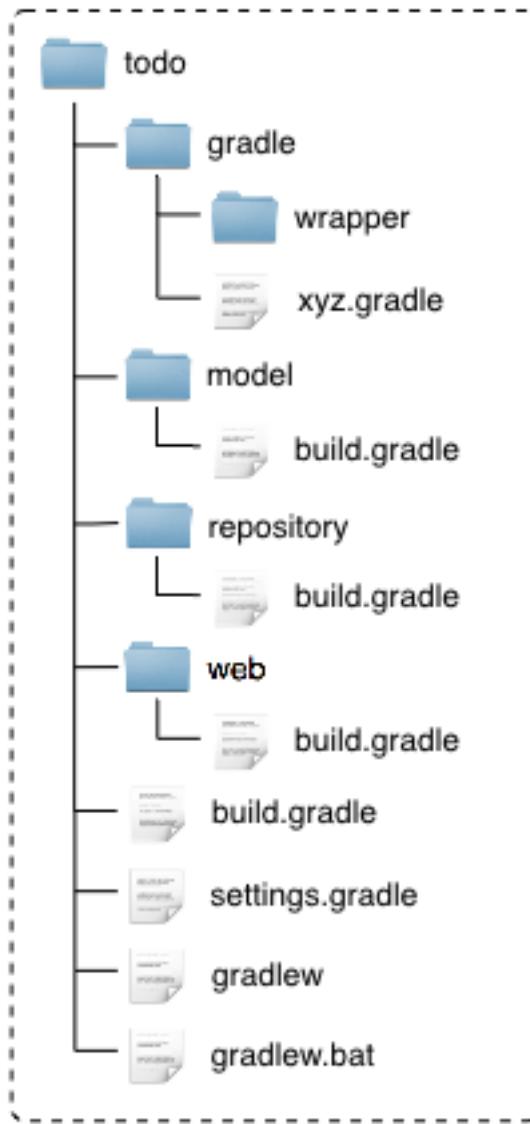
The “revolutionary” sample application



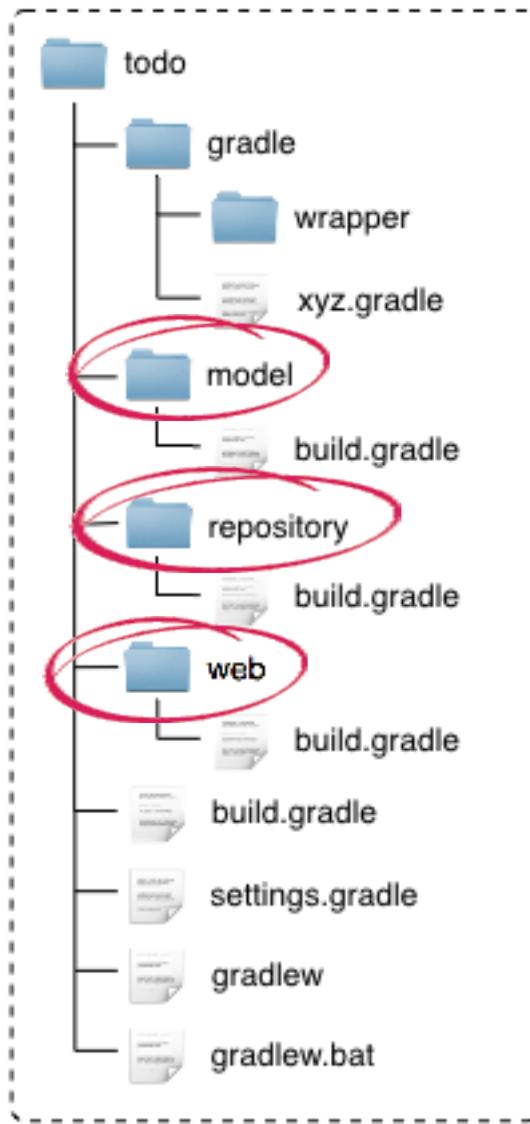
Multi-project dependencies



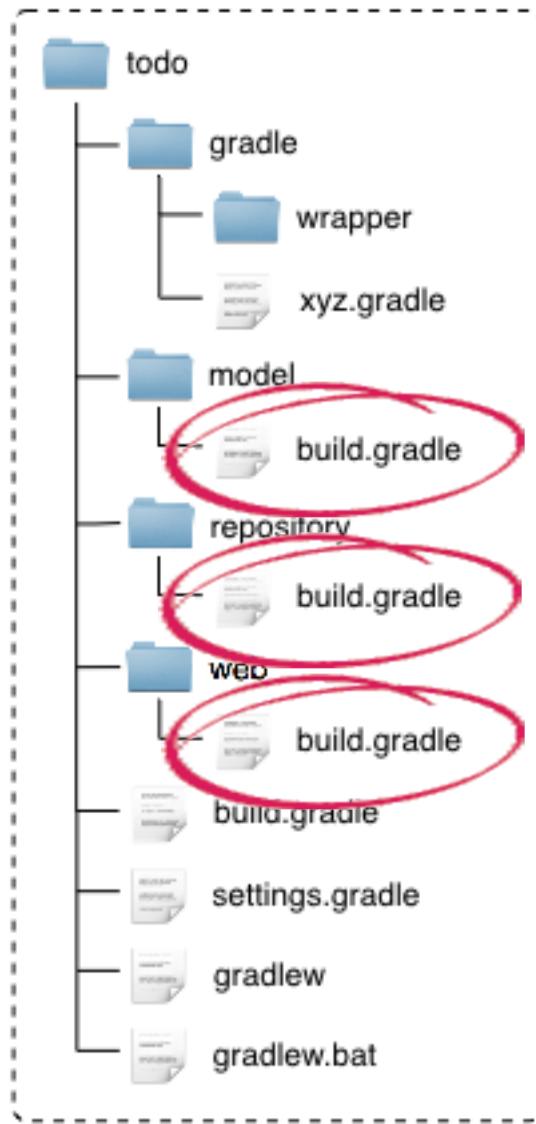
Project hierarchy



Project hierarchy

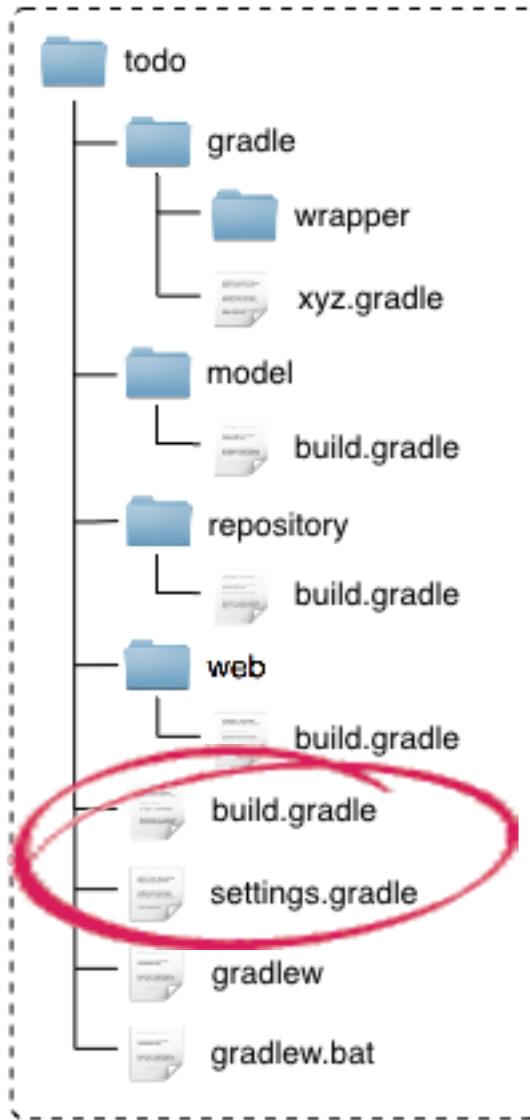


Project hierarchy



DEFINE PROJECT-SPECIFIC BEHAVIOR

Project hierarchy

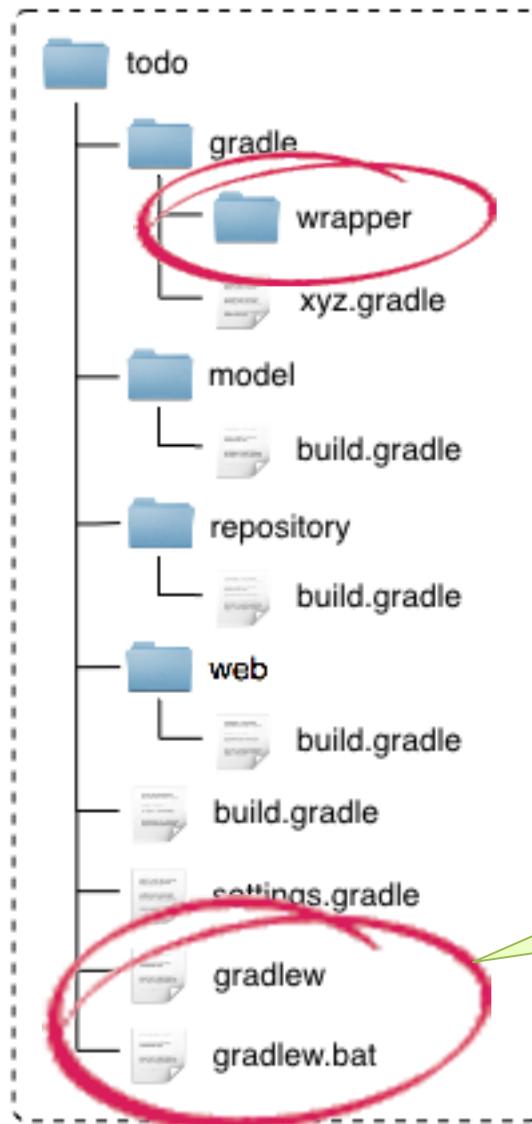


DEFINES WHICH PROJECTS
ARE TAKING PART IN THE
BUILD

settings.gradle

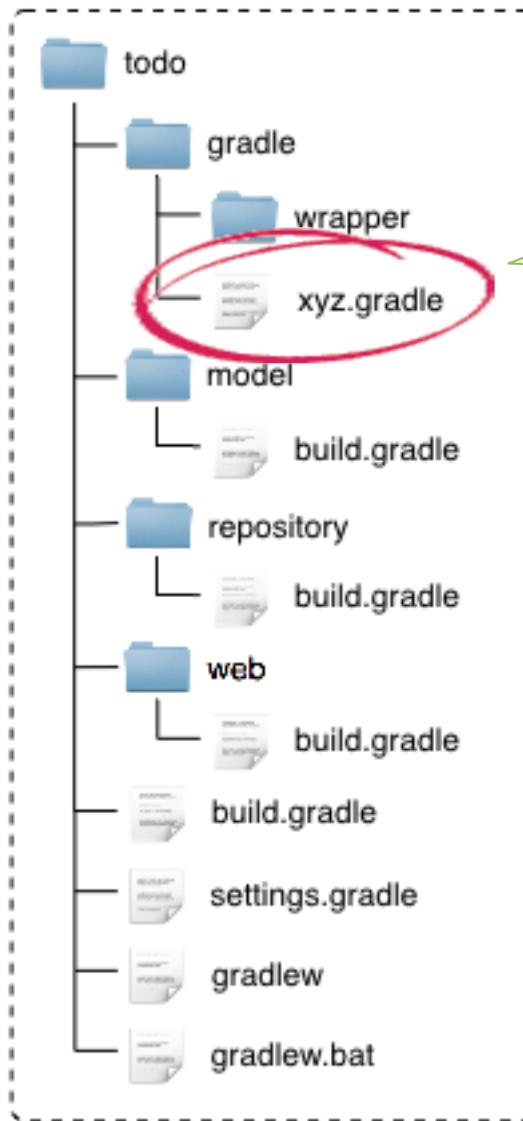
```
include 'model'  
include 'repository'  
include 'web'
```

Project hierarchy



ALWAYS USE WRAPPER
TO EXECUTE THE BUILD!

Project hierarchy

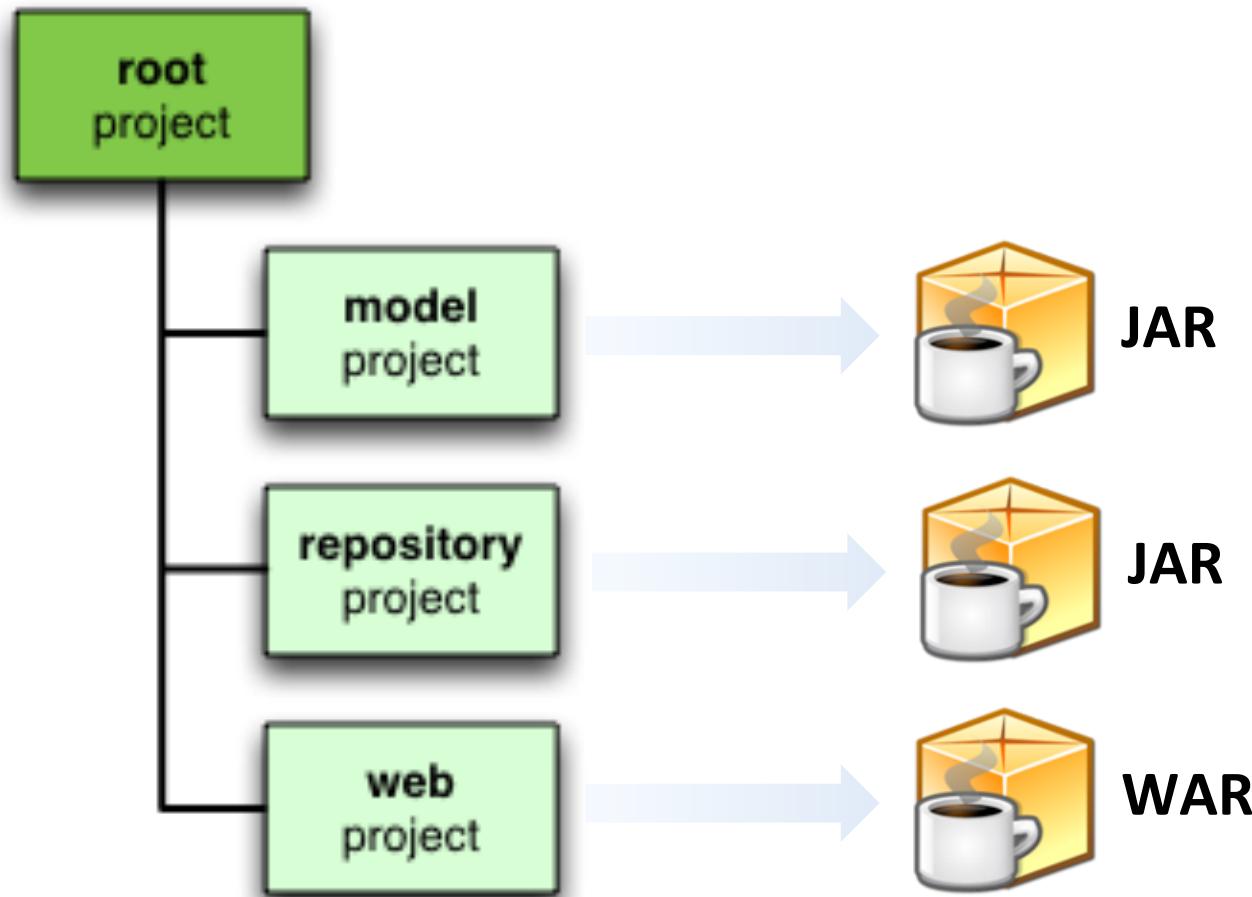


EXTERNALIZE CONCERNS INTO SCRIPT PLUGINS AND ORGANIZE THEM IN A DEDICATED DIRECTORY

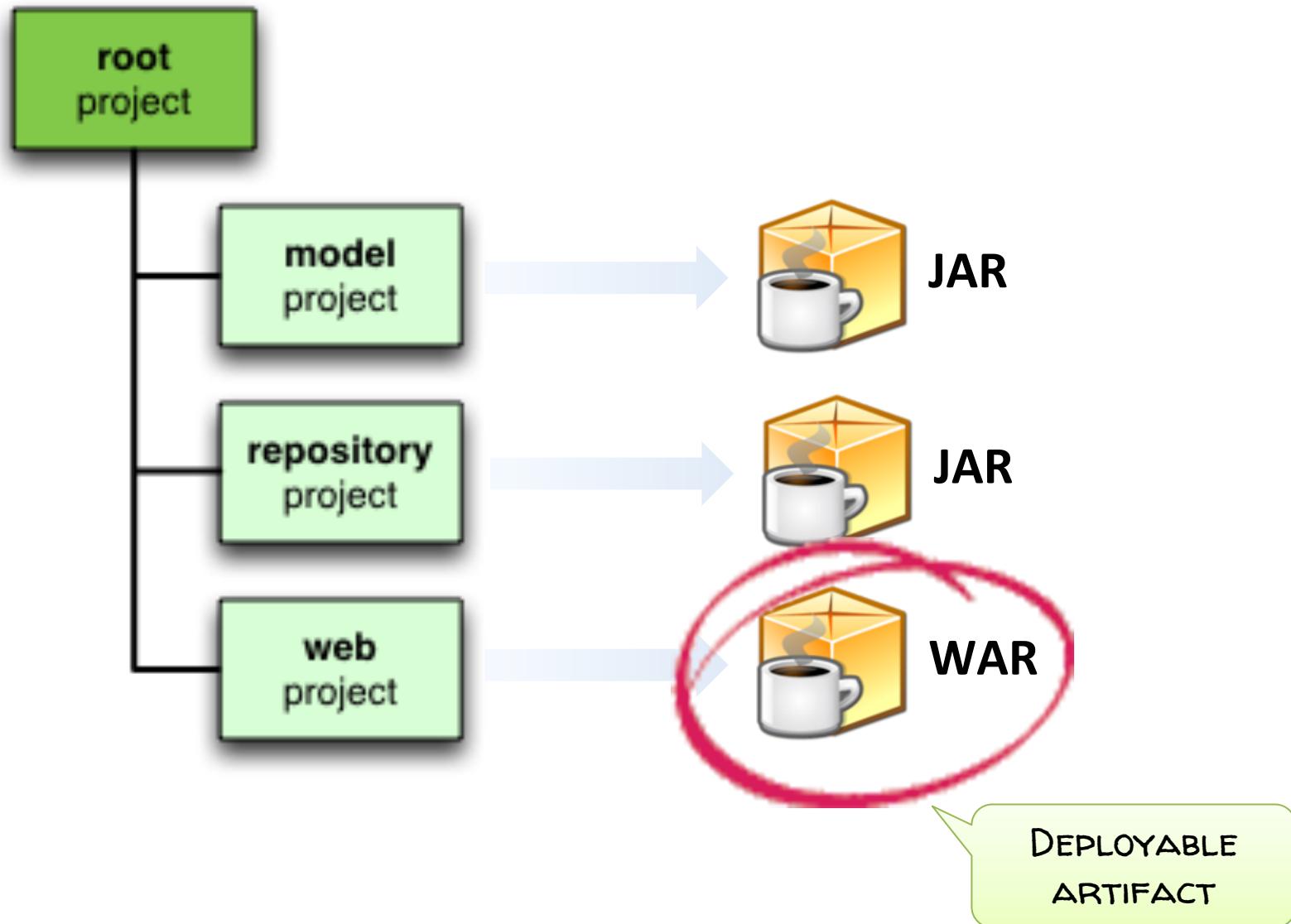
Examples:

- ✓ Versioning strategy
- ✓ Integration and functional test setup
- ✓ Deployment functionality
- ✓ ...

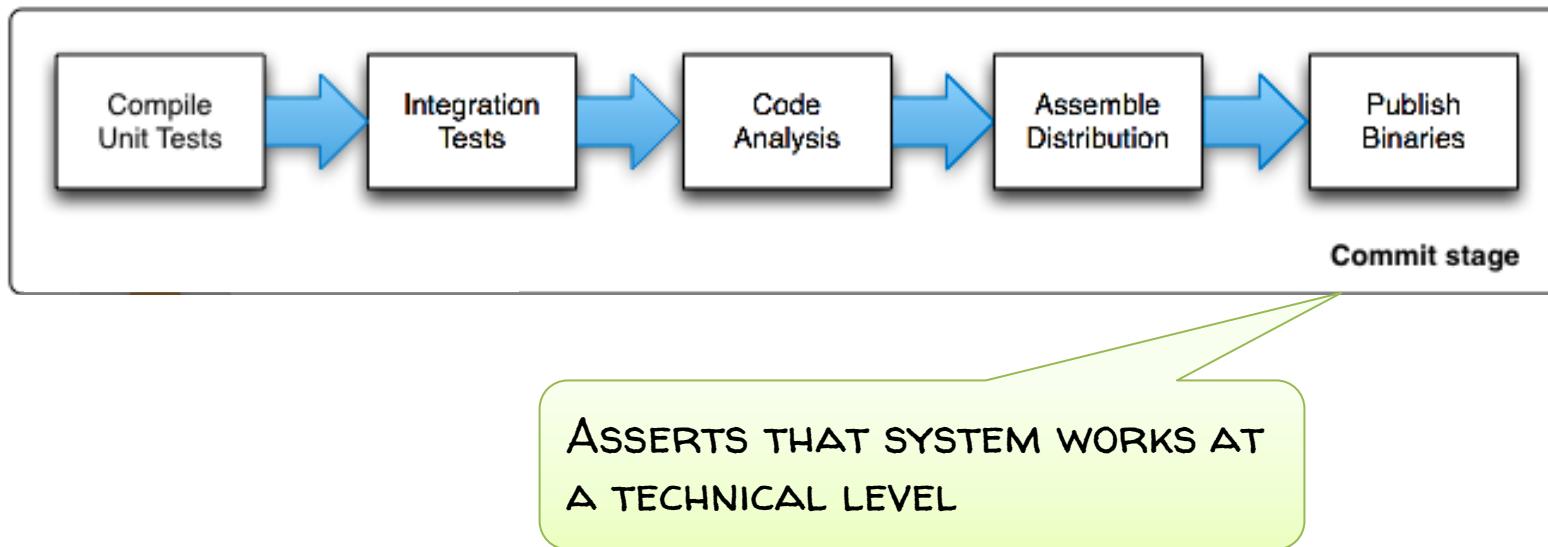
Project artifacts



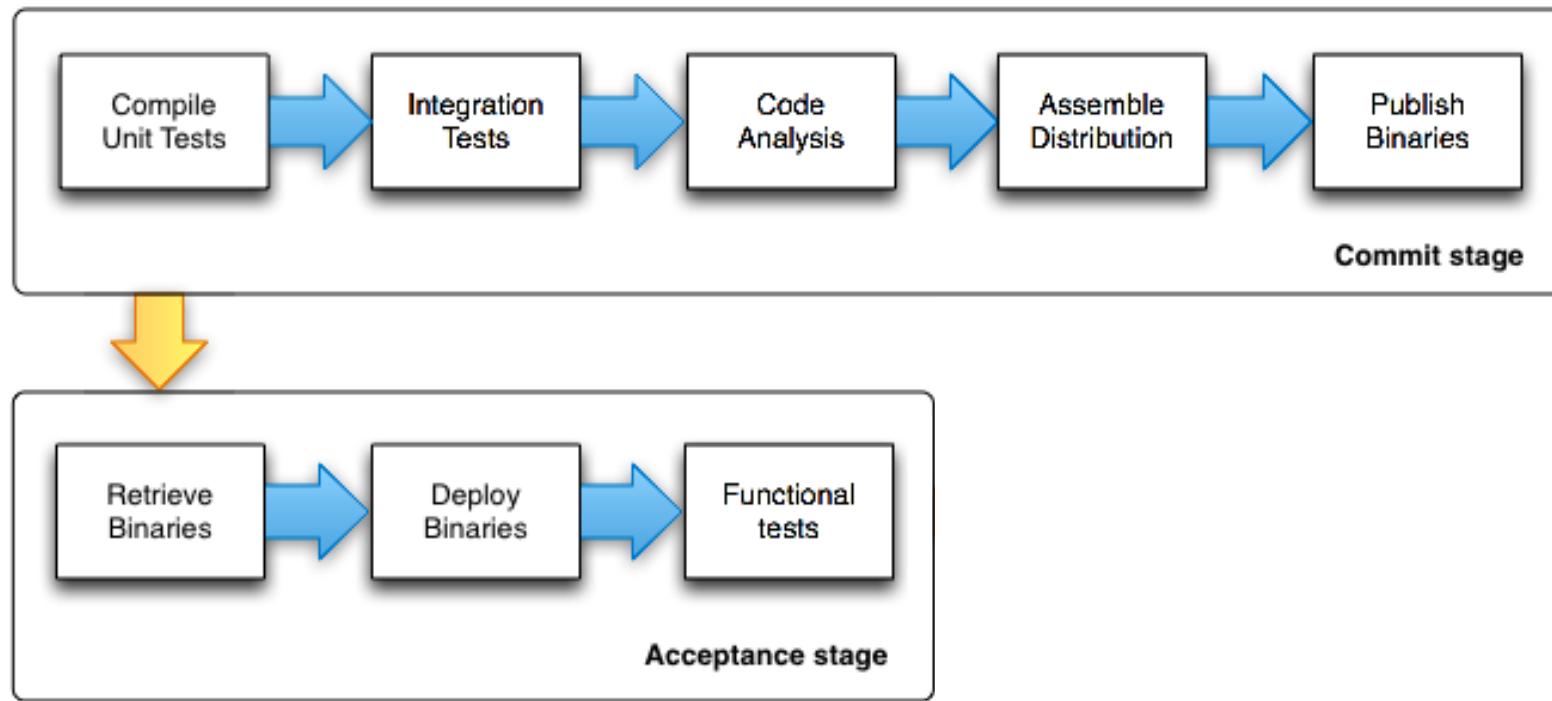
Project artifacts



Stages in build pipeline

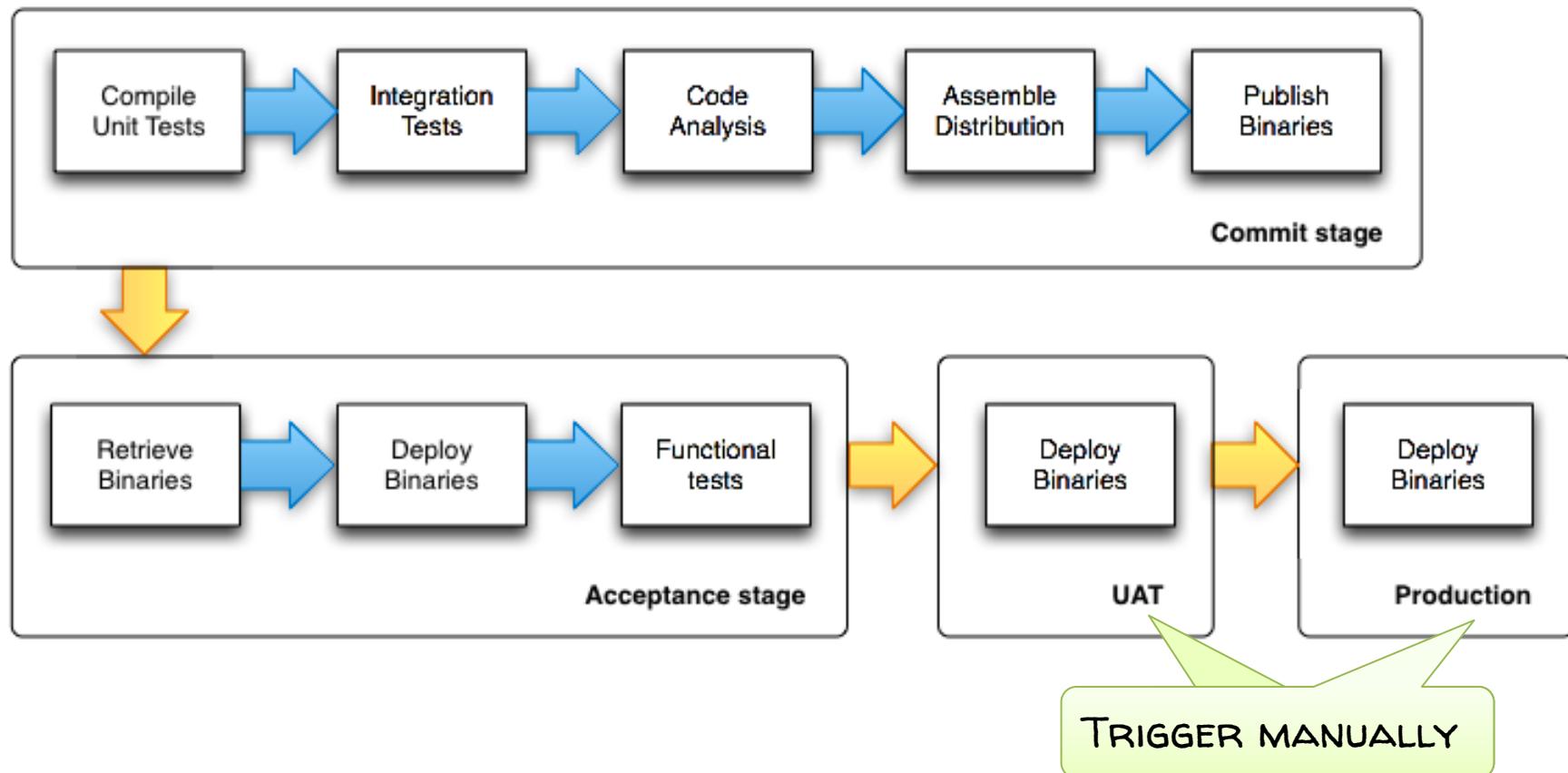


Stages in build pipeline

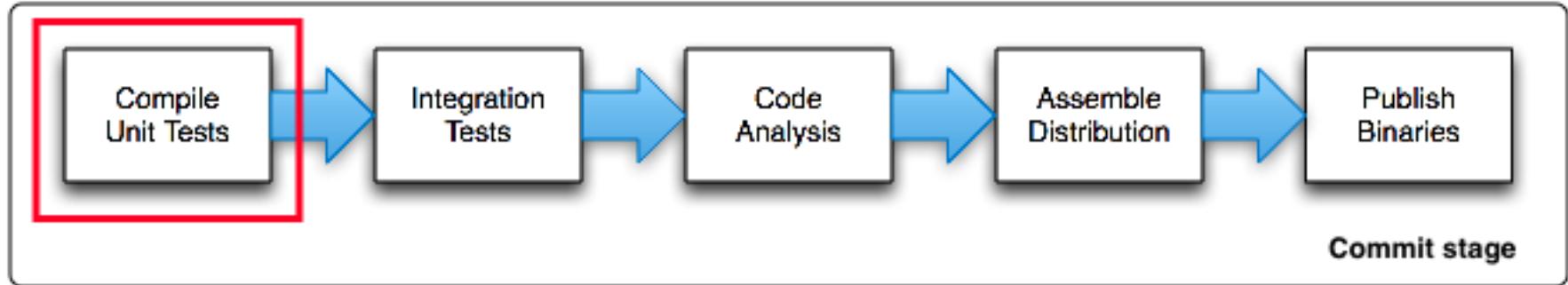


ASSERTS THAT SYSTEM WORKS ON A
FUNCTIONAL/NON-FUNCTIONAL LEVEL

Stages in build pipeline



Commit stage: Compile/unit tests



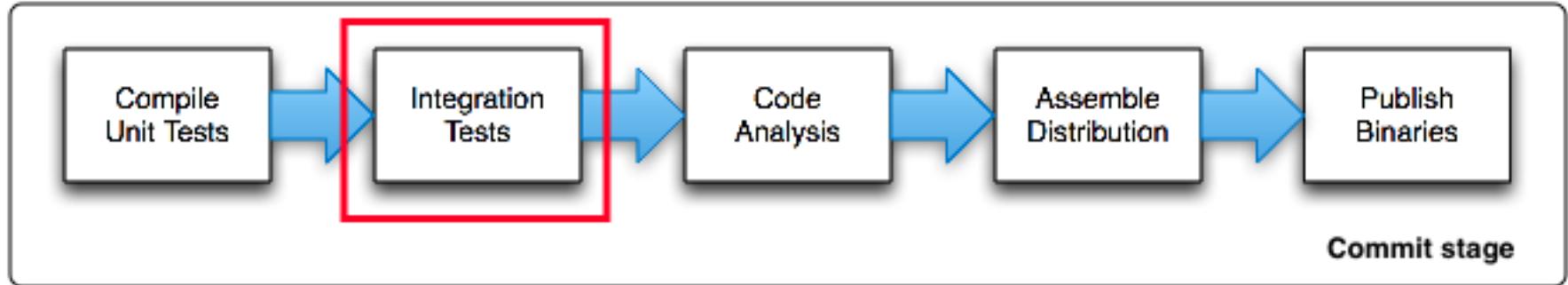
Rapid feedback (< 5 mins)

Run on every VCS check-in

Priority: fix broken build

```
17 string sInput;
18 int iLength, iN;
19 double dblTemp;
20 bool again = true;
21
22 while (again) {
23     iN = -1;
24     again = false;
25     getline(cin, sInput);
26     system("cls");
27     stringstream(sInput) >> dblTemp;
28     iLength = sInput.length();
29     if (iLength < 4) {
30         again = true;
31         continue;
32     } else if (sInput[iLength - 3] != '.') {
33         again = true;
34         continue;
35     } while (++iN < iLength) {
36         if (isdigit(sInput[iN])) {
37             continue;
38         } else if (iN == (iLength - 3)) {
39             continue;
40         }
41     }
42 }
```

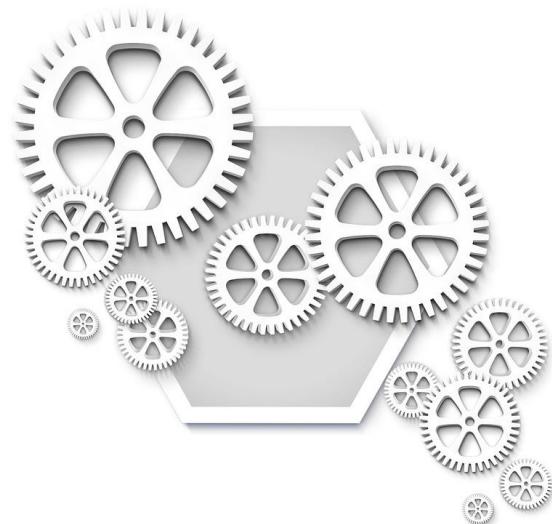
Commit stage: Integration tests



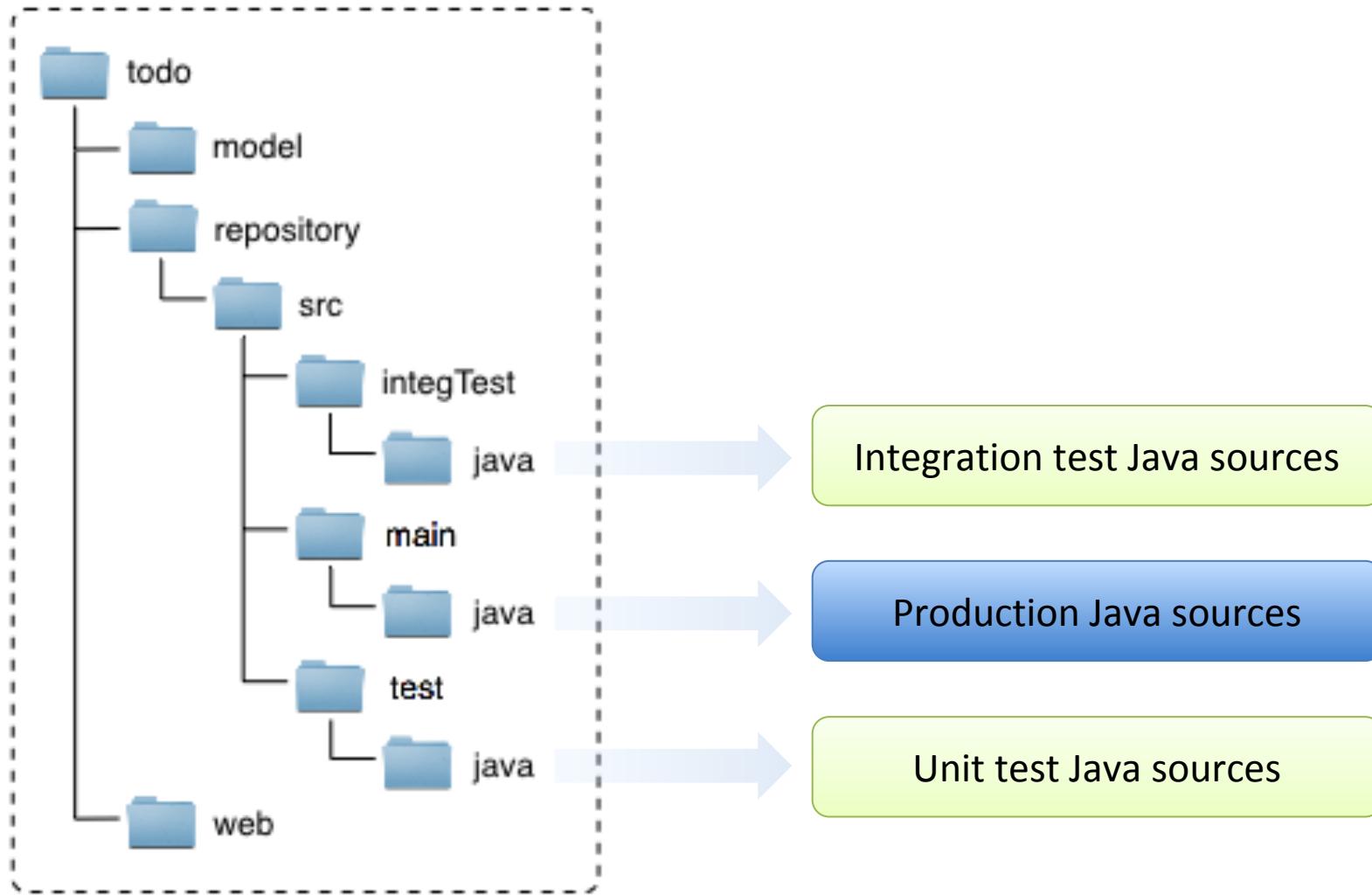
Long running tests

Environment setup

Maintenance cost



Separate tests in project layout



Separate tests with SourceSets

```
sourceSets {  
    integrationTest {  
        java.srcDir file('src/integTest/java')  
        resources.srcDir file('src/integTest/resources')  
        compileClasspath = sourceSets.main.output + configurations.testRuntime  
        runtimeClasspath = output + compileClasspath  
    }  
}  
  
task integrationTest(type: Test) {  
    description = 'Runs the integration tests.'  
    group = 'verification'  
    testClassesDir = sourceSets.integrationTest.output.classesDir  
    classpath = sourceSets.integrationTest.runtimeClasspath  
    testResultsDir = file("$testResultsDir/integration")  
}
```

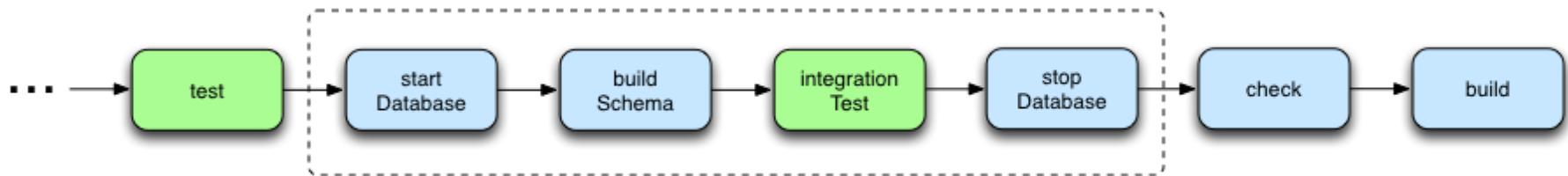
SET SOURCE AND RESOURCES DIRECTORY

SET COMPILE AND RUNTIME CLASSPATH

CUSTOM TEST RESULTS DIRECTORY

gradlew integrationTest

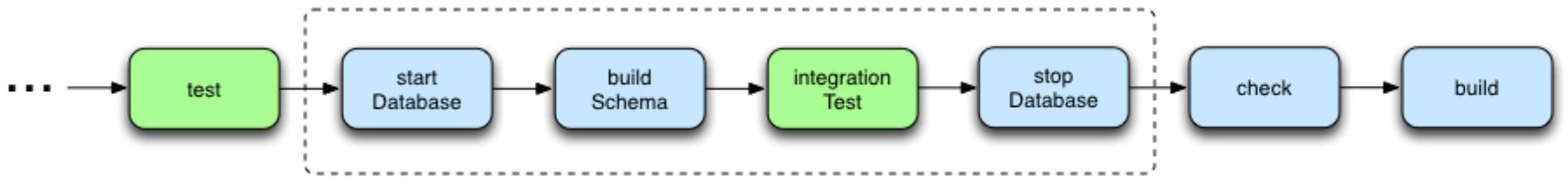
Database integration tests



LIQUI^{DB} BASE



Database integration tests



SEPARATE COMPLEX SETUP LOGIC INTO
SCRIPT PLUGIN

```
apply from: "$rootDir/gradle/databaseSetup.gradle"
```

```
integrationTest.dependsOn startAndPrepareDatabase  
integrationTest.finalizedBy stopDatabase
```

```
check.dependsOn integrationTest
```

INTEGRATE TASKS INTO BUILD
LIFECYCLE

Picking the “right” code coverage tool

Cobertura



Offline bytecode
instrumentation

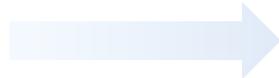


Offline bytecode
instrumentation

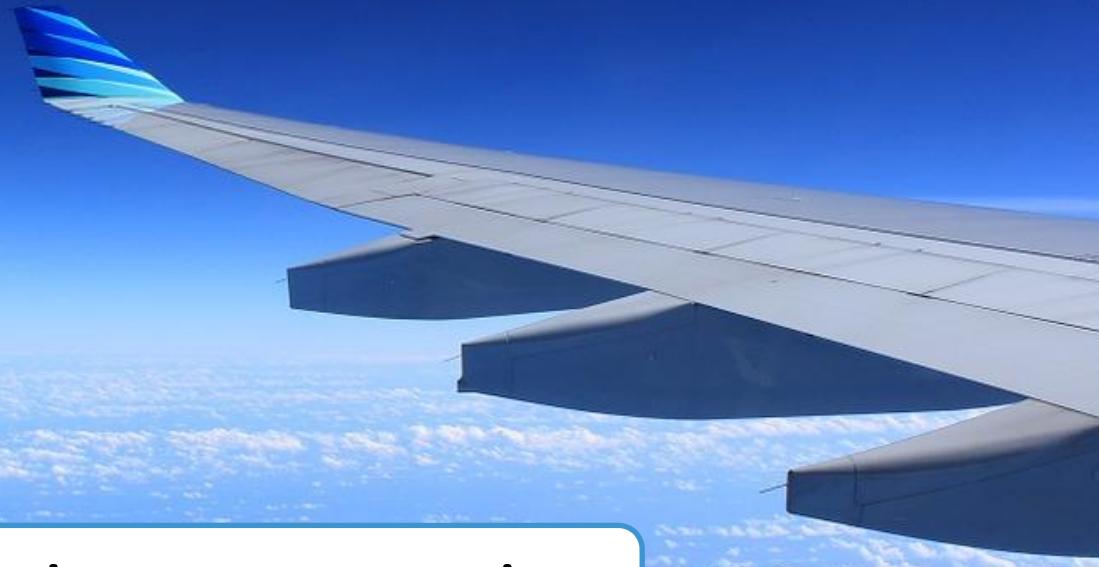
Clover



Source code
instrumentation



On-the-fly bytecode
instrumentation



On-the-fly bytecode instrumentation

No modification to source or bytecode

Code coverage with JaCoCo

```
apply plugin: "jacoco"

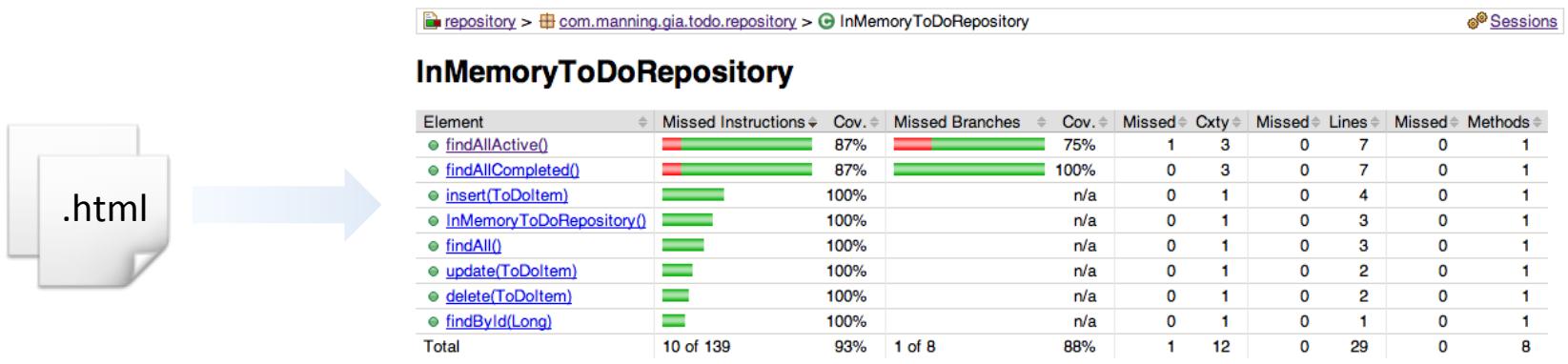
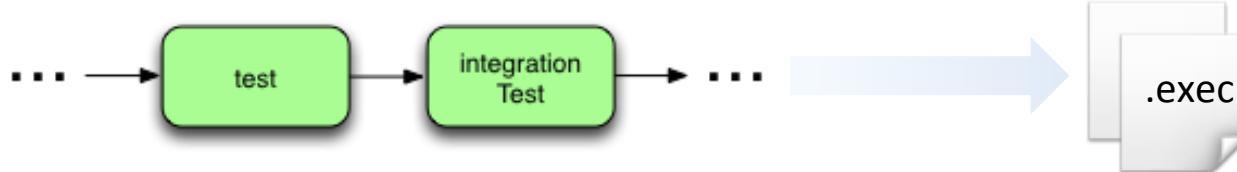
test {
    jacoco {
        append = false
        destinationFile = file "$buildDir/jacoco/jacocoTest.exec"
        classDumpFile = file "$buildDir/jacoco/testDumpFile"
    }
}

integrationTest {
    jacoco {
        append = false
        destinationFile = file "$buildDir/jacoco/jacocoInt.exec"
        classDumpFile = file "$buildDir/jacoco/intDumpFile"
    }
}
```

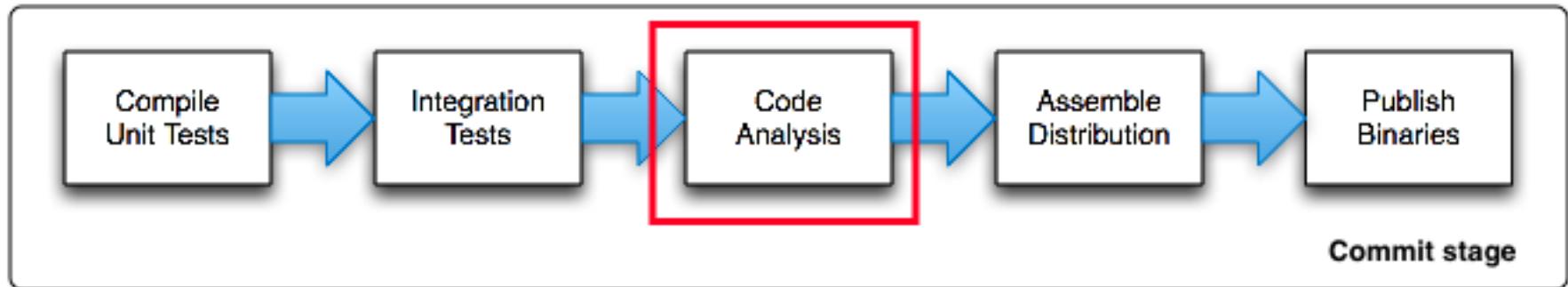
JACOCO EXTENSION IS
ADDED TO ALL TASKS
WITH TYPE TEST

CONFIGURES INSTRUMENTATION
FOR INTEGRATION TESTS AS
WELL

Generating coverage reports



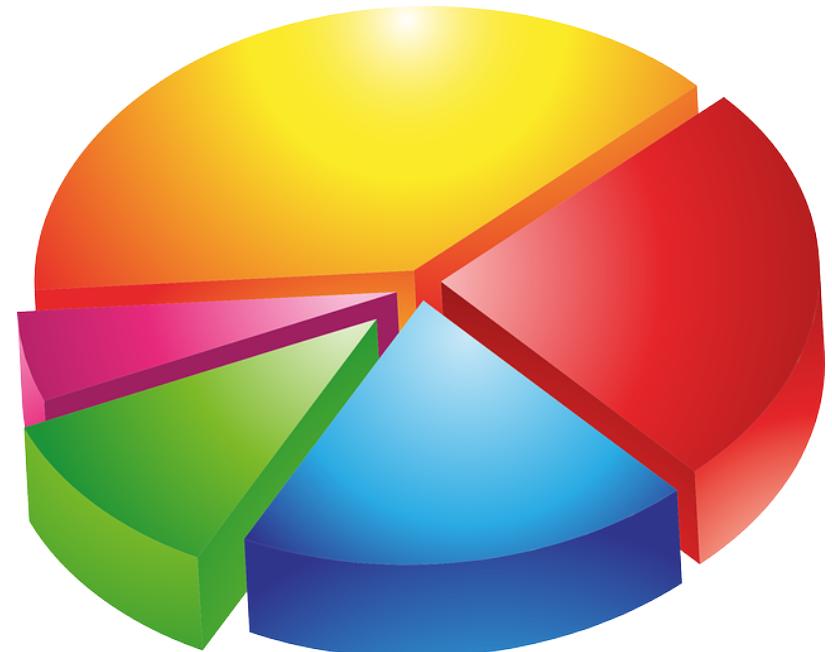
Commit stage: Code analysis



Perform code health check

Fail build for low quality

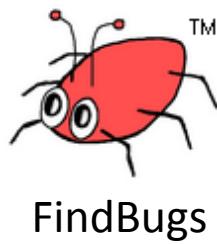
Record progress over time



Static code analysis tools



Checkstyle



FindBugs

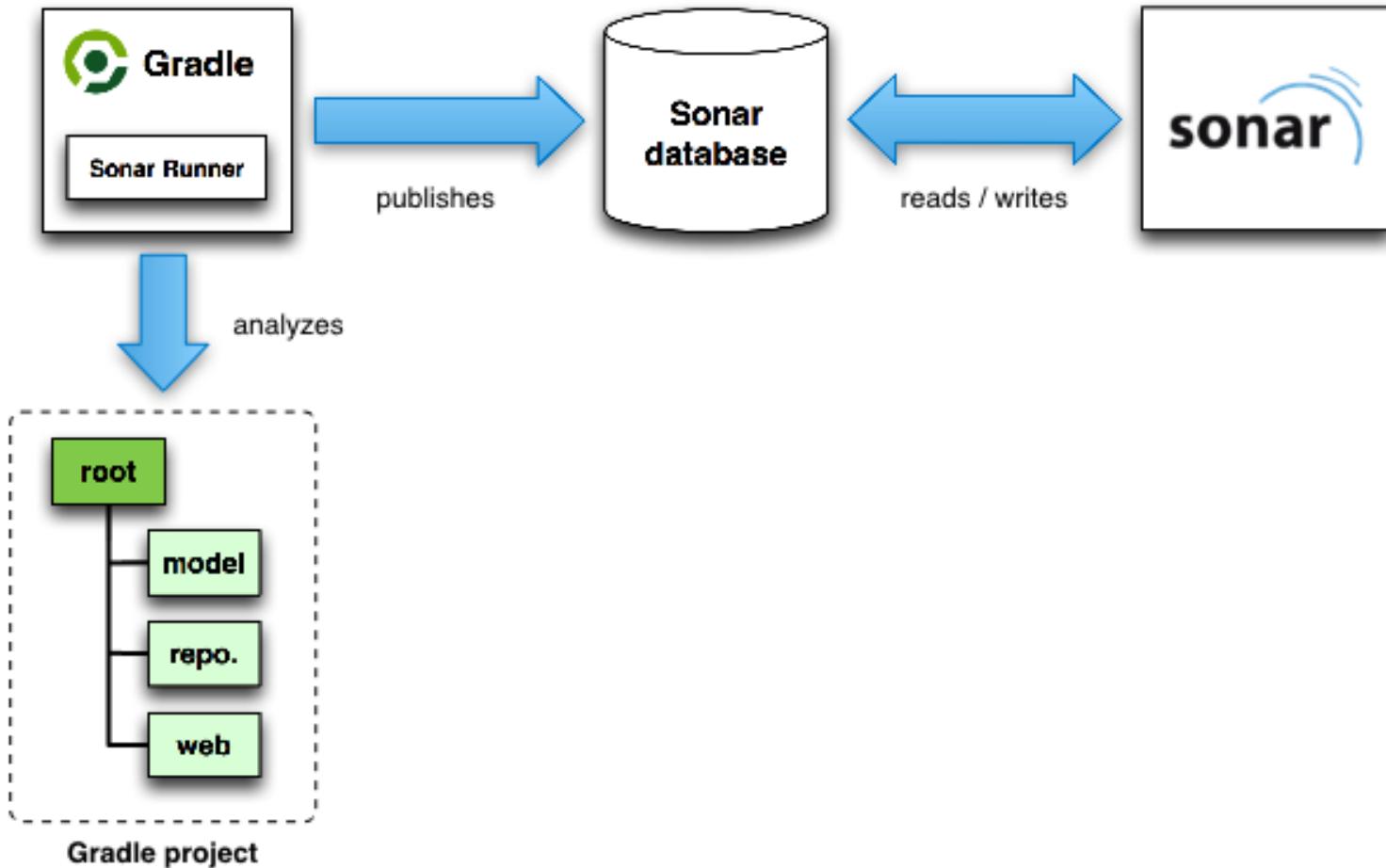
```
apply plugin: 'pmd'  
  
pmd {  
    ignoreFailures = true  
}  
  
tasks.withType(Pmd) {  
    reports {  
        xml.enabled = false  
        html.enabled = true  
    }  
}
```



```
apply plugin: 'jdepend'  
  
jdepend {  
    toolVersion = '2.9.1'  
    ignoreFailures = true  
}
```

gradlew check

Measure quality over time with Sonar



Applying the Sonar Runner plugin

```
apply plugin: 'sonar-runner'

sonarRunner {
    sonarProperties {
        property 'sonar.projectName', 'todo'
        property 'sonar.projectDescription', 'A task management
application'
    }
}

subprojects {
    sonarRunner {
        sonarProperties {
            property 'sonar.sourceEncoding', 'UTF-8'
        }
    }
}
```

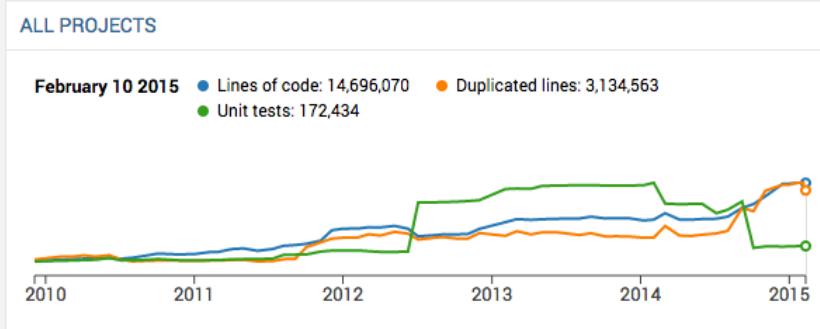
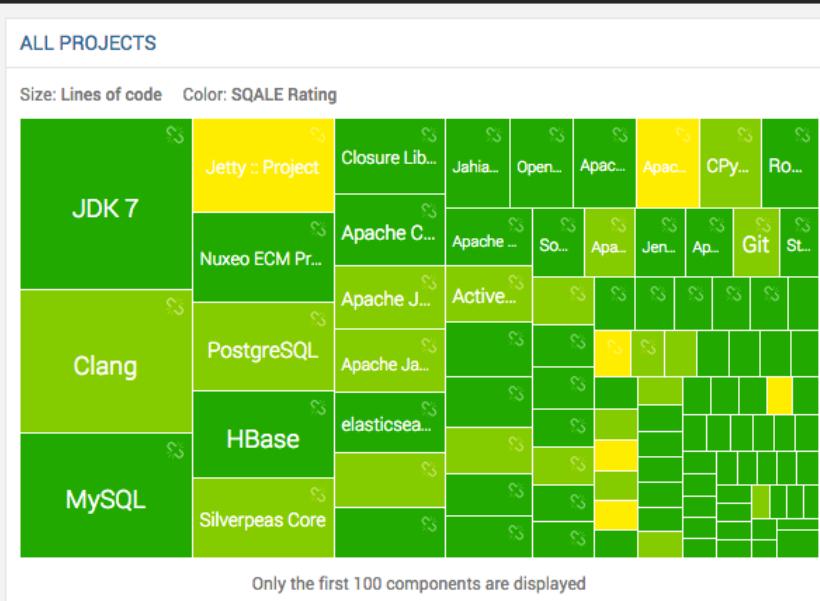
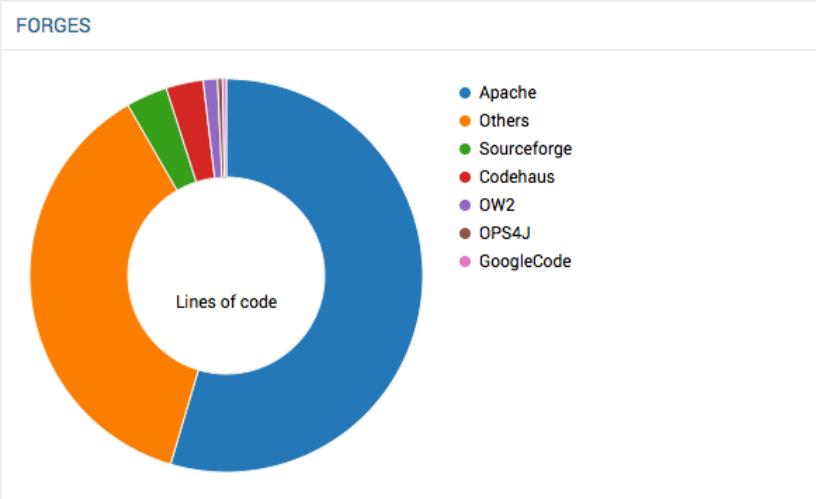
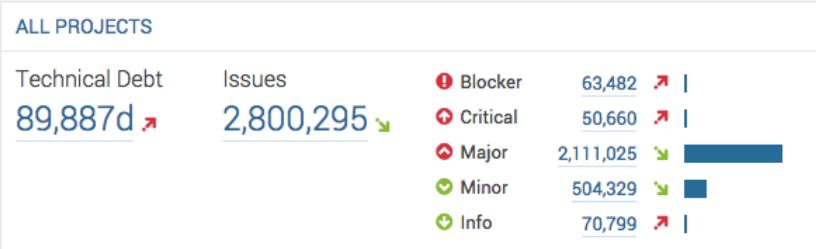
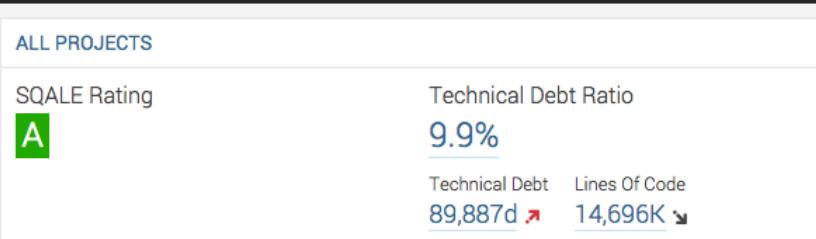
gradlew sonarRunner

Helicopter View

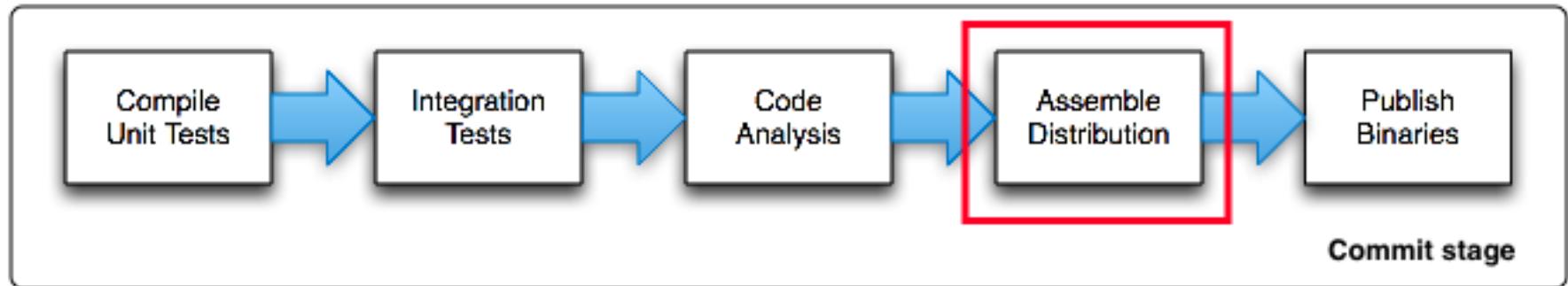
Languages Panel
Activity
SonarQube Ecosystem

TOOLS
Dependencies Compare

sonarqube
Sonar as a Service for your project with CloudBees



Commit stage: Assemble distribution



Exclude env. configuration

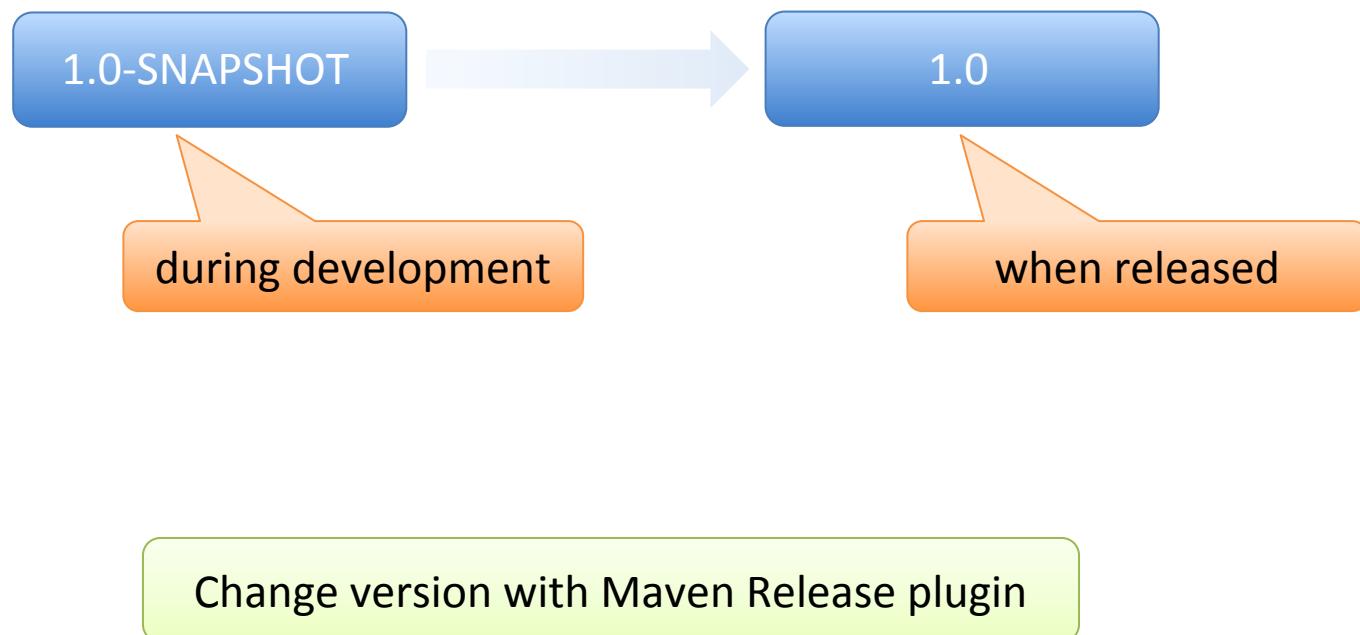
Include build information

Versioning strategy



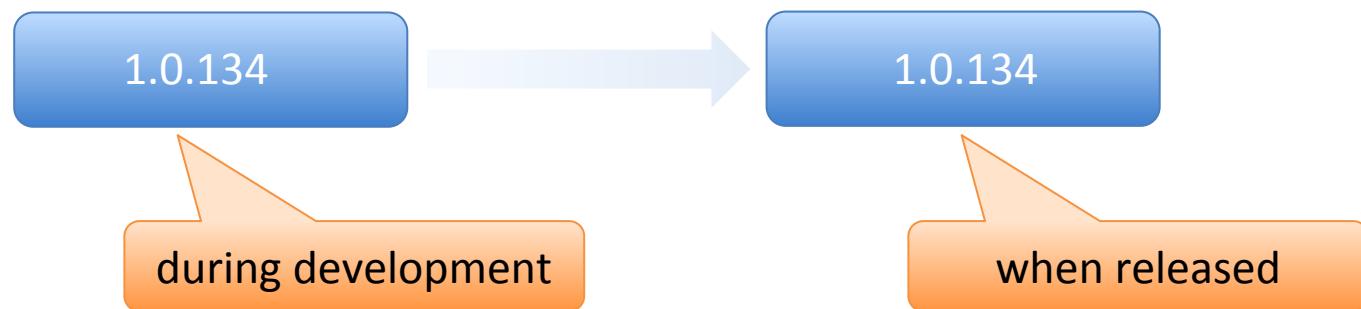
Versioning strategy

...the Maven way



Versioning strategy

...the Continuous Delivery way



1.0.134

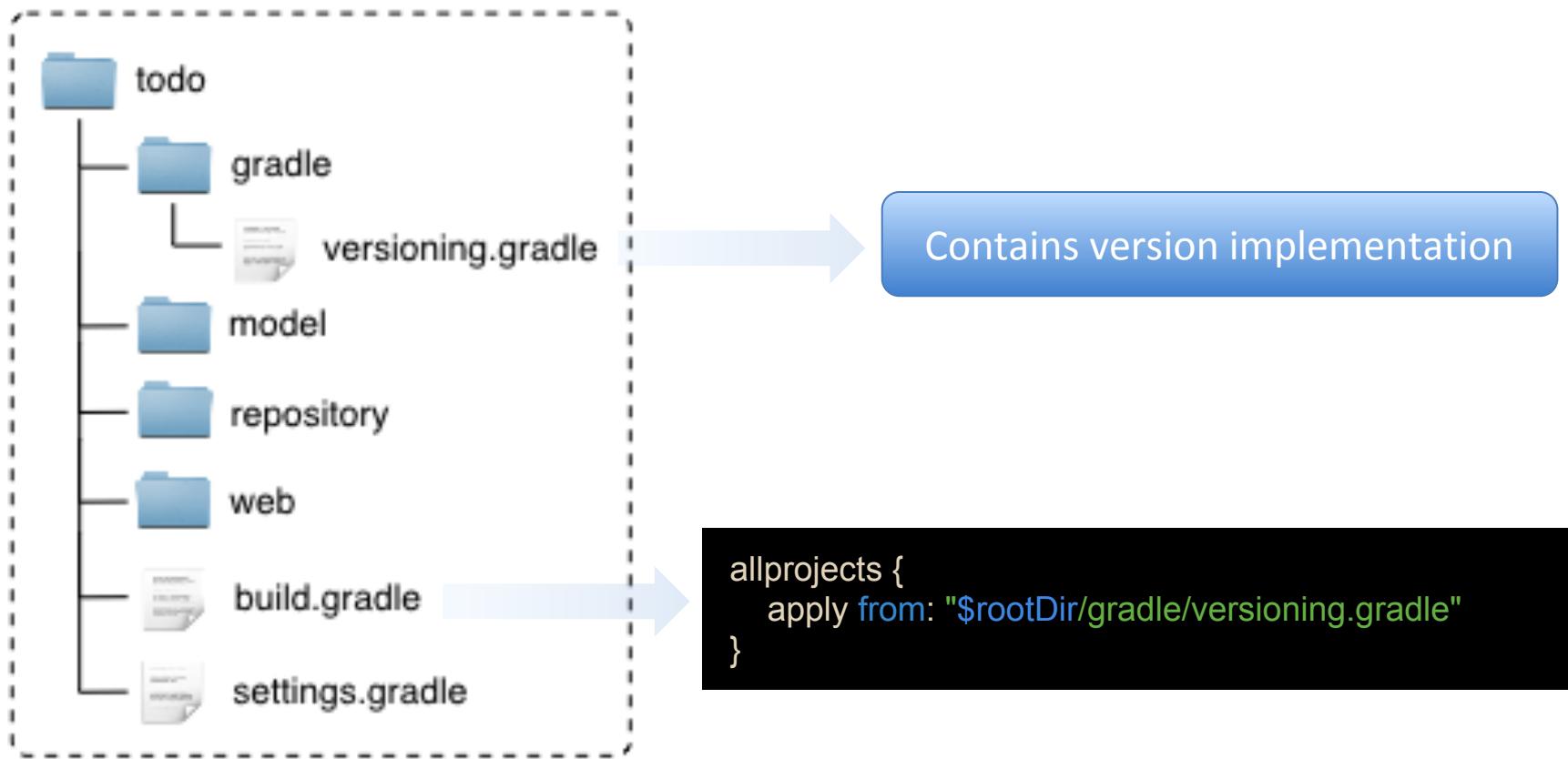
Project version number

Jenkins build number



Versioning strategy

...implemented with Gradle



Versioning strategy

...implemented with Gradle

```
ext.buildTimestamp = new Date().format('yyyy-MM-dd HH:mm:ss')

version = new ProjectVersion(1, 0, System.env.SOURCE_BUILD_NUMBER)

class ProjectVersion {
    Integer major
    Integer minor
    String build

    ProjectVersion(Integer major, Integer minor, String build) {
        this.major = major
        this.minor = minor
        this.build = build
    }

    @Override
    String toString() {
        String fullVersion = "$major.$minor"

        if(build) {
            fullVersion += ".$build"
        }

        fullVersion
    }
}
```

JENKINS BUILD NUMBER

BUILDS VERSION
STRING REPRESENTATION

Packaging the deployable artifact



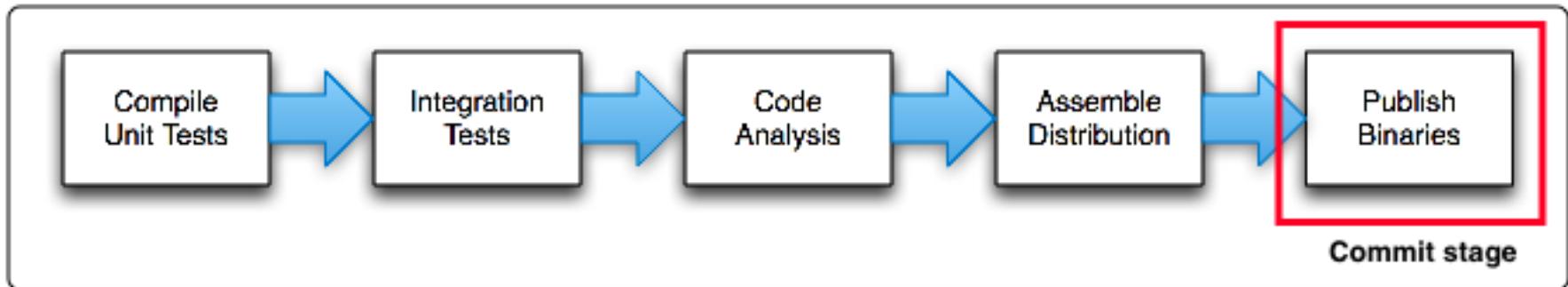
```
project(':web') {  
    apply plugin: 'war'  
  
    task createBuildInfoFile << {  
        def buildInfoFile = new File("$buildDir/build-info.properties")  
        Properties props = new Properties()  
        props.setProperty('version', project.version.toString())  
        props.setProperty('timestamp', project.buildTimestamp)  
        props.store(buildInfoFile.newWriter(), null)  
    }  
  
    war {  
        dependsOn createBuildInfoFile  
        basePath = 'todo'  
  
        from(buildDir) {  
            include 'build-info.properties'  
            into('WEB-INF/classes')  
        }  
    }  
}
```

Creates file containing build information

Include build info file into WAR distribution

gradlew assemble

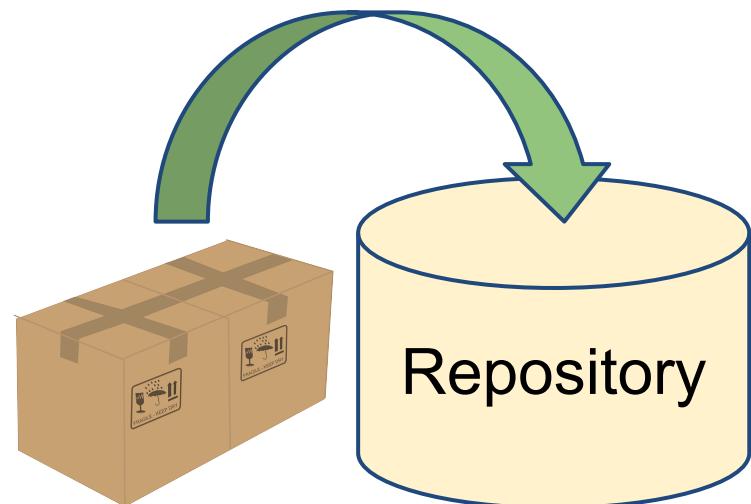
Commit stage: Publish binaries



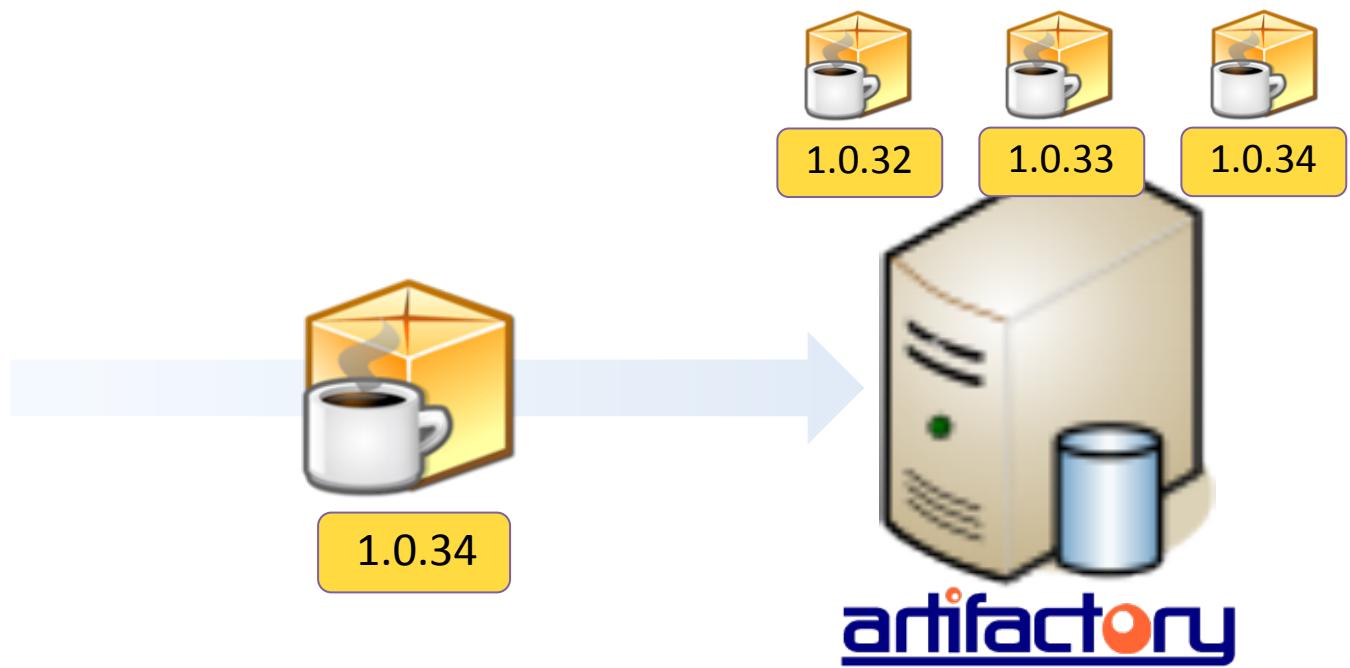
Version artifact(s)

Use binary repository

Publish once, then reuse



Publishing the deployable artifact



Defining build configuration

COMMON
CONFIGURATION

ENV-SPECIFIC
CONFIGURATION

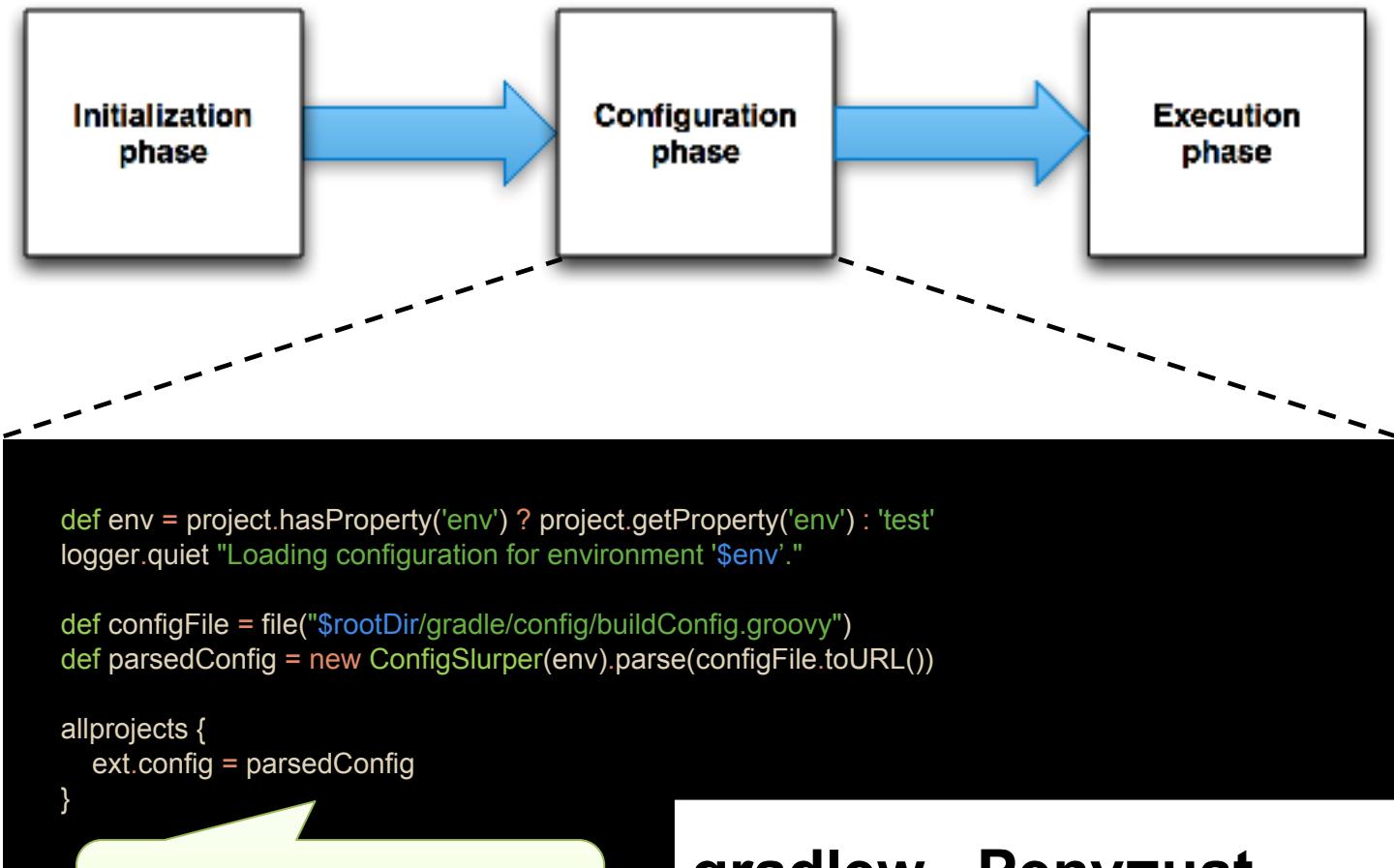
```
binaryRepository {  
    url = 'http://mycompany.bin.repo:8081/artifactory'  
    username = 'admin'  
    password = 'password'  
    name = 'libs-release-local'  
}  
  
environments {  
    test {  
        server {  
            hostname = 'mycompany.test'  
            port = 8099  
            context = 'todo'  
            username = 'manager'  
            password = 'manager'  
        }  
    }  
    uat {  
        server {  
            hostname = 'mycompany.uat'  
            port = 8199  
            context = 'todo'  
            username = 'manager'  
            password = 'manager'  
        }  
    }  
    ...  
}
```

READ CREDENTIALS FROM
GRADLE.PROPERTIES

READ CREDENTIALS FROM
GRADLE.PROPERTIES

READ CREDENTIALS FROM
GRADLE.PROPERTIES

Reading build configuration



ASSIGN CONFIGURATION
TO EXTRA PROPERTY

gradlew -Penv=uat ...

Using the Maven Publishing plugin

BUILD REPOSITORY URL
FROM CONFIGURATION

```
apply plugin: 'maven-publish'

ext.fullRepoUrl = "$config.binaryRepository.url/$config.binaryRepository.name"

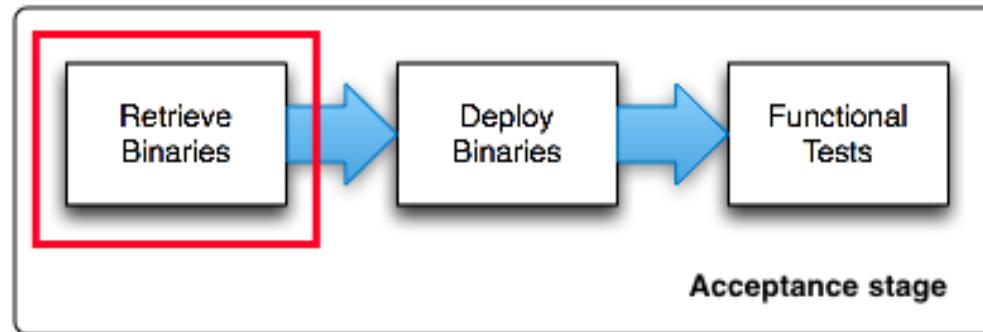
publishing {
    publications {
        webApp(MavenPublication) {
            from components.web
        }
    }

    repositories {
        maven {
            url fullRepoUrl

            credentials {
                username = config.binaryRepository.username
                password = config.binaryRepository.password
            }
        }
    }
}
```

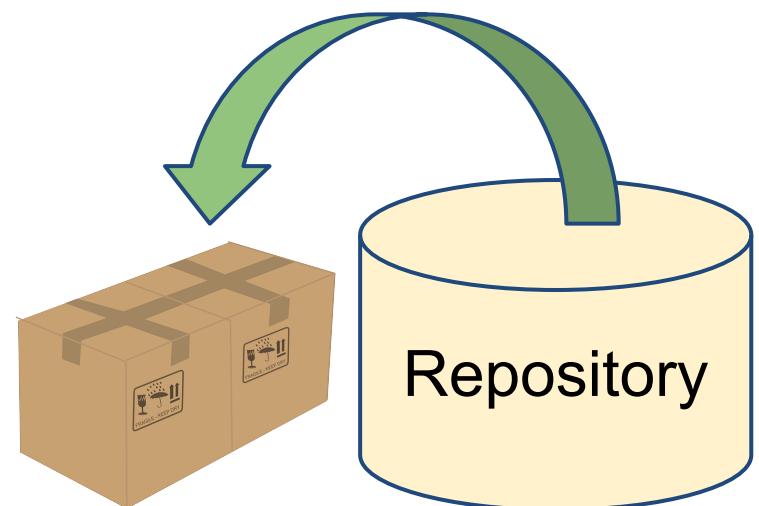
gradlew publish

Acceptance stage: Retrieve binaries

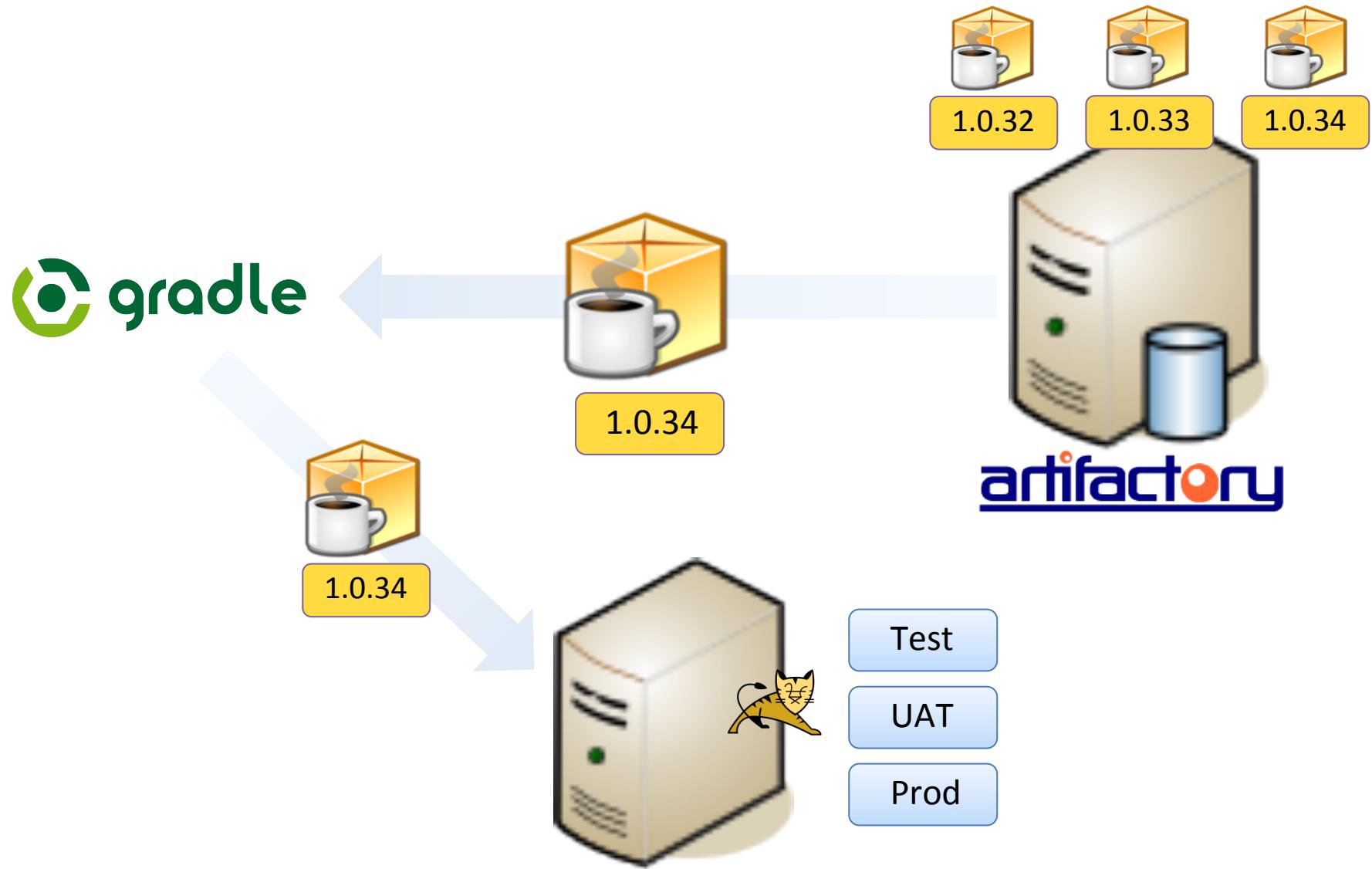


Request versioned artifact

Store in temp. directory



Downloading the deployable artifact



Task for downloading artifact

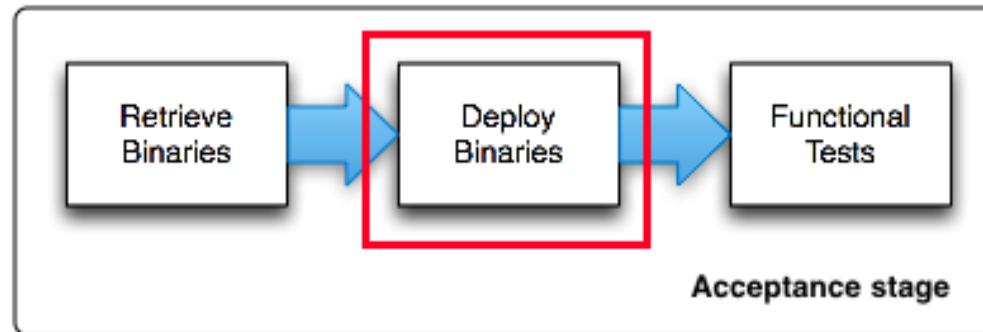
DECLARE A DEPENDENCY
ON THE WAR

```
configurations {  
    todo  
}  
  
dependencies {  
    todo group: project.group, name: project.name, version: project.version.toString(), ext: 'war'  
}  
  
ext.downloadDir = file("$buildDir/download/artifacts")  
  
task fetchToDoWar(type: Copy) {  
    from configurations.todo  
    into downloadDir  
}
```

COPY IT TO THE DOWNLOAD
DIRECTORY

gradlew fetchToDoWar

Acceptance stage: Deploy binaries



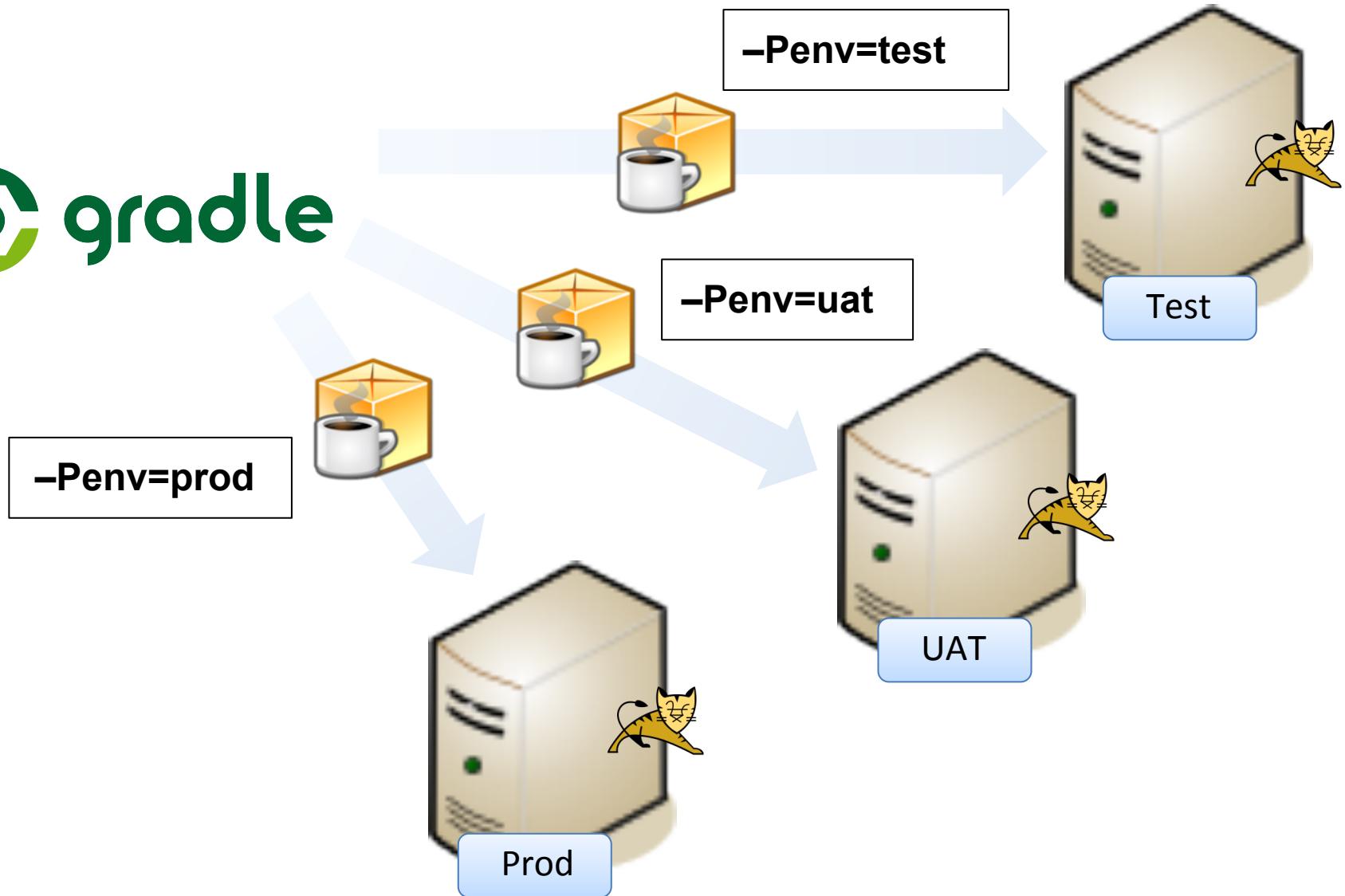
Deployment on request

Make it a reliable process

Use process for all envs.



Deploying to multiple environments



Deployment with the Cargo plugin

```
cargoDeployRemote.dependsOn fetchToDoWar, cargoUndeployRemote
```

```
cargoUndeployRemote {  
    onlyIf applicationContextStatus  
}
```

```
cargo {  
    containerId = 'tomcat7x'  
    port = config.server.port
```

```
    deployable {  
        file = downloadedArtifact  
        context = config.server.context  
    }
```

```
    remote {  
        hostname = config.server.hostname  
        username = config.server.username  
        password = config.server.password  
    }  
}
```

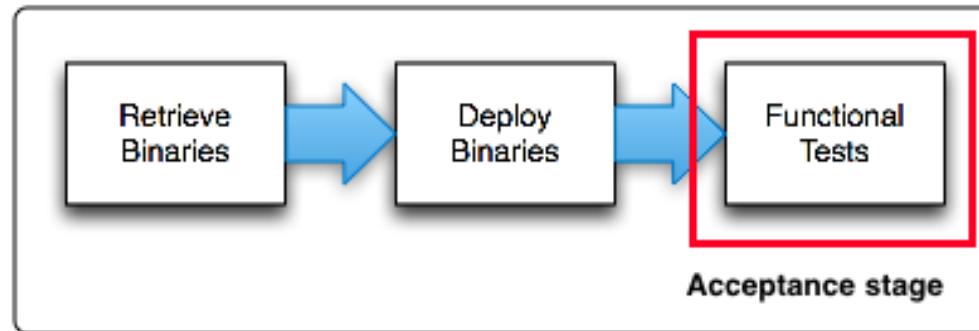
DOWNLOAD ARTIFACT FROM BINARY
REPOSITORY AND UNDEPLOY EXISTING

ONLY UNDEPLOY IF
URL CONTEXT EXISTS

USE ENVIRONMENT-SPECIFIC
CONFIGURATION

gradlew –Penv=uat cargoDeploy

Acceptance stage: Functional tests



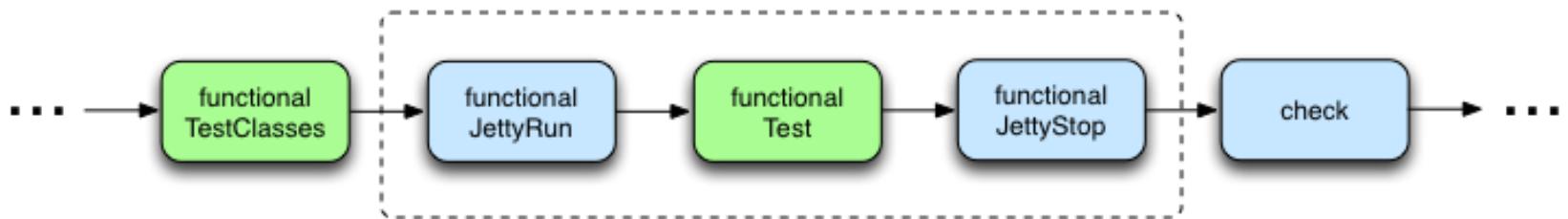
Test all UI permutations

Test important use cases

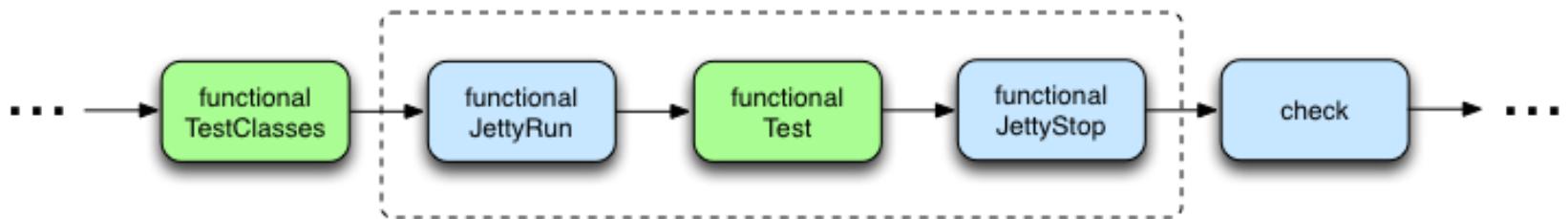
Run against different envs.



In-container functional tests



Local functional tests



```
task functionalTest(type: Test) {  
    ...  
}
```

FUNCTIONAL TEST TASK

```
task functionalJettyRun(type: org.gradle.api.plugins.jetty.JettyRun) {  
    httpPort = functionalJettyHttpPort  
    stopPort = functionalJettyStopPort  
    stopKey = functionalJettyStopKey  
    contextPath = functionalJettyContextPath  
    daemon = true  
}
```

CUSTOM JETTY RUN TASK

```
task functionalJettyStop(type: org.gradle.api.plugins.jetty.JettyStop) {  
    stopPort = functionalJettyStopPort  
    stopKey = functionalJettyStopKey  
}
```

CUSTOM JETTY STOP TASK

```
functionalJettyRun.dependsOn functionalTestClasses  
functionalTest.dependsOn functionalJettyRun  
functionalTest.finalizedBy functionalJettyStop
```

Executing remote functional tests

```
ext {  
    functionalTestReportDir = file("$testReportDir/functional")  
    functionalTestResultsDir = file("$testResultsDir/functional")  
    functionalCommonSystemProperties =  
        ['geb.env': 'firefox',  
         'geb.build.reportsDir': reporting.file("$name/geb")]  
}
```

REUSE SETUP
PROPERTIES

```
task remoteFunctionalTest(type: Test) {  
    testClassesDir = sourceSets.functionalTest.output.classesDir  
    classpath = sourceSets.functionalTest.runtimeClasspath  
    testReportDir = functionalTestReportDir  
    testResultsDir = functionalTestResultsDir  
    systemProperties functionalCommonSystemProperties  
    systemProperty 'geb.build.baseUrl',  
        "http://$config.server.hostname:$config.server.port/$config.server.context/"  
}
```

BUILD URL FROM
ENV. CONFIGURATION

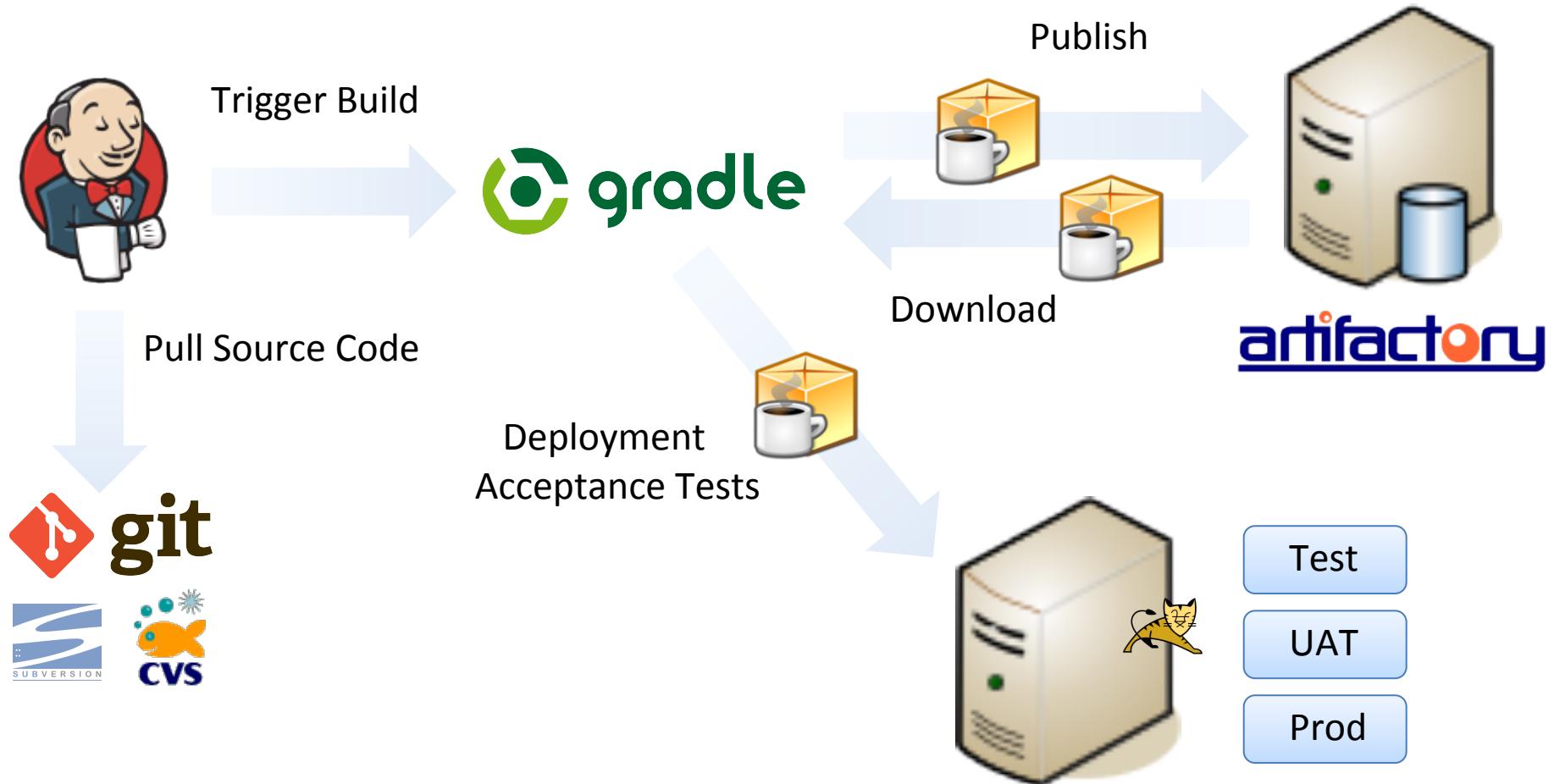
gradlew –Penv=test remoteFunctionalTest

Going further: Capacity testing

```
buildscript {  
    repositories {  
        mavenCentral()  
    }  
  
    dependencies {  
        classpath "com.github.kulya:jmeter-gradle-plugin:1.3.1-2.9"  
    }  
}  
  
apply plugin: com.github.kulya.gradle.plugins.jmeter.JmeterPlugin  
  
ext.loadTestResources = "$projectDir/src/loadTest/resources"  
  
jmeterRun.configure {  
    jmeterTestFiles = [file("$loadTestResources/todo-test-plan.jmx")]  
    jmeterPropertyFile = file("$loadTestResources/jmeter.properties")  
    jmeterUserProperties = ["hostname=${config.server.hostname},  
                           "port=${config.server.port}",  
                           "context=${config.server.context}"]  
    logging.captureStandardError LogLevel.INFO  
}
```

gradlew –Penv=uat jmeterRun

Let's bring Jenkins into play!



Model pipeline as series of jobs

		todo-acceptance-tests	15 days (todo#76)	N/A	1 min 56 sec	
		todo-code-quality	15 days (todo#76)	N/A	1 min 6 sec	
		todo-deploy-production	N/A	N/A	N/A	
		todo-deploy-test	15 days (todo#76)	N/A	51 sec	
		todo-deploy-uat	20 days (todo#65)	16 days (todo#67)	47 sec	
		todo-distribution	15 days (todo#76)	29 days (todo#41)	45 sec	
		todo-initial	15 days (todo#76)	N/A	56 sec	
		todo-integ-tests	15 days (todo#76)	N/A	46 sec	
		todo-performance-tests	15 days (todo#76)	15 days (todo#74)	1 min 28 sec	

TRIGGER JOB WHEN SCM
CHANGE IS DETECTED



Jenkins

Initial Jenkins Build Job

Jenkins / todo-initial

ENABLE AUTO REFRESH

[Back to Dashboard](#)

[Status](#)

[Changes](#)

[Workspace](#)

[Build Now](#)

[Delete Project](#)

[Configure](#)

[Coverage Trend](#)

[Git Polling Log](#)

Project todo-initial

[Workspace](#)

[Recent Changes](#)

[Latest Test Result \(no failures\)](#)

[add description](#)

[Disable Project](#)

Test Result Trend

Build	Count
todo#74	7
todo#75	7
todo#76	7
todo#77	7
todo#78	7

[\(just show failures\) enlarge](#)

Code Coverage Trend

Build	lineCovered	lineMissed
todo#74	~120	~250
todo#75	~120	~250
todo#76	~120	~250
todo#77	~120	~250
todo#78	~120	~250

[RSS for all](#) [RSS for failures](#)

Downstream Projects

[todo-integ-tests](#)

Permalinks

- [Last build \(todo#78\), 1 hr 24 min ago](#)
- [Last stable build \(todo#78\), 1 hr 24 min ago](#)
- [Last successful build \(todo#78\), 1 hr 24 min ago](#)



Build Name Setter Plugin

Jenkins search [ENABLE AUTO REFRESH](#)

[Back to Dashboard](#) [Status](#) [Changes](#) [Workspace](#) [Build Now](#) [Delete Project](#) [Configure](#) [Coverage Trend](#) [Git Polling Log](#)

Project todo-initial

[Workspace](#) [Recent Changes](#) [Latest Test Result \(no failures\)](#)

Test Result Trend

count

Build	Count
todo#74	7
todo#75	0
todo#76	0
todo#77	0
todo#78	0

[\(just show failures\) enlarge](#)

Code Coverage Trend

lineCovered lineMissed

Build	lineCovered	lineMissed
todo#74	~120	~250
todo#75	~120	~250
todo#76	~120	~250
todo#77	~120	~250
todo#78	~120	~250

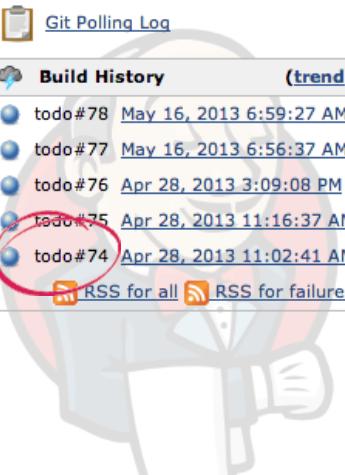
Downstream Projects

[todo-integ-tests](#)

Permalinks

- [Last build \(todo#78\), 1 hr 24 min ago](#)
- [Last stable build \(todo#78\), 1 hr 24 min ago](#)
- [Last successful build \(todo#78\), 1 hr 24 min ago](#)

[RSS for all](#) [RSS for failures](#)





JaCoCo Plugin

Jenkins / todo-initial

ENABLE AUTO REFRESH

[Back to Dashboard](#)

[Status](#)

[Changes](#)

[Workspace](#)

[Build Now](#)

[Delete Project](#)

[Configure](#)

[Coverage Trend](#)

[Git Polling Log](#)

Project todo-initial

[Workspace](#)

[Recent Changes](#)

[Latest Test Result \(no failures\)](#)

[Test Result Trend](#)

[add description](#)

[Disable Project](#)

Downstream Projects

[todo-integ-tests](#)

Permalinks

- [Last build \(todo#78\), 1 hr 24 min ago](#)
- [Last stable build \(todo#78\), 1 hr 24 min ago](#)
- [Last successful build \(todo#78\), 1 hr 24 min ago](#)

[RSS for all](#) [RSS for failures](#)

Test Result Trend

Build	Count
todo#74	7
todo#75	7
todo#76	7
todo#77	7
todo#78	7

Code Coverage Trend

Build	lineCovered	lineMissed
todo#74	~120	~250
todo#75	~120	~250
todo#76	~120	~250
todo#77	~120	~250
todo#78	~120	~250

Just show failures) enlarge

green lineCovered
red lineMissed



Parameterized Trigger Plugin

Jenkins / todo-initial

ENABLE AUTO REFRESH

[Back to Dashboard](#)

[Status](#)

[Changes](#)

[Workspace](#)

[Build Now](#)

[Delete Project](#)

[Configure](#)

[Coverage Trend](#)

[Git Polling Log](#)

Project todo-initial

[Workspace](#)

[Recent Changes](#)

[Latest Test Result \(no failures\)](#)

[add description](#)

[Disable Project](#)

Test Result Trend

count

Build	Count
todo#74	7
todo#75	7
todo#76	7
todo#77	7
todo#78	7

[\(just show failures\) enlarge](#)

Code Coverage Trend

lineCovered

lineMissed

Build	lineCovered	lineMissed
todo#74	~120	~250
todo#75	~120	~250
todo#76	~120	~250
todo#77	~120	~250
todo#78	~120	~250

Downstream Projects

todo-integ-tests

Permalinks

- [Last build \(todo#78\), 1 hr 24 min ago](#)
- [Last stable build \(todo#78\), 1 hr 24 min ago](#)
- [Last successful build \(todo#78\), 1 hr 24 min ago](#)

[RSS for all](#) [RSS for failures](#)



Gradle Plugin

Build

Invoke Gradle script



Invoke Gradle

Use Gradle Wrapper

Make gradlew executable

From Root Build Script Dir

Build step description



Switches



Tasks



Root Build script



Build File



Specify Gradle build file to run. Also, [some environment variables are available to the build script](#)



Gradle Plugin

Build

Invoke Gradle script

Invoke Gradle

Use Gradle Wrapper

Make gradlew executable

From Root Build Script Dir

Build step description

Switches

Tasks

Root Build script

Build File

ALWAYS USE THE WRAPPER!

CLEAN TASK REMOVES
EXISTING ARTIFACTS

variables are available to the



Build Name Setter Plugin

EXPRESSIVE BUILD NAME

Build History [\(trend\)](#)

- todo#78 [May 16, 2013 6:59:27 AM](#)
- todo#77 [May 16, 2013 6:56:37 AM](#)
- todo#76 [Apr 28, 2013 3:09:08 PM](#)
- todo#75 [Apr 28, 2013 11:16:37 AM](#)
- todo#74 [Apr 28, 2013 11:02:41 AM](#)

[RSS for all](#) [RSS for failures](#)

Build Environment

Set Build Name

Build Name

todo#\${BUILD_NUMBER}

JENKINS ENVIRONMENT VARIABLE



Parameterized Trigger Plugin

Trigger parameterized build on other projects

Build Triggers

Projects to build: todo-integ-tests

Trigger when build is: Stable

Trigger build without parameters:

Predefined parameters

Parameters: SOURCE_BUILD_NUMBER=\${BUILD_NUMBER}

NEXT JOB TO TRIGGER IF BUILD IS STABLE

BUILD NUMBER PARAMETER PROVIDED TO SUBSEQUENT JOBS



Clone Workspace SCM Plugin

Archive for Clone Workspace SCM

Files to include in cloned workspace ?

Files to exclude from cloned workspace ?

Criteria for build to be archived ?

Archive method ?

Override Default Ant Excludes

ARCHIVE ALL FILES

ONLY ARCHIVE IF BUILD WAS SUCCESSFUL



JaCoCo Plugin

POINT TO SEPARATED TEST RESULTS

Configure the JaCoCo plugin settings:

Publish JUnit test result report

Test report XMLs: `**/build/test-results/unit/*.xml`

Fileset 'includes' setting that specifies the generated raw XML report files, such as 'myproject/target/test-reports/junit'.
 Retain long standard output/error

Record JaCoCo coverage report

Path to exec files (e.g.: `**/target/**.exec, **/jacoco.exec`): `**/build/jacoco/test.exec`

Path to class directories (e.g.: `**/target/classDir, **/classes`): `**/build/classes`

Path to source directories (e.g.: `**/mySourceFiles`): `**/src/main/java`

Inclusions (e.g.: `**/*.class`)

Exclusions (e.g.: `**/*Test*`)

Instruction	% Branch	% Complexity	% Line	% Method	% Class
	100	70	70	70	70
	0	0	0	0	0

POINT TO JACOCO FILES AS WELL AS SOURCE AND CLASS FILES

FAIL BUILD IF QUALITY GATE CRITERIA ARE NOT MET



Clone Workspace SCM Plugin



Build Name Setter Plugin

Source Code Management

CVS
 Clone Workspace
Parent Project 

Criteria for parent build 

Git
 None
 Subversion

Build Environment

Set Build Name 

Build Name 

REUSE INITIAL WORKSPACE

REUSE INITIAL BUILD NUMBER



Build Pipeline Plugin

Invoke Gradle script

Invoke Gradle

Use Gradle Wrapper

Make gradlew executable

From Root Build Script Dir

Build step description

▼

DEFINE THE TARGET ENVIRONMENT

Switches

-Penv=uat

▼



Tasks

cargoDeployRemote

▼



Root Build script

?

Build File

?

Post-build Actions

Build Pipeline Plugin -> Manually Execute Downstream Project



Downstream Project Names

todo-deploy-production

DOWNSTREAM JOB THAT REQUIRES MANUAL EXECUTION



Build Pipeline Plugin

Build Pipeline: To Do application



VISUALIZATION OF CHAINED PIPELINE JOBS

```
> gradle qa  
:askQuestions
```

BUILD SUCCESSFUL

Total time: 300 secs