

# Java Power Tools

## *Getting it all together*

John Ferguson Smart  
Principle Consultant  
Wakaleo Consulting  
Email: [john.smart@wakaleo.com](mailto:john.smart@wakaleo.com)  
Web: <http://www.wakaleo.com>  
Twitter: wakaleo

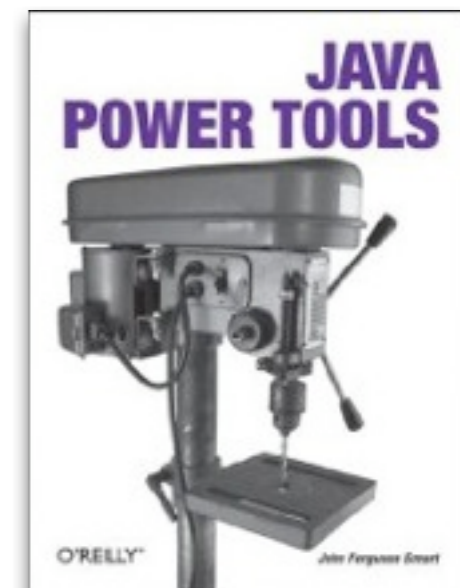


# Presentation Goals

Learn about the latest evolutions in Java development tools. In particular, learn how to improve and automate your development process using tools like Maven, Hudson, and many others.

# Speaker's qualifications

- ▶ John Ferguson Smart
  - ▶ Consultant, Trainer, Mentor, Author,...
  - ▶ Works with Enterprise Java, Web Development, and Open Source technologies
  - ▶ Author of 'Java Power Tools' (O'Reilly)
  - ▶ Writes articles for sites like JavaWorld, DevX and Java.net, and blogs on Java.net
  - ▶ Frequent speaker at conferences and Java User Groups
  - ▶ Likes to write about himself in the third person



# Agenda

- ▶ What we will cover today:
  - ▶ Industrializing your build process
  - ▶ Organizing your internal artifacts
  - ▶ Improving your release management strategy
  - ▶ Automate the build process
  - ▶ Better testing practices
  - ▶ Monitoring code coverage and code quality metrics



**HERE** is Edward Bear, coming downstairs now, bump, bump, bump, on the back of his head, behind Christopher Robin. It is, as far as he knows, the only way of coming downstairs, but sometimes he feels that there really is another way, if only he could stop bumping for a moment and think of it.

-- A. A. Milne

# Towards a better build process

## ▶ Industrializing your build process

### ▶ Why bother?

- ☑ Each team and each project using different conventions?
- ☑ Build scripts ad-hoc and difficult to understand and maintain?
- ☑ Little reuse of components between teams and projects?
- ☑ Code quality metrics and reporting are not done systematically, or not at all?
- ☑ High learning curve and maintenance costs?

### ▶ What can you do?

- ☑ Set up a development infrastructure: e.g. *Maven 2, Nexus, Hudson...*

# Towards a better build process

## ► Build Automation - the big picture



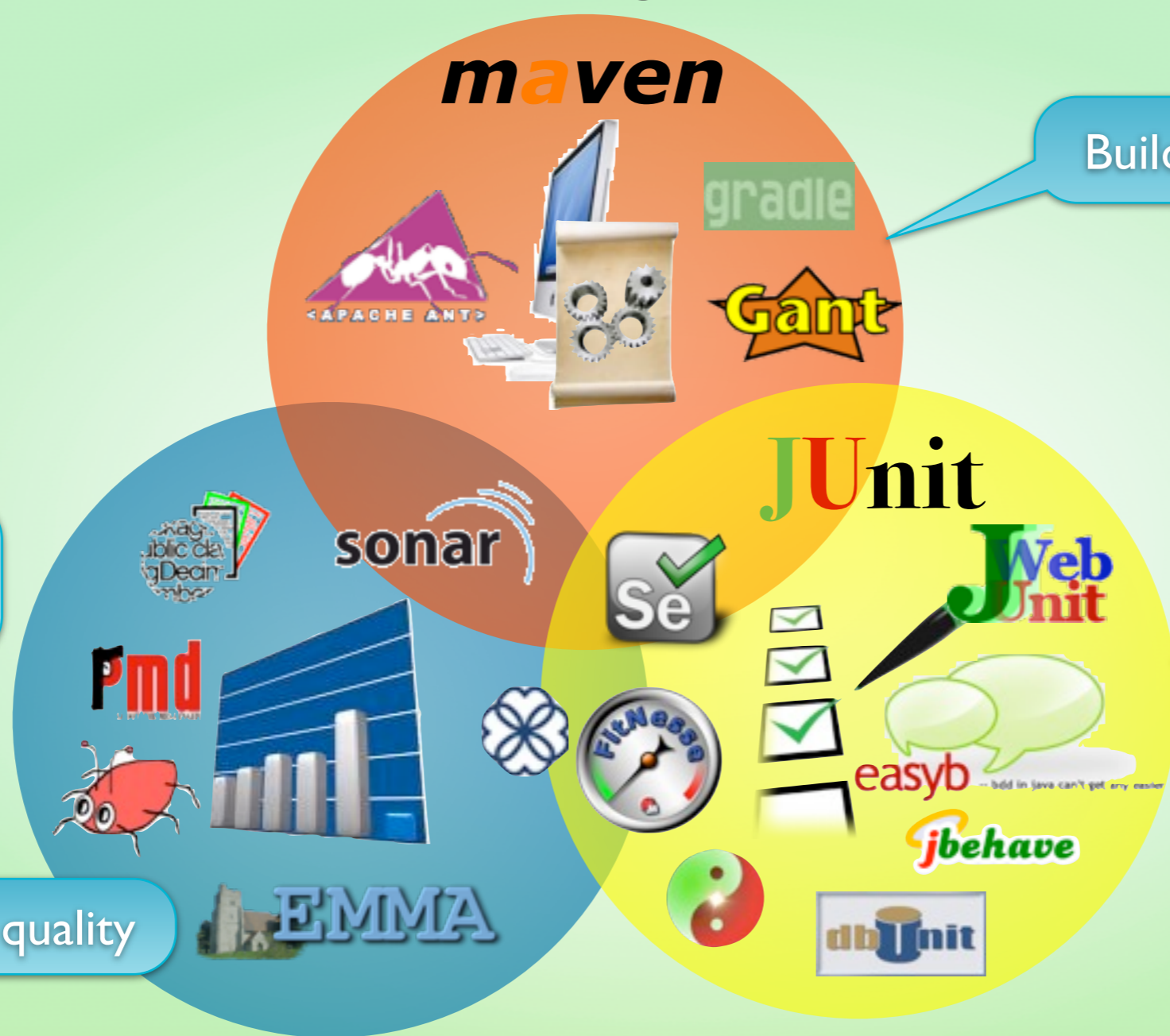
CI build server

Continuous Integration

Automated code quality

Build scripting

Automated testing



# Towards a better build process

## ▶ Maven as a standardization tool

### ▶ More than just your average build scripting tool!

- ✓ A build tool
- ✓ A dependency management tool
- ✓ A documentation generation tool
- ✓ A metrics/code quality reporting tool
- ✓ A project management tool
- ✓ And more...

A screenshot of a web page for a Maven project. It features a sidebar with a 'Main Menu' and 'Project Documentation' sections. The main content area is titled 'Project Information' and includes an 'Overview' table with columns for 'Document' and 'Description'.

Document	Description
Continuous Integration	This is a link to the definitions of all continuous integration processes that builds and tests code on a frequent, regular basis.
Dependencies	This document lists the projects dependencies and provides information on each dependency.
Issue Tracking	This is a link to the issue management system for this project. Issues (bugs, features, change requests) can be created and queried using this link.
Mailing Lists	This document provides subscription and archive information for this project's mailing lists.
Project License	This is a link to the definitions of project licenses.

Two code snippets. The first is for 'TaxCalculatorImpl' showing methods like 'getTaxRates()', 'calculateIncomeTax()', and 'calculateGST()'. The second is for 'TaxRate' showing methods like 'getMinimumRevenue()', 'getMaximumRevenue()', 'getRate()', and 'calculateTax()'.

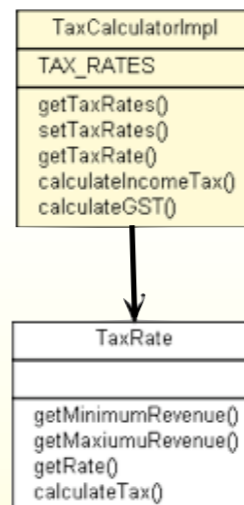
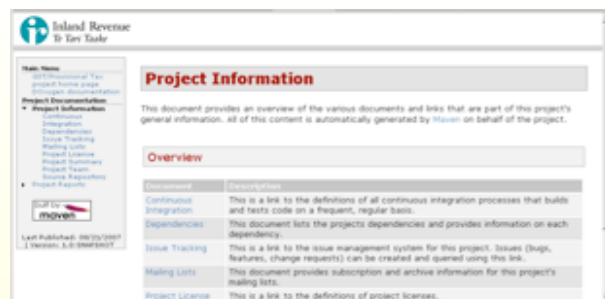
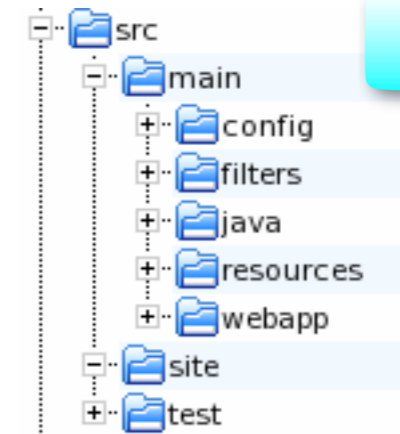
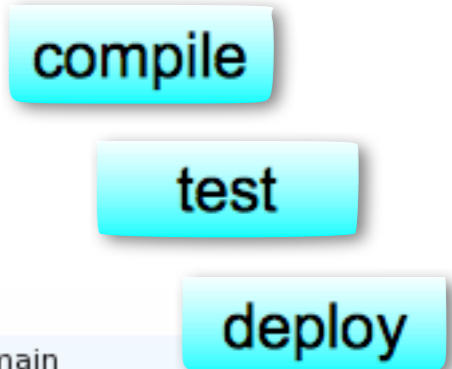
```
TaxCalculatorImpl
TAX_RATES
getTaxRates()
getTaxRates()
getTaxRate()
calculateIncomeTax()
calculateGST()

TaxRate
getMinimumRevenue()
getMaximumRevenue()
getRate()
calculateTax()
```

# Towards a better build process

## ► So what can Maven do for me?

- ✓ Standardize your build and deployment process
- ✓ A standard, but extensible, build lifecycle
- ✓ A standard directory structure
- ✓ Clear and clean dependency management
- ✓ Good technical documentation and reporting

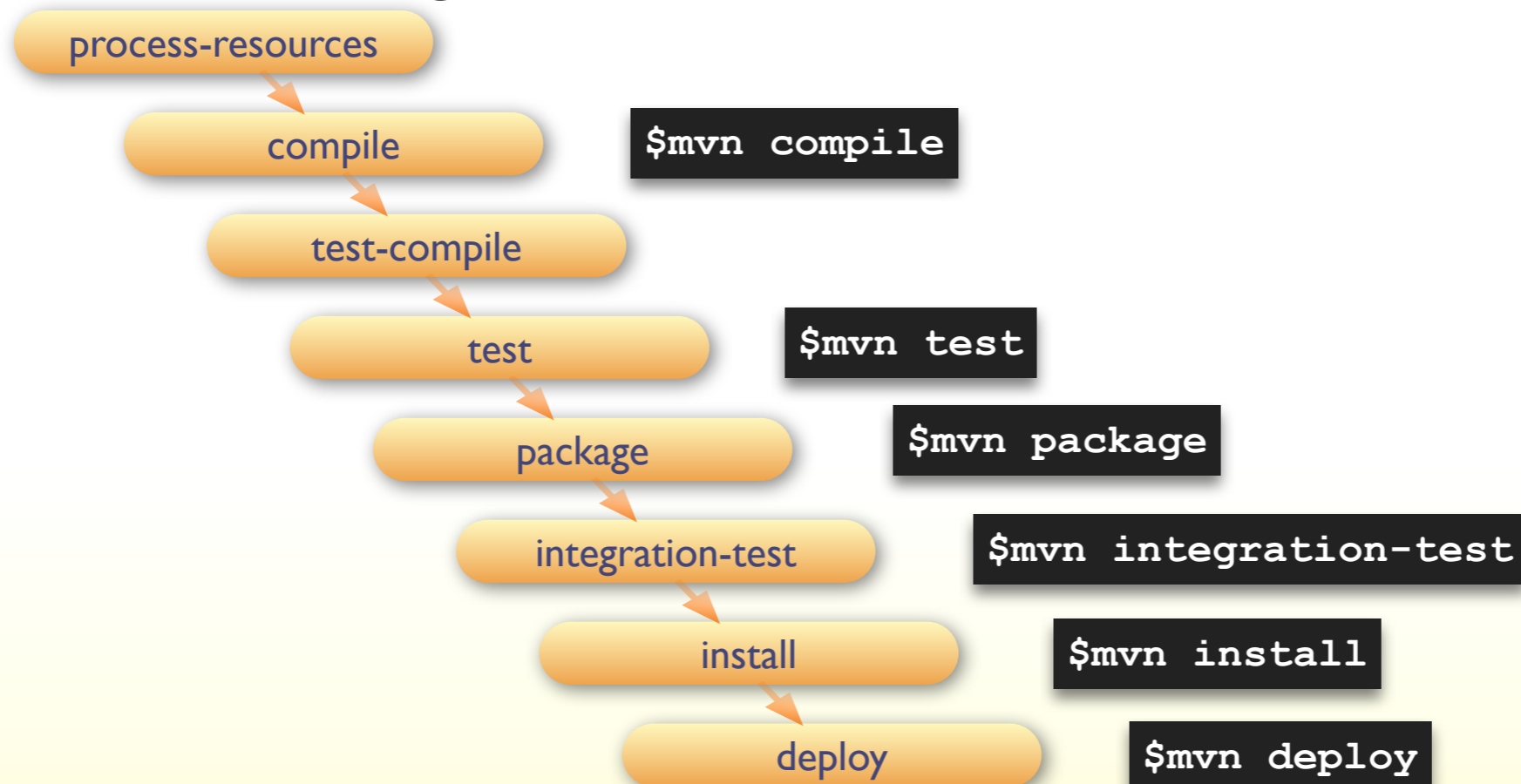
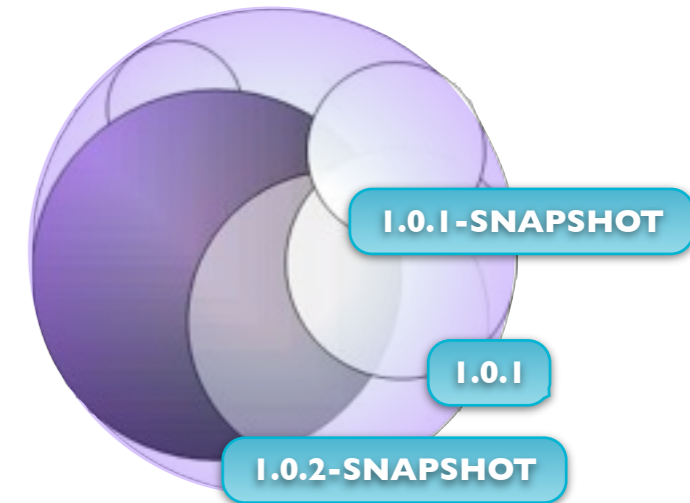


# Towards a better build process

## ▶ Maven 2 Highlights

### ▶ A Standard Life Cycle

- ✓ A standardized approach to building your software
- ✓ Familiar commands across all projects
- ✓ No re-inventing the wheel

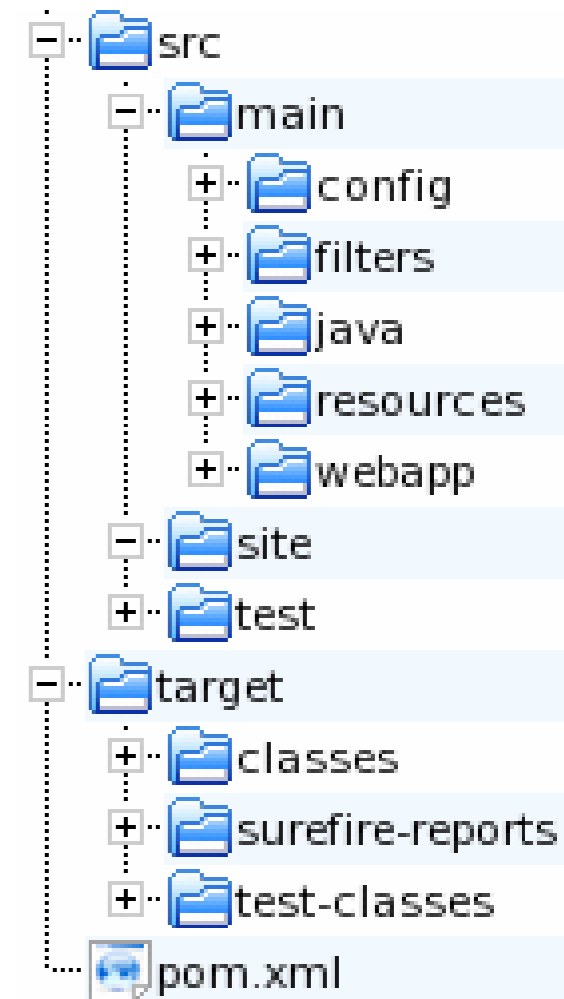


# Towards a better build process

## ▶ Maven 2 Highlights

### ▶ A Standard Directory Structure

- ✓ Familiar structure across all projects
- ✓ Lower learning curve
- ✓ Convention Over Configuration
- ✓ Less low-level scripting



# Towards a better build process

## ▶ Maven 2 Highlights

### ▶ A Standard Way of Identifying Artifacts

- ☑ Each Maven artifact has a unique identifier, or “coordinates”



```
<project...>
...
<groupId>com.mycompany.accounting</groupId>
<artifactId>accounting-core</artifactId>
<packaging>jar</packaging>
<version>1.1</version>
<name>Accounting Core package</name>
...
```

Project group name

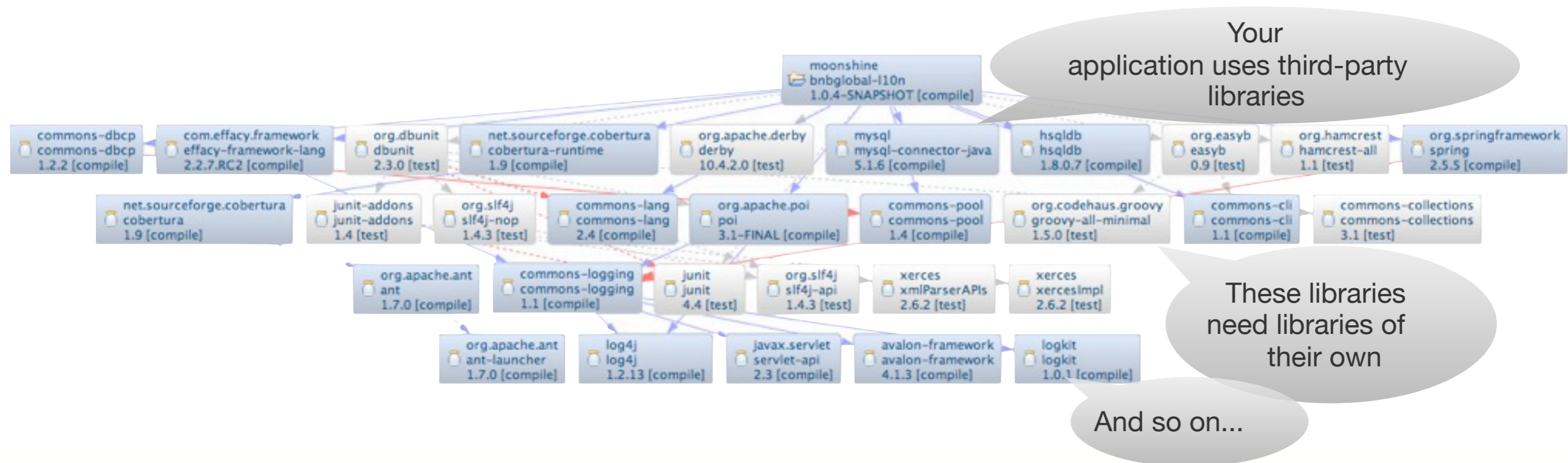
Artifact name

Version number

`com/mycompany/accounting/accounting-core/1.1/accounting-core-1.1.jar`

# Towards a better build process

- ▶ Maven 2 Highlights
- ▶ Dependency Management



# Towards a better build process

## ▶ Maven 2 Highlights

### ▶ Traditional Dependency Management

- ☑ Each project has its own set of JAR files
- ☑ Unnecessary duplication
- ☑ Hard to keep track of versions
- ☑ Errors due to incompatible JAR files
- ☑ Overloads the source code repository

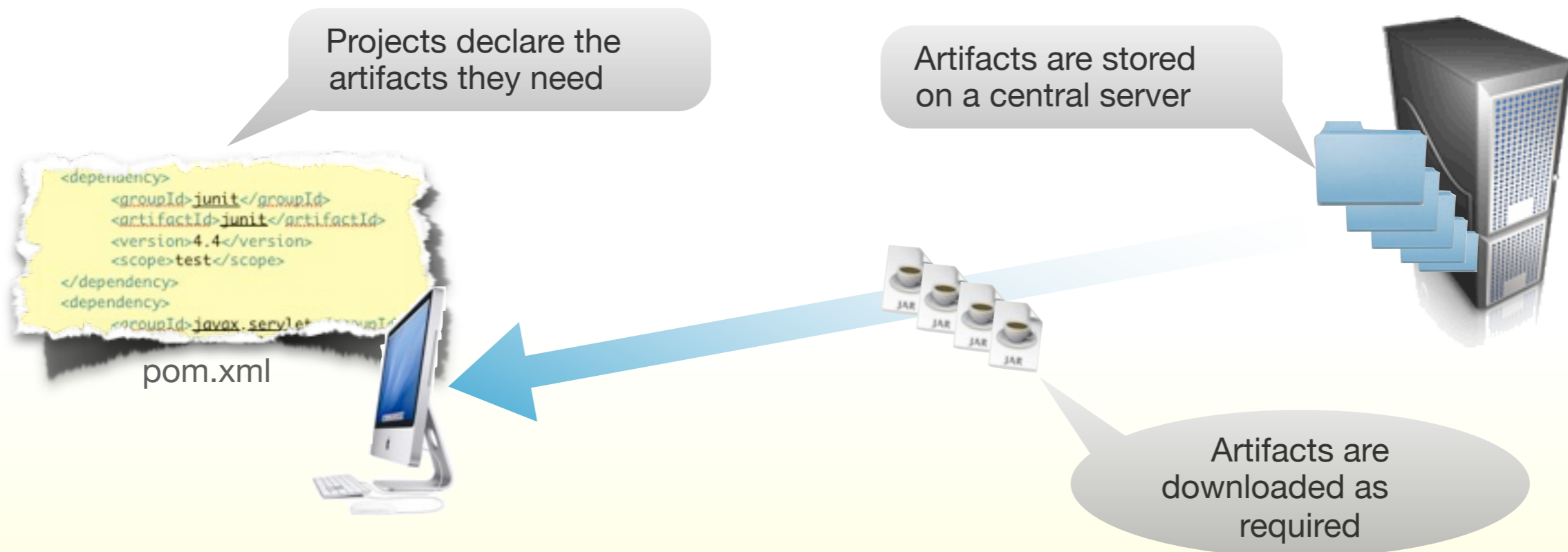


# Towards a better build process

## ▶ Maven 2 Highlights

### ▶ Declarative Dependency Management

- ✓ Artifact versions are stored on a central server
- ✓ Each project “declares” what libraries and versions it needs
- ✓ All the required dependencies are automatically downloaded



# Towards a better build process

## ▶ Maven 2 Highlights

### ▶ Declaring Dependencies in Maven

- ☑ Declared in the build script itself
- ☑ Dependencies with version numbers
- ☑ Different types of dependencies
- ☑ Compile, test, provided,...

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.4</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>2.4</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>2.5.6</version>
  </dependency>
  ...
</dependencies>
```

Version numbers

For tests only

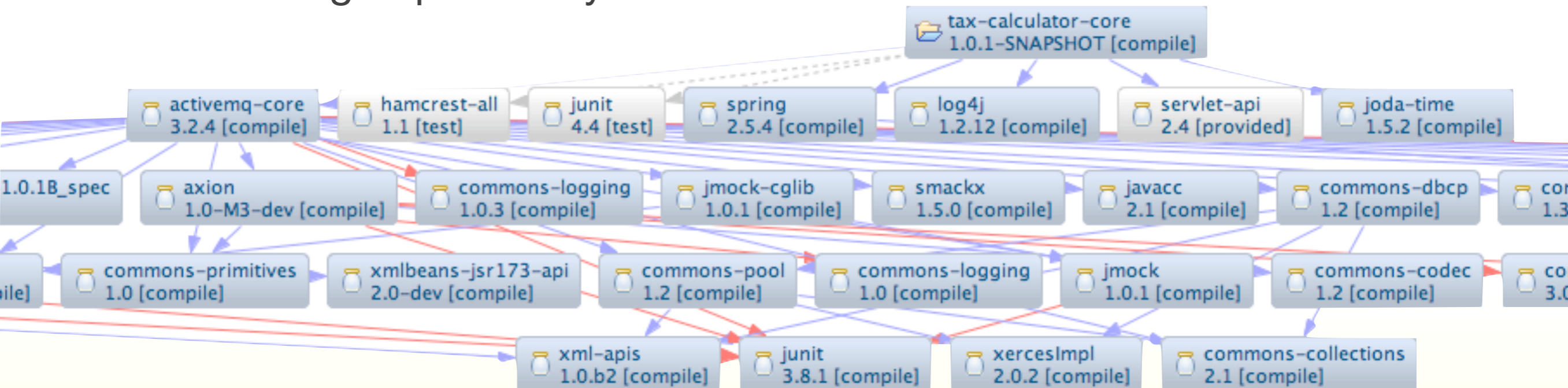
Provided by the application server

# Towards a better build process

## ▶ Managing Maven Dependencies

### ▶ It's easier with Eclipse...m2eclipse

- ✓ Adding new dependencies
- ✓ Visualizing dependencies
- ✓ Handling dependency conflict



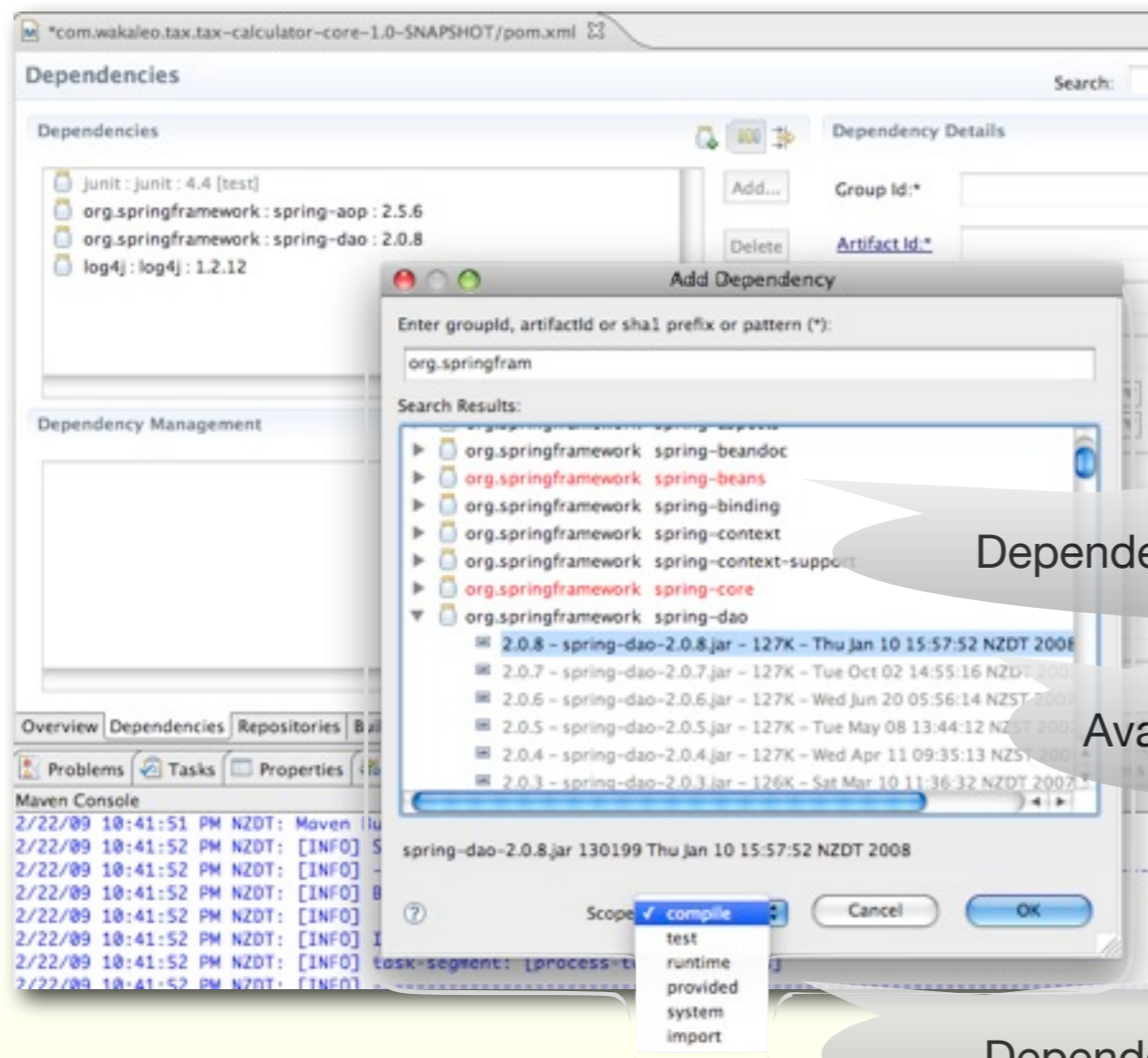
# Towards a better build process

- ▶ Managing Maven Dependencies
- ▶ Viewing Maven project dependencies



# Towards a better build process

- ▶ Managing Maven Dependencies
  - ▶ Adding new dependencies...



Dependencies already present in your project

Available versions

Dependency scopes

# Towards a better build process

- ▶ Managing Maven Dependencies
- ▶ Visualizing the Dependency Hierarchy



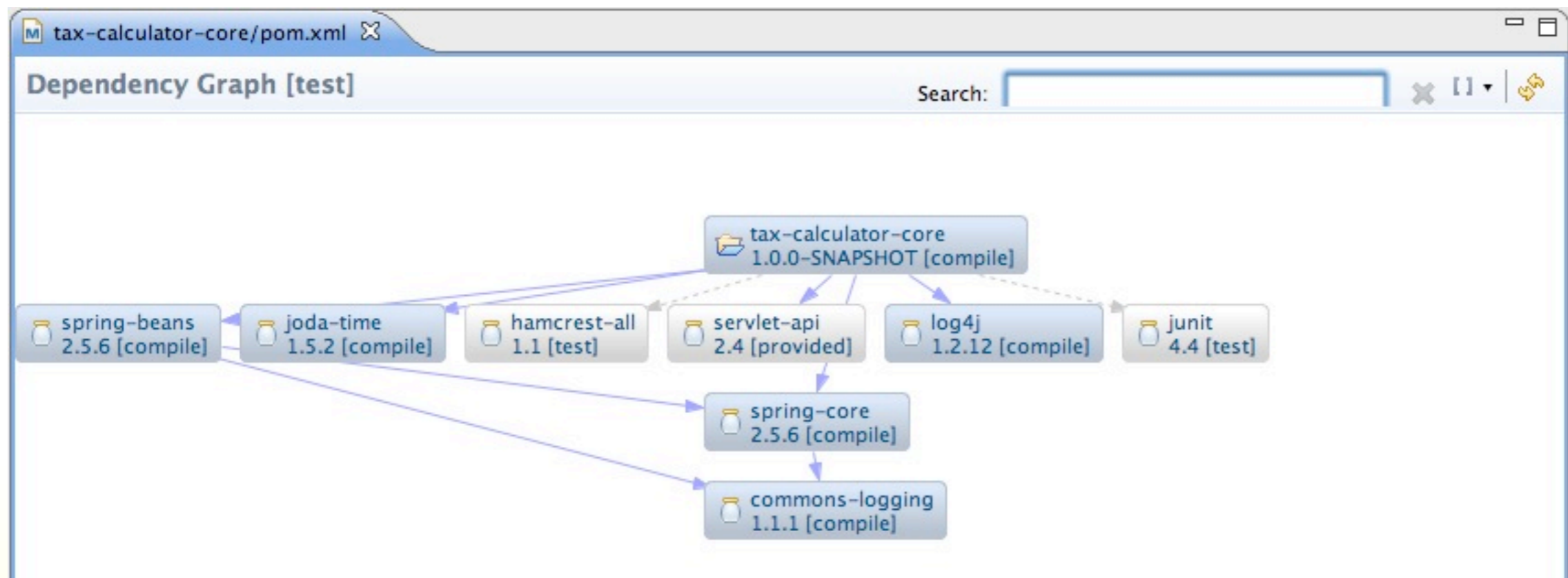
A screenshot of the Eclipse IDE showing the Maven Dependency Hierarchy and Resolved Dependencies. The Dependency Hierarchy on the left shows a tree structure of dependencies, with two instances of 'commons-logging : 1.1.1 [compile]' circled in red and labeled as conflicting. A red double-headed arrow with a question mark points to these conflicting entries. The Resolved Dependencies on the right shows a list of resolved dependencies, with 'commons-logging : 1.1.1 [compile]' circled in green. A green arrow points from the resolved dependency to the conflicting entries in the hierarchy. A search bar is visible at the top of the Resolved Dependencies panel.

Resolved dependencies

Conflicting dependencies

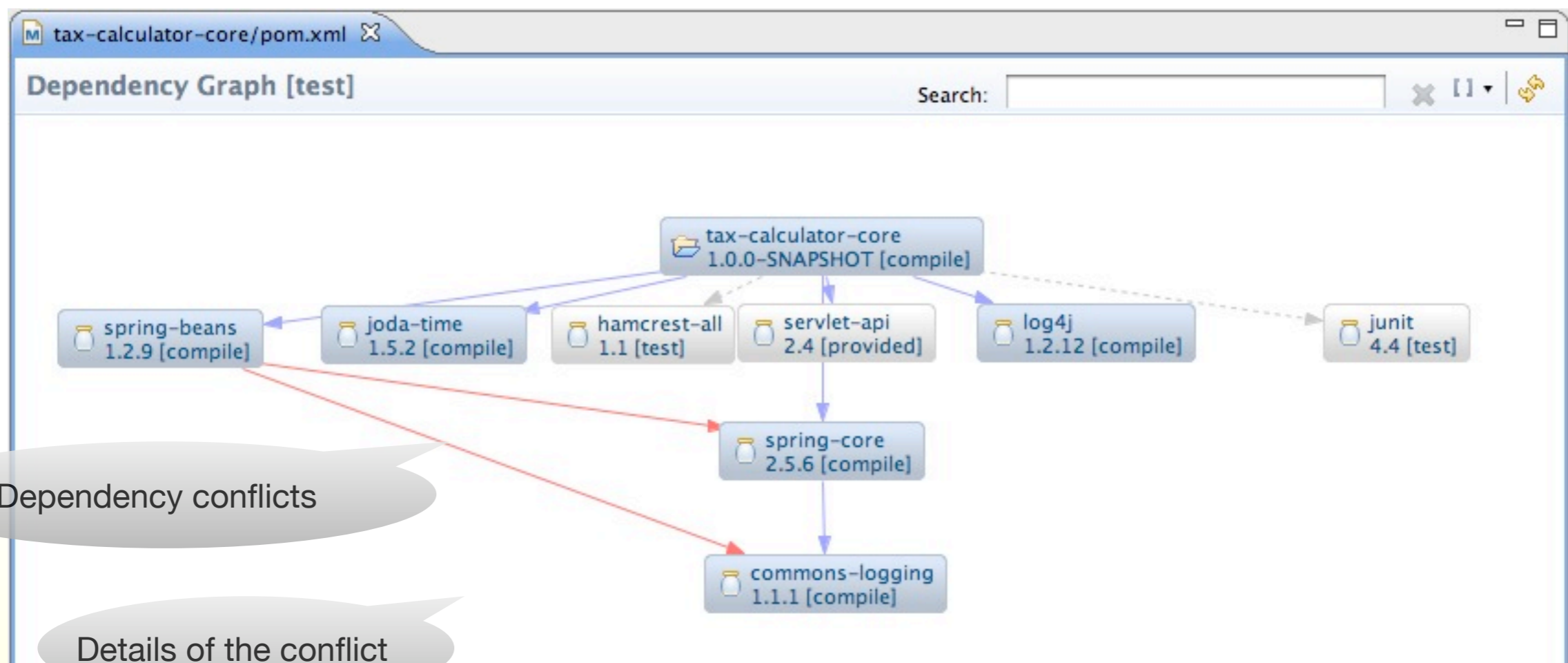
# Towards a better build process

- ▶ Managing Maven Dependencies
  - ▶ Visualizing the Dependency Graph



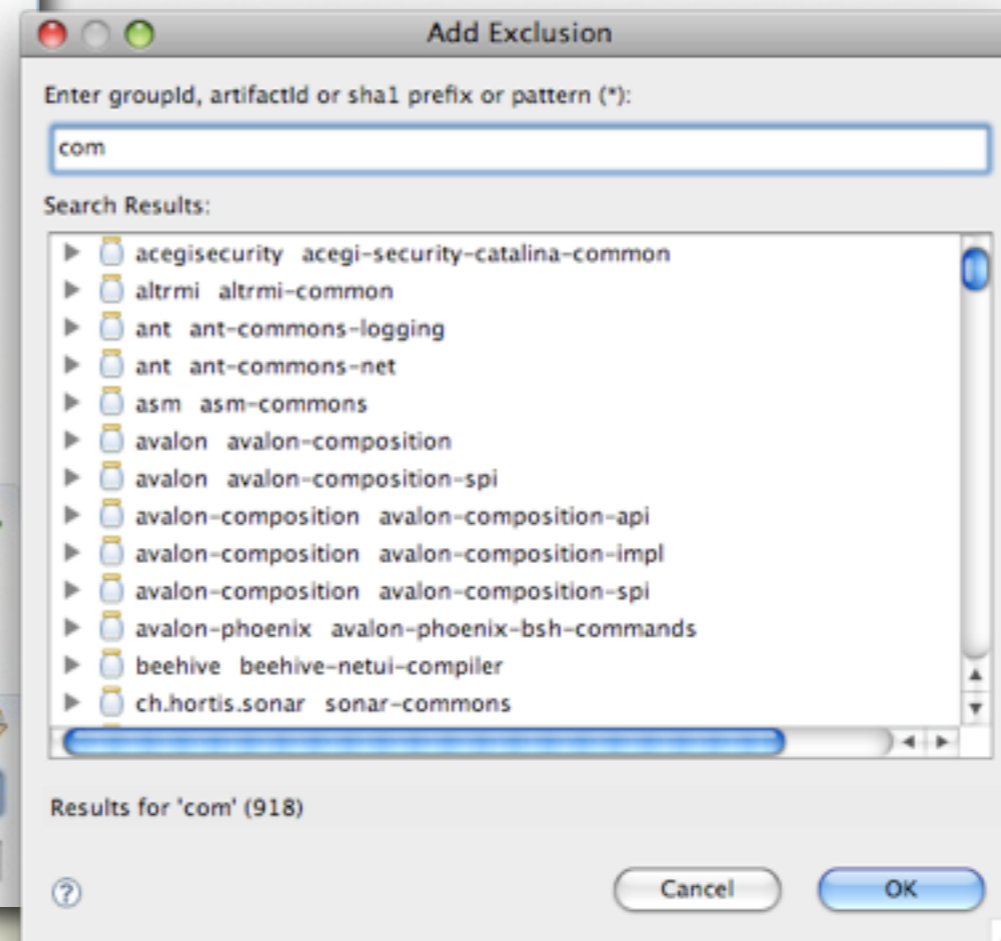
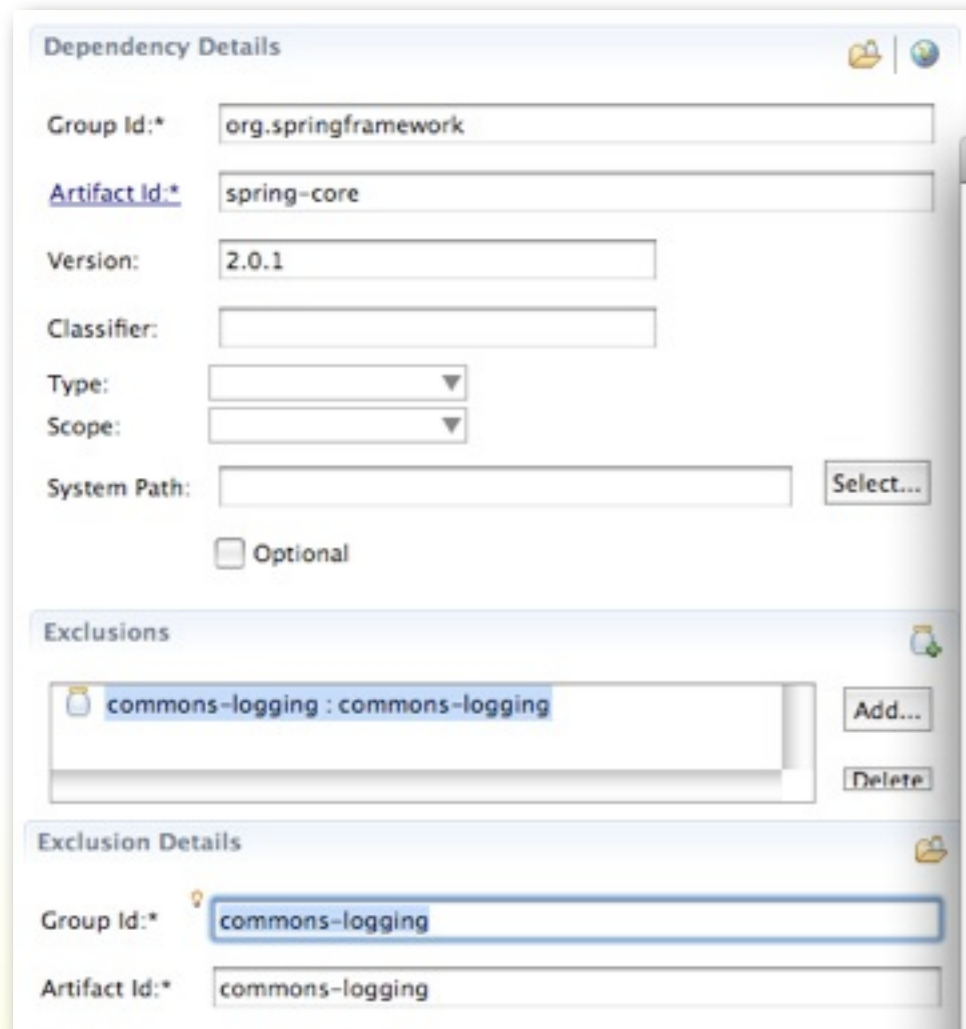
# Towards a better build process

- ▶ Managing Maven Dependencies
- ▶ Visualizing conflicts in the Dependency Graph



# Towards a better build process

- ▶ Managing Maven Dependencies
- ▶ Excluding Dependencies



# DEMO

A walk through a simple Maven 2 project

# Organizing your artifacts

- ▶ Organizing your artifacts
  - ▶ Internal releases are hard to co-ordinate
    - ☑ How do I share my API with other teams?
    - ☑ Where is the latest version of that API?
    - ☑ What version am I using, anyway?
  - ▶ What can you do?
    - ▶ Use an Enterprise Maven Repository



# Organizing your artifacts

- ▶ Organizing your artifacts
  - ▶ Maven Repositories
    - ☑ JAR files are downloaded to a local cache

# Organizing your artifacts

## ▶ Organizing your artifacts

### ▶ Maven Repositories

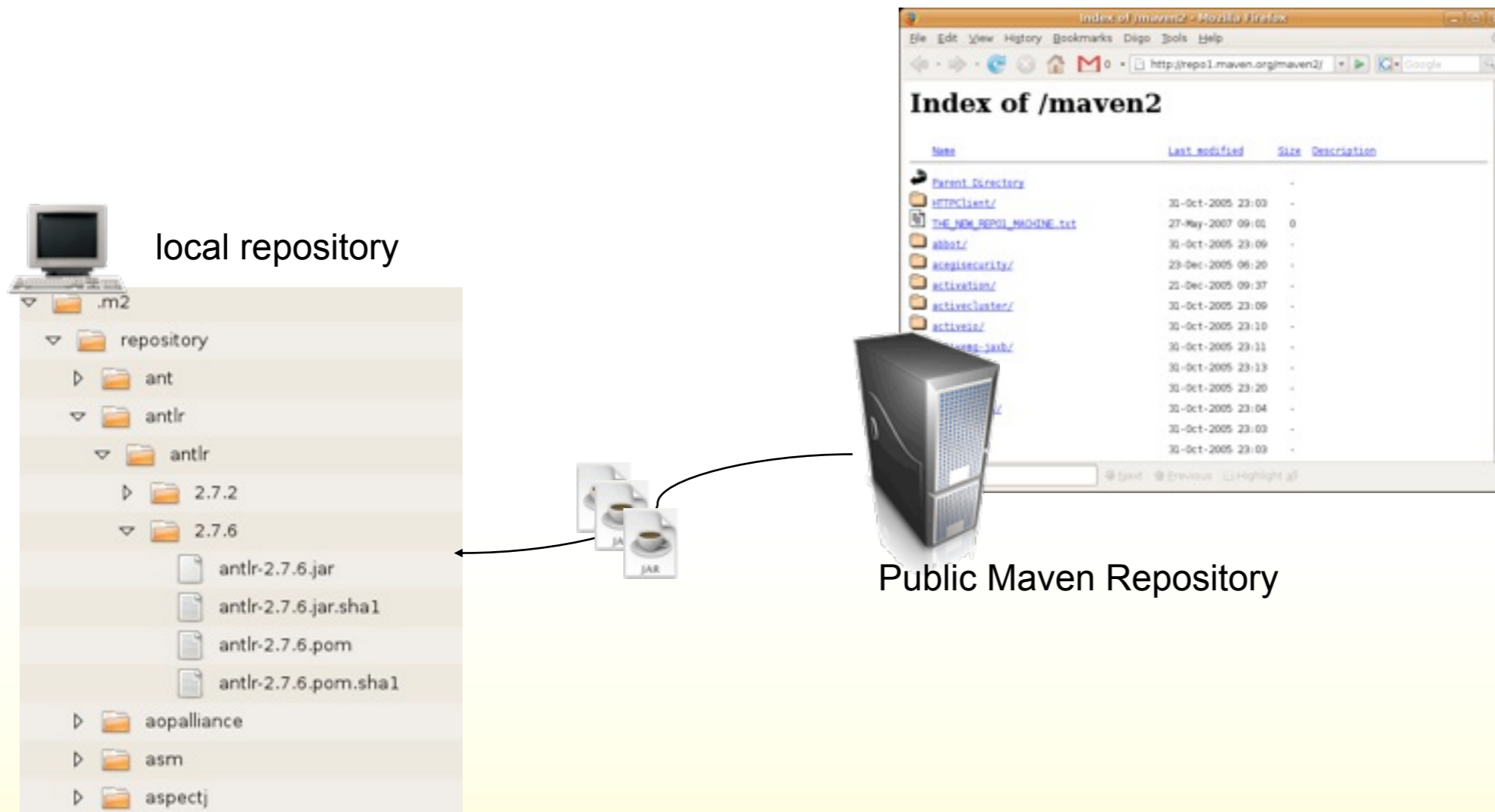
- ☑ Public web sites containing JARs for many open source projects
- ☑ Maven automatically downloads the JARs you need onto your local machine
- ☑ You can publish internal APIs on your own repository

# Organizing your artifacts

## ▶ Organizing your artifacts

### ▶ Maven Repositories

- JAR files are downloaded as required into a local cache

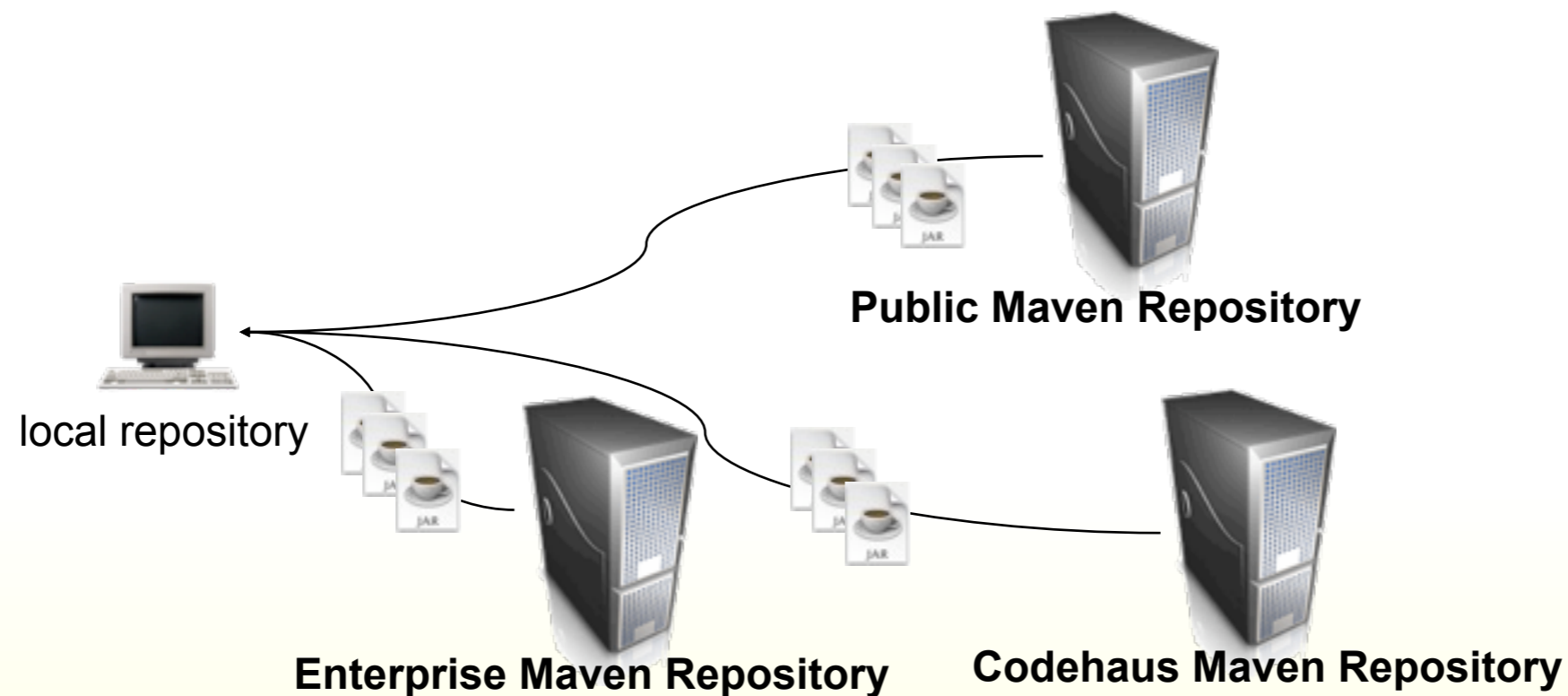


# Organizing your artifacts

## ► Organizing your artifacts

### ► Maven Repositories

- ☑ There are several public Maven repositories
- ☑ You can also share internal JARs in an Enterprise Maven Repository

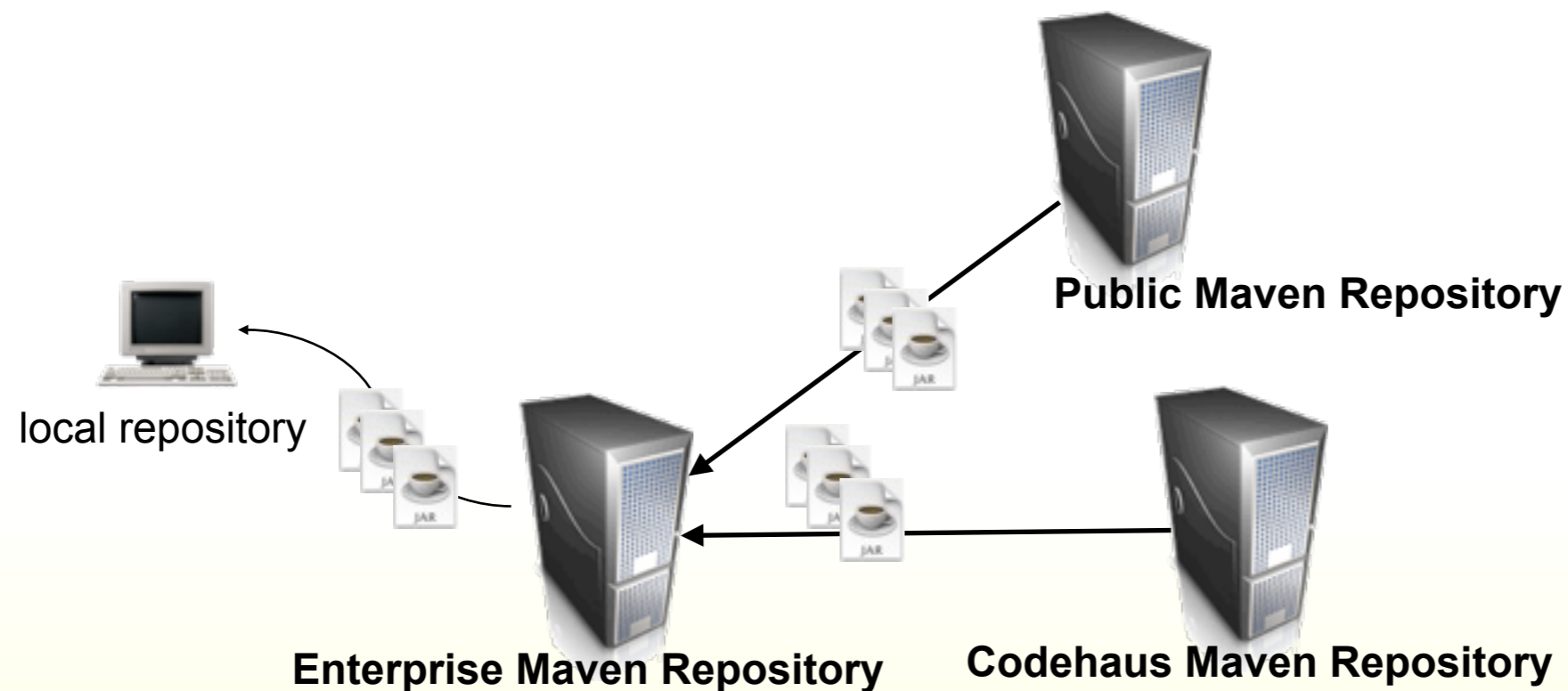


# Organizing your artifacts

## ▶ Organizing your artifacts

### ▶ Maven Repositories

- ☑ The Enterprise Maven Repository can also act as a proxy/cache to the public repositories



# Organizing your artifacts

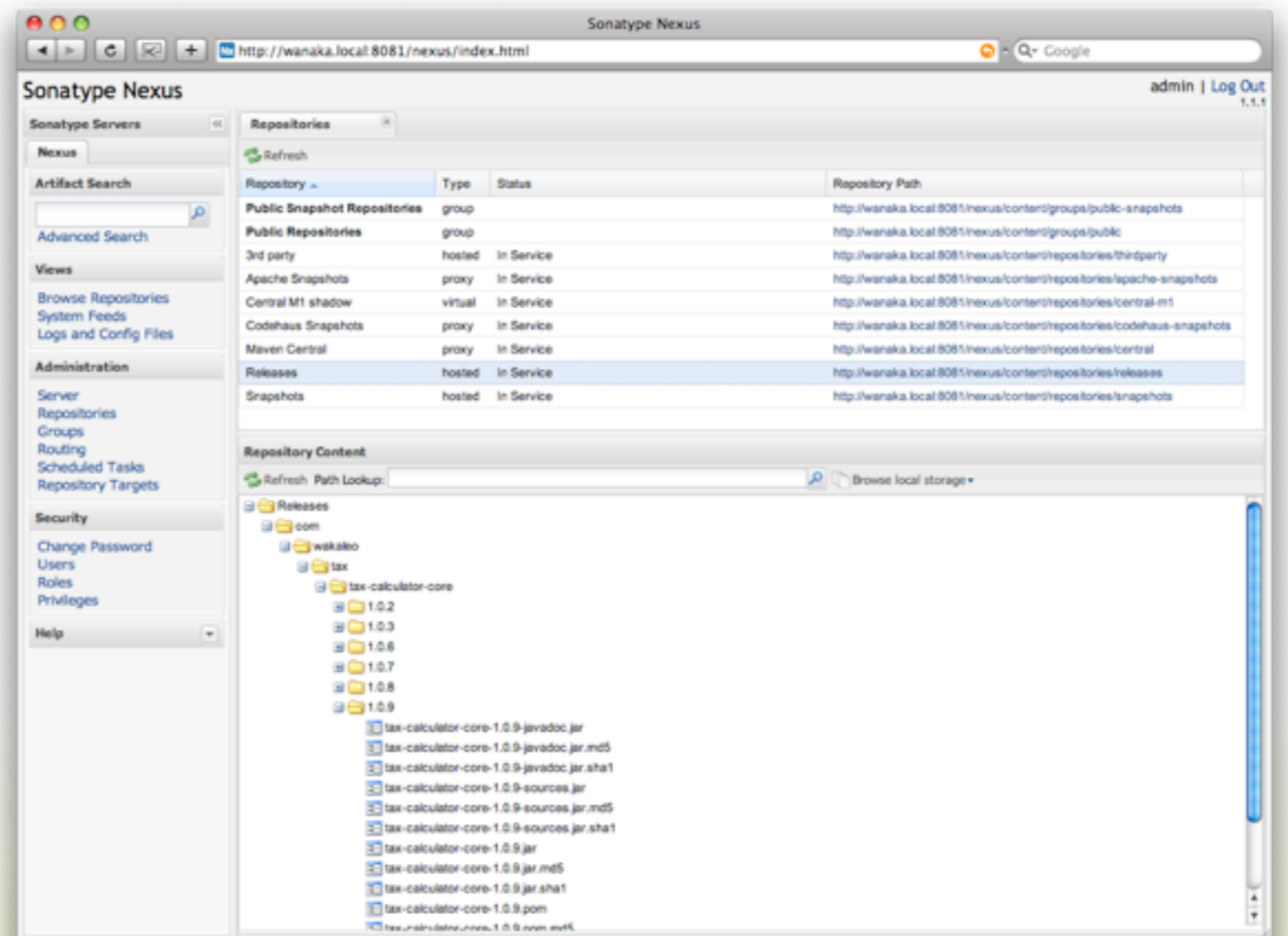
## ▶ Organizing your artifacts

### ▶ Key Best Practices

- ☑ Deploy your internal releases to a local Enterprise repository
- ☑ Store proprietary JAR files you need here as well
- ☑ Distinguish between snapshot and release versions

# Organizing your artifacts

- ▶ Organizing your artifacts
  - ▶ Using the Nexus repository manager
    - ☑ Easy to set up
    - ☑ Easy to administer
    - ☑ Proxy/cache



# DEMO

An Enterprise Repository

# Better Release Management

## ▶ Working with snapshots and releases

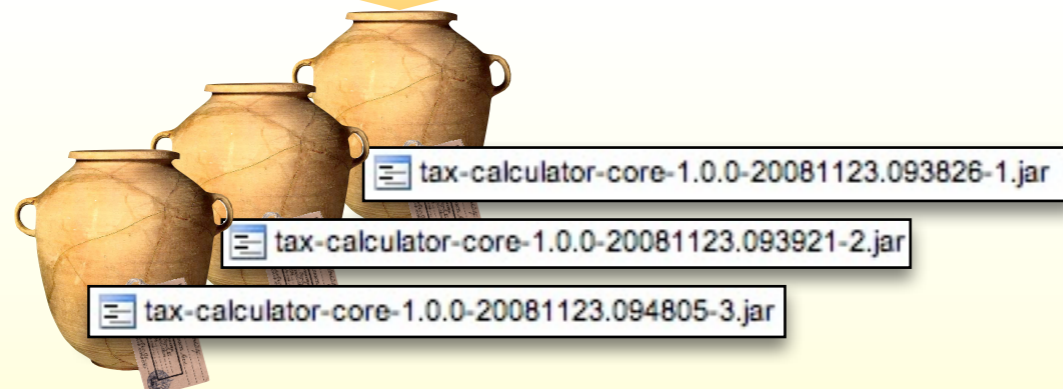
### ▶ Snapshots

- ☑ Work in progress
- ☑ A new time-stamped version deployed with each deployment



```
<artifactId>tax-calculator-core</artifactId>  
<packaging>jar</packaging>  
<version>1.0.0-SNAPSHOT</version>
```

\$mvn deploy



# Better Release Management

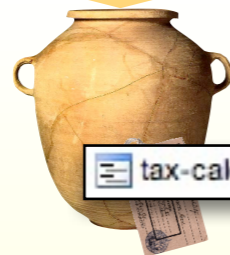
## ▶ Working with snapshots and releases

### ▶ Releases

- ☑ Stable, tested release
- ☑ The deployed artifact is unique

```
<artifactId>tax-calculator-core</artifactId>  
<packaging>jar</packaging>  
<version>1.0.0</version>
```

\$mvn deploy



tax-calculator-core-1.0.0-20081123.093826-1.jar



# Better Release Management

- ▶ Working with snapshots and releases
- ▶ Setting it all up in Maven
  - ☑ To deploy a snapshot version, you need:
    - ▶ A Snapshot repository
    - ▶ A SNAPSHOT version number
    - ▶ User authentication for the repository
    - ▶ A Snapshot configuration for the repository



```
<version>1.0.1-SNAPSHOT</version>
```

```
<settings>  
  <servers>  
    <server>  
      <id>wakaleo-snapshots</id>  
      <username>johns</username>  
      <password>secret</password>  
    </server>  
  </servers>  
</settings>
```

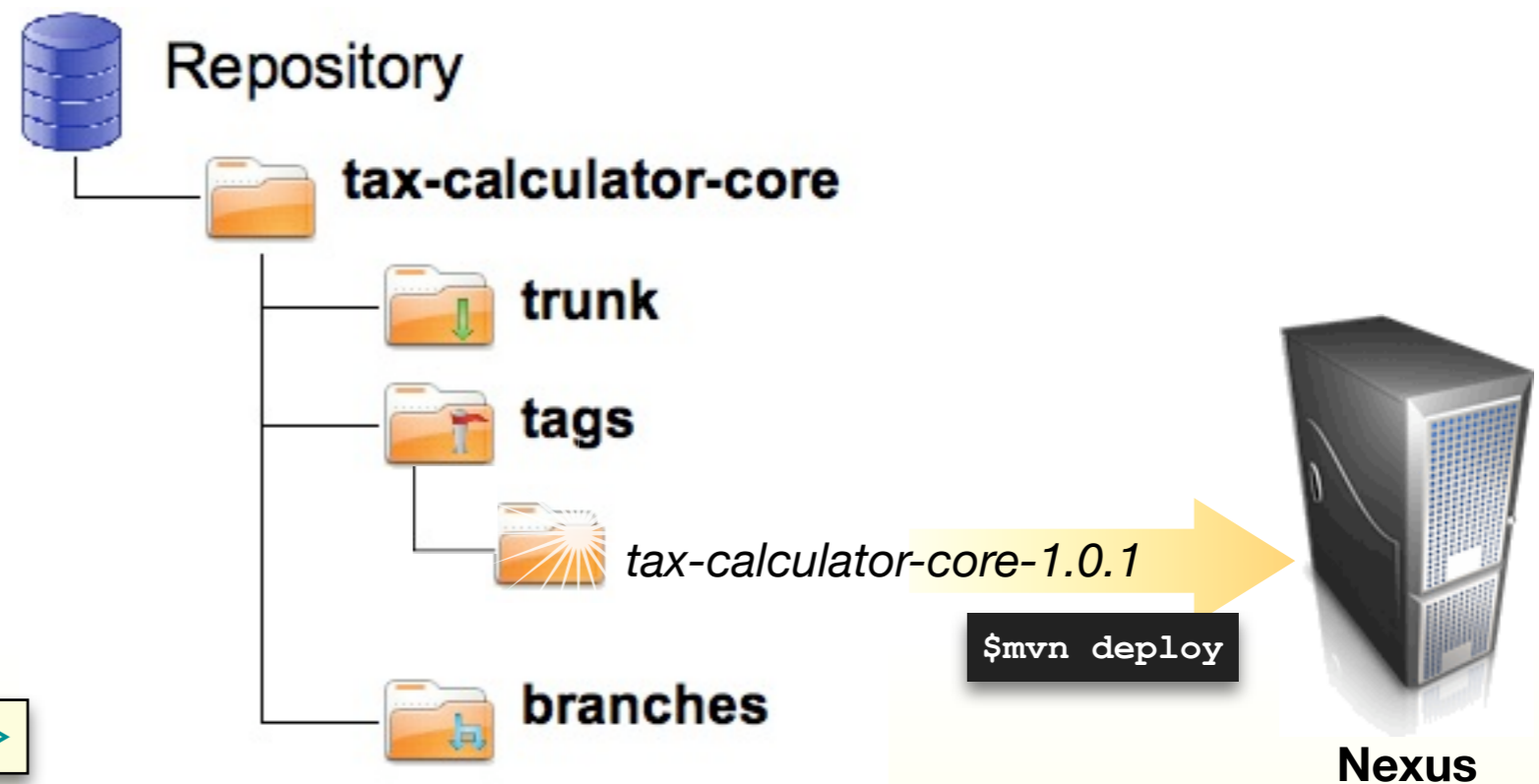
```
<distributionManagement>  
  <snapshotRepository>  
    <id>wakaleo-snapshots</id>  
    <name>Internal Snapshots</name>  
    <url>http://wanaka:8081/nexus/content/repositories/snapshots</url>  
  </snapshotRepository>  
  ...  
</distributionManagement>
```

# Better Release Management

- ▶ Working with snapshots and releases
- ▶ Automating snapshot deployments
  - ☑ Use the maven-release-plugin to automate SCM book-keeping
    - ▶ mvn:prepare
    - ▶ mvn:perform
    - ▶ mvn:rollback

pom.xml

```
<version>1.0.1-SNAPSHOT</version>  
<version>1.0.1</version>  
<version>1.0.2-SNAPSHOT</version>
```



# DEMO

## Releases, Snapshots and Repositories

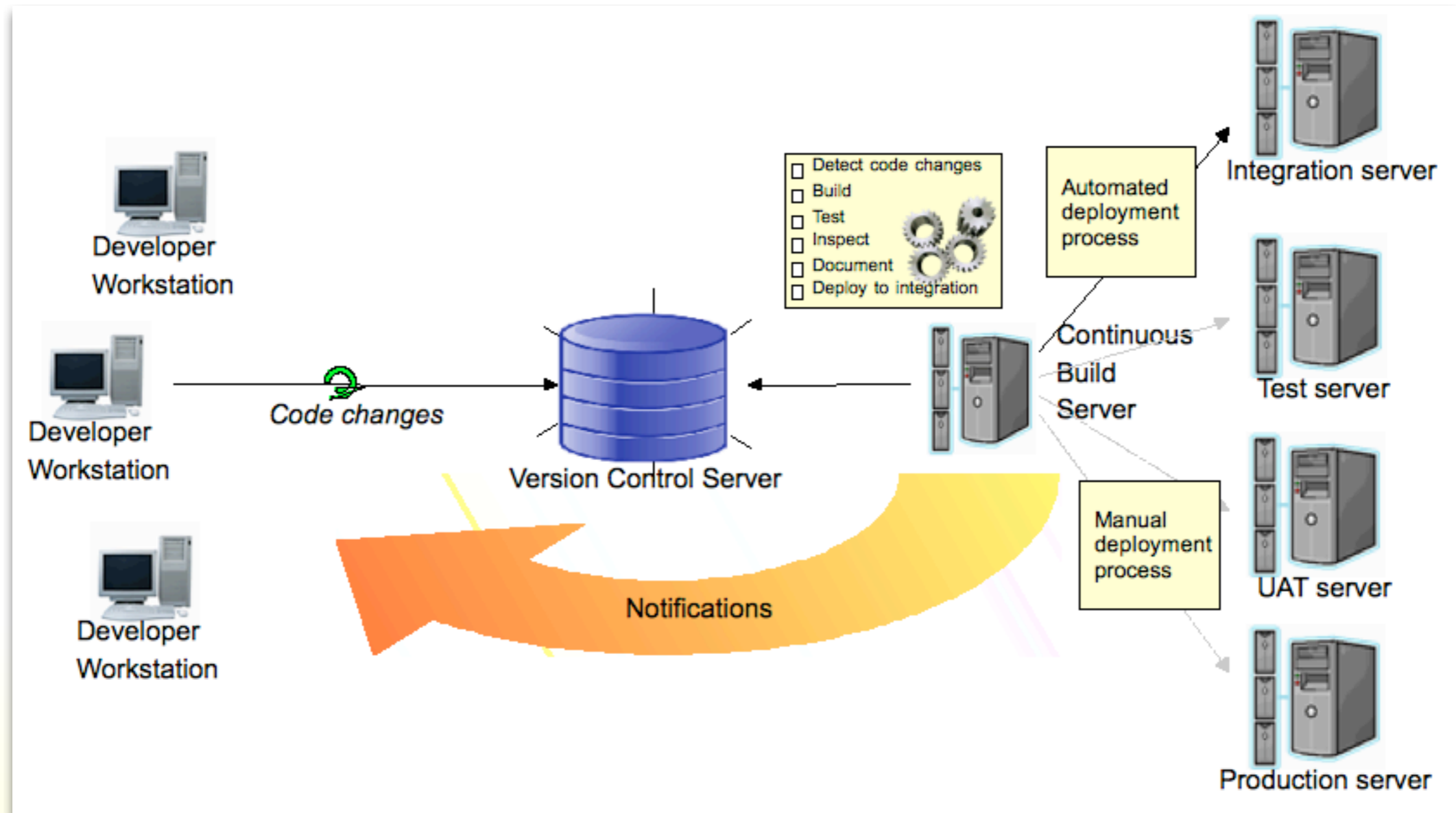
# Automating the build process

- ▶ Continuous Integration - what's the issue?
- ▶ Traditional development cycles are bad for your health:
  - ☑ Integration is long and difficult
  - ☑ Poor visibility on development progress
  - ☑ Functional tests are done too late
  - ☑ Raised issues are harder to fix
  - ☑ The client gets a sub-optimal product



# Automating the build process

## ▶ Continuous Integration - what's involved?

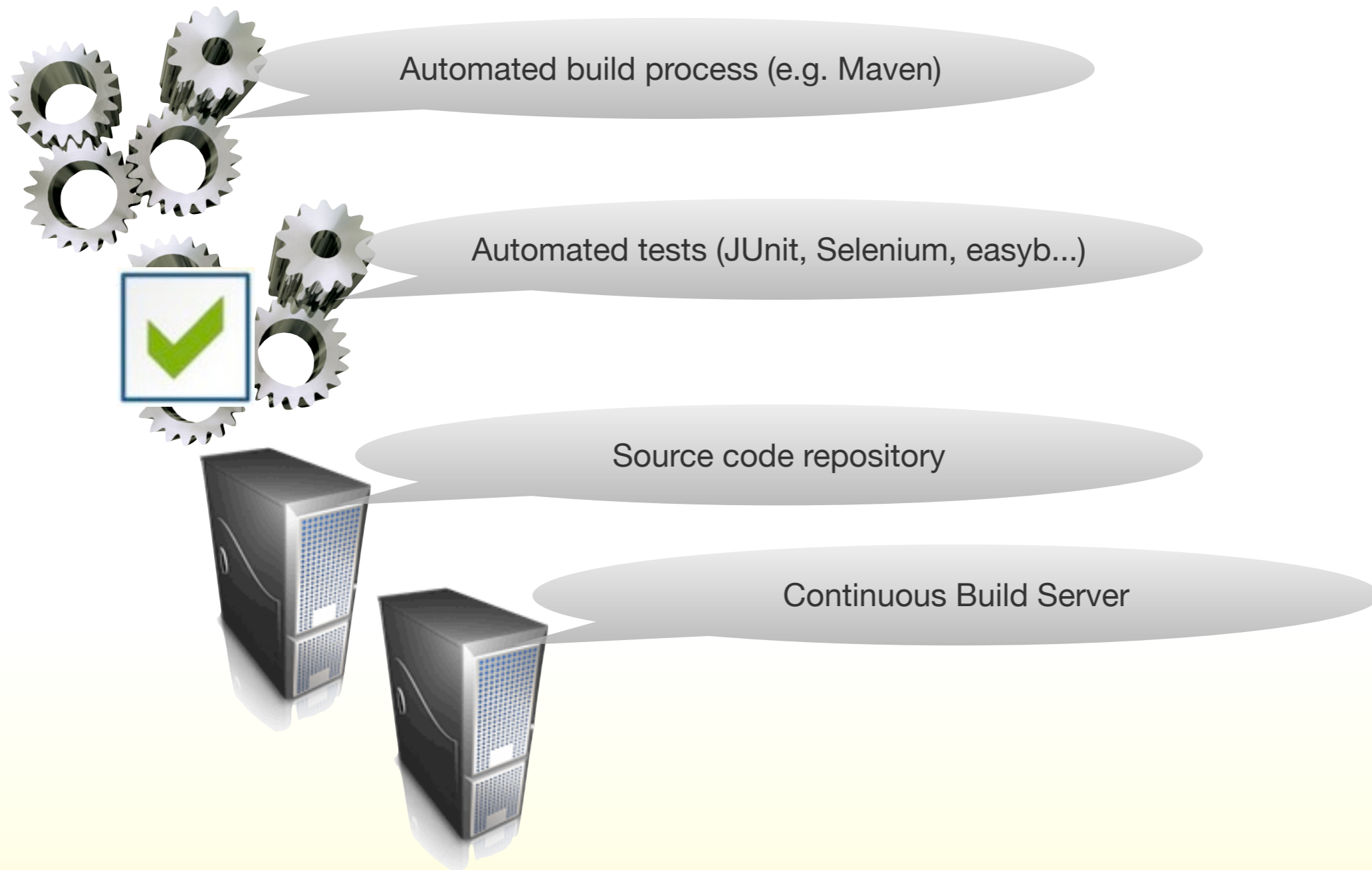


# Automating the build process

- ▶ Continuous Integration - why bother?
  - ▶ Smoother integration process
  - ▶ Automatic regression testing
  - ▶ Regular working releases
  - ▶ Earlier functional testing
  - ▶ Faster and easier bug fixes
  - ▶ Better visibility

# Automating the build process

## ▶ Continuous Integration - what you need



# Automating the build process

- ▶ Continuous Integration - what can it do?
  - ▶ More than just your average build scheduler!
    - ☑ Raise (and monitor) integration issues - fast!
    - ☑ Automatically publish Maven artifacts
    - ☑ Monitor your build process
    - ☑ Monitor and report on code quality and code coverage

# Automating the build process

- ▶ Continuous Integration - raising issues fast!
- ▶ Use an appropriate notification strategy, e.g.
  - ☑ The committer is notified for all build results
  - ☑ Team members are notified for any failed builds
  - ☑ Team leader gets special notification after 5 successive build failures

# Automating the build process

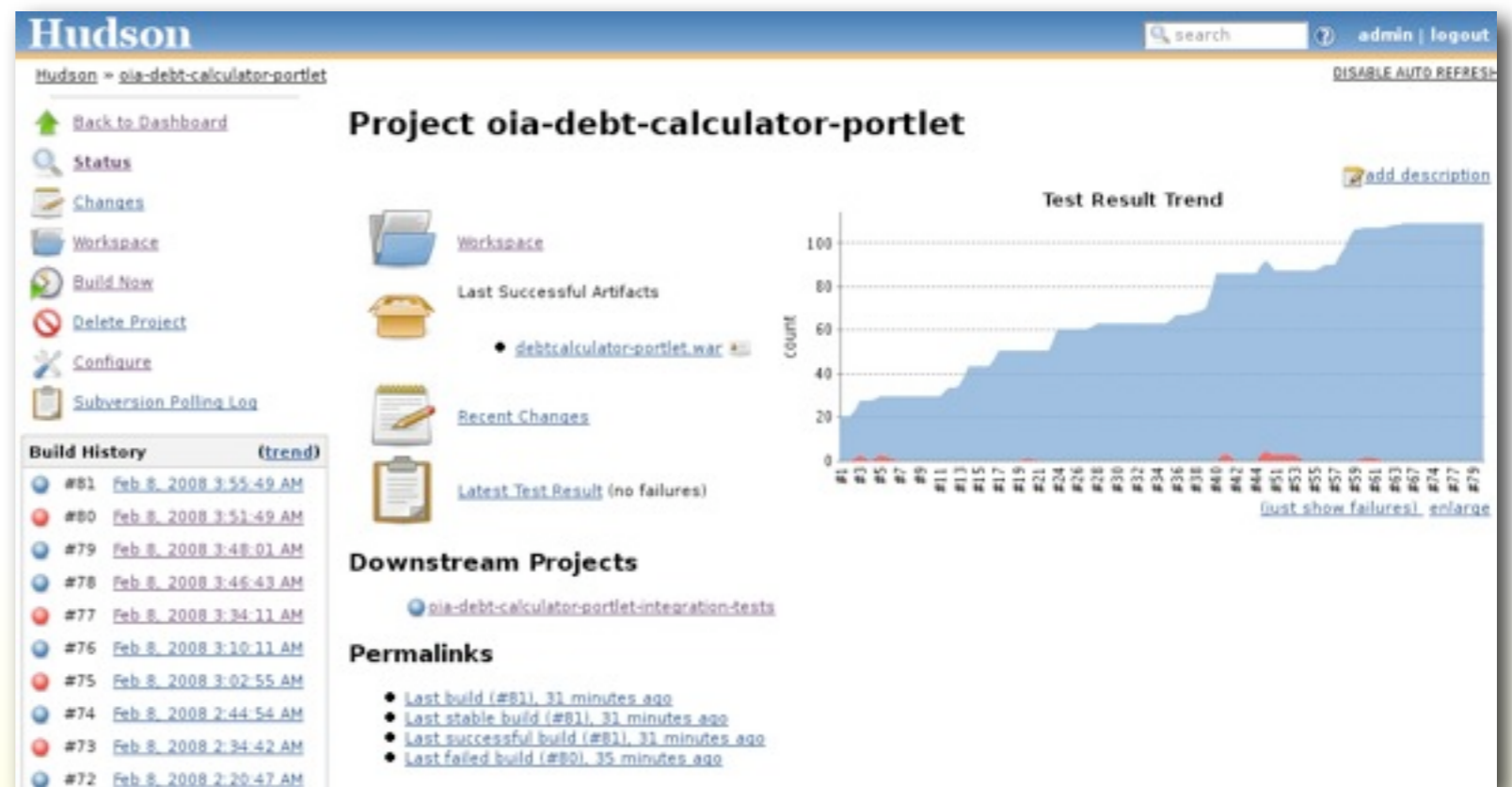
- ▶ Continuous Integration - raising issues fast!
- ▶ Use an appropriate (fast) notification mechanism
  - ☑ Instant Messaging, SMS etc. for fast notification
  - ☑ Email as a secondary channel
  - ☑ RSS for consulting build history

# Automating the build process

- ▶ Continuous Integration - raising issues fast!
- ▶ Use a good build plan strategy
  - ☑ Fast builds first, e.g.
  - ☑ Unit tests before integration tests
  - ☑ Integration tests before metrics
- ▶ Use manual builds where appropriate
  - ☑ Release builds
  - ☑ Deployments
  - ☑ ...

# Automating the build process

- ▶ Looking for a good O/S Continuous Integration tool?
- ▶ Try Hudson!
  - ☑ Easy to set up and configure
  - ☑ Good build and code quality metrics
  - ☑ Lots of plugins



# DEMO

Automating the builds

# Better Testing

- ▶ Improving your testing game
  - ▶ Innovative testing techniques
  - ▶ Automating unit tests
  - ▶ Separate unit tests and integration tests
  - ▶ Monitor test duration
  - ▶ Keep tabs on test coverage

# Better Testing

- ▶ Use innovative testing techniques
  - ▶ Unit testing
    - ☑ A cornerstone of modern software development
    - ☑ Unit tests can help you:
      - ▶ Ensure that code behaves as expected
      - ▶ Make your code more flexible and easier to maintain
      - ▶ Detect regressions
      - ▶ Document your code

# Better Testing

- ▶ Use innovative testing techniques
  - ▶ **Innovative** unit testing
    - ☑ More readable tests - Hamcrest asserts
    - ☑ More efficient tests - Parameterized tests
    - ☑ Cleaner tests - using Groovy
    - ☑ More accurate tests - Behavior Driven Development (BDD)
    - ☑ Functional and web tests
    - ☑ Use your imagination!

# Better Testing

- ▶ Use innovative testing techniques
  - ▶ Hamcrest asserts
    - ☑ More expressive and readable test assertions
    - ☑ Easier to understand
    - ☑ Less chance of test errors

# Better Testing

## ▶ Use innovative testing techniques

### ▶ Hamcrest asserts

- ✓ Traditional JUnit 3.x asserts are hard to read:
- ✓ Parameter order is counter-intuitive for English-speakers
- ✓ `x=10` is written

```
assertEquals(10, x);
```

- ✓ The statements don't read well for English-speakers
- ✓ “Assert that are equal 10 and x”
- ✓ Default error messages are sometimes limited:

```
String color = "yellow";  
assertTrue(color.equals("red") || color.equals("blue"));
```

Failure Trace

java.lang.AssertionError:

at com.wakaleo.jpt.junit.lab5.taxcalculato

# Better Testing

## ▶ Use innovative testing techniques

### ▶ Hamcrest asserts

☑ JUnit 4.4 introduces the `assertThat` statement

☑ Rather than writing:

```
import static org.junit.Assert.*;
...
assertEquals(expectedTax, calculatedTax, 0);
```

☑ You can write

```
import static org.hamcrest.Matchers.*;
...
assertThat(calculatedTax, is(expectedTax));
```

# Better Testing

- ▶ Use innovative testing techniques
  - ▶ Parameterized tests
    - ☑ Run several sets of test data against the same test case
    - ☑ Help reduce the number of unit tests to write
    - ☑ Encourage developers to test more thoroughly

# Better Testing

- ▶ Use innovative testing techniques
  - ▶ Parameterized tests
    - ☑ Example: Calculating income tax

Taxable income	PAYE rate for every \$1 of taxable income (excluding ACC earners' levy)
up to \$14,000	12.5 cents
\$14,001 to \$40,000 inclusive	21 cents
\$40,001 to \$70,000	33 cents
\$70,000 and over	39 cents

# Better Testing

## ▶ Use innovative testing techniques

### ▶ Parameterized tests

#### ☑ What you need:

- ▶ Some test data
- ▶ A test class with matching fields
- ▶ And some tests
- ▶ And an annotation

Income	Year	Expected Tax
\$0.00	2007	\$0.00
\$10,000.00	2007	\$1,950.00
\$20,000.00	2007	\$3,900.00
\$38,000.00	2007	\$7,410.00
\$38,001.00	2007	\$7,410.33
\$40,000.00	2007	\$8,070.00
\$60,000.00	2007	\$14,670.00
\$100,000.00	2007	\$30,270.00

**@RunWith(Parameterized.class)**

**TaxCalculationTest**

```
- income : double  
- year : int  
- expectedTax : double  
+ TaxCalculationTest(year : int, income : double, expectedTax : double)  
+ shouldCalculateCorrectTax()
```

```
@Test  
public void shouldCalculateCorrectTax() throws InvalidYearException {  
    TaxCalculator calculator = new TaxCalculatorImpl();  
    BigDecimal calculatedTax = calculator.calculateIncomeTax(income, year);  
    assertThat(expectedTax, is(calculatedTax));  
}
```

# Better Testing

- ▶ Use mocks and stubs
- ▶ Mockito - lightweight mocking



```
@Test
public void stubbing() {
    Stack<String> mockStack = mock(Stack.class);
    when(mockStack.pop()).thenReturn("hi there!");

    assertThat(mockStack.pop(), is("hi there!"));
}
```

Define Stubs

```
@Test
public void verifyCollaboration() {
    Stack<String> mockStack = mock(Stack.class);

    // do stuff
    mockStack.push("hi there!");
    mockStack.pop();

    // check what happened
    verify(mockStack).push("hi there!");
    verify(mockStack).pop();
}
```

Verify interactions

# DEMO

## Better Testing - Parameterized Tests and Hamcrest Asserts

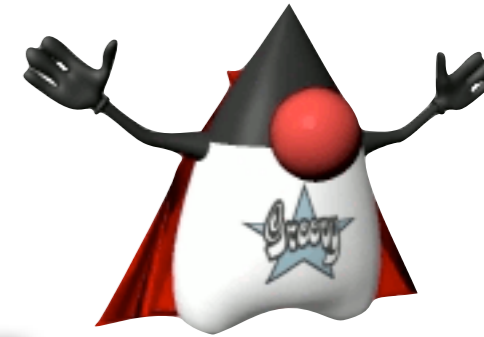
# Better Testing

- ▶ Use innovative testing techniques
  - ▶ Groovy Testing
    - ☑ Write more expressive unit tests in Groovy
    - ☑ Expressive test code
    - ☑ Quicker to write
    - ☑ More incentive to write in-depth test cases
  - ▶ But
    - ☑ Relatively limited IDE support...



# Better Testing

- ▶ Use innovative testing techniques
- ▶ Some examples...



```
public class TaxCalculatorBusinessTest {  
  
    private TaxCalculator calculator;  
  
    @Before  
    public void setup() {  
        calculator = new TaxCalculatorImpl();  
    }  
  
    @Test  
    public void shouldNotTaxLosses() throws InvalidYearException {  
        BigDecimal tax = calculator.calculateIncomeTax(new BigDecimal("-100000"), 2009);  
        assertThat(tax, is(new BigDecimal("0.00")));  
    }  
}
```

```
class GroovyTaxCalculatorBusinessTest {  
  
    def calculator = new TaxCalculatorImpl()  
  
    @Test  
    void shouldNotTaxLosses() {  
        def tax = calculator.calculateIncomeTax(-100000, 2009)  
        assertThat tax, is(0.00)  
    }  
}
```

# Better Testing

- ▶ Use innovative testing techniques
- ▶ Some examples...



```
public class TaxCalculatorBusinessTest {  
  
    private TaxCalculator calculator;  
  
    @Before  
    public void setup() {  
        calculator = new TaxCalculatorImpl();  
    }  
  
    @Test(expected=InvalidYearException.class)  
    public void shouldNotAcceptUnknownYears() throws InvalidYearException {  
        calculator.calculateIncomeTax(new BigDecimal("100000"), 1901);  
    }  
}
```

```
class GroovyTaxCalculatorBusinessTest {  
  
    final shouldFail = new GroovyTestCase().&shouldFail  
  
    @Test  
    void shouldNotAcceptUnknownYears() {  
        shouldFail(InvalidYearException) {  
            calculator.calculateIncomeTax(100000, 1901)  
        }  
    }  
}
```

# Better Testing

- ▶ Use innovative testing techniques
- ▶ Behaviour-Driven Development
  - ☑ It's not about writing tests
  - ☑ TDD and BDD is about writing better code:
    - ☑ Maintainable
    - ☑ Flexible
    - ☑ Reliable
    - ☑ Simple



# Better Testing

- ▶ Use innovative testing techniques
- ▶ Behavior-Driven Development
  - ☑ BDD uses words like “should” to describe the desired behavior
  - ☑ “Should transfer money from account A to account B”
  - ☑ Should deploy landing gear before touching ground”
  - ☑ ...



# Better Testing

- ▶ Use innovative testing techniques
  - ▶ Enter Easyb
    - ☑ A BDD testing framework for Java
    - ☑ Make testing clearer and easier to write
    - ☑ Make tests self-documenting
    - ☑ Help developers focus on the requirements



# Better Testing

- ▶ Use innovative testing techniques
  - ▶ Easyb stories:
    - ☑ Use a narrative approach
    - ☑ Describe a precise requirement
    - ☑ Can be understood by a stakeholder
    - ☑ Usually made up of a set of scenarios
    - ☑ Use an easy-to-understand structure:
      - ▶ *Given [a context]...*
      - ▶ *When [something happens]...*
      - ▶ *Then [something else happens]...*



# Better Testing

- ▶ Use innovative testing techniques
- ▶ Enter Easyb
  - ☑ A Groovy-based DSL for Behaviour-Driven Development



```
import com.wakaleo.jpt.taxcalculator.InvalidYearException

scenario "Should not tax losses", {
    given "a correctly configured tax calculator",
    when "you calculate annual income tax for a negative income",
    then "the calculated tax should be zero",
}
```

```
import com.wakaleo.jpt.taxcalculator.InvalidYearException

scenario "Should not tax losses", {
    given "a correctly configured tax calculator", {
        calculator = new TaxCalculatorImpl()
    }
    when "you calculate annual income tax for a negative income", {
        tax = calculator.calculateIncomeTax(-10000, 2009)
    }
    then "the calculated tax should be zero", {
        assert tax == 0
    }
}
```

# Better Testing

- ▶ Use innovative testing techniques
- ▶ Easyb reporting
  - ☑ Simple but readable...



**sections**

Summary

---

Stories

---

Stories Text

---

**Summary**

Behaviors	Failed	Pending	Time (sec)
4	0	1	0.708

**Stories Summary**

Stories	Scenarios	Failed	Pending	Time (sec)
2	4	0	1	0.708

**Specifications Summary**

**sections**

Summary

---

**Stories**

---

Stories Text

---

**Stories List**

Story	Scenarios	Failed	Pending	Time (sec)
create account	1	0	0	0.603
Make initial deposit onto a new account			success	0.08
given a new account				
when an initial deposit is made				
then the balance should be equal to the amount deposited			success	
withdraw money from account	3	0	1	0.105

# Better Testing

- ▶ Use innovative testing techniques
  - ▶ Continuous Testing
    - ☑ Run your tests whenever you save your code changes
    - ☑ Test failures are treated like compiler errors!
  - ▶ Two main tools:
    - ☑ Infinitest
    - ☑ JUnitMax

# DEMO

## Better Testing - Testing with Groovy and EasyB

# Better Testing

- ▶ Use innovative testing techniques
  - ▶ Integration Test strategies
    - ☑ Deploy to an embedded Jetty server for local integration tests
    - ☑ Run local integration tests with a web testing tool
      - ▶ Selenium
      - ▶ HtmlUnit
      - ▶ JWebUnit
      - ▶ Canoo WebTest
    - ☑ Deploy automatically to a publicly-visible integration server

# Better Testing

## ▶ Web Testing Tools

### ▶ Two strategies to developer web testing:

#### Test before deployment

▶ Use Jetty

▶ Run web tests locally from within the normal build

#### Test after deployment

▶ Deploy to a test server

▶ Run web tests against a remote server in a dedicated build job

▶ You probably need both...

# Better Testing

## ▶ Examples of Web Testing Tools

### ▶ Selenium

- ✓ Runs tests through a real browser
- ✓ Accurate rendition of Javascript and AJAX behavior
- ✓ High-level readable API
- ✓ More complicated to set up
- ✓ Slow



# Better Testing

## ▶ Examples of Web Testing Tools

### ▶ Selenium



```
@Test
public void testDepositAmount() throws Exception {
    Selenium selenium
        = new DefaultSelenium("localhost", 4444,
                               "*firefox",
                               "http://localhost:8080");

    selenium.start();
    selenium.open("/ebank-web");
    selenium.waitForPageToLoad("10000");
    selenium.type("depositAmount", "100");
    selenium.click("deposit");
    selenium.waitForPageToLoad("10000");
    selenium.isTextPresent("Current balance: $100");
}
```

Open up a web browser

These actions are executed in the browser

# Better Testing

## ▶ Examples of Web Testing Tools

### ▶ HTMLUnit

- ☑ Simulates a browser
- ☑ Lower level API
- ☑ Easier to set up
- ☑ Fast

# Better Testing

## ▶ Examples of Web Testing Tools

### ▶ HTMLUnit

```
@Test
public void depositCash() throws Exception {
    WebClient webClient = new WebClient();
    HtmlPage page = webClient.getPage("http://localhost:8080/ebank-web");
    assert page.asText().contains("Current Balance: $0");
    HtmlForm form = page.getForms().get(0);
    HtmlSubmitInput depositButton = form.getInputByName("deposit");
    HtmlTextInput textField = form.getInputByName("depositAmount");

    textField.setValueAttribute("100");

    HtmlPage result = depositButton.click();
    assert result.asText().contains("Current Balance: $100");
}
```

Simulates the browser

Requires knowledge of the page structure

# Better Testing

## ▶ Examples of Web Testing Tools

### ▶ JWebUnit

- ☑ Simulates the browser (build on HTMLUnit)
- ☑ Higher and more readable API
- ☑ Easy to set up
- ☑ Fast

# Better Testing

## ▶ Examples of Web Testing Tools

### ▶ JWebUnit

```
public class TestDepositStoryUI extends WebTestCase {
    public void testDepositingCashShouldAddToBalance() {
        setBaseUrl("http://localhost:9090/ebank-web");
        beginAt("/");
        assertTextPresent("Current Balance: $0");
        setTextField("depositAmount", "100");
        clickButtonWithText("deposit");
        assertTextPresent("Current Balance: $100");
    }
}
```

Simulates a browser

High-level API

# DEMO

## Better Testing - Web Testing

# Automated Code Quality

- ▶ Why use code quality metrics
  - ▶ Better quality code
  - ▶ Code is easier to maintain
  - ▶ Detect potential bugs
  - ▶ Train new staff

# Automated Code Quality

- ▶ Why automate code quality metrics
  - ▶ Global picture of code quality
  - ▶ Statistical code quality trends
  - ▶ More value-added manual reviews

# Automated Code Quality

- ▶ How automate code quality metrics
  - ▶ Integrate code quality metrics into your build scripts
  - ▶ Run a special build reserved for code metrics
  - ▶ TIP: Code quality metrics are easier with Maven!

# Automated Code Quality

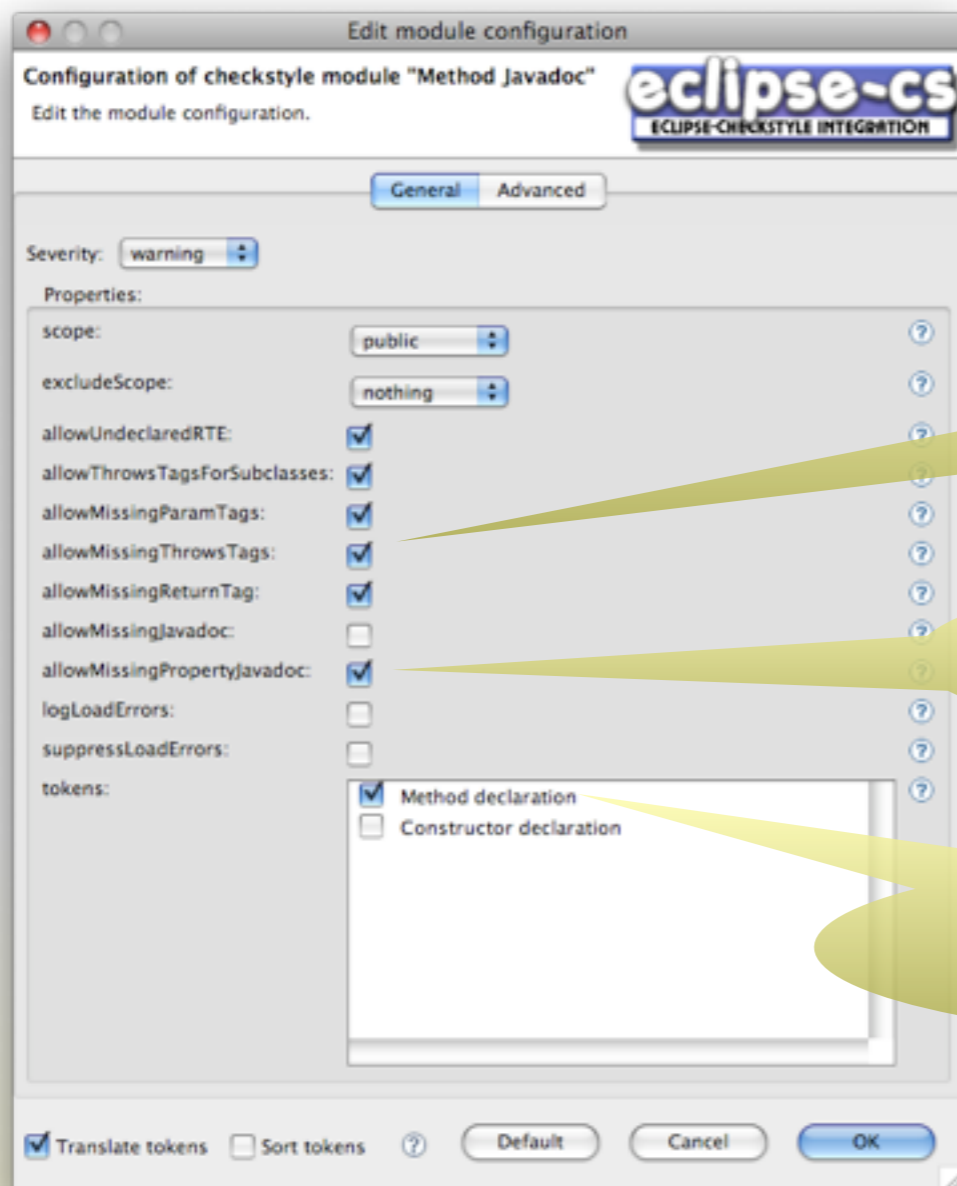
- ▶ Types of code quality metrics
  - ▶ Coding Standards
    - ☑ Naming conventions, Javadoc comments, layout,...
    - ☑ Tool: **Checkstyle**
  - ▶ Best Practices
    - ☑ Good programming habits, potential bugs,...
    - ☑ Tools: **Checkstyle, PMD, Findbugs, Crap4j**
  - ▶ Bug Detection
    - ☑ Broken code, dangerous code, bugs,...
    - ☑ Tool: **FindBugs**
  - ▶ Code Coverage
    - ☑ How much code is executed by your tests
    - ☑ Tool: **Cobertura, Emma, Clover, Crap4j...**

# Automated Code Quality

- ▶ Coding standards - how far do you go?
- ▶ Customize for your organization!
  - ☑ Configure rules to match your practices
  - ☑ Remove rules that don't apply
  - ☑ Add non-default rules where useful

# Automated Code Quality

- ▶ Coding standards - how far do you go?
- ▶ Example - Checkstyle javadoc conventions



Don't worry about @param,  
@return etc.

Don't worry about getters and  
setters

Don't enforce for constructors

# Automated Code Quality

## ▶ Team code reviews

- ☑ Review code as a group
- ☑ Long and slow if done manually
- ☑ Benefits greatly from the use of tools



# Automated Code Quality

- ▶ Automating code quality metrics with Hudson
  - ▶ Using the Violations plugin
    - ☑ Reports on Checkstyle, PMD, Findbugs, and others
    - ☑ Uses data generated with Maven or Ant

# Automated Code Quality

- ▶ Automating code quality metrics with Hudson
- ▶ Configuring the Violations plugin

Report Violations

Per file display limit

This is the number of violations to display per file (per type)

Define thresholds appropriate for your project

type			XML filename pattern
checkstyle	<input type="text" value="10"/>	<input type="text" value="999"/>	<input type="text" value="planeshowcase/target/checkstyle-result.xml"/>
cpd	<input type="text" value="10"/>	<input type="text" value="999"/>	<input type="text" value="planeshowcase/target/cpd.xml"/>
findbugs	<input type="text" value="10"/>	<input type="text" value="999"/>	<input type="text" value="planeshowcase/target/findbugs.xml"/>
fxcop	<input type="text" value="10"/>	<input type="text" value="999"/>	<input type="text"/>
pmd	<input type="text" value="10"/>	<input type="text" value="999"/>	<input type="text" value="planeshowcase/target/pmd.xml"/>
pylint	<input type="text" value="10"/>	<input type="text" value="999"/>	<input type="text"/>

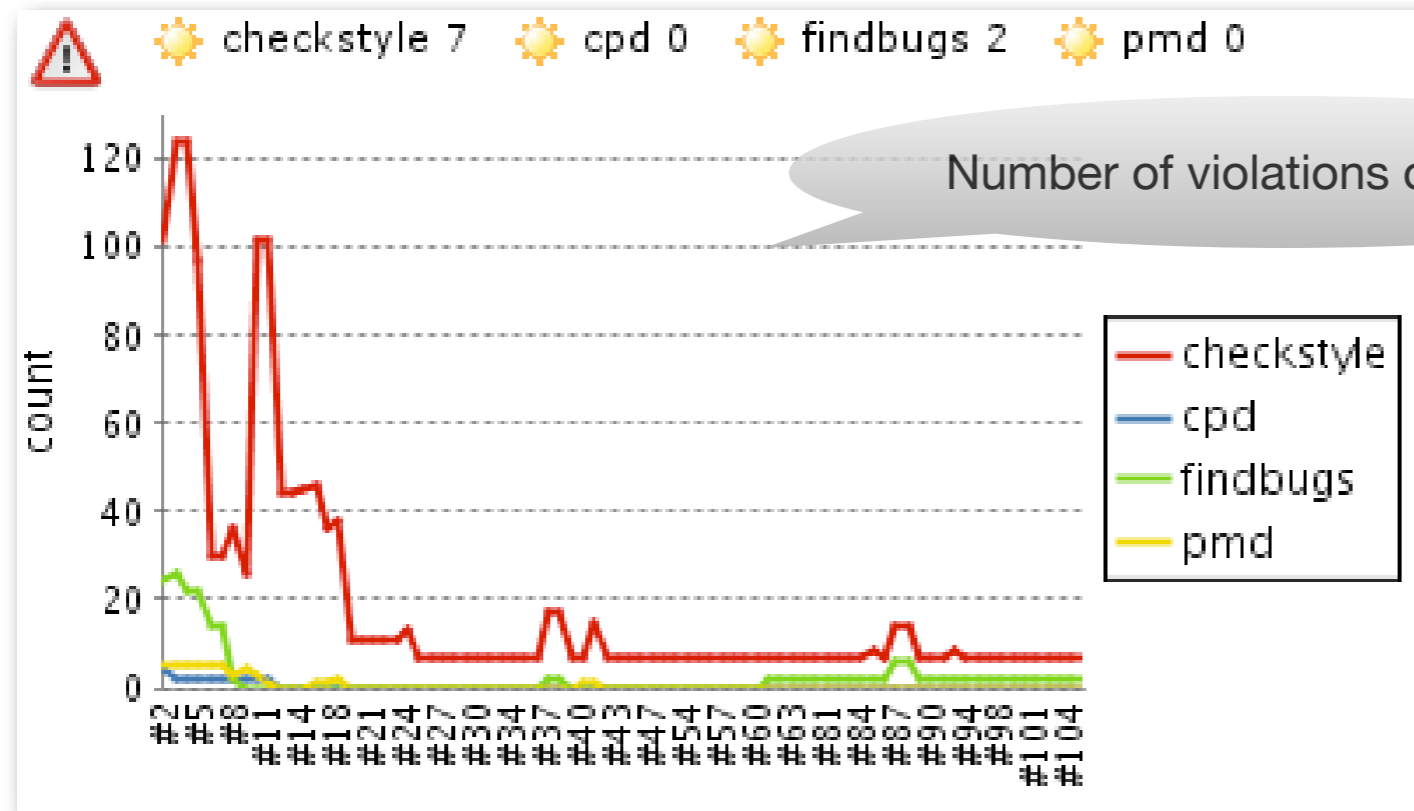
Source Path Pattern (Optional)

This is a file name pattern that can be used to resolve classes to sourcefiles (for example **\*\*/src/main/java**).

Faux Project Path (Optional)

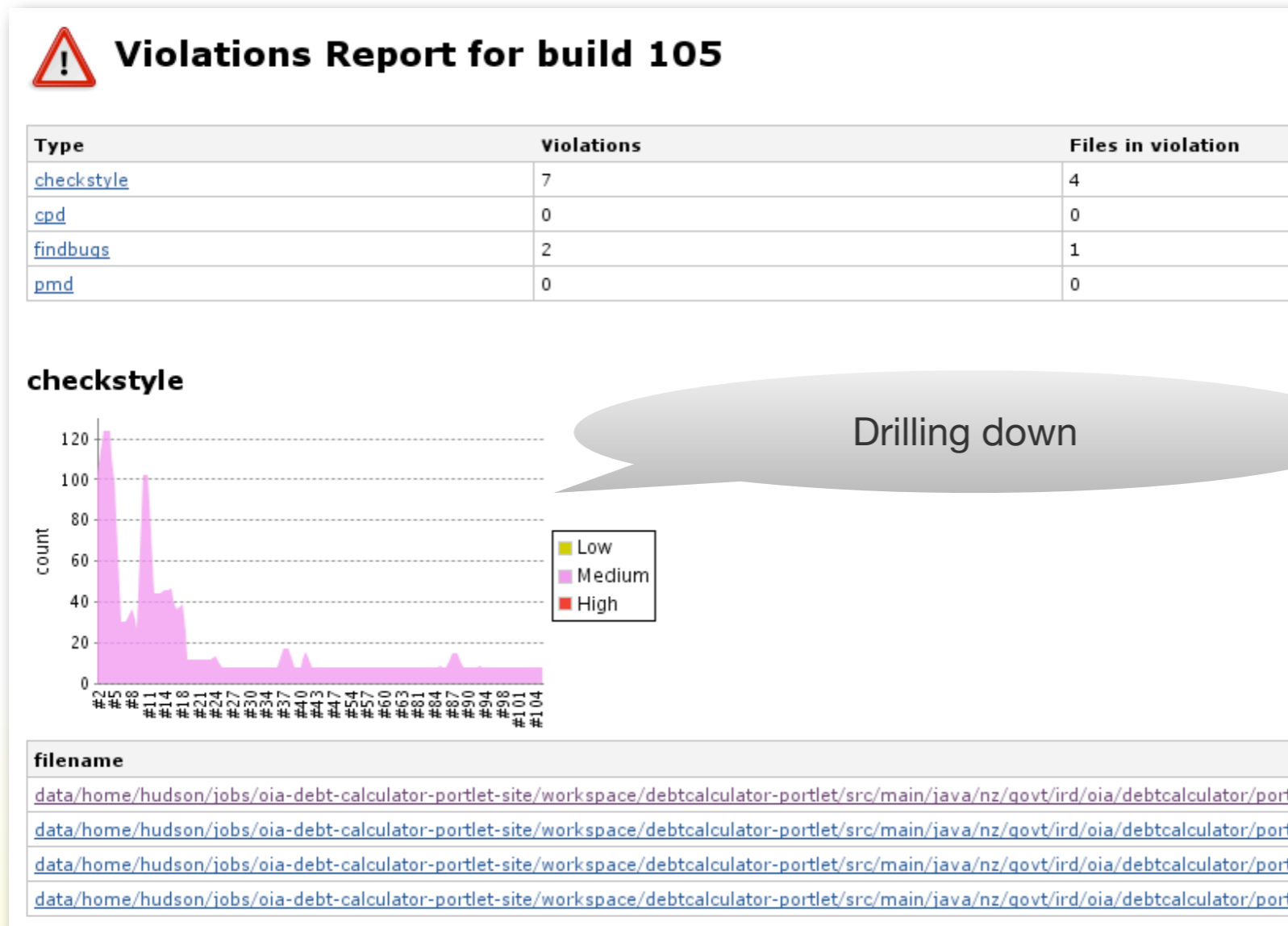
# Automated Code Quality

- ▶ Automating code quality metrics with Hudson
- ▶ Displaying the Violations reports



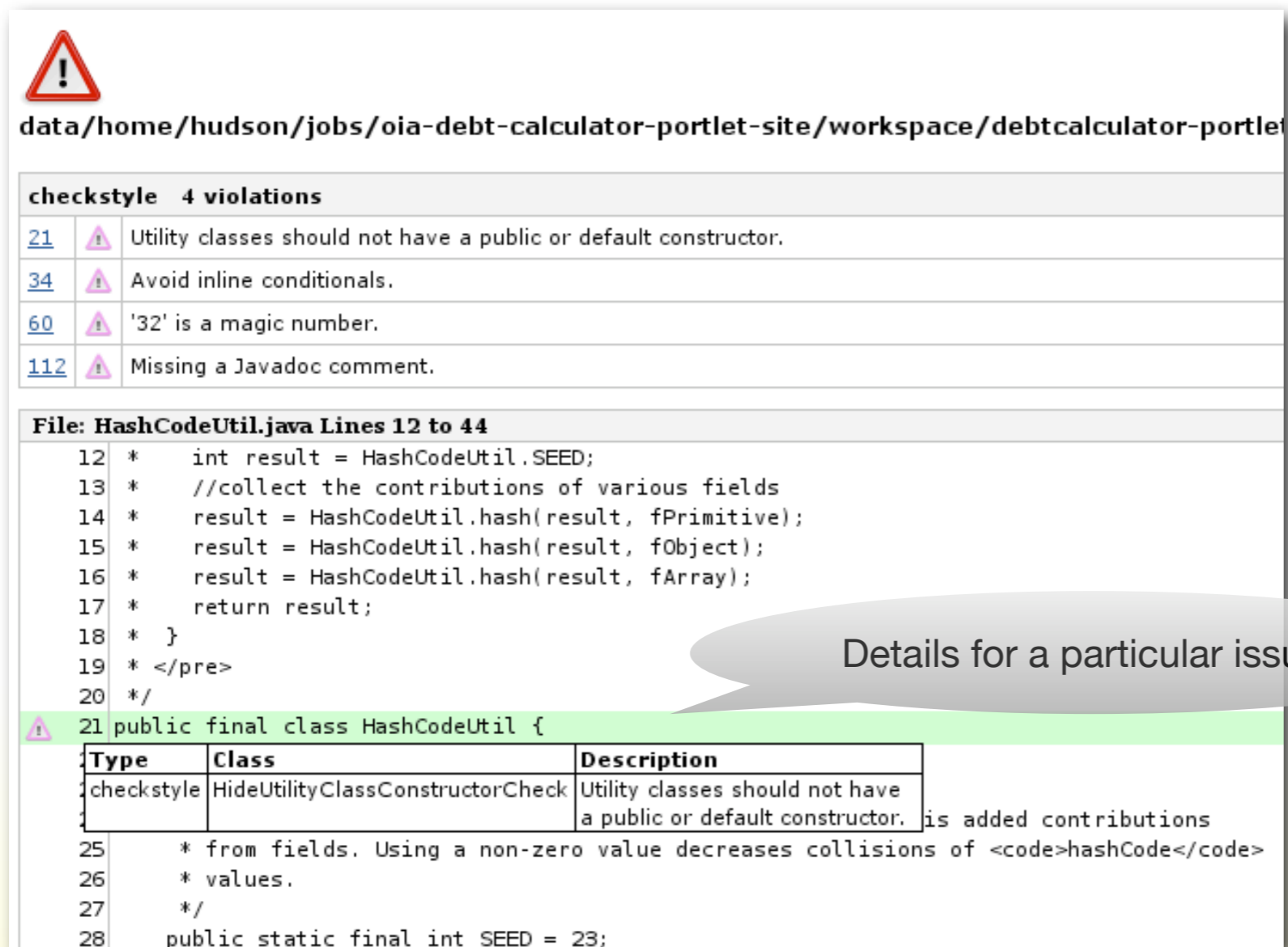
# Automated Code Quality


- ▶ Automating code quality metrics with Hudson
- ▶ Displaying the Violations reports







# Automated Code Quality

- ▶ Automating code quality metrics with Hudson
- ▶ Displaying the Violations details




  
data/home/hudson/jobs/oia-debt-calculator-portlet-site/workspace/debtcalculator-portlet

**checkstyle 4 violations**

<a href="#">21</a>		Utility classes should not have a public or default constructor.
<a href="#">34</a>		Avoid inline conditionals.
<a href="#">60</a>		'32' is a magic number.
<a href="#">112</a>		Missing a Javadoc comment.

**File: hashCodeUtil.java Lines 12 to 44**

```
12 *   int result = hashCodeUtil.SEED;
13 *   //collect the contributions of various fields
14 *   result = hashCodeUtil.hash(result, fPrimitive);
15 *   result = hashCodeUtil.hash(result, fObject);
16 *   result = hashCodeUtil.hash(result, fArray);
17 *   return result;
18 * }
19 * </pre>
20 */
21 public final class hashCodeUtil {
```

 21 public final class hashCodeUtil {

Type	Class	Description
checkstyle	HideUtilityClassConstructorCheck	Utility classes should not have a public or default constructor.

is added contributions  
25 \* from fields. Using a non-zero value decreases collisions of <code>hashCode</code>  
26 \* values.  
27 \*/  
28 public static final int SEED = 23;

Details for a particular issue

# Automated Code Quality



**TESTING**

I FIND YOUR LACK OF TESTS DISTURBING.

DIY.DESPAIR.COM

# Automated Code Quality

## ▶ Code Coverage

### ▶ What can code coverage metrics tell you:

- ☑ Indicates what code is being executed by your unit tests.
- ☑ Can help isolate untested code
- ☑ Does not guarantee that the tests that are run are of high quality
- ☑ Can be an indicator of whether tests are being written at all

# Automated Code Quality

- ▶ Code Coverage
  - ▶ Using the Hudson code coverage plugins
    - ☑ Generate data using Ant or Maven
    - ☑ Report coverage metrics in Hudson

Publish Cobertura Coverage Report

Cobertura xml report pattern

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 are using Subversion as SCM and have configured multiple modules, in which case it is relative to the workspace root. Cobertura must be configured to generate XML reports for this plugin to function.

Coverage Metric Targets

Methods	<input type="text" value="80"/>	<input type="text" value="0"/>	<input type="text" value="0"/>
Lines	<input type="text" value="80"/>	<input type="text" value="0"/>	<input type="text" value="0"/>
Conditionals	<input type="text" value="70"/>	<input type="text" value="0"/>	<input type="text" value="0"/>

Configure health reporting thresholds.  
For the ☀️ row, leave blank to use the default value (i.e. 80).  
For the ☁️ and 🟡 rows, leave blank to use the default values (i.e. 0).

# Automated Code Quality

- ▶ Code Coverage
  - ▶ Using the Hudson code coverage plugins
    - ☑ Cobertura coverage reports

```
43  */
44  public Manifest getManifest() {
45  3   if (manifest == null) {
46  0   loadMetaInf();
47  }
48  3   return manifest;
49  }
50
51  public void setManifest(Manifest manifest) {
52  0   this.manifest = manifest;
53  0  }
54
55  public ServletContext getContext() {
56  0   return context;
57  }
58
59  public void setContext(ServletContext context) {
60  3   this.context = context;
61  3  }
62
63  /**
64   * Load the META-INF file.
65   */
66  public void loadMetaInf() {
67  3   String appServerHome = context.getRealPath("/");
68  3   File manifestFile = new File(appServerHome, "META-INF/MANIFEST.MF");
```

Executed code

Unexecuted code

# Automated Code Quality

- ▶ Code Coverage
- ▶ Using the ECLiPse plugin

The screenshot shows the Eclipse IDE with the following components:

- Project Explorer:** Shows the project structure for 'babble-core' and its sub-projects.
- Code Editor:** Displays the source code for 'Babble.java'. The code is color-coded based on coverage: green for executed code and red for unexecuted code. Callouts point to these areas with the labels 'Executed code' and 'Unexecuted code'.
- Coverage View:** A table at the bottom of the IDE showing coverage statistics for the project and its components.

Element	Coverage	Covered Instructions	Total Instructions
babble-core	88.8%	404	455
src/main/java	78.9%	161	204
com.sonatype.training.babble.domain	78.9%	161	204
Babble.java	83.1%	59	71
Babbler.java	76.7%	102	133
src/test/java	96.8%	243	251

# Automated Code Quality

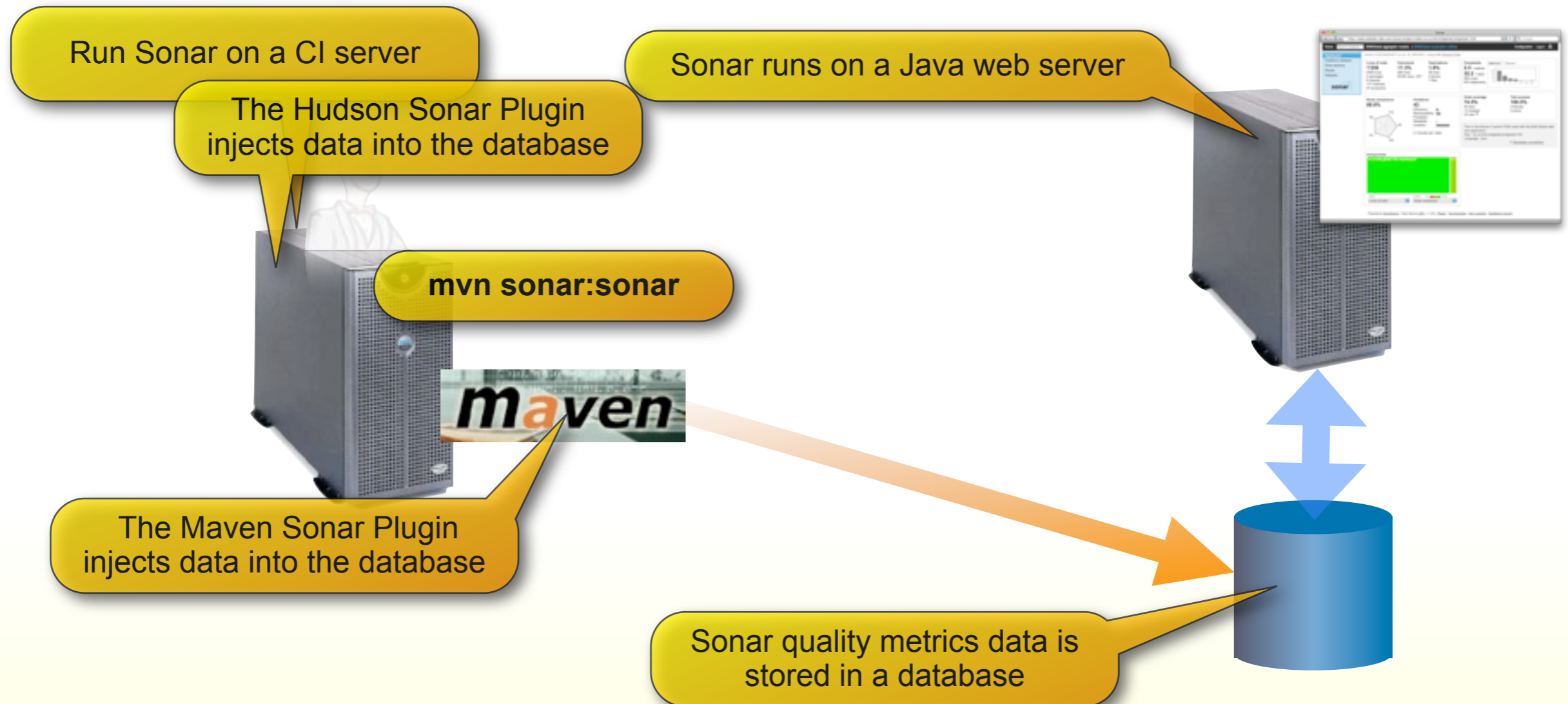
## ▶ Code Quality Management

### ▶ Check out Sonar!

- ☑ Centralized code quality management
- ☑ Generate code quality metrics in an automated build using Maven
- ☑ Store code quality metrics in a database
- ☑ Code quality metrics can be consulted on a web site

# Automated Code Quality

- ▶ Code Quality Management
- ▶ Sonar architecture



# Automated Code Quality

- ▶ Code Quality Management
  - ▶ Sonar centralizes many code quality metrics

Source code metrics

Code complexity metrics

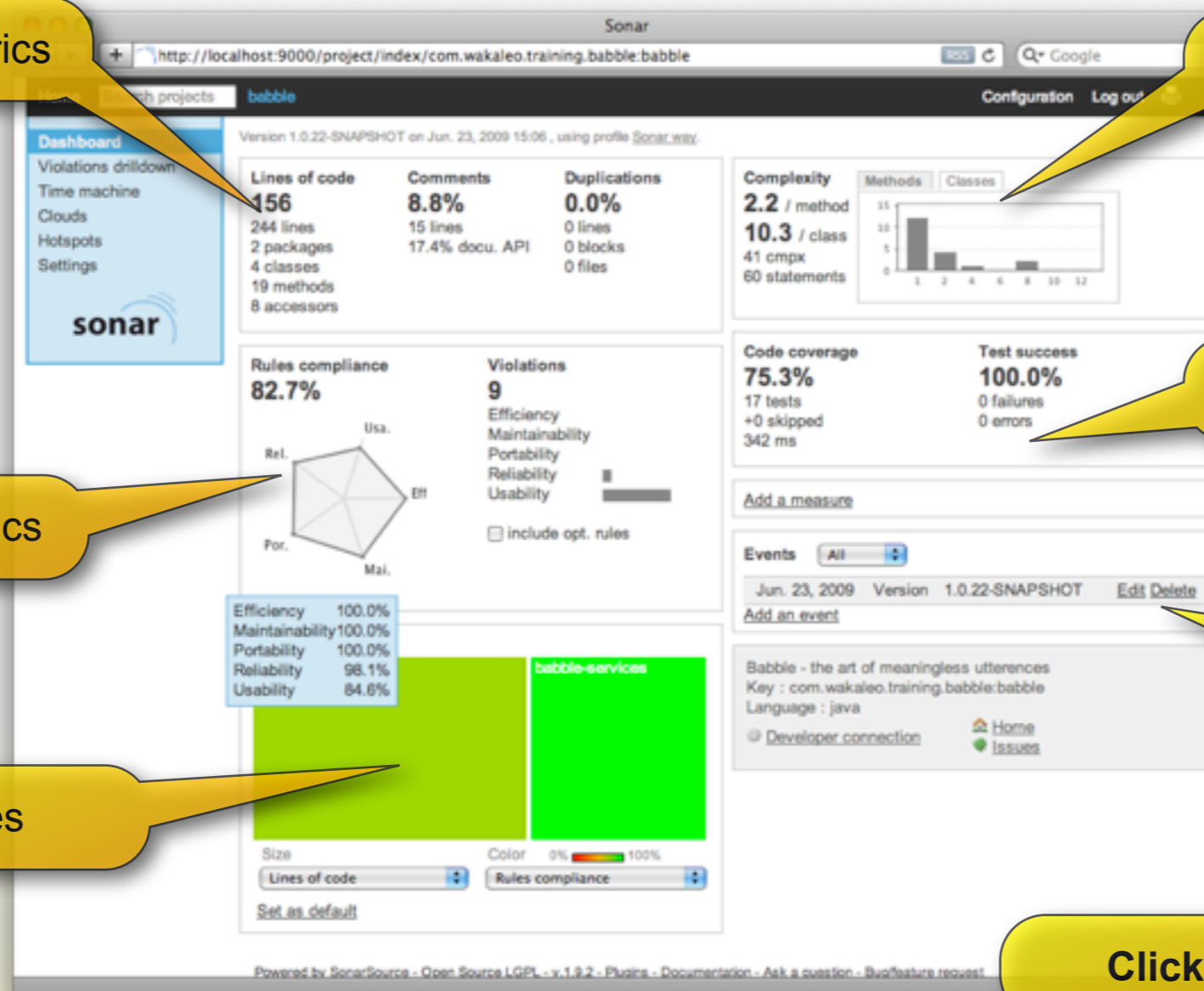
Code quality metrics

Test results and code coverage

Modules

Build history

Click anywhere to drill down



# Automated Code Quality

- ▶ Code Quality Management
- ▶ You can drill down to view the details for each type of issue

The screenshot displays the Sonar web interface. At the top, there's a navigation menu with 'Home', 'Search projects', and 'babble'. Below this, a 'Violations drilldown' section shows a table of categories and their counts: Mandatory (9), Optional (18), Usability (8), Reliability (1), Maintainability (0), Portability (0), and Efficiency (0). A 'Rules' table lists various rules like 'Local Final Variable Name', 'If Else Stmt Must Use Braces', 'Constant Name', and 'Constructor Calls Overridable Method'. A callout bubble labeled 'Overview' points to this table.

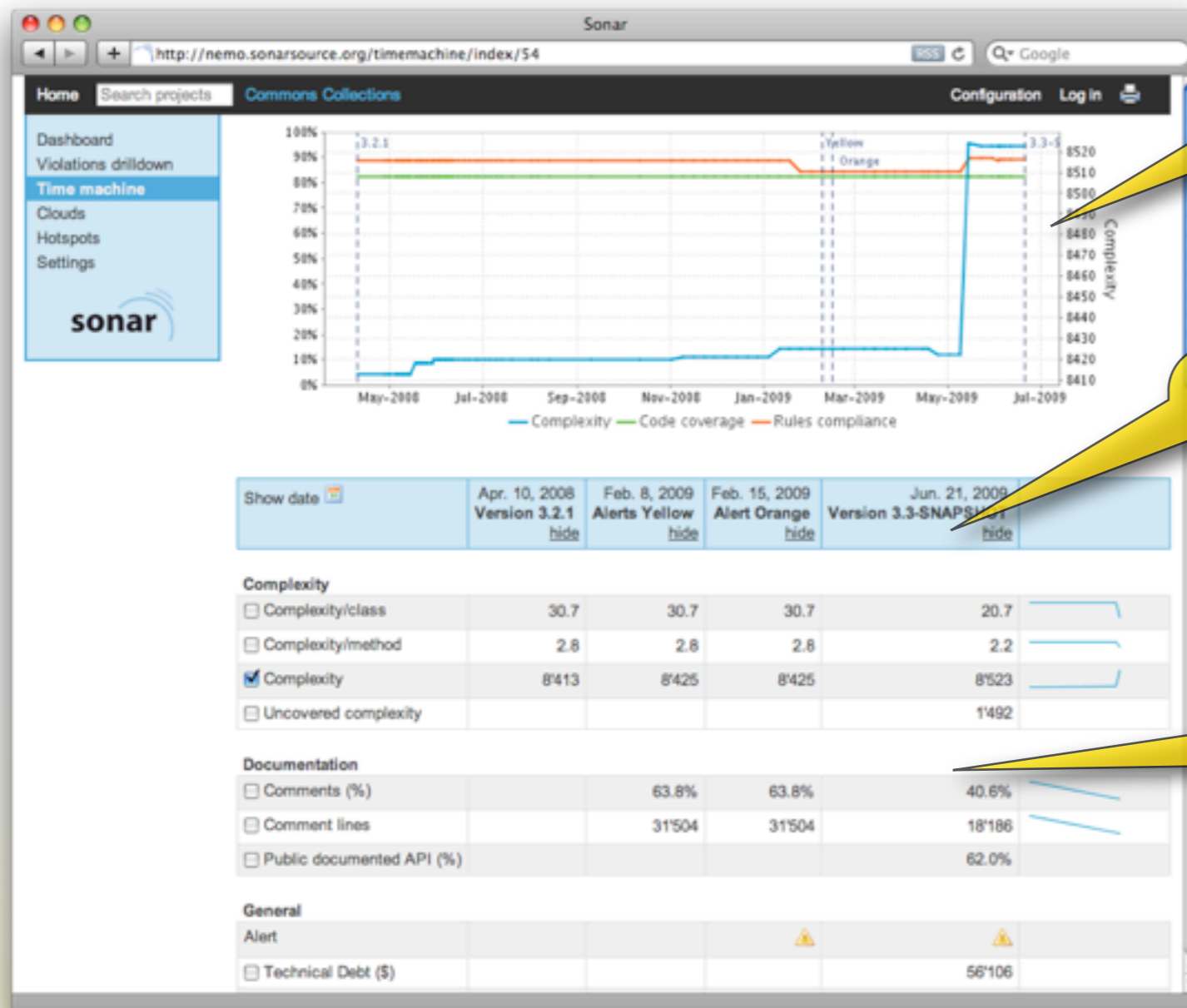
The main content area shows a 'Violations in this class' section for the 'Babble' class. It includes a table with columns for Sources, Code coverage, Violations, and Duplications. Below this, there's a code editor showing the source code of the 'Babble' class. A callout bubble labeled 'Violations in this class' points to this section.

The code editor shows several violations highlighted in yellow and red. A callout bubble labeled 'Different types of violations' points to these. The violations include: 'Constructor Calls Overridable Method: Overridable method 'setBabbler' called during object construction', 'Design For Extension: Method 'getUtterance' is not designed for extension - needs to be abstract, final or empty.', 'Design For Extension: Method 'setUtterance' is not designed for extension - needs to be abstract, final or empty.', 'Design For Extension: Method 'getTime' is not designed for extension - needs to be abstract, final or empty.', 'Design For Extension: Method 'setTime' is not designed for extension - needs to be abstract, final or empty.', 'Design For Extension: Method 'compareTo' is not designed for extension - needs to be abstract, final or empty.', and 'Local Final Variable Name: Name 'BEFORE' must match pattern '[a-z][a-zA-Z0-9]\*\$'.

At the bottom of the code editor, a callout bubble labeled 'Violation details' points to the 'Local Final Variable Name' violation.

# Automated Code Quality

- ▶ Code Quality Management
- ▶ You can also view historical data



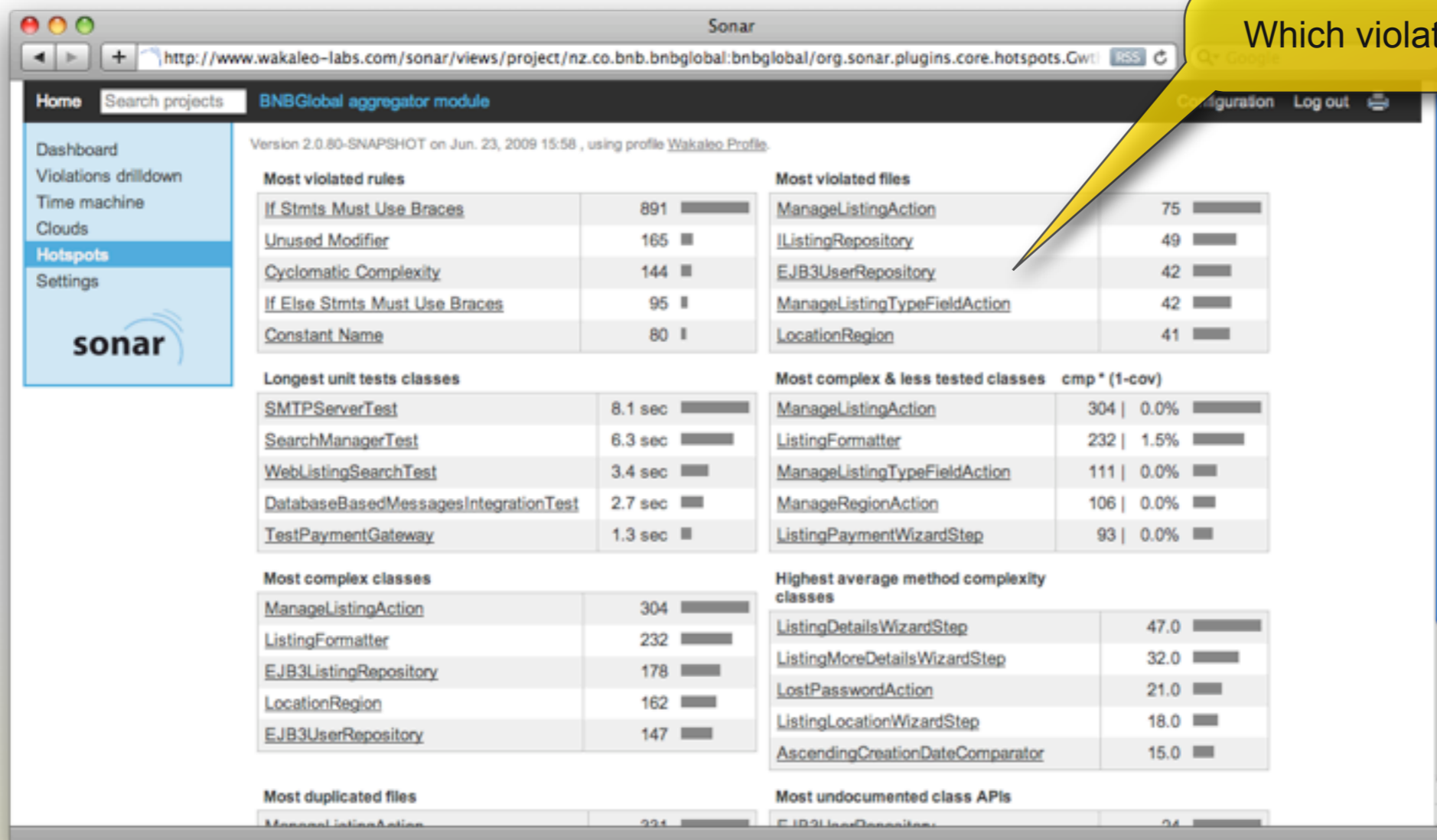
Quality metrics over time

Which builds to display

Which metrics to display

# Automated Code Quality

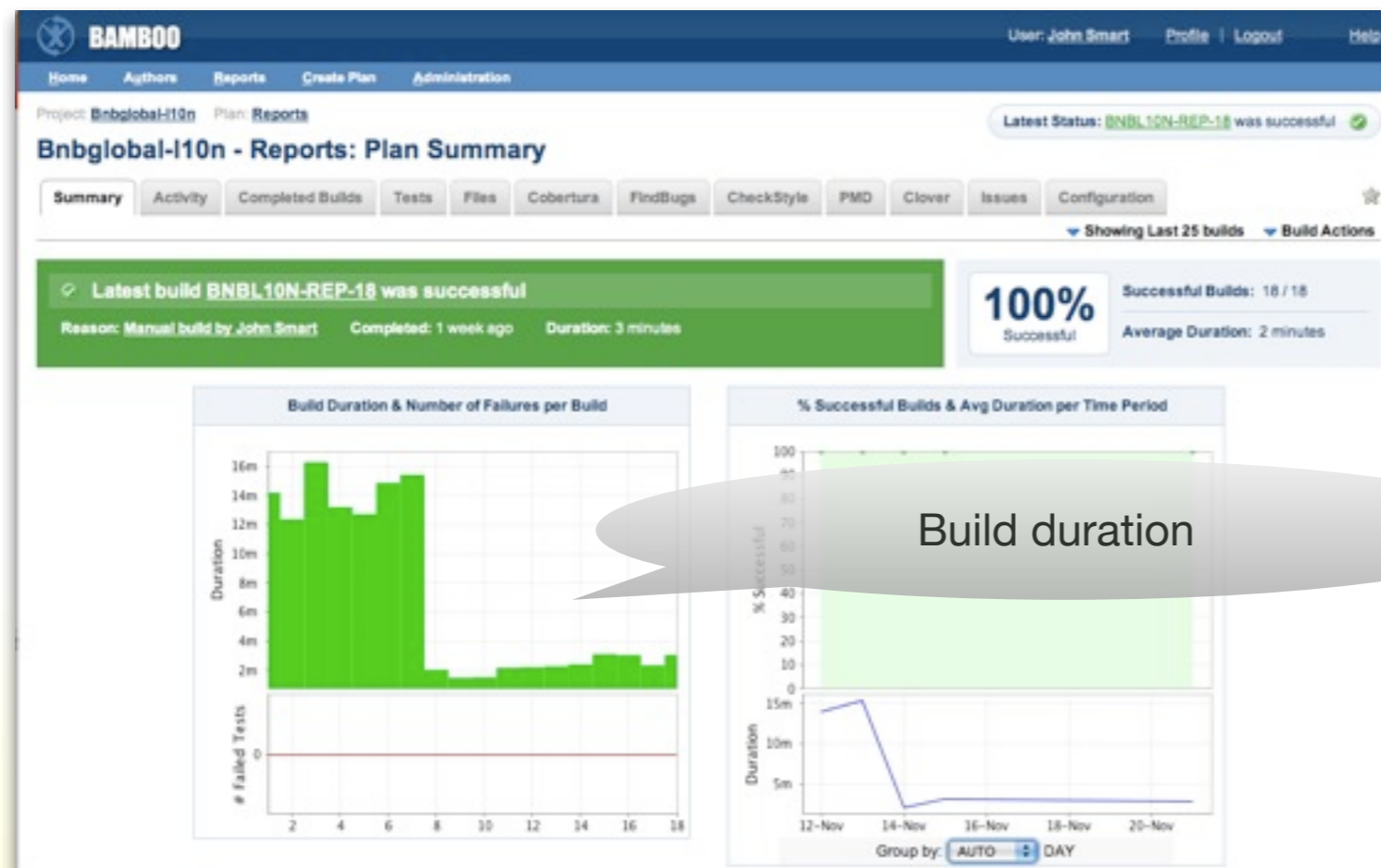
- ▶ Code Quality Management
- ▶ You can also view historical data



Which violations occur the most?

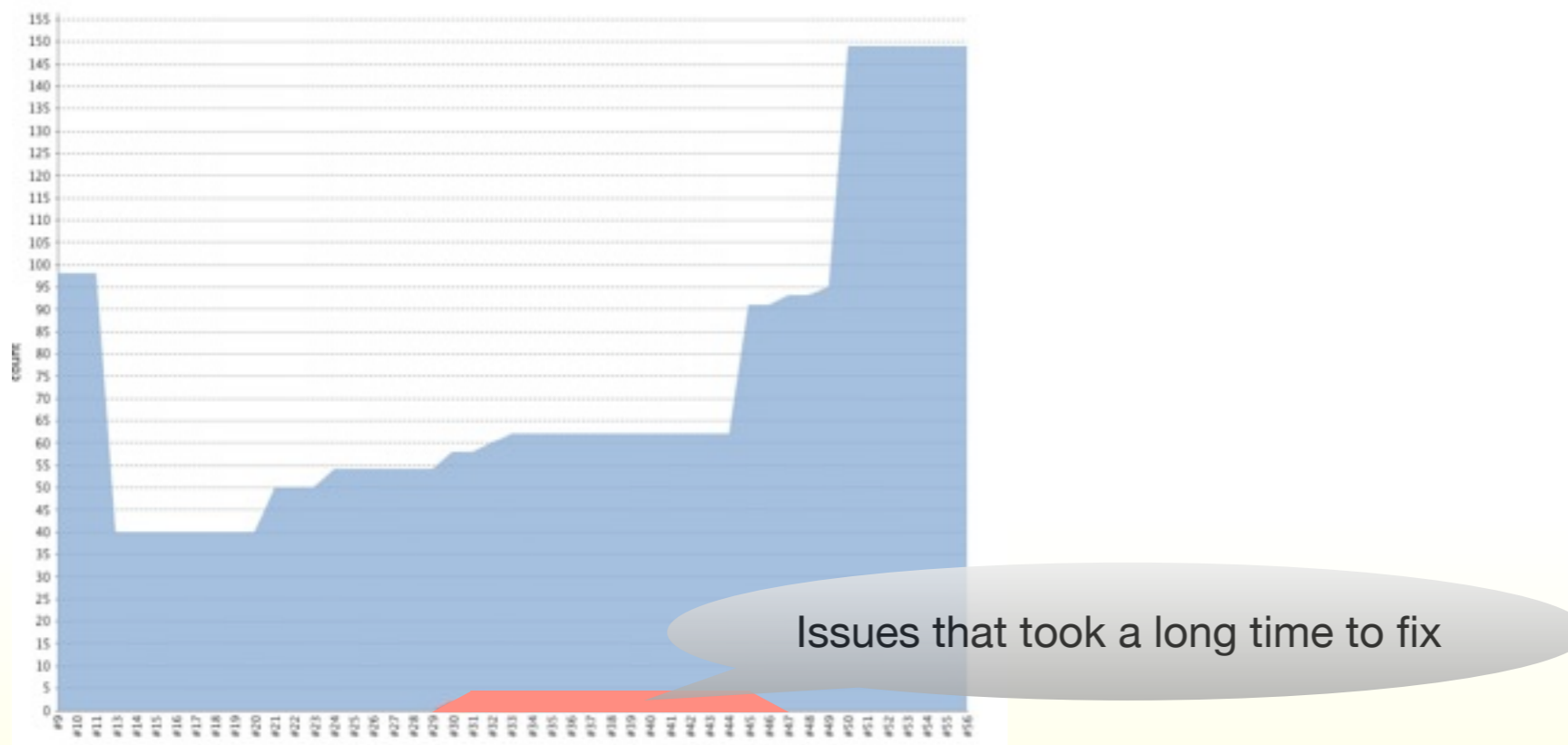
# Automated Code Quality

- ▶ More advanced code quality metrics
  - ▶ How long do your tests run
    - ☑ Overly-long tests slow down your build process
    - ☑ They may also indicate a performance issue



# Automated Code Quality

- ▶ More advanced code quality metrics
  - ▶ Are your tests failing repeatedly
    - ☑ May indicate a difficult technical issue
    - ☑ Could result in unreliable “quick-fix” solutions



# DEMO

## Automated Code Quality

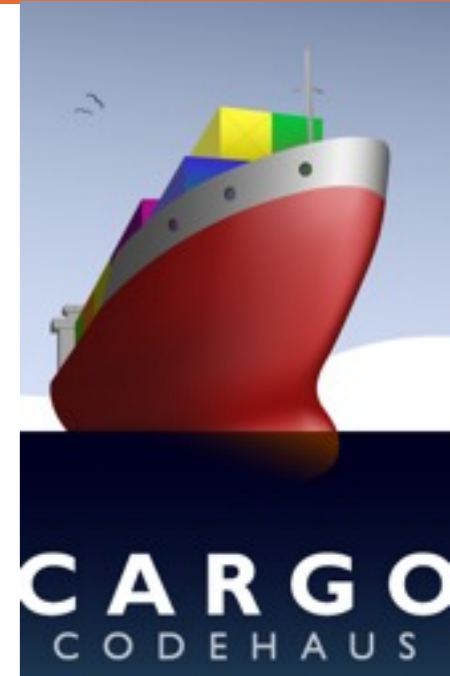
# Automated Deployment

- ▶ Automate your deployment process
  - ▶ Reproducible process
  - ▶ Save time
  - ▶ Audit trace



# Automated Deployment

- ▶ Automating the deployment process with Cargo
  - ▶ Start, stop and install application servers
  - ▶ Deploy to different application servers
  - ▶ Run a stand-alone instance
  - ▶ Or deploy to an existing server



Agile2009 - Making Agile Real

# Summary

- ▶ There are many ways to improve a development process
  - ▶ Improve your build process and dependency management
  - ▶ Install a CI server and publish your APIs automatically
  - ▶ Use code quality metrics to isolate bottlenecks and fix problems
  - ▶ Automate as much as possible!

John Ferguson Smart  
Email: john.smart@wakaleo.com  
Web: http://www.wakaleo.com  
Twitter: wakaleo

# Thanks for your attention!

