

# Software Engineering and Architecture

## Behavior Driven Development (BDD)

---

Olivier Liechti

HEIG-VD

[olivier.liechti@heig-vd.ch](mailto:olivier.liechti@heig-vd.ch)

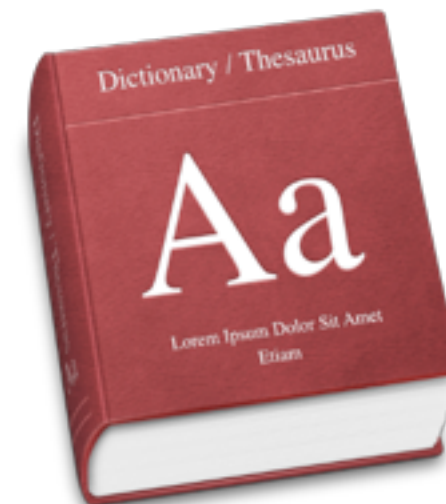


MASTER OF SCIENCE  
IN ENGINEERING

# Reminder: Building Value

---

- When we presented agile development concepts, we insisted on the notion of “**value**”.
- **Building the right system** is at least as important as **building the system right**.
- When requirements are managed with a **product backlog**, the order of items in the backlog should reflect their value.
- On paper, this sounds like common sense. In practice, it is not obvious for the **whole team** to have this **mindset**.
- **Therefore, anything that can reinforce this principle is useful. It has a lot to do with communication and collaboration.**



# Reminder: Are we Done?

---

- Agile methodologies rely on the definition of “done” for stories (or other **units of delivery**).
- From a **development** perspective, we have seen that a story is done when it has been **implemented, tested and documented**.
- With **continuous delivery**, the scope of “done” has been extended. A feature is “done” when it is in production and available to users.
- In practice, it is challenging for developers to **be sure that they are done** at the end of the sprint. Are all the scenario variations covered?
- **Practices and tools that help clarify, track and validate the acceptance criteria are important.**



# The Testing Onion

---

- There are **different types of tests**. They need to be run very, very often (along the delivery pipeline). The results must be broadcasted to the team.
- **Unit tests** must be **extremely fast** (no network, no DB access, no file access). They must be run before committing changes.
- **Integration tests** are slower and test **end-to-end** flows in the entire system. They are typically run after commit, on an integration server.
- **Automated acceptance tests** validate the **behavior** of the system, from an **end-user perspective**. They also need to be run on a frequent basis, but they usually take a long time to execute.
- **User acceptance tests** give feedback that is difficult to get with an automated approach (subjective evaluation, exploratory testing, usability, etc.)
- **What kind of tools can we use to manage automated acceptance tests?**





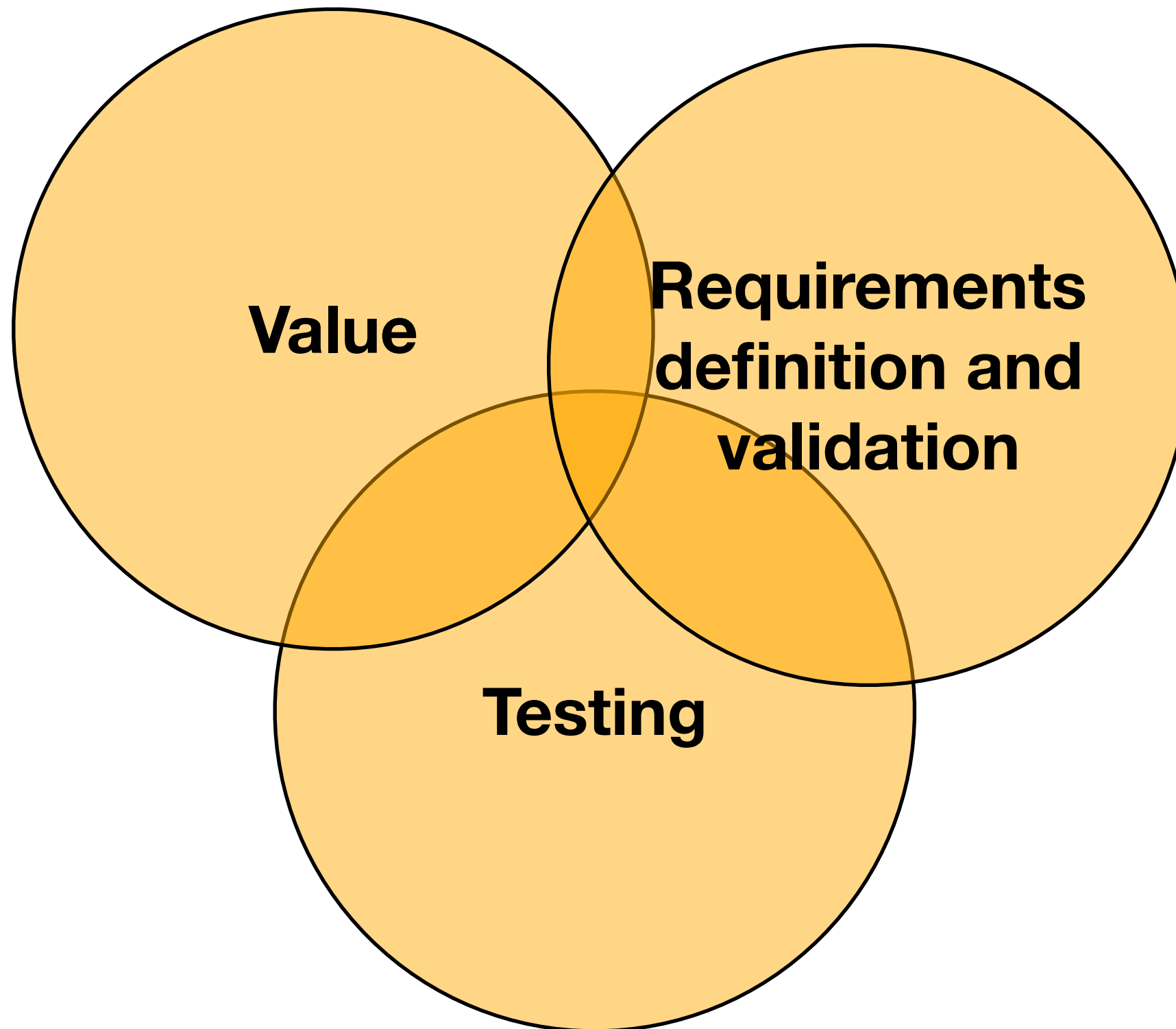
# Testing is not an After-the-Fact Activity

---

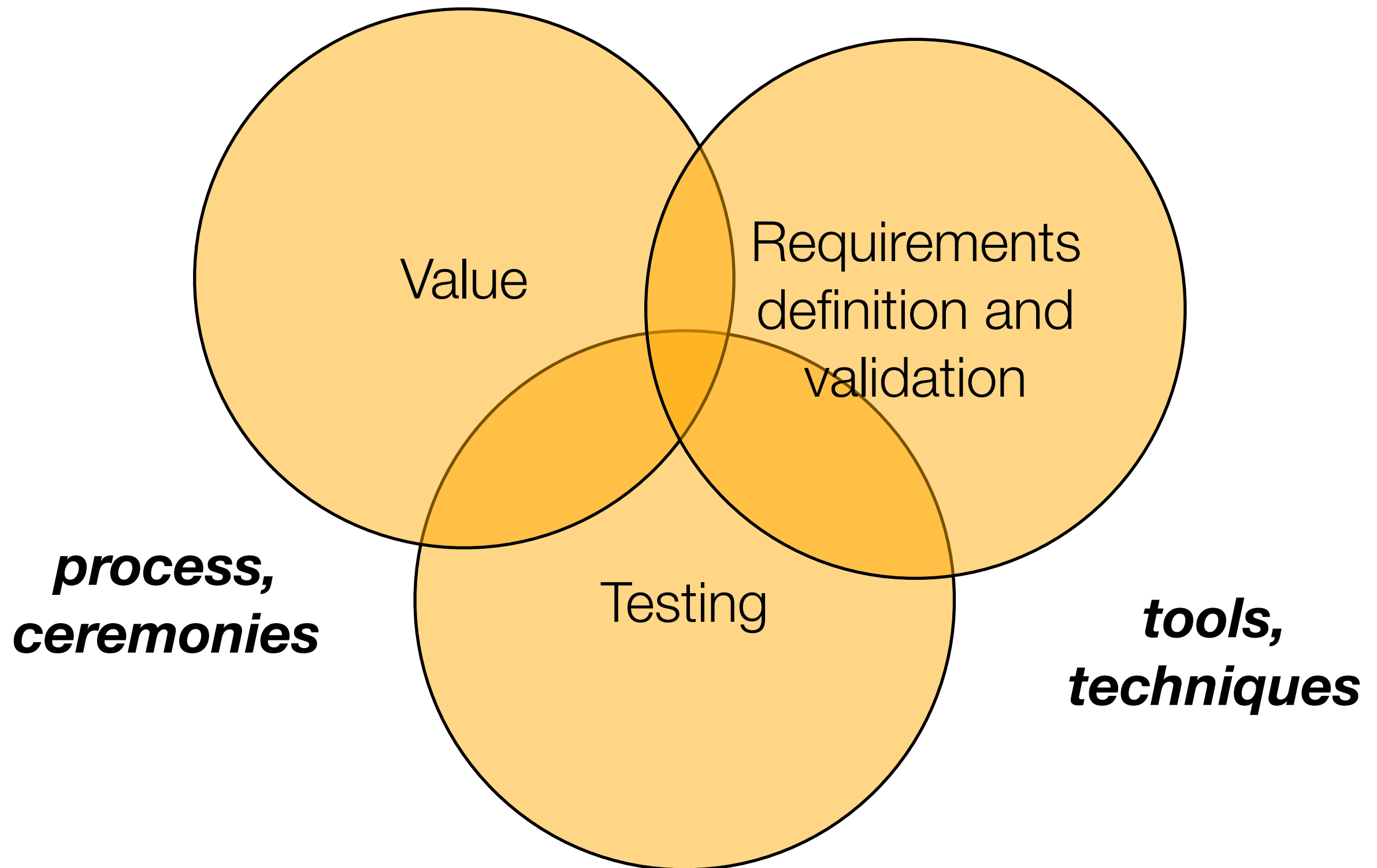
- **Writing tests** if not only about checking that a piece of code is doing what it is supposed to do.
- **Writing unit tests** helps us **design** better software (more modular, easier to maintain, easier to evolve). This is the core idea of Test-Driven Development (TDD).
- Writing unit tests, i.e. building a **test harness**, allows us to **refactor** our code with confidence.
- **If unit tests help us design, could other tests help us specify the... behavior of our system? Could they also give us confidence that we don't impact end-users with our changes?**



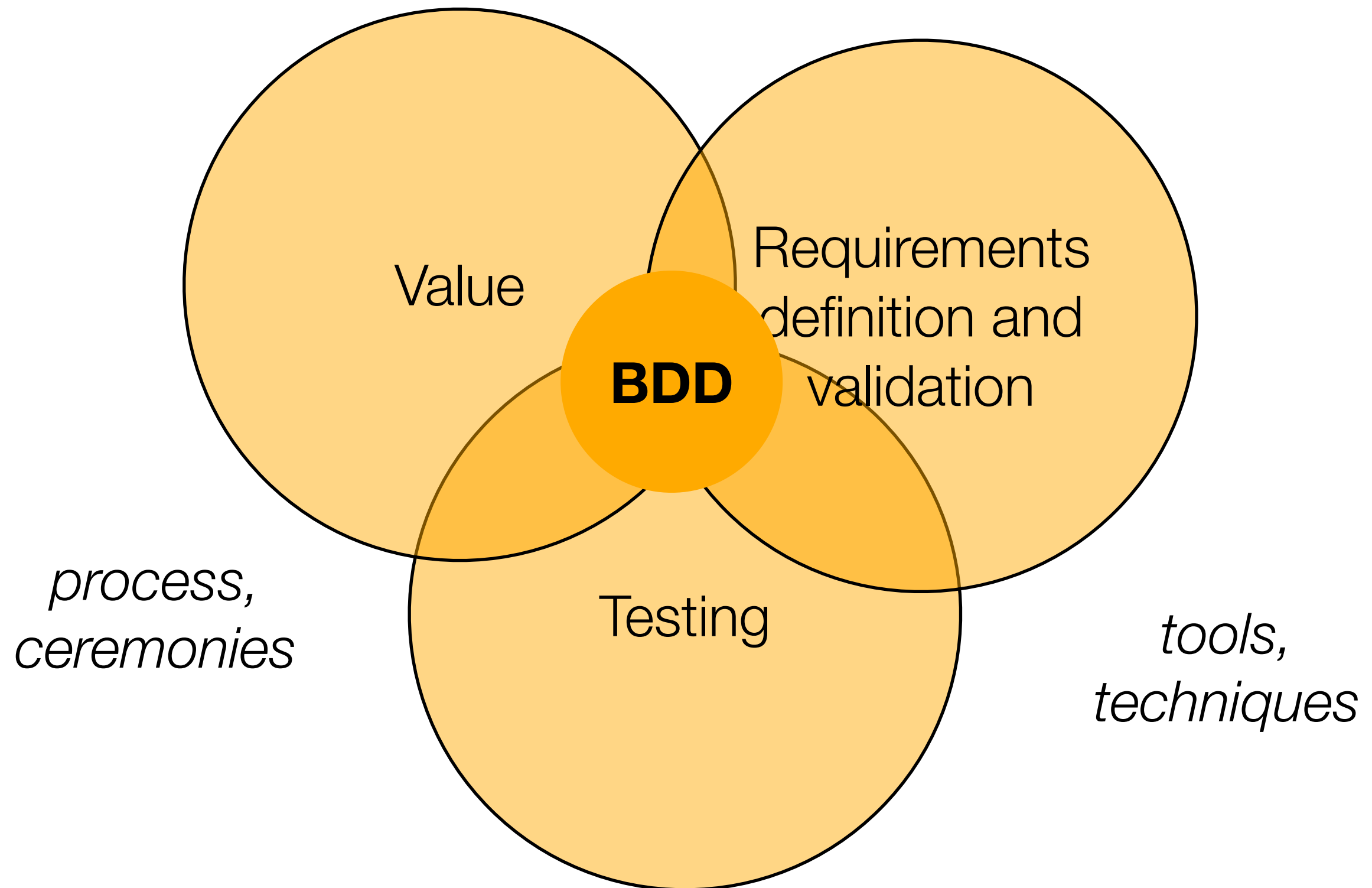
*A harness can be useful and cool.*



***mindset,  
common language***



*mindset,  
common language*





<http://dannorth.net/introducing-bdd/>

***“I had a problem.** While using and teaching agile practices like test-driven development (TDD) on projects in different environments, I kept coming across the same **confusion** and **misunderstandings**.*

*Programmers wanted to know **where to start, what to test** and **what not to test, how much to test in one go, what to call their tests, and how to understand why a test fails.**”*

*Dan North, 2006*

*“The deeper I got into TDD, the more I felt that **my own journey** had been less of a wax-on, wax-off process of gradual mastery than **a series of blind alleys**.*

*I remember thinking “If only someone had told me that!” far more often than I thought “Wow, a door has opened.” I decided it must be possible to **present TDD in a way that gets straight to the good stuff** and avoids all the pitfalls.”*

*Dan North, 2006*

*“My response is **behaviour-driven development** (BDD). It has evolved out of established agile practices and is designed to make them more accessible and effective for teams new to agile software delivery.*

*Over time, BDD has grown to **encompass the wider picture of agile analysis and automated acceptance testing.**”*

*Dan North, 2006*

# BDD: Naming & Vocabulary Matters

---

- “Test method names should be sentences”.
  - Compare the two representations of the same “specification”. It suggests that tools can support communication by emphasizing a common language for the domain.
- see <http://agiledox.sourceforge.net/>

```
public class FooTest extends TestCase {  
    public void testIsASingleton() {}  
    public void testAreallyLongNameIsAGoodThing() {}  
}
```

```
Foo  
- is a singleton  
- a really long name is a good thing
```



# BDD: “Ubiquitous Language” for Analysis

- BDD proposes a template to describe the intended behavior of a system. The template is used to specify the acceptance criteria for a given user story.

**Given** some initial context (the givens),  
**When** an event occurs,  
**then** ensure some outcomes.

## USER STORY

**As a** customer,  
**I want to** withdraw cash from  
an ATM,  
**so that** I don't have to wait  
in line at the bank.

## ACCEPTANCE CRITERIA

**Given** the account is in credit  
**A** And the card is valid  
**G** And the dispenser contains cash  
**A** **When** the customer requests cash  
**A** **Then** ensure the account is debited  
**W** And ensure cash is dispensed  
**T** And ensure the card is returned  
**A**  
And ensure the card is returned

# BDD: Executable Specifications

---

- “Acceptance criteria should be executable”
- We need tools that allow:
  - **analysts** to write the acceptance criteria in plain english, following the previous template;
  - **developers** to write test fixtures that act as intermediary between the specification and the system to test;
  - the **continuous delivery pipeline** to execute the specifications automatically, to integrate the test results in the “live” specification, to notify the team about the results.



# Process : When will be done?

---

**Scenario:** trader is not alerted below threshold

**Given** a stock of symbol STK1 and a threshold of 10.0

**When** the stock is traded at 5.0

**Then** the alert status should be OFF



Executable  
Specifications



*Acceptance criteria for stories are defined as scenarios.*

# Process : linking the specs with the system



Executable  
Specifications

**Scenario:** trader is not alerted below threshold

**Given** a stock of symbol STK1 and a threshold of 10.0

**When** the stock is traded at 5.0

**Then** the alert status should be OFF

Test Fixtures

```
public class TraderSteps { // look, Ma, I'm a POJO!!

    private Stock stock;

    @Given("a stock of symbol $symbol and a threshold
of $threshold")
    public void aStock(String symbol, double threshold)
    {
        stock = new Stock(symbol, threshold);
    }

    @When("the stock is traded at $price")
    public void theStockIsTradedAt(double price) {
        stock.tradeAt(price);
    }

    @Then("the alert status should be $status")
    public void theAlertStatusShouldBe(String status) {
        ensureThat(stock.getStatus().name(),
equalTo(status));
    }
}
```

System Under  
Test  
(SUT)



# Process : let's see if we are done...

**Scenario:** trader is not alerted below threshold

**Given** a stock of symbol STK1 and a threshold of 10.0

**When** the stock is traded at 5.0

**Then** the alert status should be **OFF**



Executable  
Specifications



*The test results are displayed directly in the “living” specs  
(other reports and notifications are also useful!)*

# Process : yeah!!!!!!

**Scenario:** trader is not alerted below threshold

**Given** a stock of symbol STK1 and a threshold of 10.0

**When** the stock is traded at 5.0

**Then** the alert status should be **OFF**



Executable  
Specification



*The test results are displayed directly in the “living” specs  
(other reports and notifications are also useful!)*

# Process : nooooooooooooo....

---

**Scenario:** trader is not alerted below threshold

**Given** a stock of symbol STK1 and a threshold of 10.0

**When** the stock is traded at 5.0

**Then** the alert status should be **OFF**



Executable  
Specifications

**REJECTED**



*The test results are displayed directly in the “living” specs  
(other reports and notifications are also useful!)*



***I can't wait to get started... what should I do?***

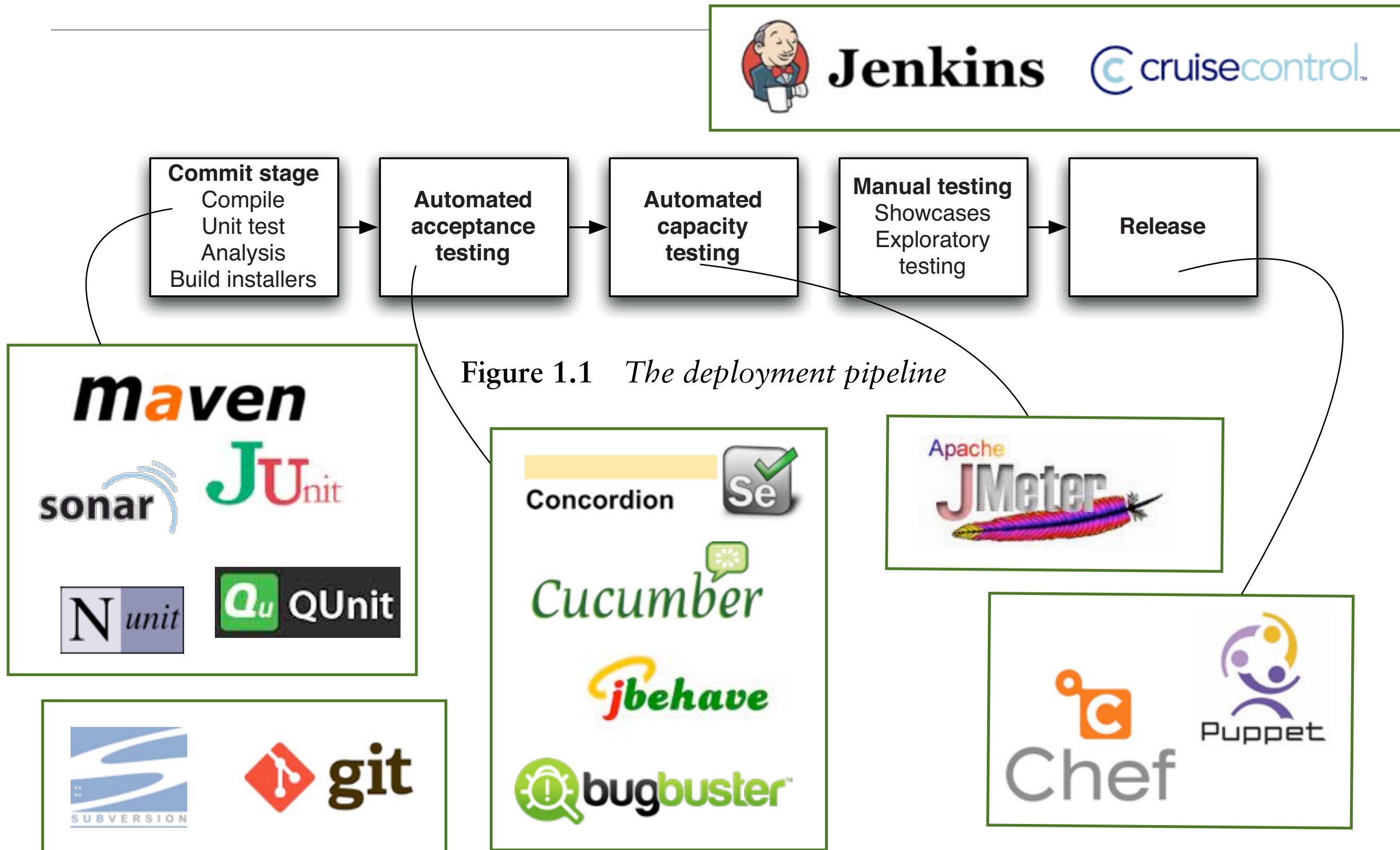


**MOTIVATION.**

Get off your ass.



# Tools



# Tools

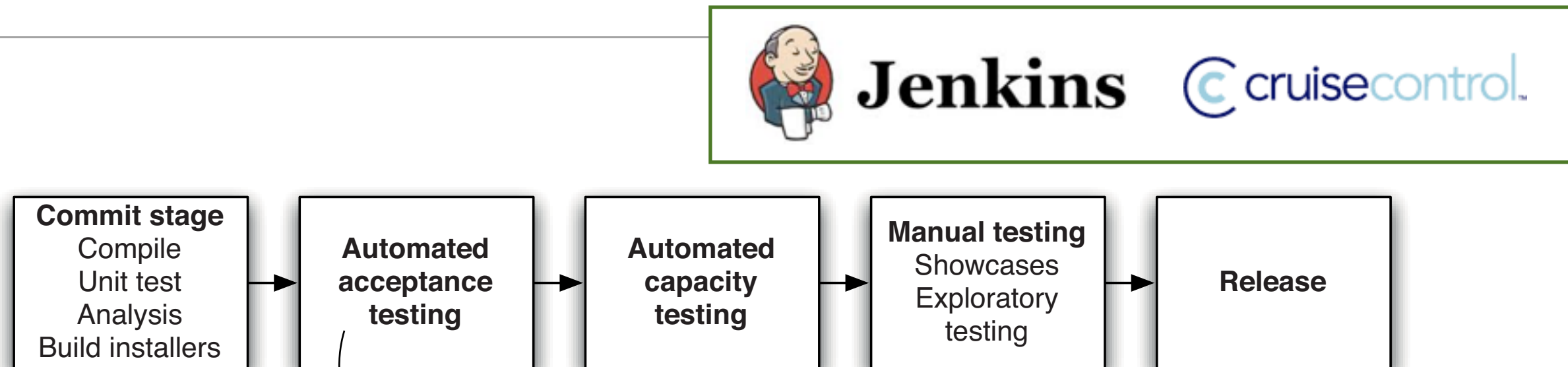
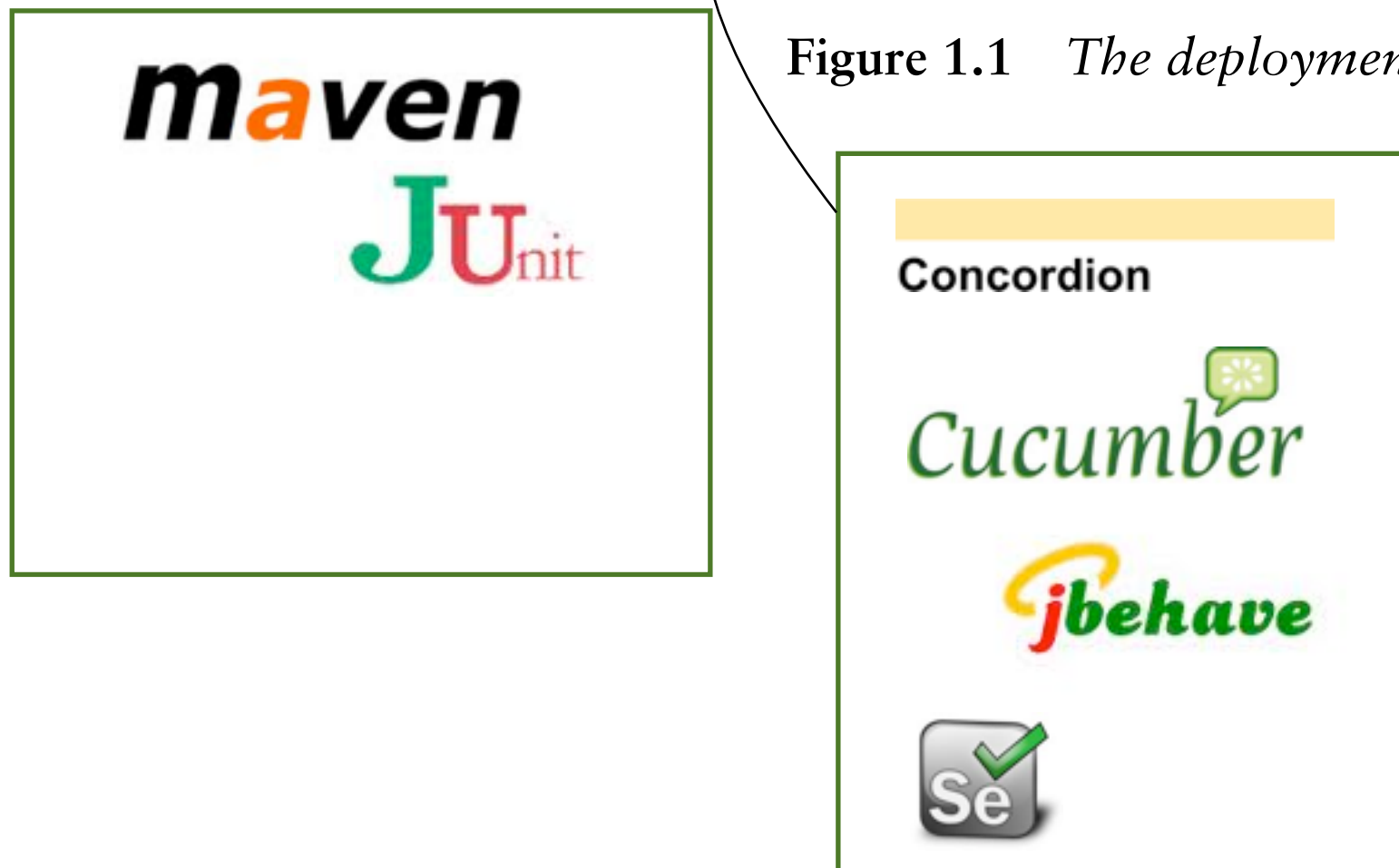


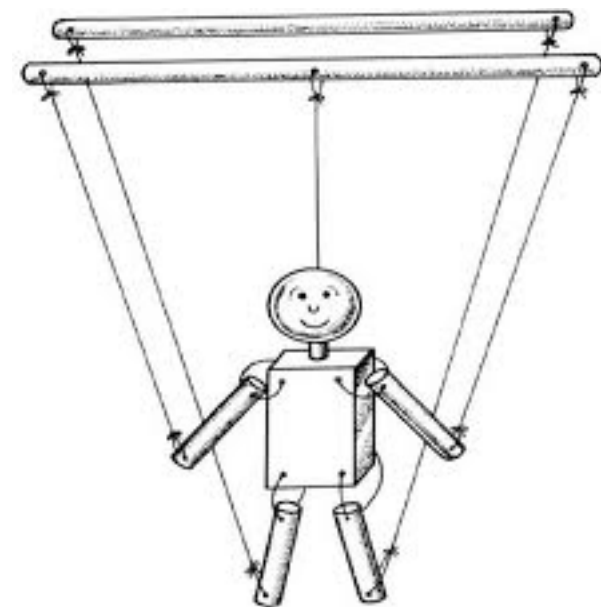
Figure 1.1 *The deployment pipeline*



# Selenium

---

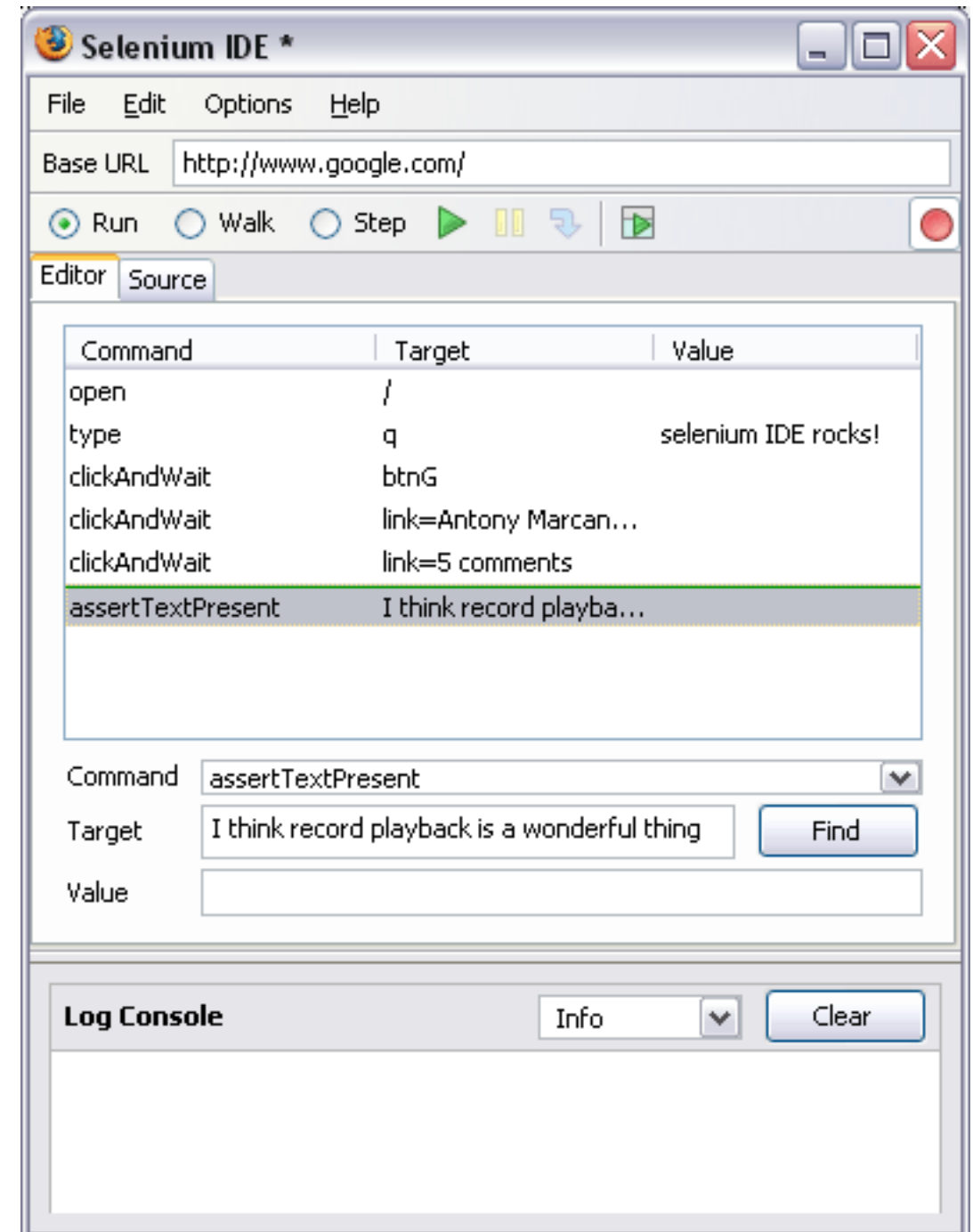
- Selenium provides means to **control web browsers** and to automate tasks, including testing.
- Selenium is a **suite of tools**:  
Selenium IDE, Selenium Web Driver, Selenium Server.
- With Selenium, it is possible to **record** browsing sessions and to **replay** them. An **API** is also available.
- Selenium is one of the most popular tools for **automated testing of web applications**.



<http://seleniumhq.org/>

# Selenium IDE

- With Selenium IDE, you can record web browsing sessions.
- You can then replay them (Selenium is actually controlling the web browser, so this is testing the real user experience).
- You can also edit your scripts and add tests and assertions.



# Selenium Web Driver

- Web Driver provides you with an API to write automated web tests.
- In this case, Selenium still controls a real web browser.
- There are patterns to write good, maintainable tests (in particular, the “Page Object” pattern)
- This is important, because the Web UI is often very dynamic (in the sense that it changes often). You don’t want to have to rewrite all of your tests when a page layout is redone...

```
public class Selenium2Example {
    public static void main(String[] args) {
        // Create a new instance of the Firefox driver
        WebDriver driver = new FirefoxDriver();

        // And now use this to visit Google
        driver.get("http://www.google.com");

        // Find the text input element by its name
        WebElement element = driver.findElement(By.name("q"));

        // Enter something to search for
        element.sendKeys("Cheese!");

        // Now submit the form. WebDriver will find the form for us from
        the element
        element.submit();

        // Check the title of the page
        System.out.println("Page title is: " + driver.getTitle());

        // Google's search is rendered dynamically with JavaScript.
        // Wait for the page to load, timeout after 10 seconds
        (new WebDriverWait(driver, 10)).until(new
        ExpectedCondition<Boolean>() {
            public Boolean apply(WebDriver d) {
                return d.getTitle().toLowerCase().startsWith("cheese!");
            }
        });

        // Should see: "cheese! - Google Search"
        System.out.println("Page title is: " + driver.getTitle());

        //Close the browser
        driver.quit();
    }
}
```

```
public class Selenium2Example {
    public static void main(String[] args) {
        // Create a new instance of the Firefox driver
        WebDriver driver = new FirefoxDriver();

        // And now use this to visit Google
        driver.get("http://www.google.com");

        // Find the text input element by its name
        WebElement element = driver.findElement(By.name("q"));

        // Enter something to search for
        element.sendKeys("Cheese!");

        // Now submit the form. WebDriver will find the form for us from the element
        element.submit();

        // Check the title of the page
        System.out.println("Page title is: " + driver.getTitle());

        // Google's search is rendered dynamically with JavaScript.
        // Wait for the page to load, timeout after 10 seconds
        (new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>() {
            public Boolean apply(WebDriver d) {
                return d.getTitle().toLowerCase().startsWith("cheese!");
            }
        });

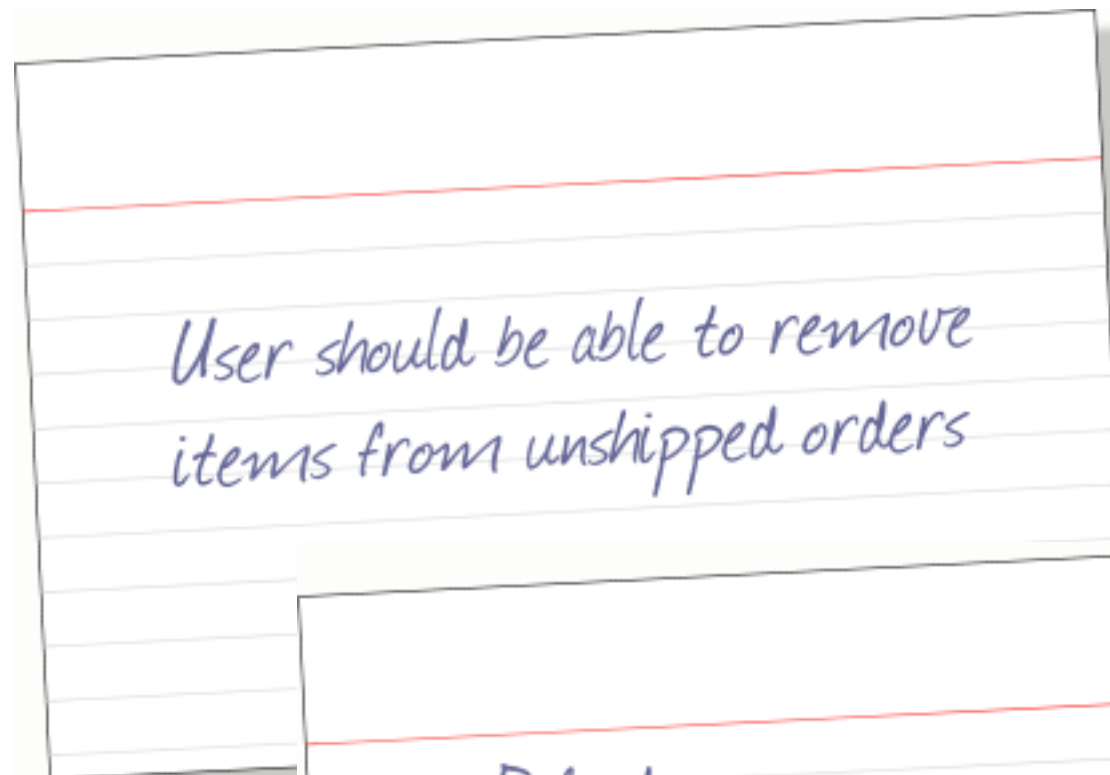
        // Should see: "cheese! - Google Search"
        System.out.println("Page title is: " + driver.getTitle());

        //Close the browser
        driver.quit();
    }
}
```

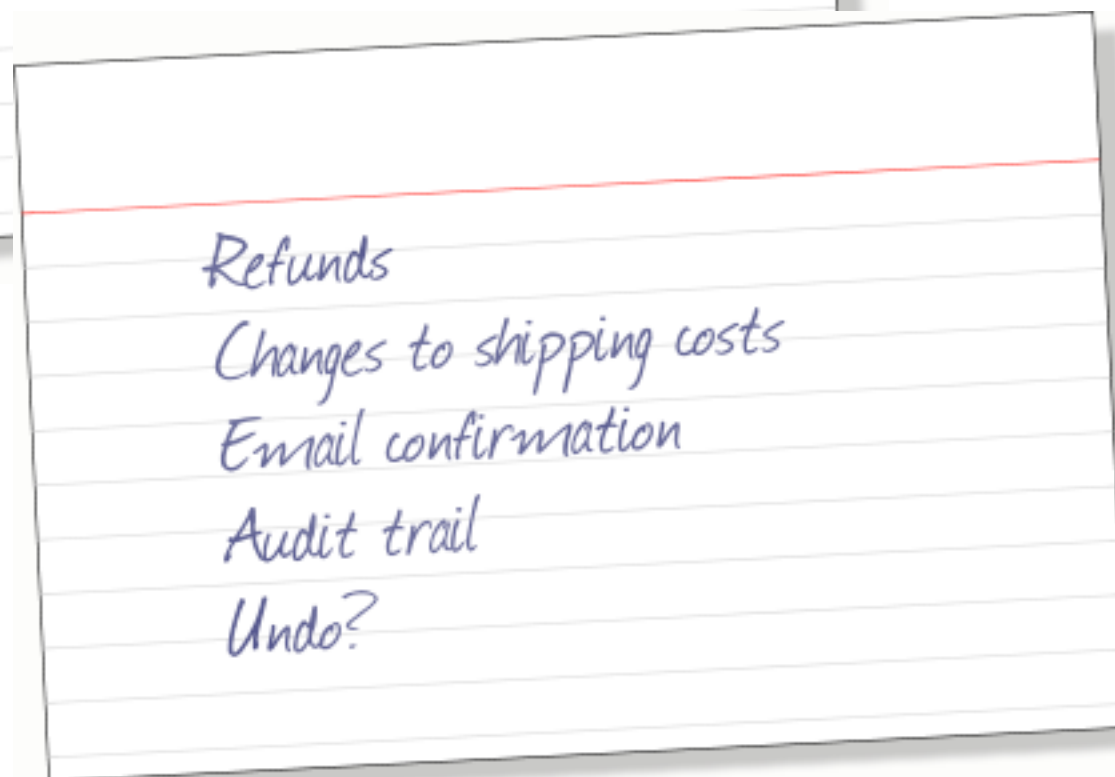


# Concordion

---



*front of the story card*



*back of the story card  
(acceptance criteria)*

# Concordion

User should be able to remove items from unshipped orders

[Stories](#) > [Iteration 19](#) >

## Removing Items From Unshipped Orders

**Owner:** Sharon Hargreaves

**QA:** Manish Gupta

### Automated

- [Check item removal is only available for unshipped orders](#)
- [Check price is updated \(refunds / shipping costs\)](#)
- [Check e-mail confirmation is sent](#)
- [Check changes are audited](#)

### Manual

- Check user-interface changes are consistent with rest of system
- Attempt to remove items from a shipped order (incl. URL manipulation)

### Out of Scope

- "Undo" functionality
- Analysis / reporting of suspicious activity
- Handling bounced e-mail

# Concordion

Refunds

Changes to shipping costs

Email confirmation

Audit trail

Undo?

[Online Shop](#) > [Orders](#) > [Unshipped](#) > [Item Removal](#) >

## Item Refund

For orders that are not subject to shipping charges, removal of an item results in a refund of the item's price.

### Example

Given an unshipped order with no shipping charges, containing the following items:

Item Description	Price Incl. Tax (£)
Kettle	33.25
Dictionary	18.49
Camera	249.95

If the shopper removes the **Dictionary** from her order then she will be refunded **£18.49**.

The order now contains:

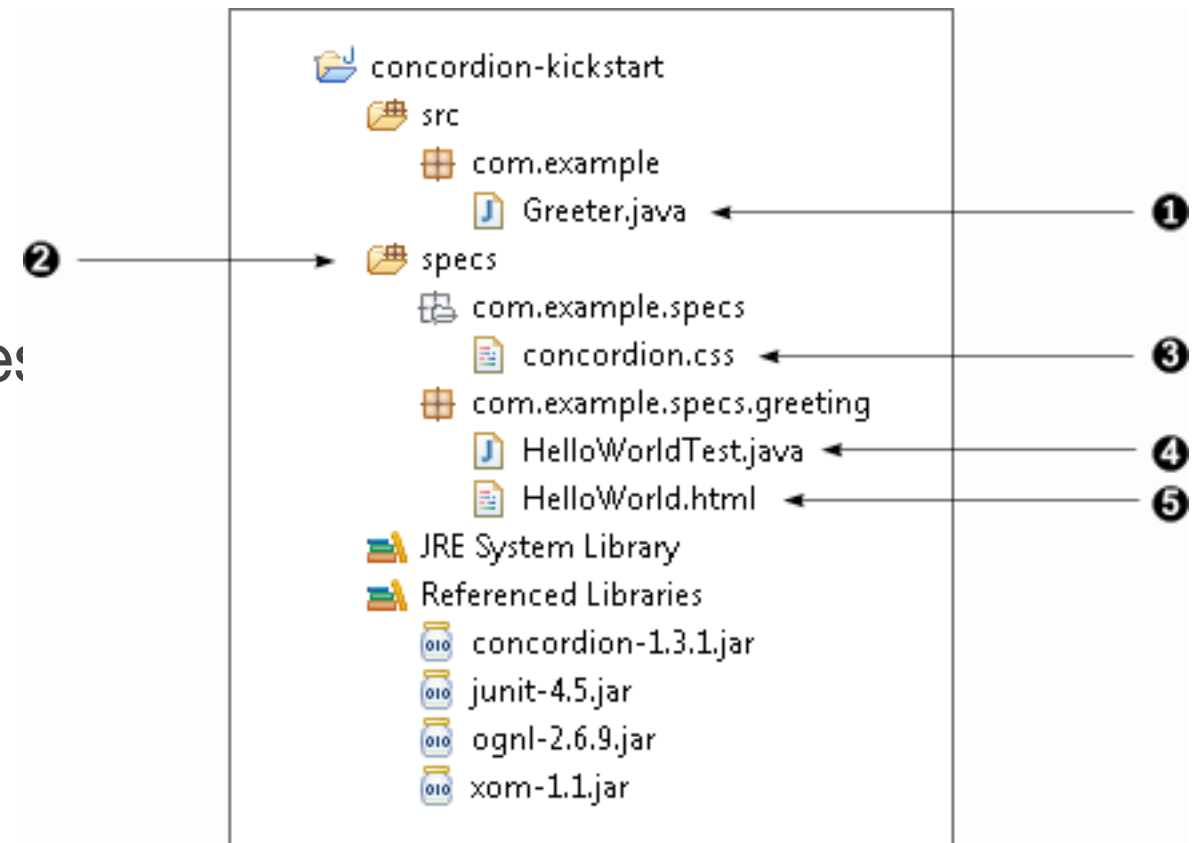
Item Description	Price Incl. Tax (£)
Kettle	33.25
Camera	249.95

### Further details

- [How are shipping charges handled?](#)
- [What happens if the order only contains one item?](#)
- [How are the refunds processed?](#)

# Concordion

- You write **acceptance tests in HTML** documents.
- You use **special HTML tags** to invoke test fixtures and to make assertions.
- You write **text fixtures as JUnit** tests.
- Executing the JUnit tests generates standard reports AND generates a collection of HTML documents (the living spec).
- You can use (and write) extensions to integrate **Selenium tests** (and even include snapshots in the living specs!)



# Acceptance Test & Fixture

Hello World!

Results generated by **Concordion™**  
in 97 ms on 18-Sep-2007 at 16:20:49 BST

HelloWorld.html

```
<html xmlns:concordion="http://www.concordion.org/2007/concordion">
  <body>
    <p concordion:assertEquals="getGreeting()">Hello World!</p>
  </body>
</html>
```

HelloWorldTest.java

```
package example;

import org.concordion.integration.junit3.ConcordionTestCase;

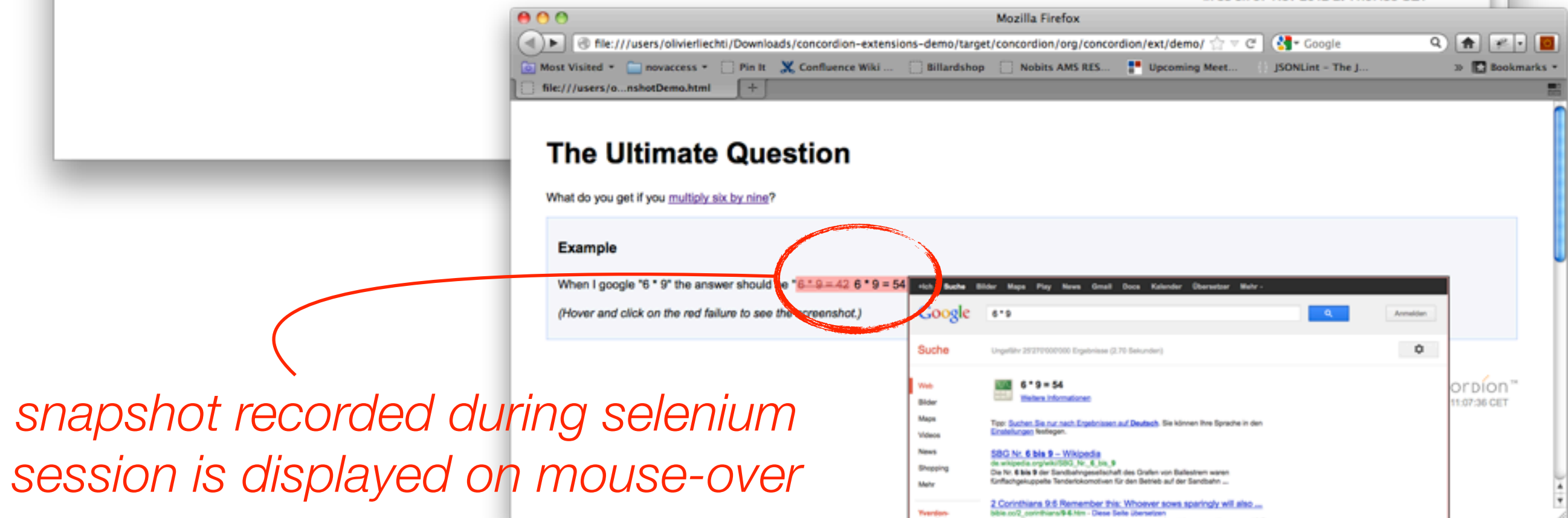
public class HelloWorldTest extends ConcordionTestCase {

    public String getGreeting() {
        return "Hello World!";
    }

}
```









## 1. Write story

Plain  
text

Scenario: A trader is alerted of status

Given a stock and a threshold of 15.0

When stock is traded at 5.0

Then the alert status should be OFF

When stock is traded at 16.0

Then the alert status should be ON

## 2. Map steps to Java

POJO

```
public class TraderSteps {  
    private TradingService service; // Injected  
    private Stock stock; // Created  
  
    @Given("a stock and a threshold of $threshold")  
    public void aStock(double threshold) {  
        stock = service.newStock("STK", threshold);  
    }  
    @When("the stock is traded at price $price")  
    public void theStockIsTraded(double price) {  
        stock.tradeAt(price);  
    }  
    @Then("the alert status is $status")  
    public void theAlertStatusIs(String status) {  
        assertThat(stock.getStatus().name(), equalTo(status));  
    }  
}
```

### 3. Configure Stories

Only  
once

```
public class TraderStories extends JUnitStories {  
  
    public Configuration configuration() {  
        return new MostUsefulConfiguration()  
            .useStoryLoader(new LoadFromClasspath(this.getClass()))  
            .useStoryReporterBuilder(new StoryReporterBuilder()  
                .withCodeLocation(codeLocationFromClass(this.getClass()))  
                .withFormats(CONSOLE, TXT, HTML, XML));  
    }  
  
    public List<CandidateSteps> candidateSteps() {  
        return new InstanceStepsFactory(configuration(),  
            new TraderSteps(new TradingService()))  
            .createCandidateSteps();  
    }  
  
    protected List<String> storyPaths() {  
        return new StoryFinder().findPaths(codeLocationFromClass(this.getClass()),  
            "**/*.story");  
    }  
}
```

### 4. Run Stories

With  
any of



IntelliJ**IDEA** **maven**

## 5.View Reports

HTML

### Scenario: A trader is alerted of status

Given a stock and a threshold of 15.0  
When stock is traded at 5.0  
Then the alert status is OFF  
When stock is traded at 16.0  
Then the alert status is ON

***Ok, now what?***



REALITY

is overrated. Who needs it?