A Journey in Software Development

An overview of methods and tools (part 2)

Mathieu Acher

Maître de Conférences

mathieu.acher@irisa.fr

UNIVERSITÉ DE RENNES 1

istic Informatique Électronique

Material

https://github.com/acherm/teaching/tree/master/PDL/

Previously

Project

- A1 and B1
  - advices prototyping UIs and modeling
- C1
  - documentation, refactoring and a bit of testing
- Deadlines are still hard
- Bonus for C1
  - Deadline is still 18th december
    - Evaluation of the december version
  - But you can deliver a new version the 10th of January
    - And get 2 bonus points

Today

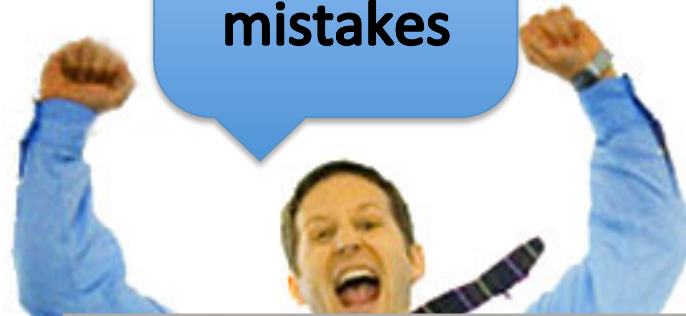
- Testing (I want to insist)
- Debugging
- The links between documenting, testing, refactoring, debugging, and design patterns
- A few words about Maven
- Revision Control Systems
  - SVN (centralized)
  - GIT (distributed)

Testing

…the activity of finding out whether a piece of code (a method, class or program) produces the intended behavior

Your **hope** as a programmer

« A program does exactly what you expected to do »


I don't
make
mistakes

Microsoft Visual Studio
Package Load Failure
Package 'ArtOfTest.WebAiiVSIP.WebAiiVSIP, ArtOfTest.WebAiiVSIP, Version=1.0.0.0, Culture=neutral, PublicKeyToken=14176981d1873d86' has failed to load properly (GUID = {D5EDEC6D-8025-40CF-987C-B8EABA2C7451}). Please contact package vendor for assistance. Application restart is recommended, due to possible environment corruption. Would you like to disable loading this package in the future? You may use 'devenv /resetskippkgs' to re-enable package loading.
Yes
No

Test phases

Unit testing on individual units of source code (=smallest testable part).

Integration testing on groups of individual software modules.

System testing on a complete, integrated system (evaluate compliance with requirements)

Running example

1. Set of products
2. Number of products
3. Balance

1. Add product

Subtotal: CDN$ 10.94

Make any changes below? Update

Shopping Cart Items--To Buy Now		Price:	Qty:
Item added on April 26 2007 Save for later Delete	Harry Potter and the Half-Blood Prince (Book 6) [Adult Edition] - J. K. Rowling; **Mass Market Paperback** In Stock	CDN$ 10.94 You Save: CDN$ 4.05 (27%)	1

2. Remove product

Init

Constructor + Set up and tear down of fixture.

```
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

public class ShoppingCartTest extends TestCase {

    private ShoppingCart _bookCart;

    // Creates a new test case
    public ShoppingCartTest(St
        super(name);
    }
```

```
// Creates test environment (fixture).
// Called before every testX() method.
protected void setUp() {
    _bookCart = new ShoppingCart();

    Product book = new Product("Harry Potter", 23.95);
    _bookCart.addItem(book);
}

// Releases test environment (fixture).
// Called after every testX() method.
protected void tearDown() {
    _bookCart = null;
}
```

Assertions

`fail(msg)` – triggers a failure named *msg*

`assertTrue(msg, b)` – triggers a failure, when condition *b* is false

`assertEquals(msg, v1, v2)` – triggers a failure, when $v1 \neq v2$

`assertEquals(msg, v1, v2, ϵ)` – triggers a failure, when $|v1 - v2| > \epsilon$

`assertNull(msg, object)` – triggers a failure, when *object* is not *null*

`assertNonNull(msg, object)` – triggers a failure, when *object* is *null*

Example #1

```
// Tests adding a product to the cart.
public void testProductAdd() {
    Product book = new Product("Refactoring", 53.95);
    _bookCart.addItem(book);

    assertTrue(_bookCart.contains(book));

    double expected = 23.95 + book.getPrice();
    double current = _bookCart.getBalance();

    assertEquals(expected, current, 0.0);

    int expectedCount = 2;
    int currentCount = _bookCart.getItemCount();

    assertEquals(expectedCount, currentCount);
}
```

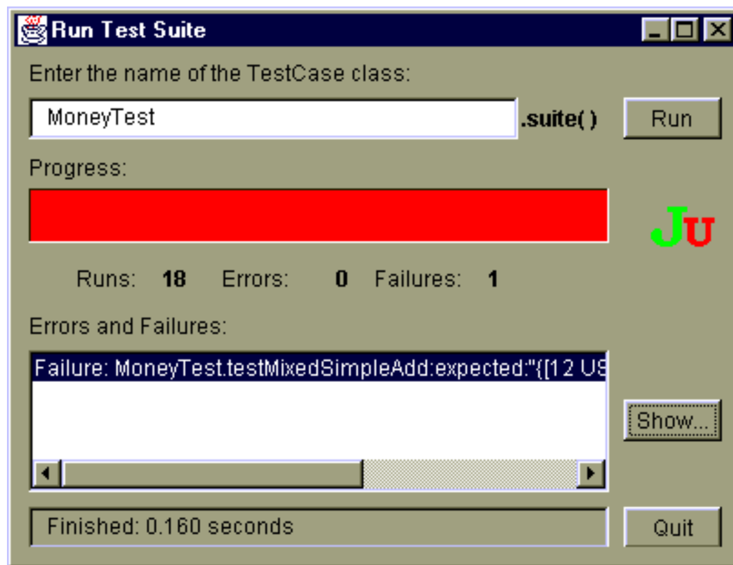
Example #2

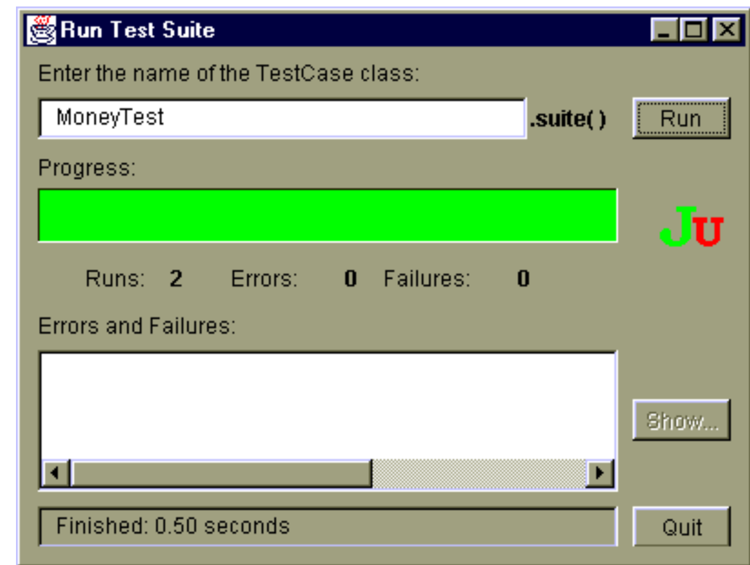
```java
// Tests removing a product from the cart.
public void testProductRemove() throws NotFoundException {
        Product book = new Product("Harry Potter", 23.95);
        _bookCart.removeItem(book);

        assertTrue(!_bookCart.contains(book));

        double expected = 23.95 - book.getPrice();
        double current = _bookCart.getBalance();

        assertEquals(expected, current, 0.0);

        int expectedCount = 0;
        int currentCount = _bookCart.getItemCount();

        assertEquals(expectedCount, currentCount);
}
```

```java
public static Test suite() {

    // Here: add all testX() methods to the suite (reflection).
    TestSuite suite = new TestSuite(ShoppingCartTest.class);

    // Alternative: add methods manually (prone to error)
    // TestSuite suite = new TestSuite();
    // suite.addTest(new ShoppingCartTest("testEmpty"));
    // suite.addTest(new ShoppingCartTest("testProductAdd"));
    // suite.addTest(...);

    return suite;
}
```

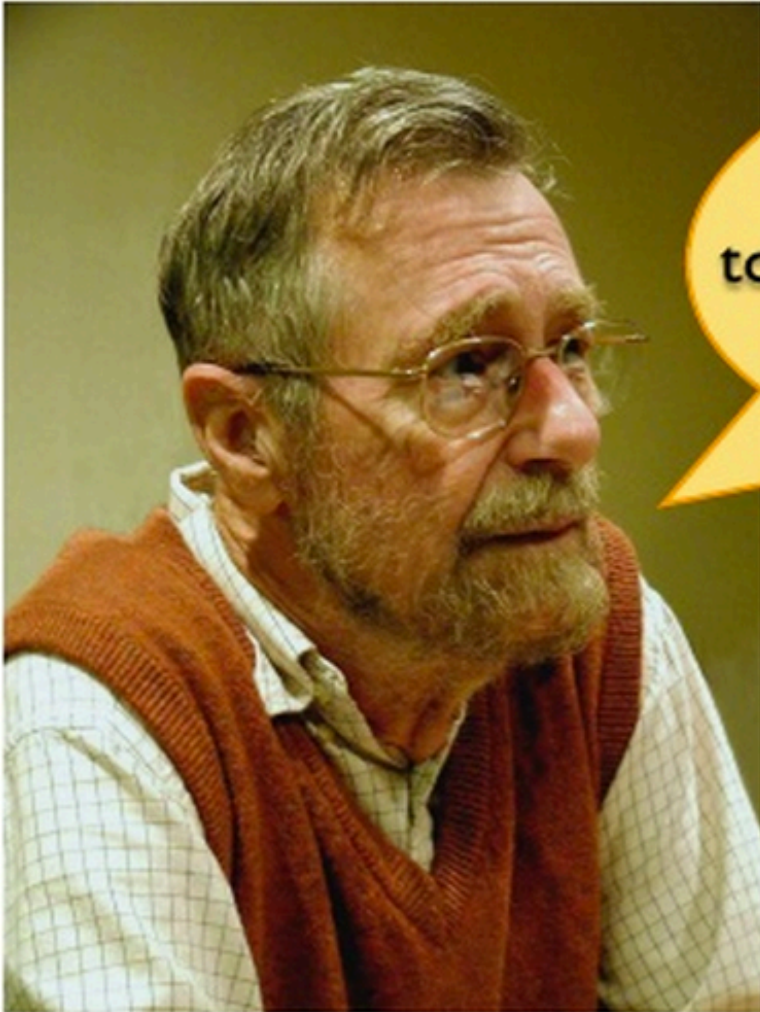
Unit Test
JUnit 3 and 4 http://www.junit.org

- Test pattern
  - Test, TestSuite, TestCase
  - Assertions (assertXX) that must be verified
- TestRunner
  - Chain tests and output a report.





Djikstra



Program testing can be used to show the presence of bugs, but never to show their absence!

You can't test everything
(so one advice by Martin Fowler)

Whenever you are tempted to type something into a print statement or a debugger expression, **write it as a test instead.**

Logging

Debugging

- Symbolic debugging
  - javac options: -g, -g:source,vars,lines
  - command-line debugger : jdb (JDK)
    - commands look like those of dbx
  - graphical « front-ends » for jdb (AGL)
  - Misc
    - Multi-threads, Cross-Debugging (-Xdebug) on remote VM , ...

Monitoring

- Tracer
  - TRACE options of the program
  - can slow-down .class with TRACE/←TRACE tests
    - solution : use a pre-compiler (excluding trace calls)
  - Kernel tools, like OpenSolaris DTrace (coupled with the JVM)

Logging

- Logging is chronological and systematic record of data processing events in a program
  - e.g. the Windows Event Log
- Logs can be saved to a persistent medium to be studied at a later time
- Use logging in the development phase:
  - Logging can help you **debug** the code
- Use logging in the production environment:
  - Helps you **troubleshoot problems**

Logging, why? (claims)

- Logging is easier than debugging
- Logging is faster than debugging
- Logging can work in environments where debugging is not supported
- Can work in production environments
- Logs can be referenced anytime in future as the data is stored

Logging Methods, How?

- The evil System.out.println()
- Custom Solution to Log to various datastores, eg text files, db, etc…
- Use Standard APIs
  - Don't reinvent the wheel

Log4J

- Popular logging frameworks for Java
- Designed to be reliable, fast and extensible
- Simple to understand and to use API
- Allows the developer to control which log statements are output with arbitrary granularity
- Fully configurable at runtime using external configuration files

Log4J Architecture

- Log4J has three main components: loggers, appenders and layouts
  - Loggers
    - Channels for printing logging information
  - Appenders
    - Output destinations (console, File, Database, Email/ SMS Notifications, Log to a socket, and many others…)
  - Layouts
    - Formats that appenders use to write their output
- Priorities

Logger

- Responsible for Logging
- Accessed through java code
- Configured Externally
- Every Logger has a name
- Prioritize messages based on level
  - TRACE, DEBUG, INFO, WARN, ERROR & FATAL
- Usually named following dot convention like java classes do.
  - Eg com.foo.bar.ClassName
- Follows inheritance based on name

Logger API

- Factory methods to get Logger
  - `Logger.getLogger(Class c)`
  - `Logger.getLogger(String s)`
- Method used to log message
  - `trace(), debug(), info(), warn(), error(), fatal()`
  - Details
    - `void debug(java.lang.Object message)`
    - `void debug(java.lang.Object message, java.lang.Throwable t)`
  - Generic Log method
    - `void log(Priority priority, java.lang.Object message)`
    - `void log(Priority priority, java.lang.Object message, java.lang.Throwable t)`

Root Logger

- The root logger resides at the top of the logger hierarchy. It is exceptional in two ways:
  1. it always exists,
  2. it cannot be retrieved by name.

- Logger.getRootLogger()

Appender

- Appenders put the log messages to their actual destinations.
- No programatic change is require to configure appenders
- Can add multiple appenders to a Logger.
- Each appender has its Layout.
- ConsoleAppender, DailyRollingFileAppender, FileAppender, JDBCAppender, JMSAppender, NTEventLogAppender, RollingFileAppender, SMTPAppender, SocketAppender, SyslogAppender, TelnetAppender

Layout

- Used to customize the format of log output.
- Eg. HTMLLayout, PatternLayout, SimpleLayout, XMLLayout
- Most commonly used is PatternLayout
  - Uses C-like syntax to format.
    - `Eg. "%-5p [%t]: %m%n`
    - `DEBUG [main]: Message 1 WARN [main]: Message 2`

Log4j Basics

- Who will log the messages?
  - The Loggers
- What decides the priority of a message?
  - Level
- Where will it be logged?
  - Decided by Appender
- In what format will it be logged?
  - Decided by Layout

Log4j in Action

```
// get a logger instance named "com.foo"
Logger  logger = Logger.getLogger("com.foo");

// Now set its level. Normally you do not need to set the
// level of a logger programmatically. This is usually done
// in configuration files.
logger.setLevel(Level.INFO);

Logger barlogger = Logger.getLogger("com.foo.Bar");

// This request is enabled, because WARN >= INFO.
logger.warn("Low fuel level.");

// This request is disabled, because DEBUG < INFO.
logger.debug("Starting search for nearest gas station.");

// The logger instance barlogger, named "com.foo.Bar",
// will inherit its level from the logger named
// "com.foo" Thus, the following request is enabled
// because INFO >= INFO.
barlogger.info("Located nearest gas station.");

// This request is disabled, because DEBUG < INFO.
barlogger.debug("Exiting gas station search");
```

Log4j Optimization & Best Practises

- User logger as private static variable
- Only one instance per class
- Name logger after class name
- Don’t use too many appenders
- Don’t use time-consuming conversion patterns (see javadoc)
- Use Logger.isDebugEnabled() if need be
- Prioritize messages with proper levels

You can't test everything
(so one advice by Martin Fowler)

> Whenever you are tempted to type something into a print statement or a debugger expression, **write it as a test instead.**

Documenting, Testing, Design Patterns, Refactoring, Debugging

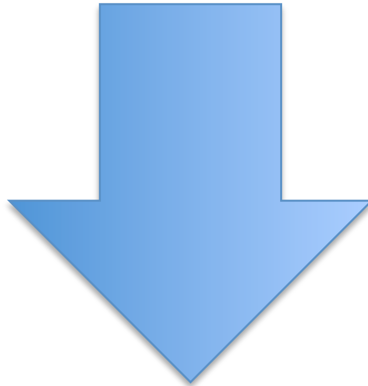
#1 What is the link?

- Documenting
  - Understanding (readability, maintainability)
- Refactoring
  - Improving the design (readability, maintainability, extensibility)
- The activity of documenting can somehow be replaced by the activity of refactoring
  - if the code and architecture is comprehensible by itself

refactoring.com

Documentation and Refactoring

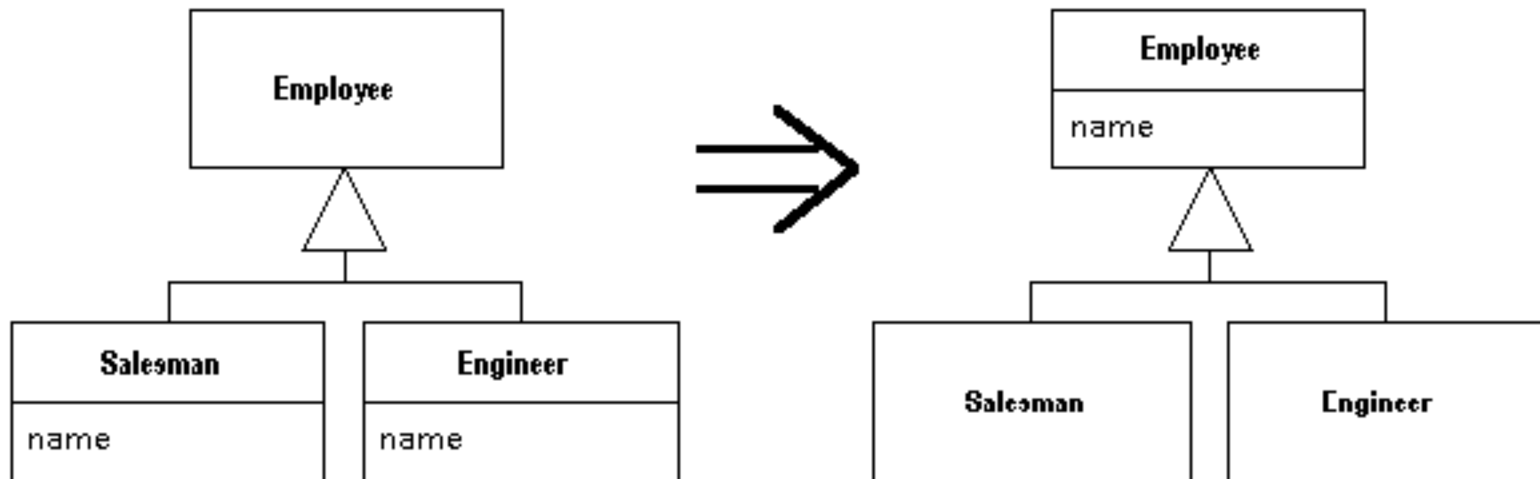
```
if ( (platform.toUpperCase().indexOf("MAC") > -1) &&  // platform is MacOS
      (browser.toUpperCase().indexOf("IE") > -1) && // brower is IE
       wasInitialized() && resize > 0 )
 {
   // do something
 }
```



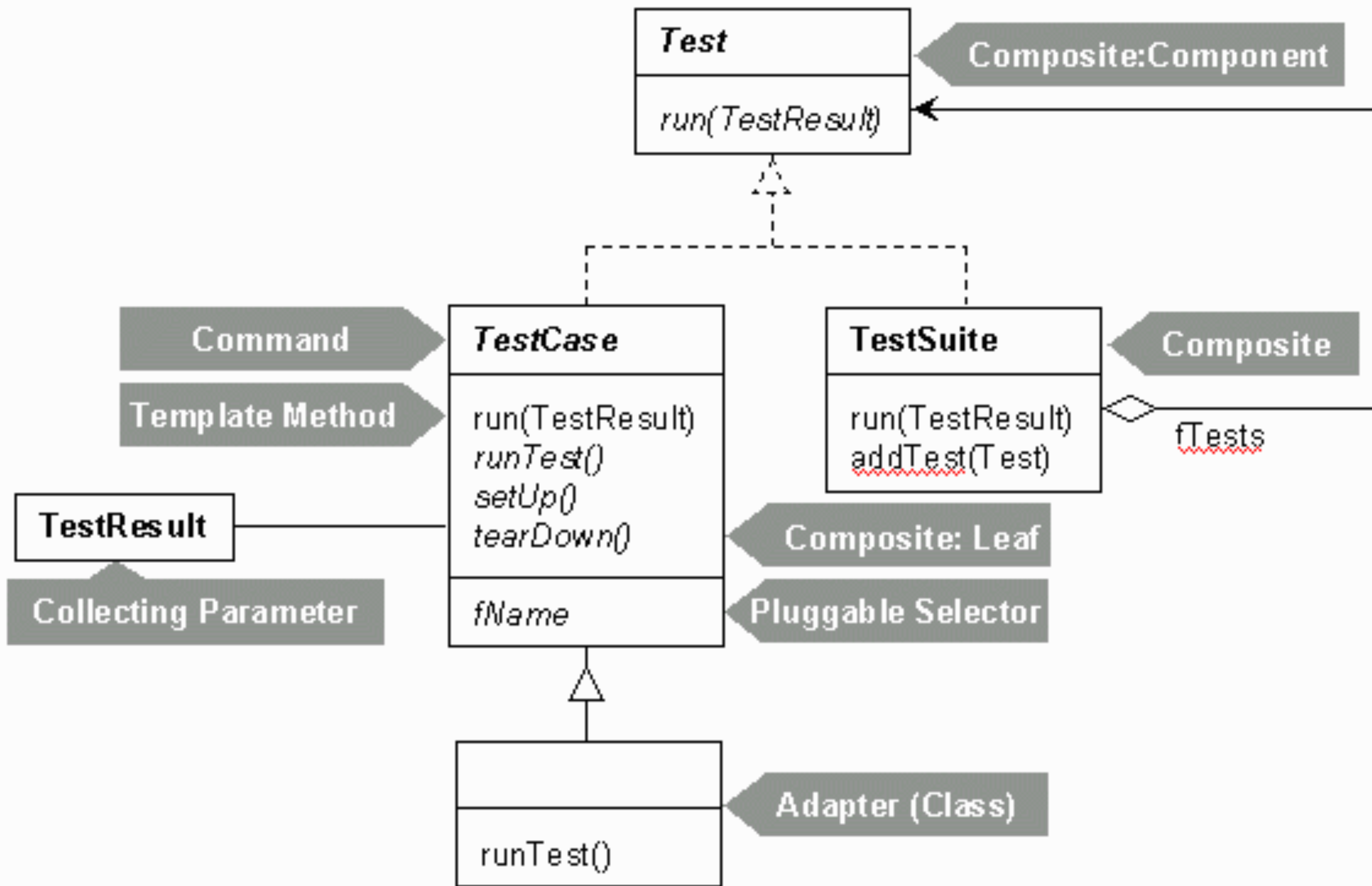
```
final boolean isMacOs     = platform.toUpperCase().indexOf("MAC") > -1;
final boolean isIEBrowser = browser.toUpperCase().indexOf("IE")  > -1;
final boolean wasResized  = resize > 0;

if (isMacOs && isIEBrowser && wasInitialized() && wasResized)
{
        // do something
}
```

#2 What is the link?

Design patterns: there are refactorings



JUnit and... Design patterns



Worth reading!

http://junit.sourceforge.net/doc/cookstour/cookstour.htm

#3 What is the link?

- Testing: “the activity of finding out whether a piece of code produces the intended behavior”
  - Debugging can help
  - Testing is better than debugging

Whenever you are tempted to type something into a print statement or a debugger expression, **write it as a test instead.**

What is the link?

- Testability
  - degree to which a system or component **facilitates the establishment** of test criteria and the performance of tests to determine whether those criteria have been met.”
  - Controllability + Observability
- **Controllability** ability to manipulate the software’s input as well as to place this software into a particular state
- **Observability** deals with the possibility to observe the outputs and state changes that
- How to improve Testability?
  - Refactoring, Design patterns

What is the link?

Testing/Refactoring/Design Patterns

- How to improve testability?
- Test-driven Development
  - Write tests first ~ Test-driven design

Let say your first piece of code is… a test

```
// Tests removing a product from the cart.
public void testProductRemove() throws NotFoundException {
    Product book = new Product("Harry Potter", 23.95);
    _bookCart.removeItem(book);

    assertTrue(!_bookCart.contains(book));

    double expected = 23.95 - book.getPrice();
    double current = _bookCart.getBalance();

    assertEquals(expected, current, 0.0);

    int expectedCount = 0;
    int currentCount = _bookCart.getItemCount();

    assertEquals(expectedCount, currentCount);
}
```

What is the link?

- Testing
- Documenting

- Unit tests are one of the best source of documentation
  - One of the entry point to understand a framework
  - It documents the properties of methods, how objects collaborate, etc.

What is the link?

Documenting

Refactoring

Debugging

Testing

Readability
Understandibility
Maintainability

Design

Document, refactor… Execute your tests… Debug.. Write test.. And so on!

Documenting

Debugging

Refactoring

Testing

```
package de.vogella.eclipse.ide.first;

public class MyFirstClass {

    public static void main(String[] args) {
        System.out.println("Hello Eclipse!");
        int sum = 0;
        for (int i = 0; i <= 100; i++) {
            sum += i;
        }
        System.out.println(sum);
    }
}
```

Extract Method

Method name: calculateSum

Access modifier: public protected default private

Parameters:

Type	Name
int	sum

Edit… Up Down

Generate method comment

Replace additional occurrences of statements with method

Method signature preview:

```
private static int calculateSum(int sum)
```

Preview > OK Cancel

Problems | Javadoc | Declaration | Console | Error Log

<terminated> MyFirstClass [Java Application] C:\Program Files\Java\jre7\bin\javaw.e

```
Hello Eclipse!
5050
```

Run Test Suite

Enter the name of the TestCase class:

MoneyTest .suite() Run

Progress:

JU

Errors and Failures:

Show…

Finished: 0.50 seconds Quit

With modern IDE and tools!

Compile Chain & Organizations

Compile chain

- Sometimes hidden in the IDE
  - But generally speaking, you need to master your “compile” chain
- Tools
  - make, gmake, nmake (Win),
  - Apache ANT, Apache MAVEN, Freshmeat 7Bee ...
- To **automate**:
  - pre-compilation, obfuscation, verification
  - generation of .class and .jar
    - normal, tracing, debug, …
  - documentation generation
  - « stubs » generation (rmic, idl2java, javacard …)
  - test
  - 3rd party libraries/dependencies

… And a **combination** of all these tasks

Maven

- Goal
  - Separation of concerns applied to project build
    - Compilation, code generation, unit testing, documentation, …
  - Handle project dependencies with versions (artifacts)
- Project object model (POM)
  - abstract description of the project
  - Property inheritance from POM parents
- Tools (called plugin)
  - To compile, generate documentation, automate test …

- Note: more and more useful !

Maven and POM

aka project's configurations

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.8.2</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Maven facilities and lifecycle

validate: validate the project is correct and all necessary information is available
compile: compile the source code of the project
test: test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
package: take the compiled code and package it in its distributable format, such as a JAR.
integration-test: process and deploy the package if necessary into an environment where integration tests can be run
verify: run any checks to verify the package is valid and meets quality criteria
install: install the package into the local repository, for use as a dependency in other projects locally
deploy: done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

clean: cleans up artifacts created by prior builds

site: generates site documentation for this project

Build the Project

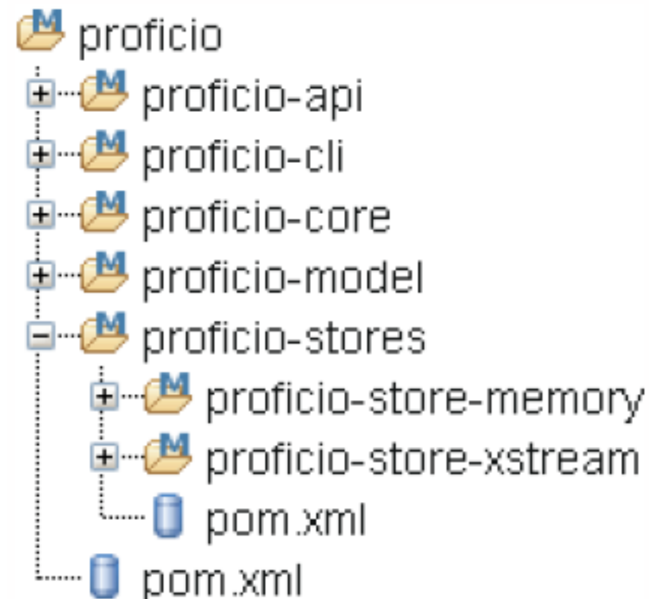
```
mvn package
```

Generating the Site

```
mvn site
```

Project hierarchies

- Motivations
  - Organize development in sub-projects
    - With N levels (N>=1)
- Technique
  - Create a super POM (type *pom*) for each nesting level
    - Place shared plugins/goals at the same level
  - Subprojects (called modules) inherit from this super pom
- Example



- Command
  - mvn clean install
    - Global construction

Maven

- Abstract project model (POM)
  - Object oriented, inheritance
  - Separation of concerns
- Default lifecycle
  - Default state (goals) sequence
    - plugins depend on states
- Give a project « standard » structure
  - Standard naming conventions
  - Standard lifecycle
- Automatic handling of dependencies between projects
  - Chargement des MAJ
- Project repositories
  - public or private, local or remotes
  - caching and proxy
- Extensible via external plugins

Maven plugins

- Core
  - clean, compiler, deploy, install, resources, site, surefire, verifier
- Packaging
  - ear, ejb, jar, rar, war, bundle (OSGi)
- Reporting
  - changelog, changes, checkstyle, clover, doap, docck, javadoc, jxr, pmd, project-info-reports, surefire-report
- Tools
  - ant, antrun, archetype, assembly, dependency, enforcer, gpg, help, invoker, one (interop Maven 1), patch, plugin, release, remote-resource, repository, scm
- IDEs
  - eclipse, netbeans, idea
- Others
  - exec, jdepend, castor, cargo, jetty, native, sql, taglist, javacc, obr …

Maven plugin for JAVA IDE

- Maven plugins exists for
  - Eclipse
  - Intellij
  - NetBeans
  - …

Relationship with PDL (your project)

Impacts

- Use/experiment with a subset of these tools
  - IDE in general (Eclipse, IntelliJ, etc.) and all services…
  - Refactoring
  - Testing
  - Documentation
  - Ant/Maven
  - Versioning systems
- You will have to in your professional career!