Tools to support development
in Java

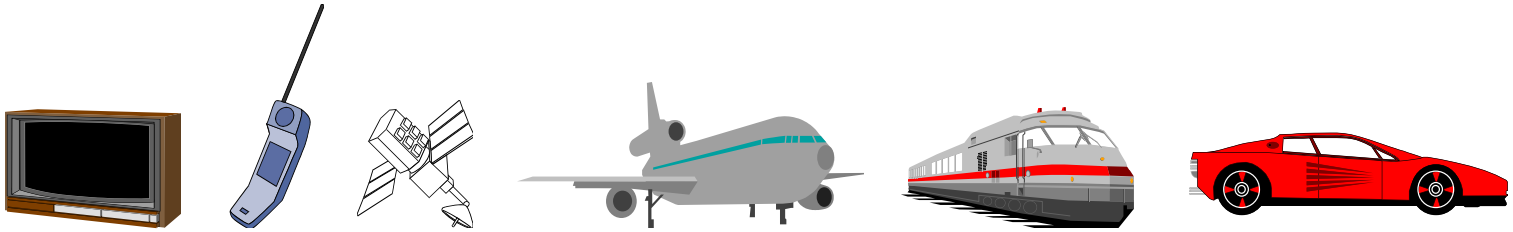
Mathieu Acher
Maître de Conférences
mathieu.acher@irisa.fr

Material

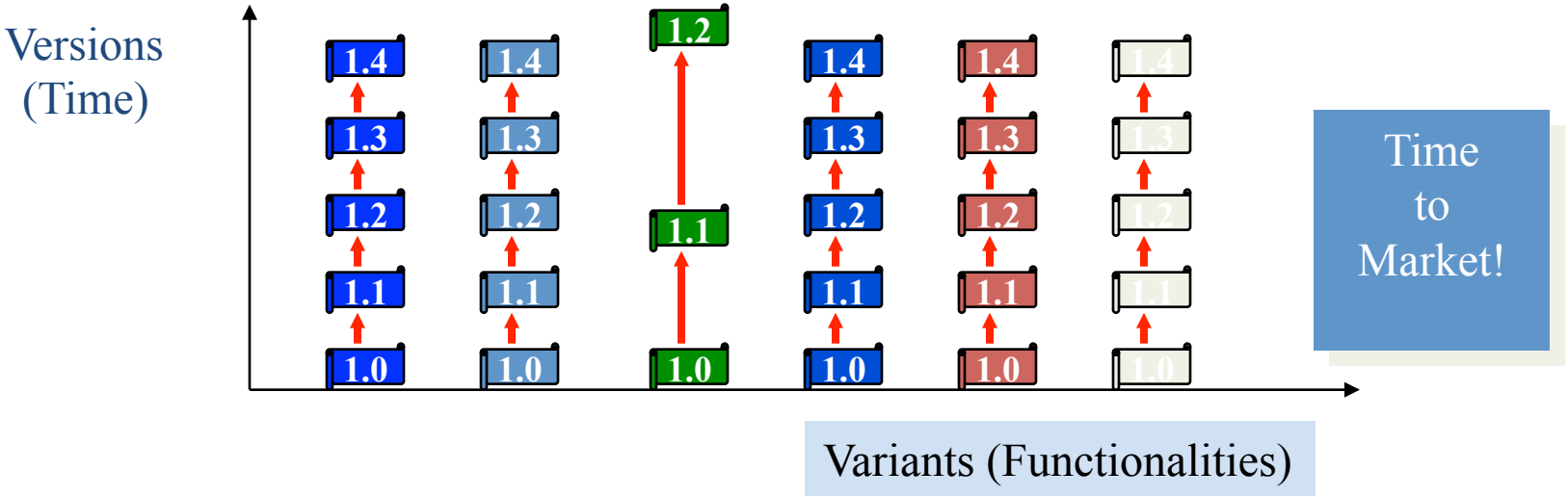
http://mathieuacher.com/teaching/iPDL-MIAGE1/

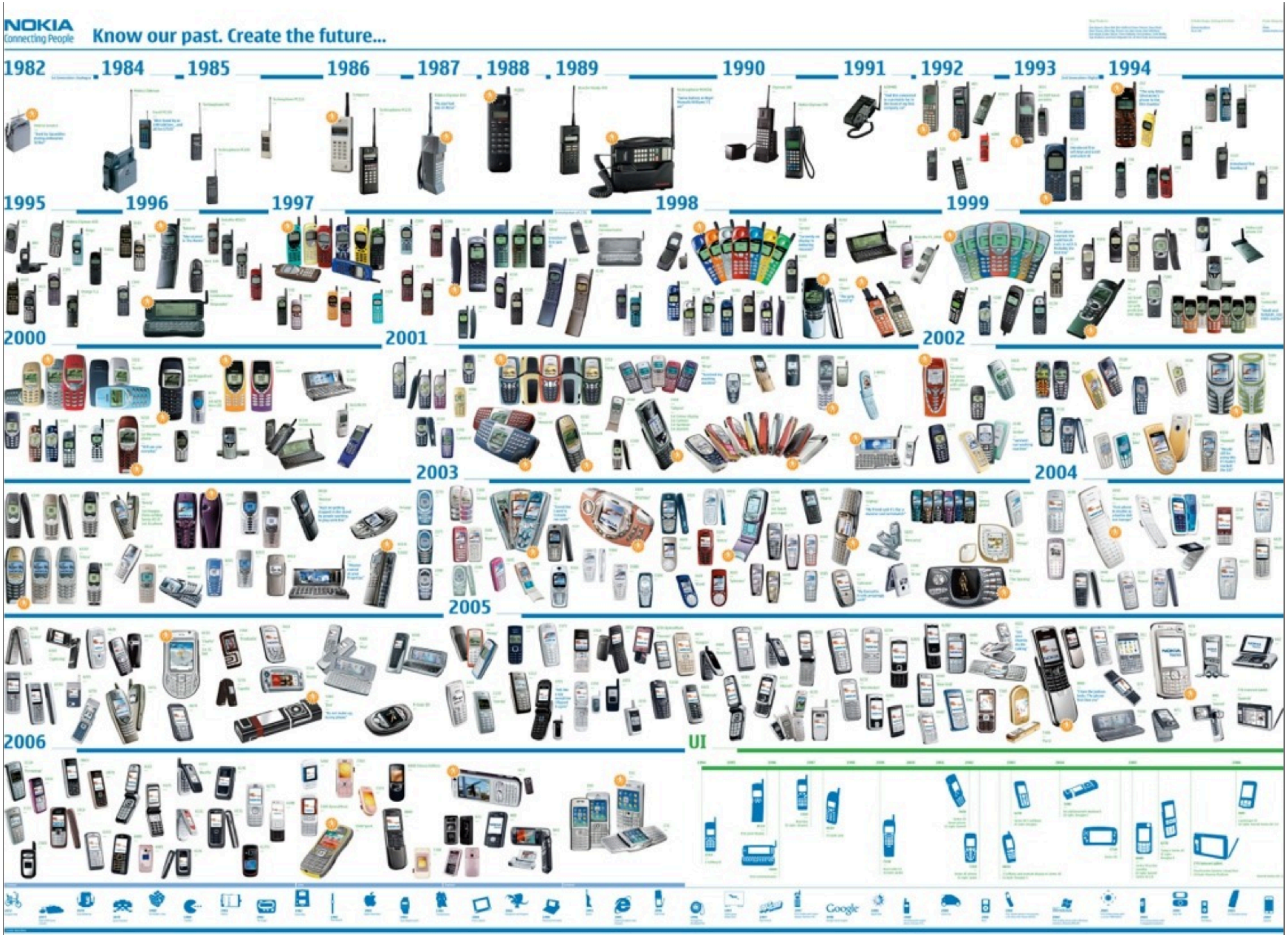
Motivation

Software Engineering



- More and more complex
  - Distributed systems
  - Quality of service: reliability, performance, security
- Explosion of functionalities
  - Product lines (space, time)




NOKIA
Connecting People
Know our past. Create the future...
1982
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
UI
Google

Software Engineering

An overview

- Documentation, Coding Conventions
- Compile Chain and Organizations
- Debugging / Logging
- Testing
- Refactoring
- Versioning

Documentation and Source Code

Documentation

- Source code: one of the best artefact for documenting a project
- Javadoc (JDK)
  - Automatic **generation** of HTML documentation
  - Using comments in java files
- Syntax

```
/**
 * This is a <b>doc</b> comment.
 * @see java.lang.Object
 * @todo fix {@underline this !}
 */
```

- Includes
  - class hierarchy, interfaces, packages
  - detailed summary of class, interface, methods, attributes
- Note
  - Add doc generation to your favorite **compile chain**

Coding Conventions

- Rules on the coding style :
  - Apache, Oracle and others template
    - e.g. http://www.oracle.com/technetwork/java/codeconv-138413.html
    - http://geosoft.no/development/javastyle.html
- Verification tools
  - CheckStyle, PMD, JackPot, Spoon Vsuite…
  - Some integrated into IDEs

Why Coding Standards are Important?

- Lead to greater **consistency** within your code and the code of your teammates
- Easier to **understand**
- Easier to **develop**
- Easier to **maintain**
- Reduces overall cost of application

Example

8. Private class variables should have underscore suffix.

```
class Person
{
  private String name_;
  ...
}
```

Apart from its name and its type, the *scope* of a variable is its most
higher significance than method variables, and should be treated w

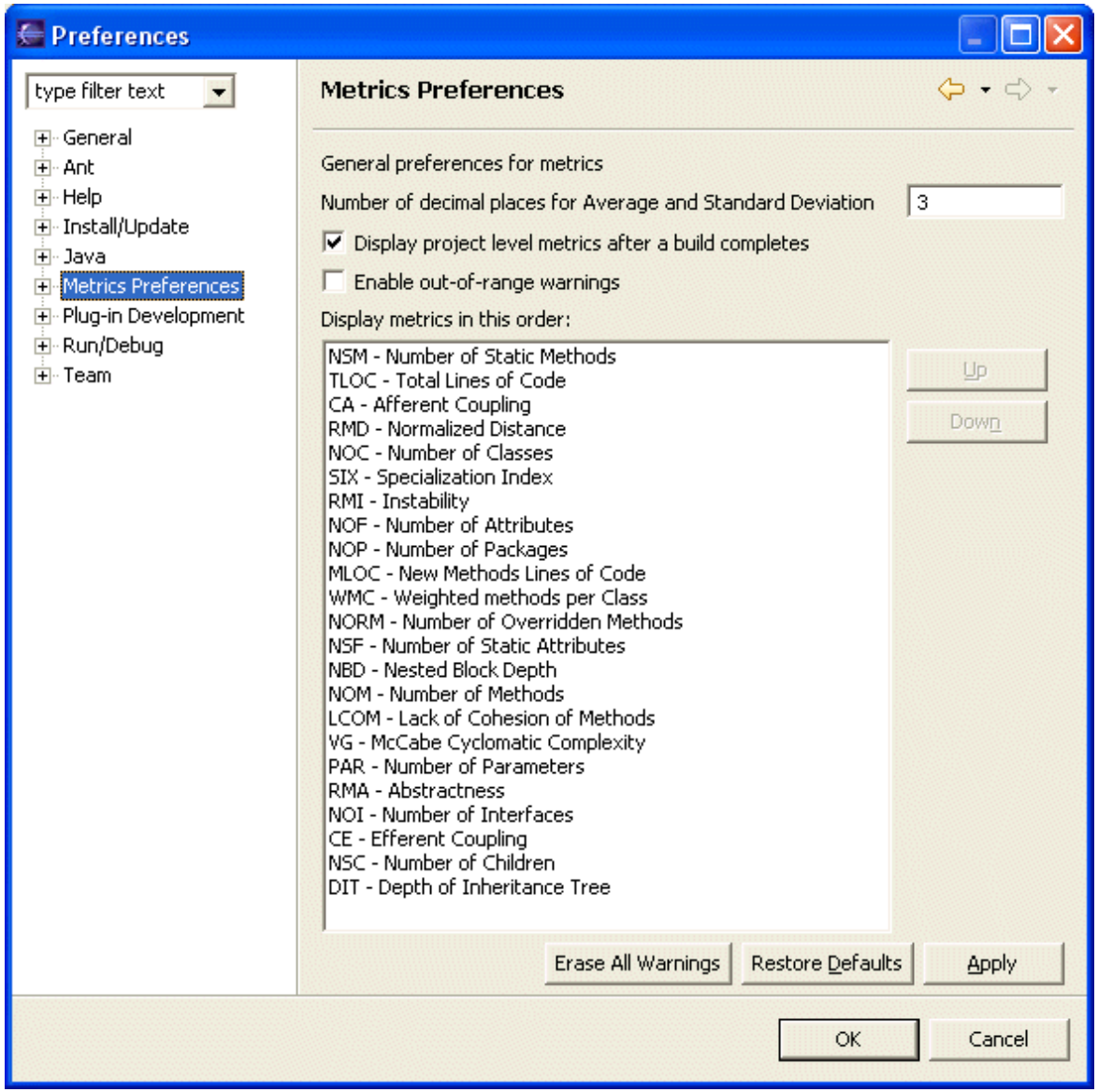
A side effect of the underscore naming convention is that it nicely r

```
  void setName(String name)
  {
    name_ = name;
  }
```

Tools to Improve your Source code

- Formatting tools
  - Indenteurs (Jindent), beautifiers, stylers (JavaStyle), …
- « Bug fixing » tools
  - Spoon VSuite, Findbugs (sourceforge) …
- Quality report tools : code metrics
  - Number of Non Comment Code Source), Number of packages, Cyclomatic numbers, …
    - JavaNCCS, Eclipse Metrics …

Code Quality Metrics



Code Quality Metrics

- Others (standalone or as IDE plugins)
  - http://metrics.sourceforge.net/
  - http://qjpro.sourceforge.net/
  - http://www.geocities.com/sivaram_subr/index.htm
  - …

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
  - …
  - 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 Motivation

- 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 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>
```

Kind of packaging

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

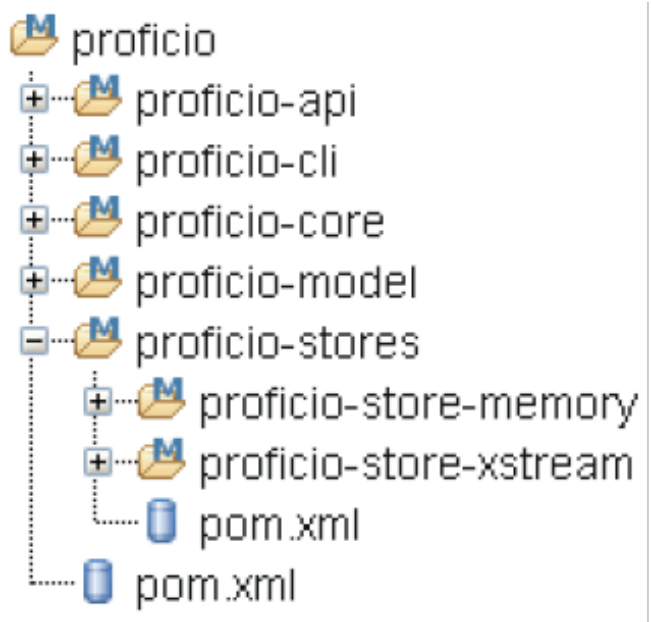
```
mvn package
```

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 plugin for JAVA IDE

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

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

Testing

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


This part is largely inspired by Thomas Zimmermann slides

Djikstra


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


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

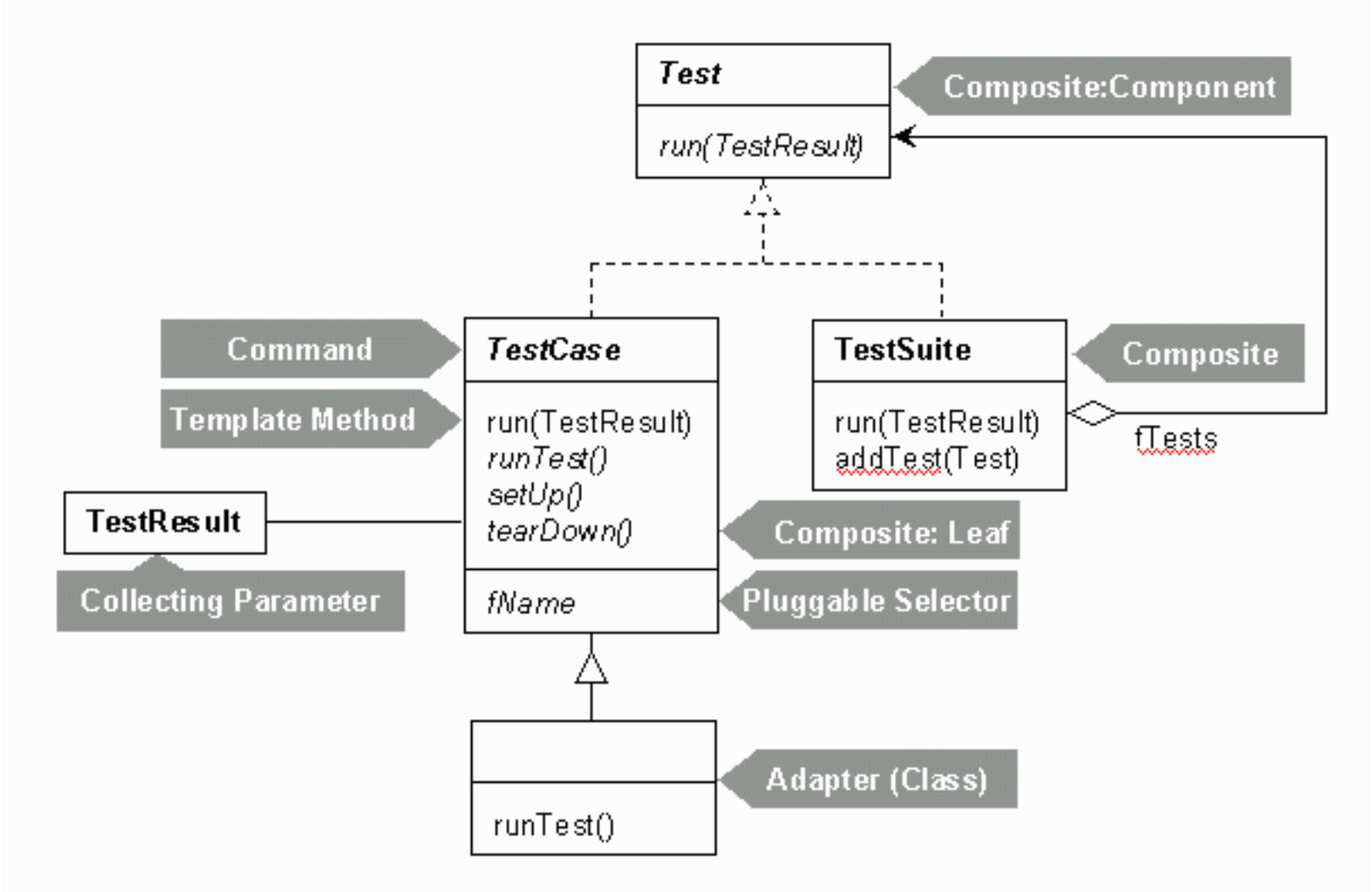
Unit testing on individual units of source code (=smallest testable part).

Integration testing on groups of individual software modules.

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

Junit and… Design Patterns

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



Running example

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

1. Add product

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

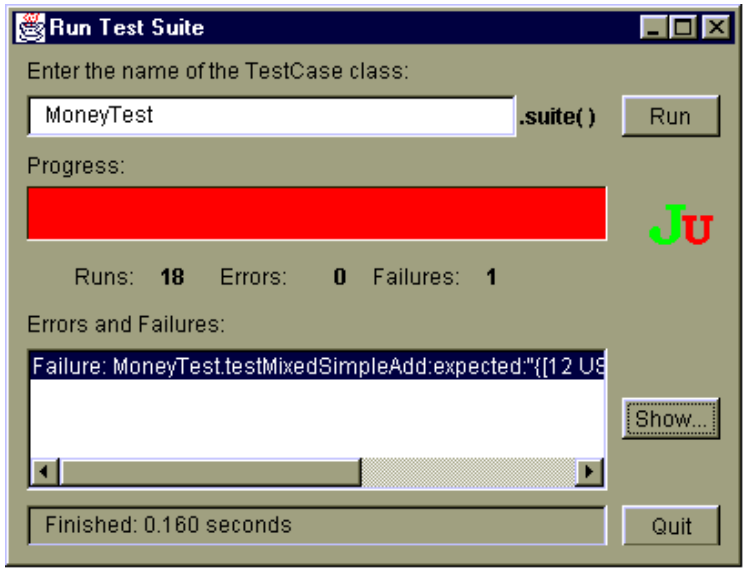
```
// Tests removing a product from the cart.
public void testProductRemove() throws NotFoundException {
    Product book = new Product("Harry Potter", 23.95);
    _bookCart.removeItem(book);

    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);
}
```

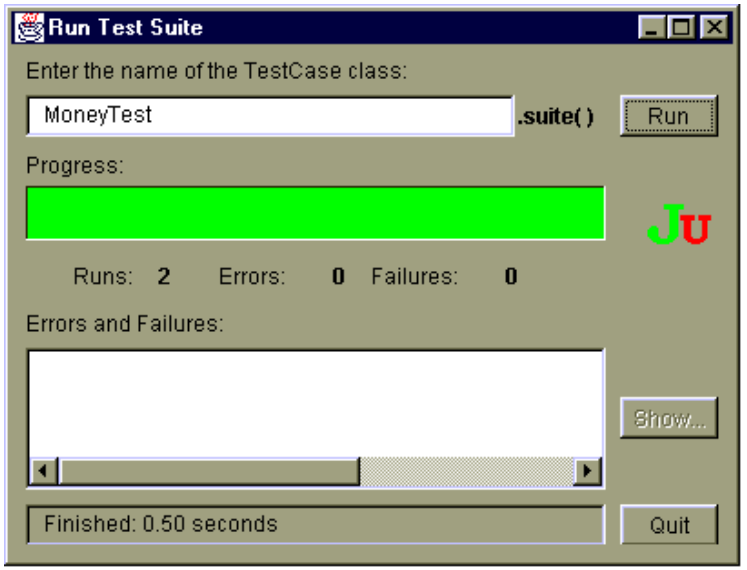
```
public static Test suite() {

    // 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.





- See JUnit course:
  - http://membres-liglab.imag.fr/donsez/cours/junit.pdf

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.

Test toolbox

- Tools to manage tests
  - « Test » every possible case
- Unit Test
  - JUNIT (http://www.junit.org) the most famous
  - Cactus (for Servlets)
  - Jcover (CodeWork)
- Coverage test
  - See methods or code branches not covered by tests.
  - Quilt (http://quilt.sourceforge.net)
- Continuous integration

Other

- Assertion
  - Pre-Condition, Post-Condition, Invariant
    - EIFFEL, CLU ... built-in
    - Java since SE 1.4
- Tools for J2SE 1.3 et less (J2ME, JavaCard, …)
  - AssertMate (Reliable Software Technologies)
    - http://www.ddj.com/articles/1998/9801d/9801d.htm#rel
  - JML (Java Modeling Language)
    - http://www.eecs.ucf.edu/~leavens/JML/
  - iContract (Reliable Systems)
  - …

Design by Contract with JML (by Gary T. Leavens and Yo

Refactoring

What's Code Refactoring?

"A series of *small* steps, each of which changes the program's *internal structure* without changing its *external behavior*"

Martin Fowler

Example

Which code segment is easier to read?

Sample 1:

```
if (markT>=0 && markT<=25 && markL>=0 && markL<=25){
        float markAvg = (markT + markL)/2;
        System.out.println("Your mark: " + markAvg);
}
```

Sample 2:

```
if (isValid(markT) && isValid(markL)){
        float markAvg = (markT + markL)/2;
        System.out.println("Your mark: " + mark);
}
```

Why do we Refactor?

- Improves the design of our software
  - Design pattern!
- Minimizes technical debt
- Keep development at speed
- To make the software easier to understand
- To help find bugs
- To “Fix broken windows”

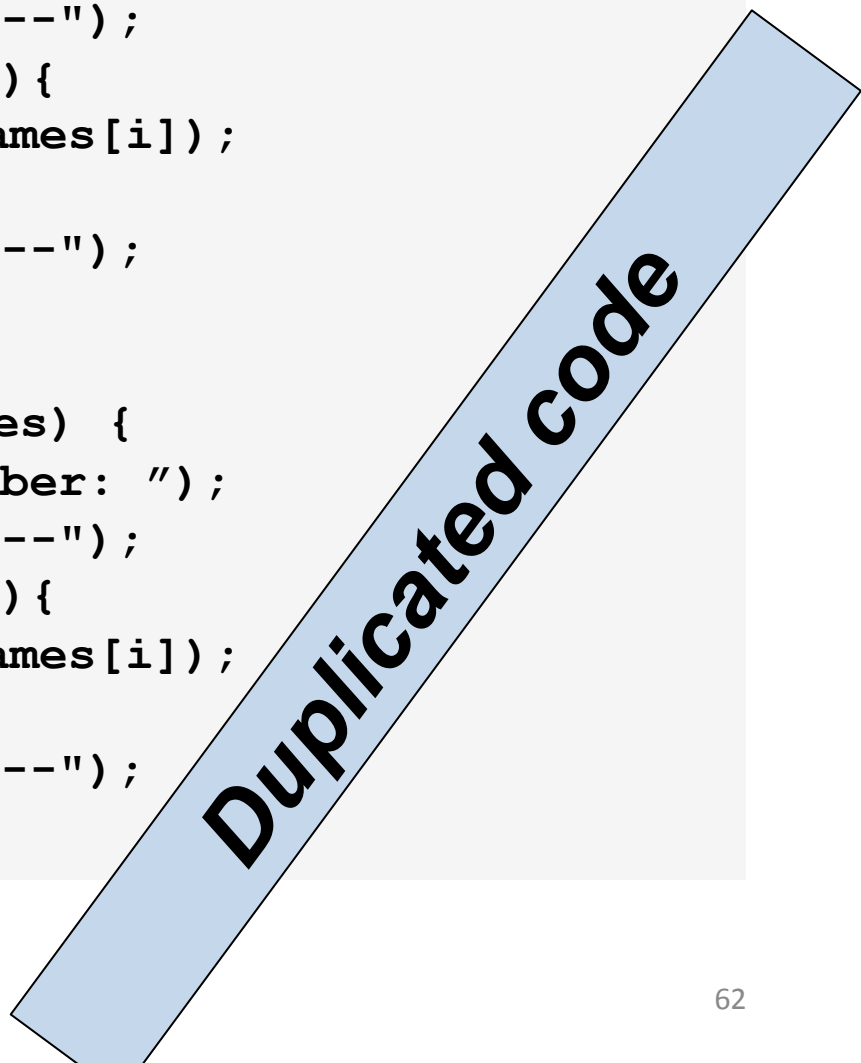
Non exhaustive (code smell)

(and not necessarily smells in all situations)

- Duplicated code
- Feature Envy
- Inappropriate Intimacy
- Comments
- Long Method
- Long Parameter List
- Switch Statements
- Improper Naming

Code Smell examples (1)

```
public void display(String[] names) {
    System.out.println("--------------");
    for(int i=0; i<names.length; i++){
        System.out.println(" + " + names[i]);
    }
    System.out.println("--------------");
}

public void listMember(String[] names) {
    System.out.println("List all member: ");
    System.out.println("--------------");
    for(int i=0; i<names.length; i++){
        System.out.println(" + " + names[i]);
    }
    System.out.println("--------------");
}
```



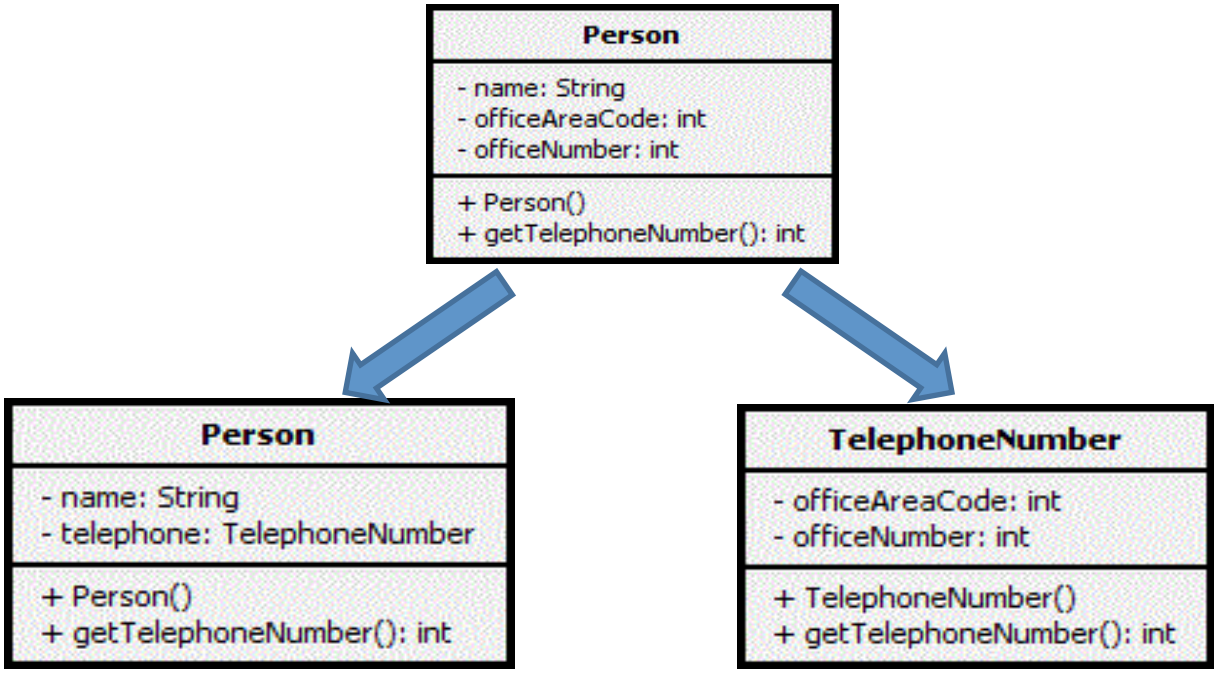
Code Smell examples (2)

```
public String formatStudent( int id,
                             String name,
                             Date dob,
                             String province,
                             String address,
                             String phone ){
   //TODO:
   return null;
}
```

Long list of parameters

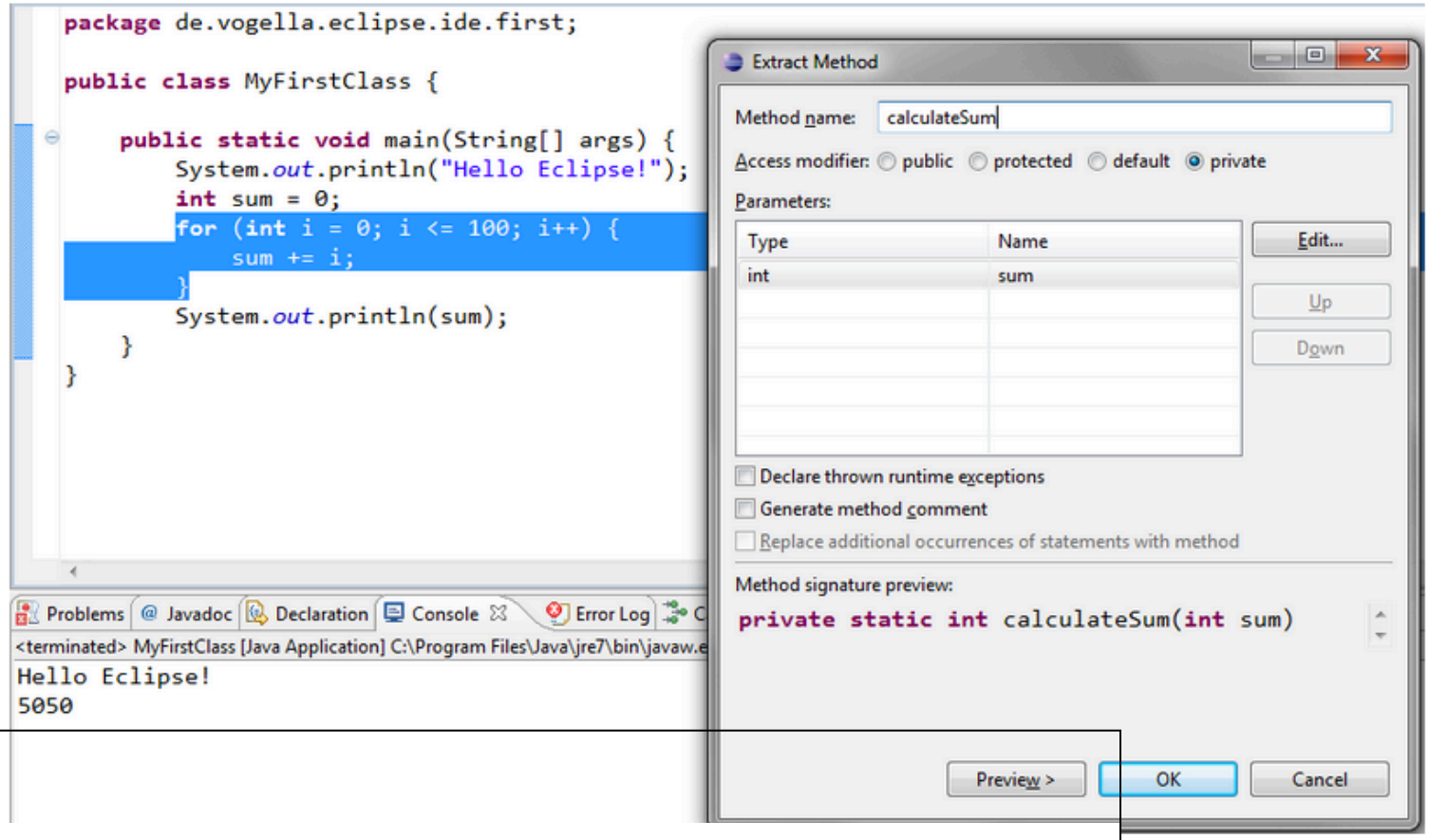
Improving design

- Move Method or Move Field – move to a more appropriate Class or source file
- Rename Method or Rename Field – changing the name into a new one that better reveals its purpose
  - Pull Up – in OOP, move to a superclass
  - Push Down – in OOP, move to a subclass



How do we Refactor?

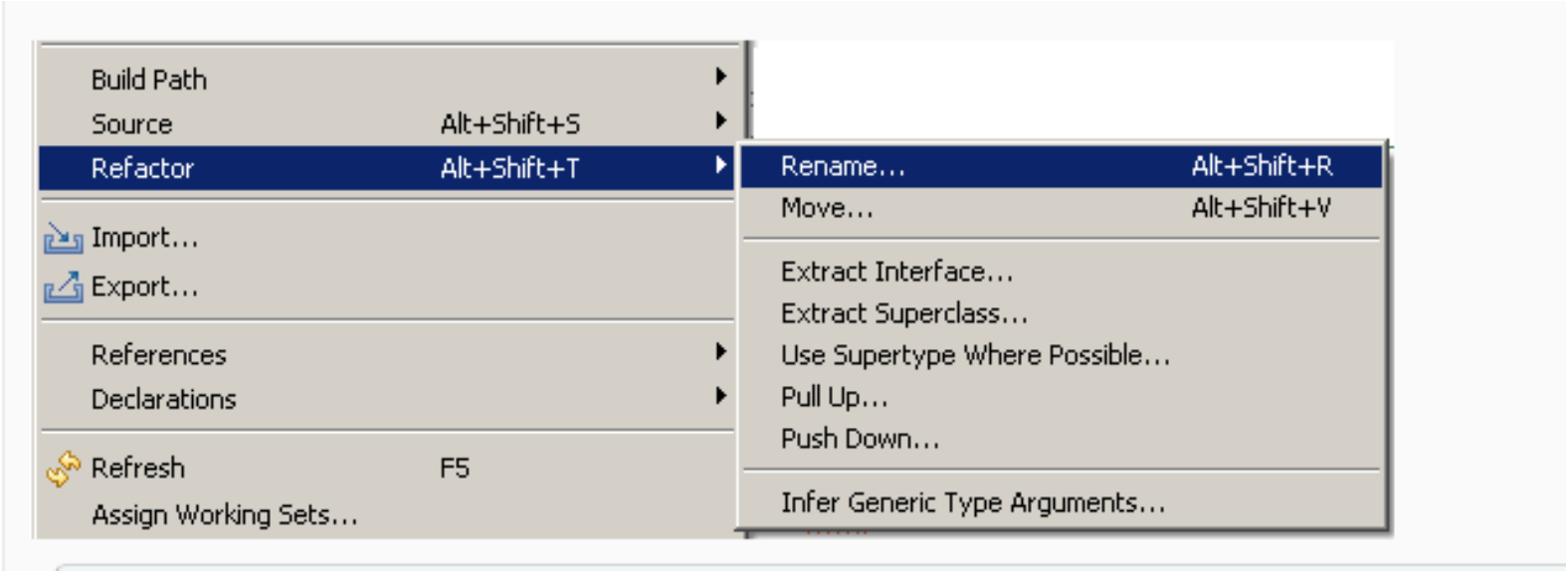
- Manual Refactoring
  - Code Smells
- Automated/Assisted Refactoring
  - Refactoring by hand is time consuming and prone to error
  - Tools (IDE)
- In either case, **test your changes**




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

public class MyFirstClass {

  public static void main(String[] args) {
    System.out.println("Hello Eclipse!");
    int sum = 0;
    sum = calculateSum(sum);
    System.out.println(sum);
  }

  private static int calculateSum(int sum) {
    for (int i = 0; i <= 100; i++) {
      sum += i;
    }
    return sum;
  }
}
```

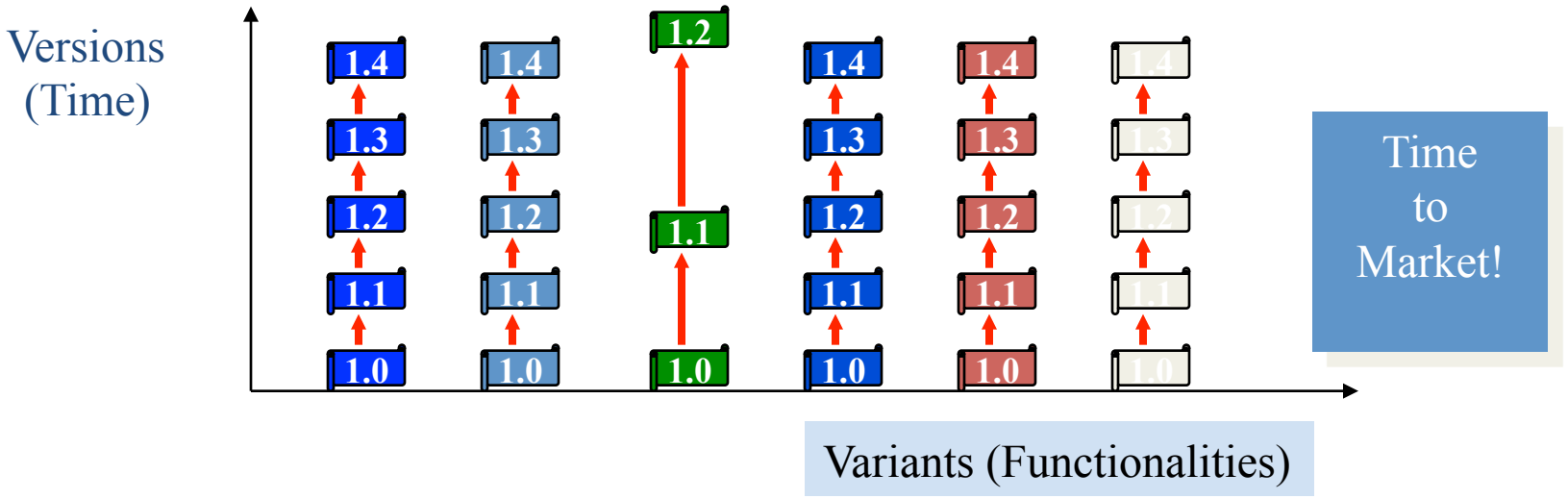
Typical refactoring patterns

- Rename variable / class / method / member
- Extract method
- Extract constant
- Extract interface
- Encapsulate field



Versioning

Software Engineering (back)



The 3 Dimensions of Software Configuration Management

[Estublier et al. 95]

- The Revision dimension
  - Evolution over time
- The Concurrent Activities dimension
  - Many developers are authorized to modify the same configuration item
- The Variant dimension
  - Handle environmental differences

- Even with the help of sophisticated tools, the complexity might be daunting
  - Try to simplify it by reifying the variants of an OO system

Versioning of source code

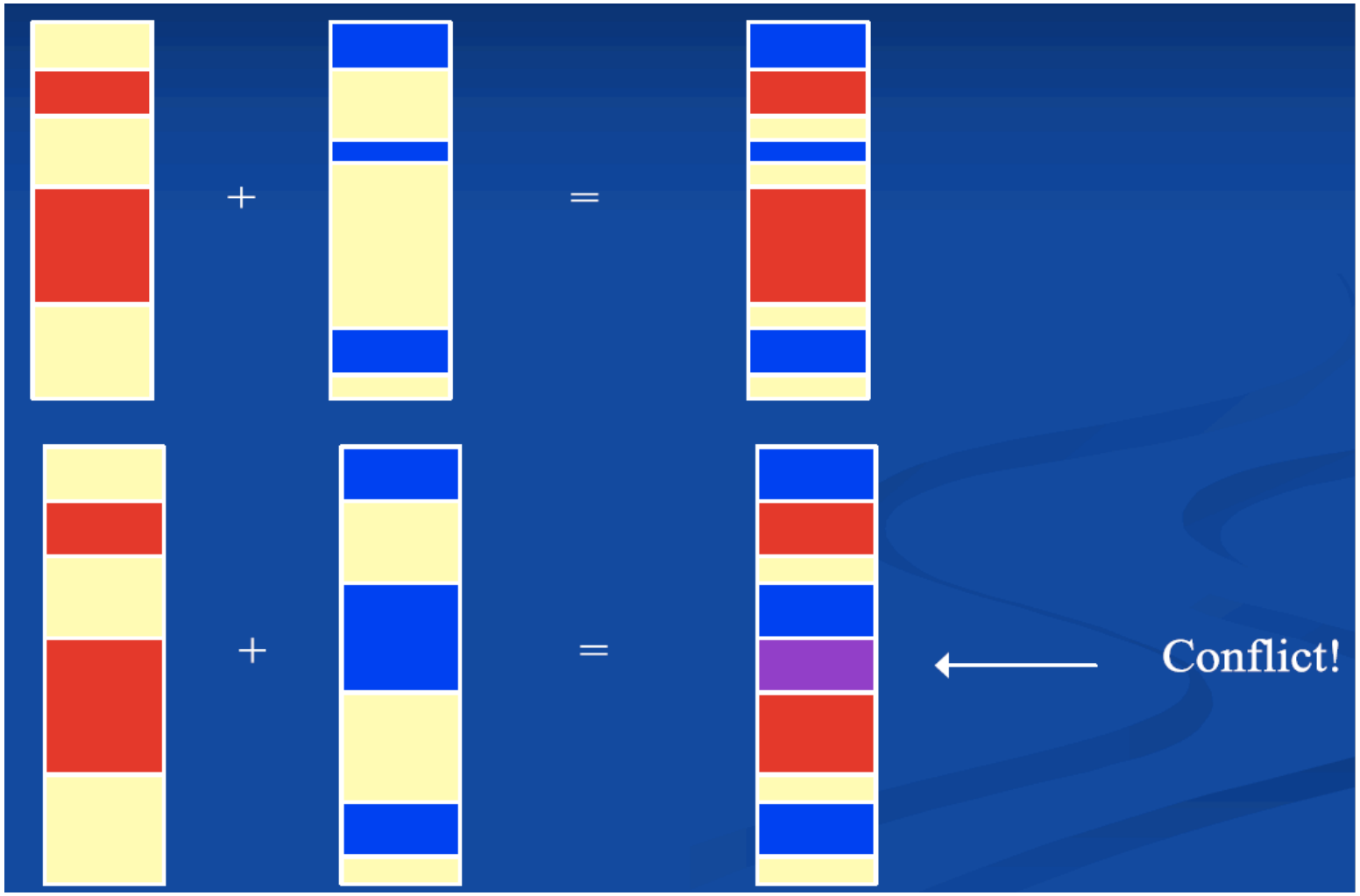
- **Collaborative** software engineering
- To master
  - software development by very large developer teams
  - parallel implementations (experiments, vendors)
- Goals
  - Increase productivity of developers and software robustness
  - Low-down development costs
- Manage software system configuration
  - to control software systems evolution
  - evolution tracking (time-machine)
  - issue and bug tracking

Versioning : What for?

- **History** of versions
  - back to an older version in case of errors
- **Alternative** versions (branching)
  - different design/implementations (maybe experimentals) for the same module
- **Collaborative** access by many developers
  - audit modification history
    - how many commits by X ?
    - when most of the commits are done?
    - ...

Concurrency management

with a simple picture



Concurrency control

- Doing nothing!
- Lock-Modify-Unlock (Pessimistic)
  - SCCS, RCS
  - Decrease productivity
- Copy-Modify-Merge (Optimistic)
  - Conflicts resolution when concurrent modifications (which are actually rare)
    - Merge, Selection, …
  - CVS, SVN : Client level resolution
- Policy-based
  - Merging and validation process for each code contribution

Concept of Version

- Trunk
  - main development
- Branches
  - Alternatives to trunk
    - Different design/implementation (experimental), vendor-specific
- Revisions
  - Sequence of versions
- Tags
  - Symbolic references to revisions (Tiger, LongHorn, …)
    - Represent a public release (R), a milestone (M)
- Branch merging

Tools

- Pioneers
  - SCCS, RCS, PVCS
- Current alternatives
  - CVS
  - SubVersion
  - Git
  - MS Visual SourceSafe
  - ChangeMan (Serena)
  - AllFusion Harvest (CA)
  - ClearCase (IBM Rational)
  - Perforce
  - CM Synergy (Telelogic)
  - Source Integrity (MKS)
  - PVCS (Merant)
  - TeamCode (Interwoven)
  - Surround CM (Seapine)
- Web-Oriented protocols
  - WebDAV/DeltaV

SVN

This part is based on this course.

- Didier DONSEZ
- Université Joseph Fourier (Grenoble 1)
- PolyTech'Grenoble LIG/ADELE
- **Didier.Donsez@imag.fr**
  Didier.Donsez@ieee.org

Main features (1)

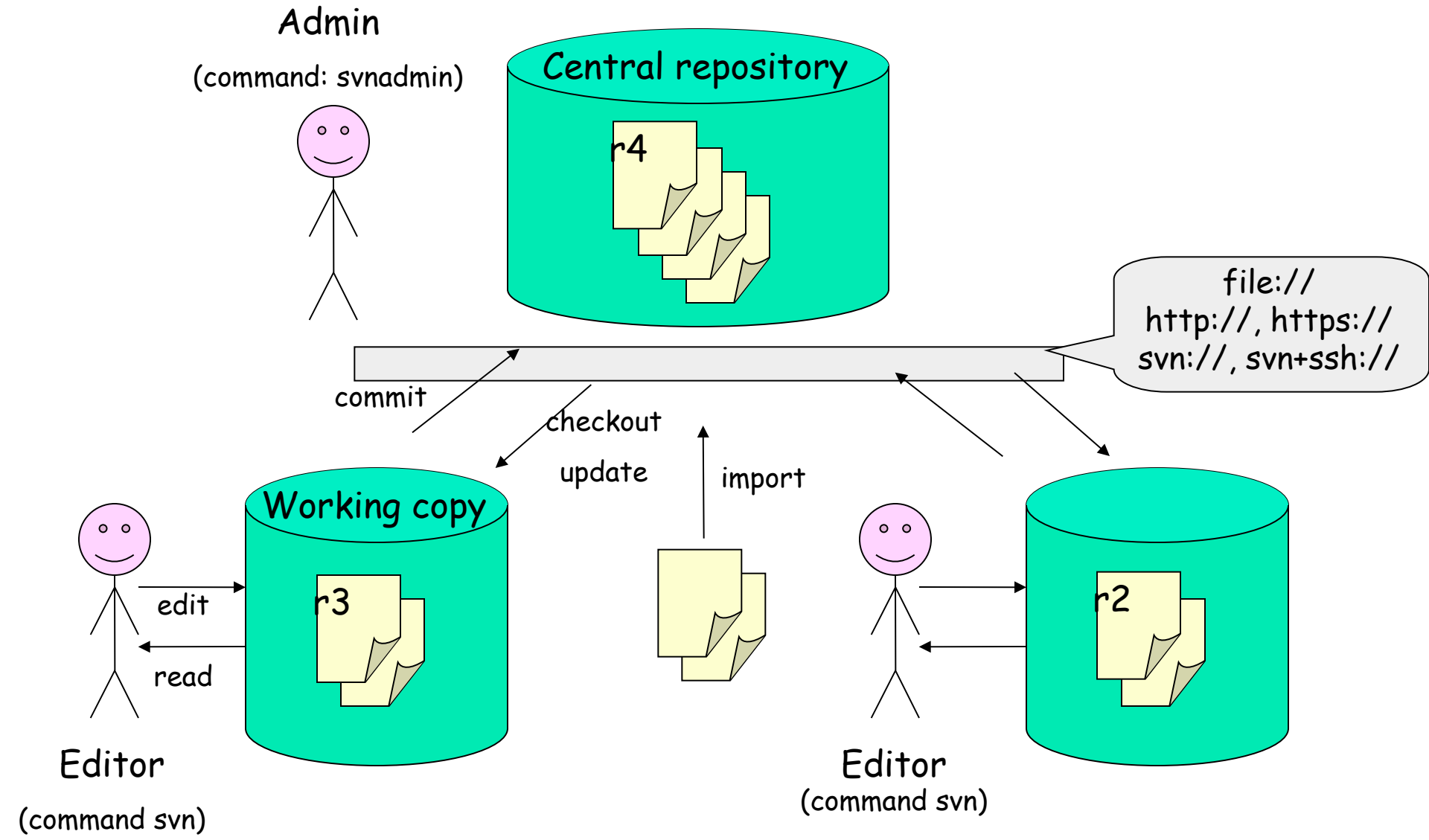
- Optimistic concurrency (Copy-Modify-Merge)
  - But possible to lock (lockable)
- Revision given to a group of modifications
- Versioning of files, directories and metadatas
- Commit/Import/Update atomic
  - grouped revision for multiple files
- Concept of remove, move, copy of subtrees

- Properties (metadatas on files or directories)
  - to be revised, specific..
- There's no concept of tag or branch

Main features (2)

- Differential storage of binary files
- Lazy copy of subtrees
- Works offline
  - Because Web-oriented and big storage capacity available on workstations.
  - Folder `./.svn`
- Exchange only diffs between client and server
- Several access modes
  - File system (`file:`)
  - WebDAV/DeltaV (`http:`,`https:`)
  - proprietary protocols (`svn:`,`svn+ssh:`)
- Browsing and usage (commit) via WebDAV clients and WebDAV/DeltaV
  - Auto-versioning support for WebDAV clients

SubVersion: Architecture



Main commands

- Editor
  - svn
  - svnlook
  - svnshell
- System Administrator
  - svnadmin
  - svndumpfilter
  - svnlook
  - svnshell
- Server
  - svnserve, mod_dav_svn for Apache HTTPD

Revisions

- Revision by group of actions
  - modify/create/delete/move files/directories/metadatas
  - differs from CVS
    - One revision per file
  - Increasing integer number
    - rev=0 for repository creation
      - new revision created at each commit
- Shortcuts

HEAD → The latest (or "youngest") revision in the repository.

BASE → The revision number of an item in a working copy.

COMMITTED → most recent revision prior to, or equal to, BASE, in which an item changed.

PREV → immediately before the last revision in which an item changed. = COMMITTED-1.

How to use shortcuts

- svn diff -r PREV:COMMITTED Hello.java
  - shows the last change committed to Hello.java
- svn log -r HEAD
  - shows log message for the latest repository commit
- svn diff -r HEAD
  - compares your working copy (with all of its local changes) to the latest version of that tree in the repository
- svn diff -r BASE:HEAD Hello.java
  - compares the unmodified version of Hello.java with the latest version of Hello.java in the repository
- svn log -r BASE:HEAD
  - shows all commit logs for the current versioned directory since you last updated
- svn update -r PREV Hello.java
  - rewinds the last change on Hello.java, decreasing Hello.java's working revision
- svn diff -r BASE:32 Hello.java
  - compares the unmodified version of Hello.java with the way Hello.java looked in revision 32

Dates and Revisions

- svn checkout -r {2007-09-17}
- svn checkout -r {16:30}
- svn checkout -r {16:30:00.200000}
- svn checkout -r {"2007-09-17 16:30"}
- svn checkout -r {"2007-09-17 16:30 +0230"}
- svn checkout -r {2007-09-17T16:30}
- svn checkout -r {2007-09-17T16:30Z}
- svn checkout -r {2007-09-17T16:30-04:00}
- svn checkout -r {20070917T1630}
- svn checkout -r {20070917T1630Z}
- svn checkout -r {20070917T1630-0500}

- Note
  - "2007-09-17" == "2007-09-17 00:00"
    - Reference only revisions created before 16 at mid-night

Creating a repository

- Creation
  - svnadmin create c:\repository
    - the --fs-type option let you specify the file system to use for storing files (BerkeleyDB by default or FSFS)
- Initial import
  - of files from project held in the repository
  - svn import hello file:///c:/repository/hello --message "initial import"
- Inspect
  - svnlook youngest repository
  - svnlook tree repository (~ --revision HEAD)
  - svn info file:///c:/repository/hello
  - svn log –r10 file:///c:/repository/hello
  - svn cat –r10 file:///c:/repository/hello/README.txt

Creating a working copy

- Creation (checkout)
  - svn checkout file:///c:/repository/hello workingcopy1
  - svn status workingcopy1 --verbose
- Adding, renaming and deleting files
  - echo This is a readme file > workingcopy1\ README.txt
  - svn add workingcopy1\README.txt
  - svn rename workingcopy1\ README.txt workingcopy1\ README.txt
  - svn delete workingcopy1\ README.txt
  - svn status workingcopy1 --verbose
- Validate modifications on the repository
  - svn commit workingcopy1 --message "some modifications"
- Synchronize with latest revision from repository
  - svn update workingcopy1

File status

- A → Resource is scheduled for Addition
- D → Resource is scheduled for Deletion
- M → Resource has local Modifications
- C → Resource has Conflicts
  - changes have not been completely merged between the repository and working copy version
- X → Resource is eXternal to this working copy
  - may come from another repository.
- ? → Resource is not under version control
- ! → Resource is missing or incomplete
  - removed by another tool than SubVersion

Collaborative editing

- Conflits after a commit (status=C)
  - Another editor modified and validated a file that was currently under modifications
- 3 possibilities
  - Dismiss
    - svn revert README.txt
      - abandon modifications done in the working copyl
  - Over-write
    - svn update README.txt
    - cp README.txt.mine README.txt
    - svn resolved README.txt
    - svn commit -m "discard the other revision"
  - Manual merge
    - svn update README.txt
    - edit README.txt
    - svn resolved README.txt
    - svn commit -m "merge the conflicted revisions"
- Note
  - When there's a conflict, svn update creates 3 files
    - `filename.ext.mine` : file image before the update
    - `filename.ext.rOLDREV` : the BASE revision before the update
    - `filename.ext.rNEWREV` : the HEAD immediately after the commit
    - `filename.ext` contient les marques indiquant les lignes en conflits

Pessimistic approach (Lock→Modify→Unlock) Unlock)

- Subversion authorize exclusive locks
- Commands client-side
  - svn ps svn:needs-lock '*' README.txt
    - Mark the file as readonly if another editor updates before editing it
    - Optional, but useful !
  - svn lock README.txt -m "for tomorrow's release"
  - svn info README.txt
  - edit README.txt
  - svn unlock README.txt

- Administrative commands
  - Developers (or because of system crashes) forget to remove their locks
    - svn cleanup
    - svnadmin lslocks c:\repository
    - svnadmin rmlocks c:\repository /README.txt
    - svn info file:///c:/repository/README.txt

Create and apply a patch

- Motivations
  - Analyze diffs between my copy and a given revision (generally HEAD)
  - Analyze diffs between 2 revisions
  - To send diffs to another editor, committer, …
  - Attach diffs to a bug-tracking system (JIRA, …)
- Creating a patch
  - svn diff README.txt >> r.patch
    - between HEAD and the current working copy
  - svn diff -r 2:3 README.txt >> r23.patch
    - between rev2 and rev3
  - svn diff --diff-cmd /usr/bin/diff --extension "-i -b" README.txt >> r.patch
    - Using external diff command
- Applying a patch (on the working-copy or else)
  - On Unix and CygWin: diff, diff3
  - On Windows, ExamDiff, KDiff3, WinMerge, Araxis

Key-word replacement

- Motivations
  - Insert in files informations about the revisions
    - Updated every update
- Methods
  - Proprierty svn:keywords
    - Specify the list of key words to replace
  - Replaceble key-workds
    - Revision (or Rev): latest revision before editing the file
    - HeadURL: URL of latest file revision
    - Date (or LastChangedDate): date of latest rev before editing the file
    - Author: author of the latest modification
    - Id: composite identifier (file rev date author)
- Exemple
  - echo Id $HeadURL$ Rev $Author$ $Date$ >> README.txt
  - type README.txt
  - svn propset svn:keywords "Id Rev HeadURL Date Author" README.txt
  - svn commit README.txt
  - svn update README.txt
  - type README.txt

Properties

- Metadatas attached to files/directories
  - Pairs <key alphanum, ASCII value or binary>
  - Editors define the semantic or specific to svn (svn:)
- SVN special properties:
    - svn:ignore : list of file patterns to ignore by versioning
    - svn:externals : specify that directory comes from an external repository
    - svn:mime-type : file's MIME type
    - svn:needs-lock : file must be locked before being edited
  - Can't be versioned
    - svn:log, svn:revision, svn:author, svn:date, svn:autoversioned
- Examples
  - svn proplist Main.java # list current properties
  - svn propget copyright Main.java # current value of copyright property
  - svn propdel contributors Main.java # delete a properties
  - svn propset copyright "(c) 2007 Didier Donsez" *.java
  - svn propset license -F /path/to/LICENSE.txt *.java
  - svn propedit copyright *.java # use editor %SVN_EDITOR% by default
  - svn propedit svn:log –r10 –revprop # edit entry 10 of history

Typical structure of a project

- Subversion do not include the concept of tag or branches

→ Tags and branches can be created from the trunk (or another tag/branch) into a given revision, using the copy command

  - svn copy project1/trunk project1/tags/release-1.0 --revision 234

- Suggested repository structure

```
project1/trunk/
project1/tags/
project1/tags/release-1.0/
project1/tags/release-1.1/
project1/tags/release-2.0/
project1/branches/
project1/branches/java7/
project1/branches/aop/
project1/branches/vendor/
project2/
project2/trunk/
…
sandbox/
sandbox/valerio
…
```

- Notes
  - Tags must be read-only! (hooks)
  - Subversion uses a lazy copy to limit copy cost.

Composite repositories

- Motivation
  - Organize a repository so that some directories correspond to external repositories coming from external entities
    - Useful to develop plugins as third-parties, …
- Property svn:externals (multilines)
  - Specify the list of external repositories
    - svn propset svn:externals "contrib http://svn.other.com/repos/contribs@r123" project1
    - svn propget svn:externals project1
    - svn checkout http://svn.example.com/repos/project1
      - Retrieve files from a central repository and from the external repositorie

Advices on using a SVN repository

- Commit logical changesets
- Commit early, commit often
- Trunk should be stable
- Use a sane repository layout
- Use the issue-tracker wisely
- Track merges manually
  - When committing the result of a merge, write a descriptive log

- To be read
  - http://svn.collab.net/repos/svn/trunk/doc/user/svn-best-practices.html

Dump and load of repo's

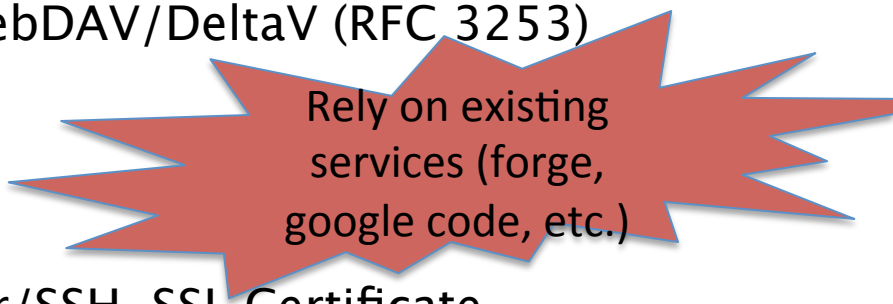
- Creating a repository dump
  - svnadmin dump repository > repository.svndump
- Creating a new partial repository and loading of repository using a dump
  - svndumpfilter include hello/trunk --drop-empty-revs --renumber-revs
    < repository.svndump > hello-trunk.svndump
  - svnadmin create hellorepo
  - svnadmin load hellorepo < hello-trunk.svndump

Dealing with binary files

- Subversion is optimized for dealing with text files (source code, LaTeX documents, etc)
- But, it can deal with binary files
  - Will not diff nor merge
  - Will not change EOL nor apply keywords
- SVN has a binary detection algorithm, but it sometimes fails (PDF have a text header)
  - Need to set svn:mime-type property manually to application/octet-stream

Repository administration

- Configuration
  - svnserve
    - Deamon/service/standalone using proprietary protocol (port=3690)
  - Apache 2.0 + mod_dav_svn
    - Mod Apache 2.0 based on WebDAV/DeltaV (RFC 3253)
      - Support auto-versioning
- Security
  - Authorization
    - username+password in clear/SSH, SSL Certificate
  - Access control
    - By directory (see file ./conf/authz)
- Hooks (inside directory repo/hook)
  - Scripts (sh,pl,py,bat,…) executed by the server at given occurrences of transaction steps
    - Fine control-grain (tags creation, …) , notification (mail,IM) …



Tools

- Clients and plugin IDE
  - TortoiseSVN, Subclipse, …
  - Ant tasks, plugin Maven SCM, plugin MS VSS, …
- Clients WebDAV and WebDAV/DeltaV
  - Revision support
- Navigation Web du dépôt SVN
  - ViewCVS, WebSVN, …
- Misc
  - Convert CVS to SVN (http://cvs2svn.tigris.org/)
- Language bindings
  - Java (JNI), Perl, Python, C++, C# …

Plugin SCM Maven http://maven.apache.org/scm/

- Plugin offering common API for main SCM
  - Commands
    - Changelog – command to show the source code revisions
    - Checkin – command for commiting changes
    - Checkout – command for getting the source code
    - Diff – command for showing the difference of the working copy with the remote ones
    - Edit – command for starting edit on the working copy
    - Status – command for showing the scm status of the working copy
    - Tag – command for tagging the certain revision
    - UnEdit – command for to stop editing the working copy
    - Update – command for updating the working with the latest changes
    - Validate – validates the scm information on the pom
  - Supported SCM
    - Subversion, CVS, Starteam, Clearcase, Perforce, bazaar
  - URL SCM
    - scm:<scm_provider><delimiter><provider_specific_part>
  - Client
    - java -jar target\maven-scm-client-1.1-jar-with-dependencies.jar checkout c:\temp\maven-scm-client scm:svn:http://svn.apache.org/repos/asf/maven/scm/trunk/maven-scm-client

References

- Subversion
  - http://subversion.tigris.org
- Books
  - Ben Collins-Sussman, Brian W. Fitzpatrick & C. Michael Pilato, Version Control with Subversion, Pub. O'Reilly, 2004, ISBN 0596004486
    - Version en ligne http://svnbook.red-bean.com/nightly/en/svn-book.pdf
  - Garrett Rooney; Practical Subversion; Apress; ISBN 1-59059-290-5 (1st edition, paperback, 2005)
  - Mike Mason; Pragmatic Version Control Using Subversion; Pragmatic Bookshelf; ISBN 0-9745140-6-3 (1st edition, paperback, 2005)
  - William Nagel; Subversion Version Control: Using the Subversion Version Control System in Development Projects; Prentice Hall; ISBN 0-13-185518-2 (1st edition, paperback, 2005)

Git

Git

- version control system
  - designed to handle very large projects with speed and efficiency
  - mainly for various open source projects, most notably the Linux kernel.
- http://git.or.cz/

Tools to use Git

- SmartSVN
- GitHub => provide the whole forge with a GIT installed
  - Free for open-source project

Centralized Version Control

- Traditional version control system
  - Server with database
  - Clients have a working version
- Examples
  - CVS
  - Subversion
  - Visual Source Safe
- Challenges
  - Multi-developer conflicts
  - Client/server communication

Distributed Version Control

- Authoritative server by convention only
- Every working checkout is a repository
- Get version control even when detached
- Backups are trivial

- Other distributed systems include
  - Mercurial
  - BitKeeper
  - Darcs
  - Bazaar

Git Advantages

- Resilience
  - No one repository has more data than any other
- Speed
  - Very fast operations compared to other VCS (I'm looking at you CVS and Subversion)
- Space
  - Compression can be done across repository not just per file
  - Minimizes local size as well as push/pull data transfers
- Simplicity
  - Object model is very simple
- Large userbase with robust tools

Git Architecture

- Index
  - Stores information about current working directory and changes made to it
- Object Database
  - Blobs (files)
    - Stored in .git/objects
    - Indexed by unique hash
    - All files are stored as blobs
  - Trees (directories)
  - Commits
    - One object for every commit
    - Contains hash of parent, name of author, time of commit, and hash of the current tree
  - Tags

Some Commands

- Getting a Repository
  - git init
  - git clone
- Commits
  - git add
  - git commit
- Get changes with
  - git fetch (fetches and merges)
  - git pull
- Propagate changes with
  - git push

Relationship with PDL (your project)

Fournisseurs

Gestionnaire

Clients

Système de gestion de catalogue (web) de produits

Plateforme d'échange

Gestion des produits

configurateur

Impacts

- Use/experiment with a subset of these tools
  - IDE
  - Versioning systems
  - Refactoring
  - Testing
  - Documentation
- You will have to in your professional career!
- My proposal: use existing system like github or google code
  - Collaborative work
  - You will keep your work somewhere
  - You will be able to demonstrate your skills