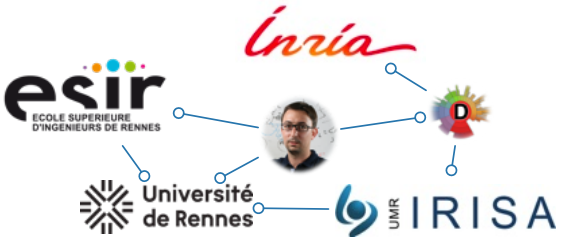
TOOLS FOR (JAVA) DEVELOPMENT INDUSTRIALIZATION

UNIV. RENNES 1, ESIR, 2023-2024

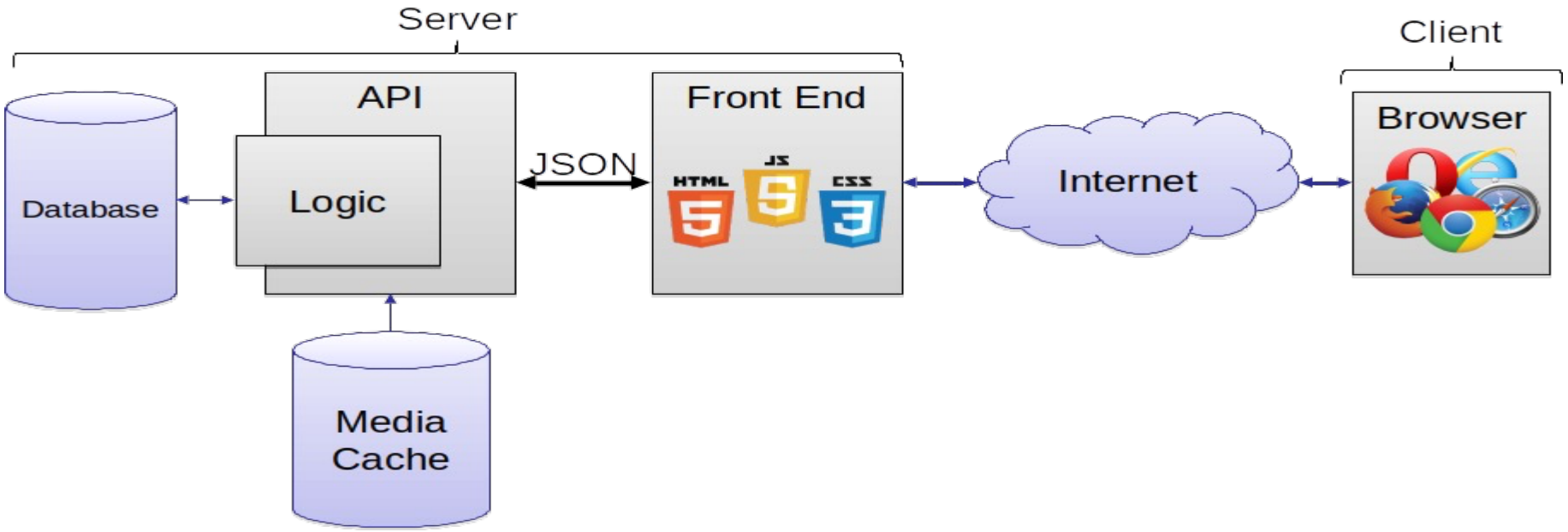
BENOIT COMBEMALE
FULL PROFESSOR, UNIVERSITY OF RENNES, FRANCE

HTTP://COMBEMALE.FR
BENOIT.COMBEMALE@IRISA.FR
@BCOMBEMALE

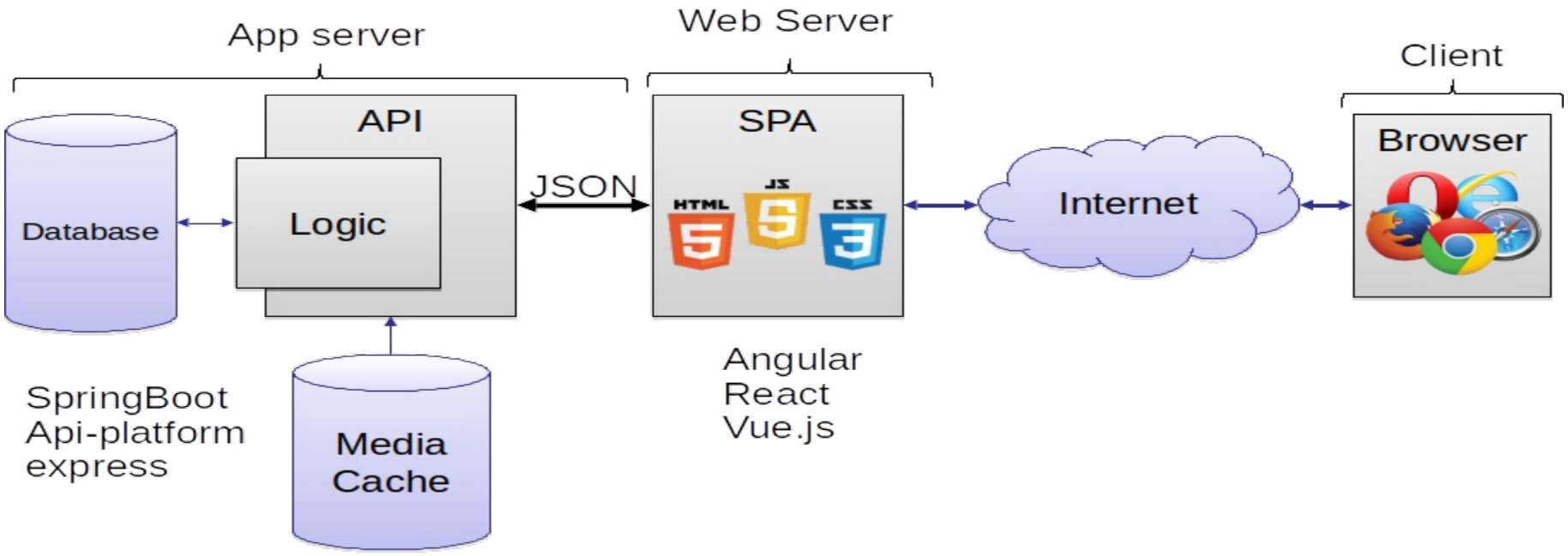


Exemple d’organisation de projet dans la pratique…

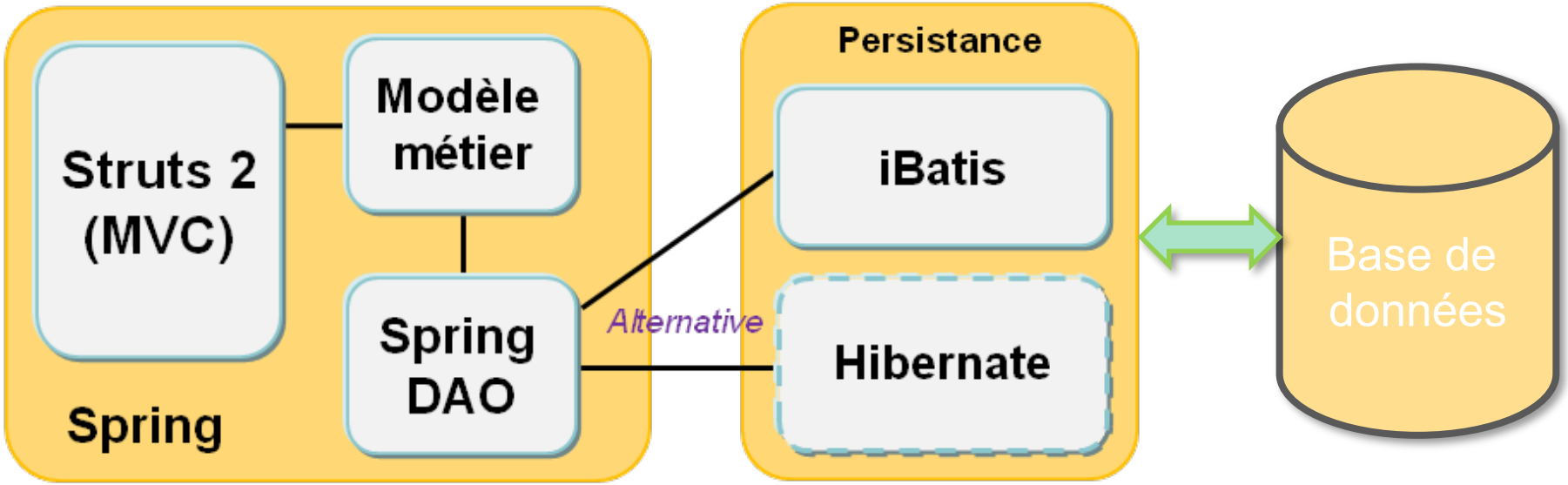
Architecture moderne



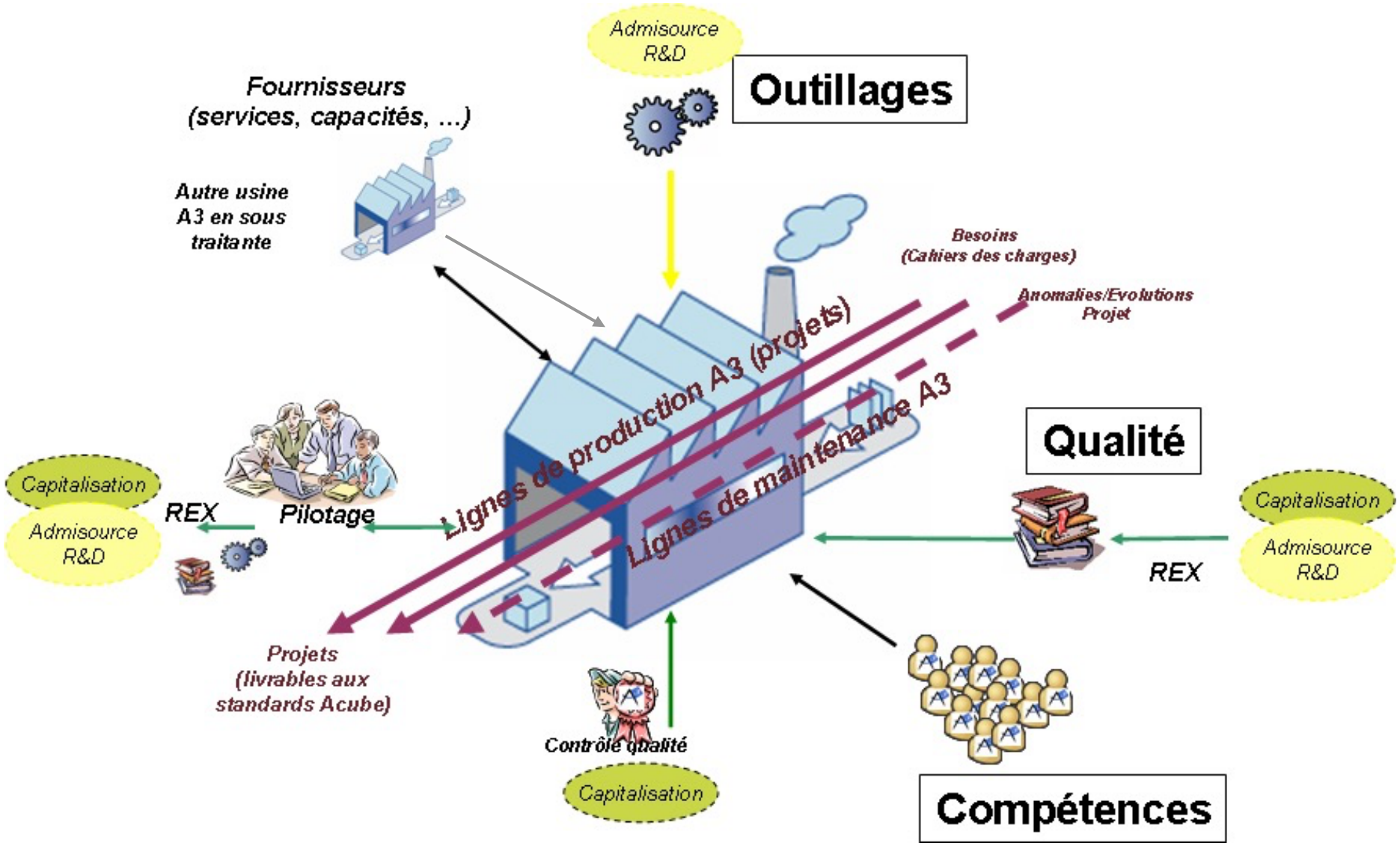
Architecture moderne



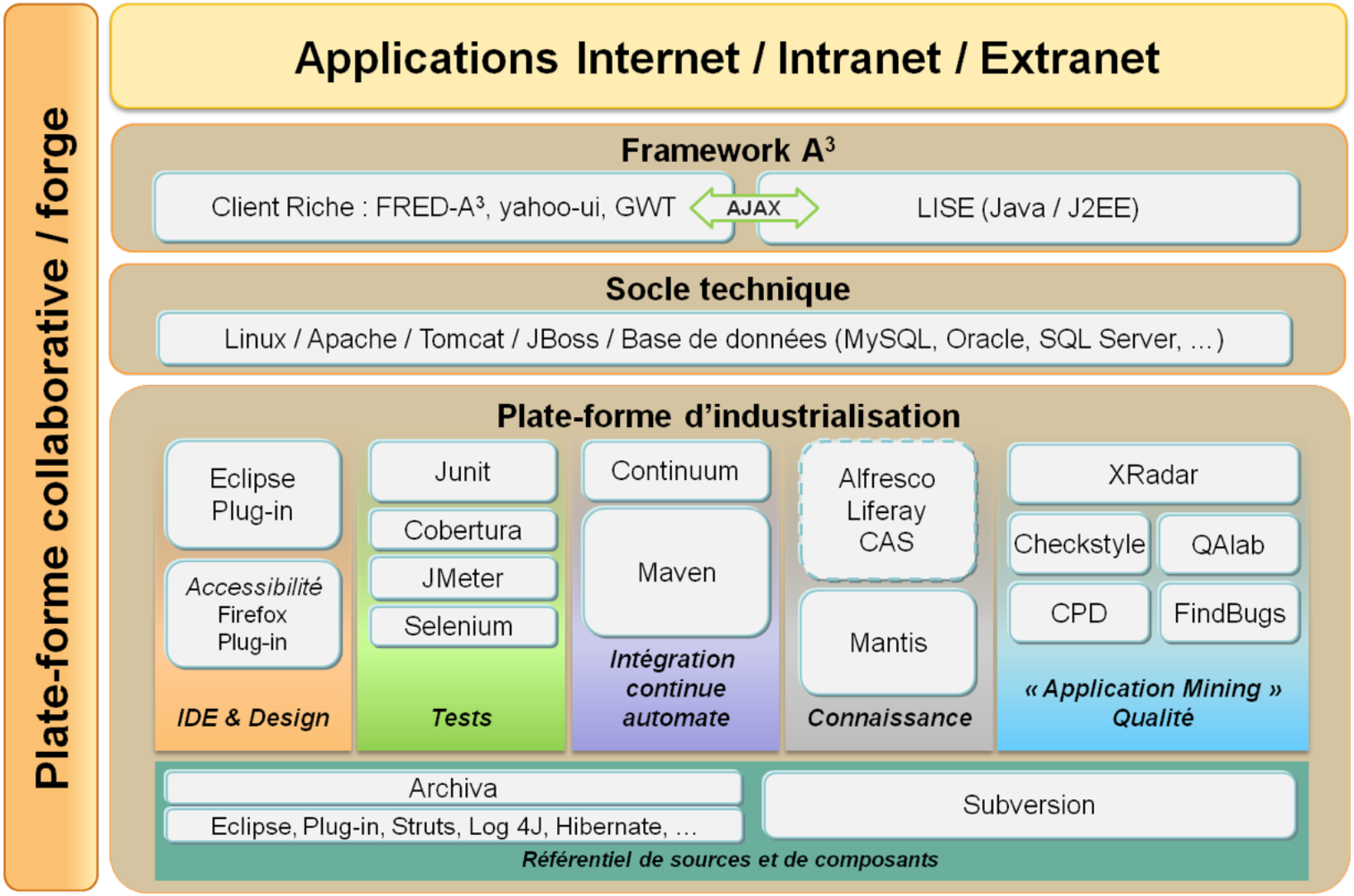
Serveur JEE (exemple)



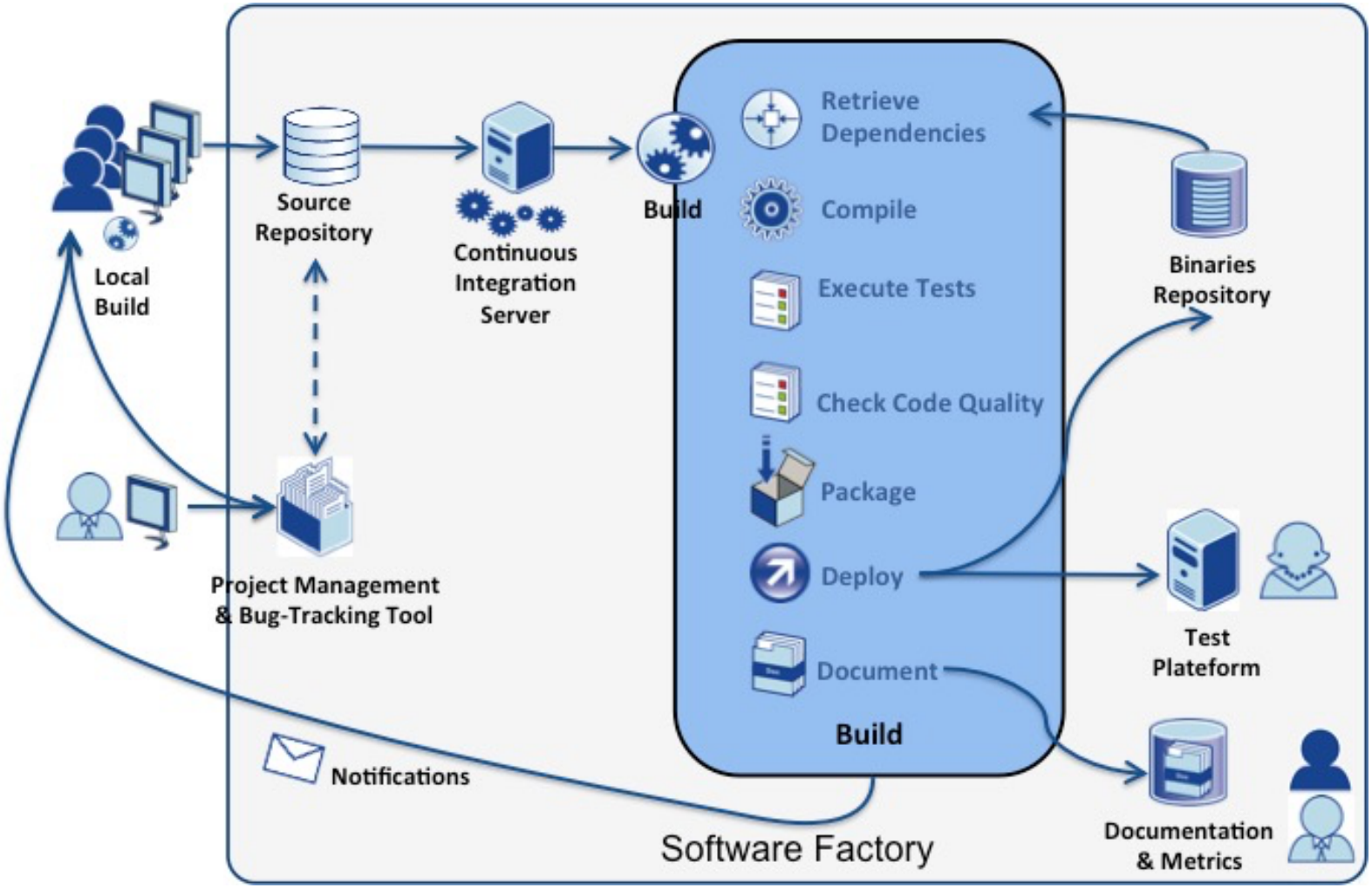
Usine Logicielle



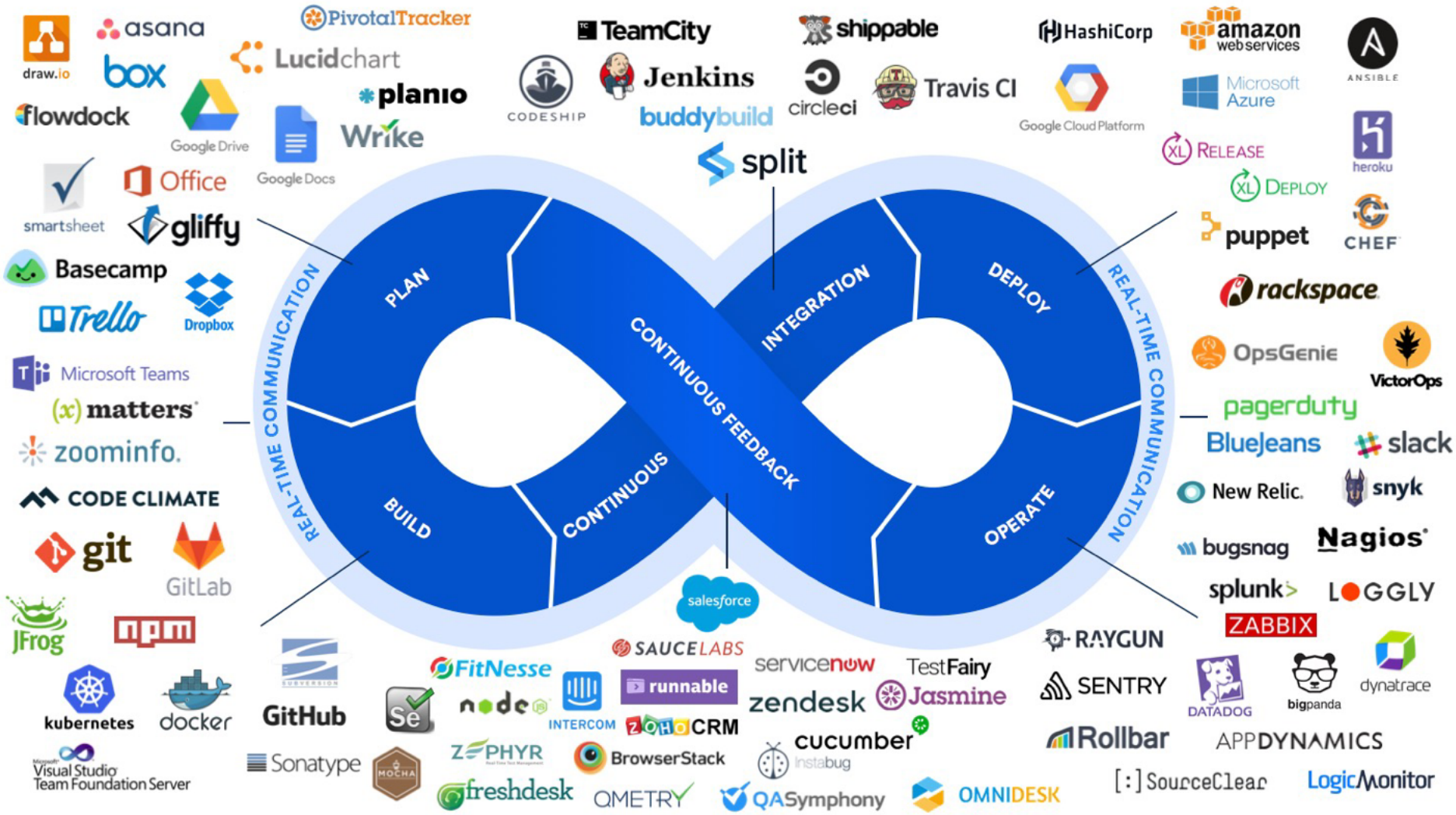
Approche industrielle – un cas classique



Usine Logicielle



DevOps (exemple)



Approche méthodologique 1/2

- Intérêts pour l'entreprise
  - Standardisation pour une meilleure maintenabilité
  - Cohérence entre les projets
  - Capitalisation

Approche méthodologique 2/2

- Intérêt pour le client
  - Voir l'application se construire
  - Faire ses remarques au fur et à mesure
  - Prioriser les exigences

Software Engineering

Code::Blocks Studio

eclipse

IJ

git

SUBVERSION

CVS

Method Summary

http://logging.apache.org

Logging Services™

Apache LOG4J™

<APACHE ANT>

Maven™

JUnit

sonar

Highlights

- Documentation
- Logging
- Testing
- Static analysis
- Refactoring
- Build automation
- Versioning
- Continuous integration
- *And towards DevOps and modern architectures*

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**

Package javax.swing

Provides a set of "lightweight" (all-Java language) components that, to the maximum degree possible, work the same on all platforms.

See:

Description

Interface Summary

Action	The `Action` interface provides a useful extension to the `ActionListener` interface in cases where the same functionality may be accessed by several contro
BoundedRangeModel	Defines the data model used by components like Sliders and ProgressBars.
ButtonModel	State Model for buttons.
CellEditor	This interface defines the methods any general editor should be able to implement.
ComboBoxEditor	The editor component used for JComboBox components.
ComboBoxModel	A data model for a combo box.
DesktopManager	DesktopManager objects are owned by a JDesktopPane object.
Icon	A small fixed size picture, typically used to decorate components.
JComboBox.KeySelectionManager	The interface that defines a `KeySelectionManager`.
ListCellRenderer	Identifies components that can be used as "rubber stamps" to paint the cells in a JList.
ListModel	This interface defines the methods components like JList use to get the value of each cell in a list and the length of the list.
ListSelectionModel	This interface represents the current state of the selection for any of the components that display a list of values with stable indices.
MenuElement	Any component that can be placed into a menu should implement this interface.
MutableComboBoxModel	A mutable version of `ComboBoxModel`.
Renderer	Defines the requirements for an object responsible for "rendering" (displaying) a value.
RootPaneContainer	This interface is implemented by components that have a single JRootPane child: JDialog, JFrame, JWindow, JApplet, JInternalFrame.
Scrollable	An interface that provides information to a scrolling container like JScrollPane.
ScrollPaneConstants	Constants used with the JScrollPane component.
SingleSelectionModel	A model that supports at most one indexed selection.
SpinnerModel	A model for a potentially unbounded sequence of object values.
SwingConstants	A collection of constants generally used for positioning and orienting components on the screen.
UIDefaults.ActiveValue	This class enables one to store an entry in the defaults table that's constructed each time it's looked up with one of the `getXXX(key)` methods.
UIDefaults.LazyValue	This class enables one to store an entry in the defaults table that isn't constructed until the first time it's looked up with one of the `getXXX(key)` methods.
WindowConstants	Constants used to control the window-closing operation.

public class **JFrame**
extends Frame
implements WindowConstants, Accessible, RootPaneContainer

An extended version of `java.awt.Frame` that adds support for the JFC/Swing component architecture. You can find task-oriented documentation about using `JFrame` in *The Java Tutorial*, in the section How to Make Frames.

The `JFrame` class is slightly incompatible with `Frame`. Like all other JFC/Swing top-level containers, a `JFrame` contains a `JRootPane` as its only child. The **content pane** provided by the root pane should, as a rule, contain all the nc the AWT `Frame` case. For example, to add a child to an AWT frame you'd write:

```
frame.add(child);
```

However using `JFrame` you need to add the child to the `JFrame's` content pane instead:

```
frame.getContentPane().add(child);
```

The same is true for setting layout managers, removing components, listing children, and so on. All these methods should normally be sent to the content pane instead of the JFrame itself. The content pane will always be non-null. A exception. The default content pane will have a BorderLayout manager set on it.

update

```
public void update(Graphics g)
```

Just calls `paint(g)`. This method was overridden to prevent an unnecessary call to clear the background.

Overrides:

`update` in class `Container`

Parameters:

`g` - the Graphics context in which to paint

See Also:

`Component.update(Graphics)`

Kornel Kisielewicz @epyoncf 12 Aug

ProTip: "//" is the speedup operator. Use // before the statement you want to speed up. Works in C++, Java and a few others!

Retweeted by Mathieu Acher

Collapse Reply Retweeted Favorite More

1,253 RETWEETS **295** FAVORITES

12:31 AM - 12 Aug 13 · Details

Coding Conventions

- Rules on the coding style :
  - Apache, Oracle and others template
    - https://www.oracle.com/java/technologies/javase/codeconventions-contents.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

```
  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 …

Logging

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

- 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

- Popu
- Desig
- Simp
- Allow
  stater
- Fully c
  config

Apache LOG4J™

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 (possibly)
- 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*

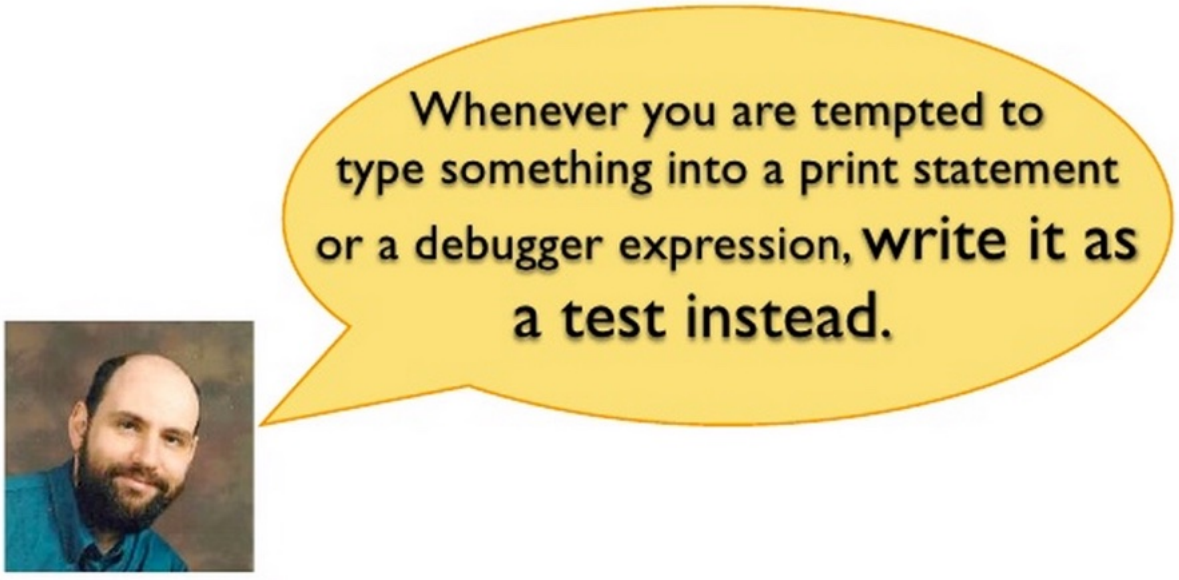
Log4j in Action

```
// 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

- Use 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
- Use Logger.isDebugEnabled() if need be
- Prioritize messages with proper levels

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



Testing and Static Analysis

…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 »

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.WebAii.VSIP.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

10. HealthCare.gov didn't have enough testing before going live.

This became clear in a series of Congressional hearings, where federal contractors testified that end-to-end testing only began in the final weeks of September, right before the Oct. 1 launch. When pressed on how much time would have been ideal for testing, one contractor told lawmakers that "months would have been nice."

http://www.washingtonpost.com/blogs/wonkblog/wp/2013/11/01/thirty-one-things-we-learned-in-healthcare-govs-first-31-days/

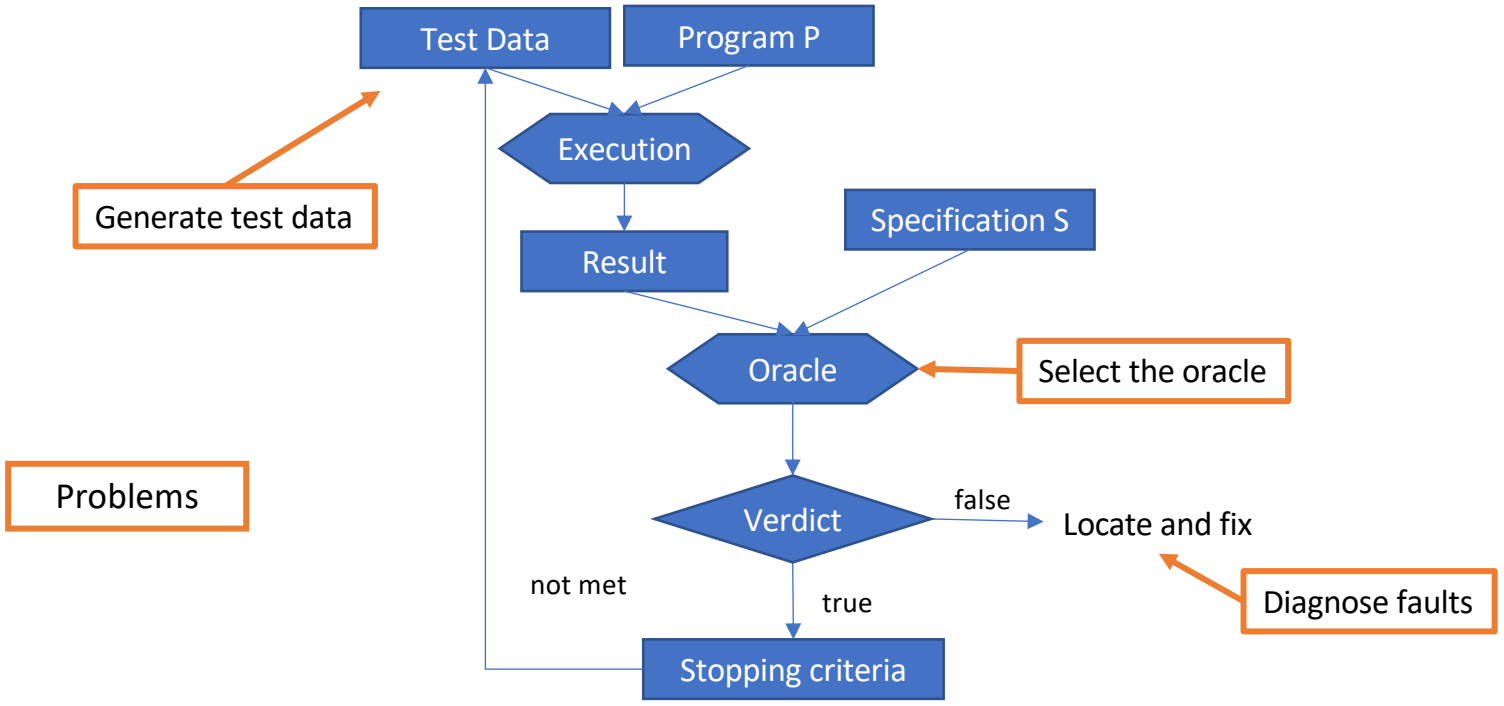
Test phases

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)

Le test dynamique : processus



Test unitaire OO

- Tester une unité isolée du reste du système
- L'unité est la classe
  - Test unitaire = test d'une classe
- Test du point de vue client
  - les cas de tests appellent les méthodes depuis l'extérieur
  - on ne peut tester que ce qui est public
  - Le test d'une classe se fait à partir d'une classe extérieure
- Au moins un cas de test par méthode publique
- Il faut choisir un ordre pour le test
  - quelles méthodes sont interdépendantes?

Test unitaire OO

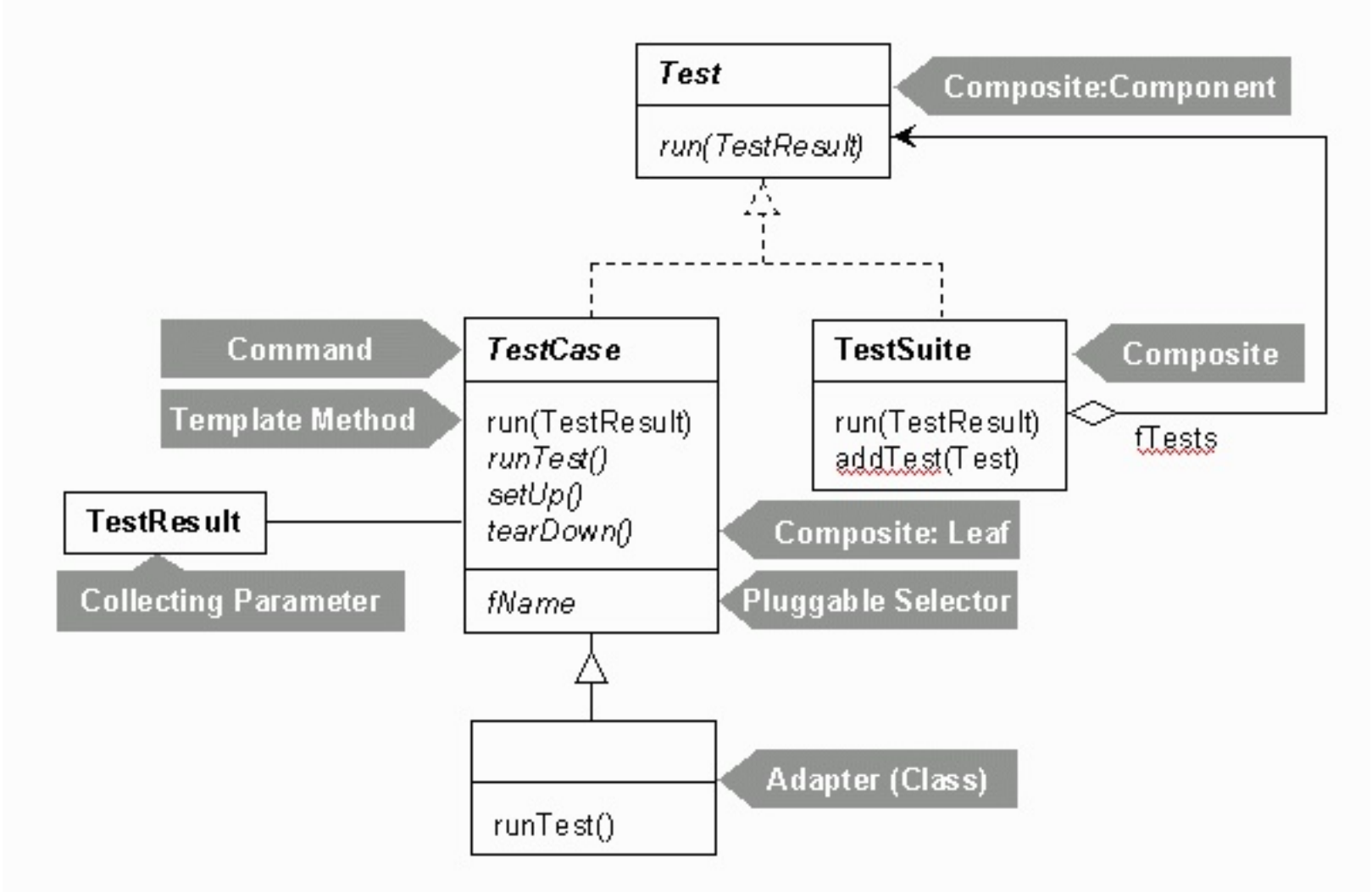
- Problème pour l'oracle :
  - Encapsulation : les attributs sont souvent privés
  - Difficile de récupérer l'état d'un objet
- Penser au test au moment du développement (« testabilité »)
  - prévoir des accesseurs en lecture sur les attributs privés
  - des méthodes pour accéder à l'état de l'objet

Cas de test unitaire

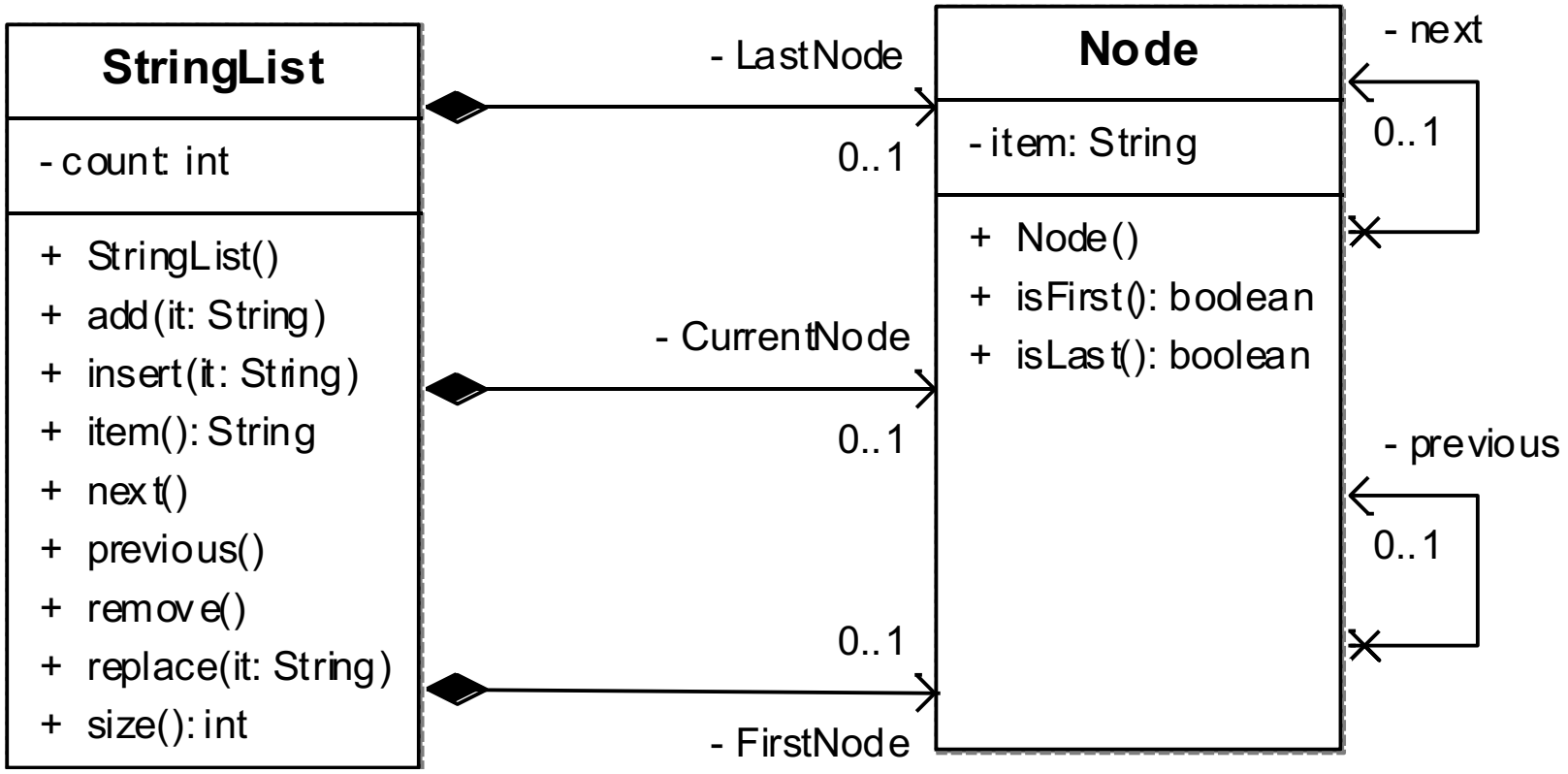
- Cas de test = une méthode
- Corps de la méthode
  - Configuration initiale
  - Une donnée de test
    - un ou plusieurs paramètres pour appeler la méthode testée
  - Un oracle
    - il faut construire le résultat attendu
    - ou vérifier des propriétés sur le résultat obtenu
- Une classe de test pour une classe testée
  - Regroupe les cas de test
  - Il peut y avoir plusieurs classes de test pour une classe testée

Junit and… Design Patterns

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

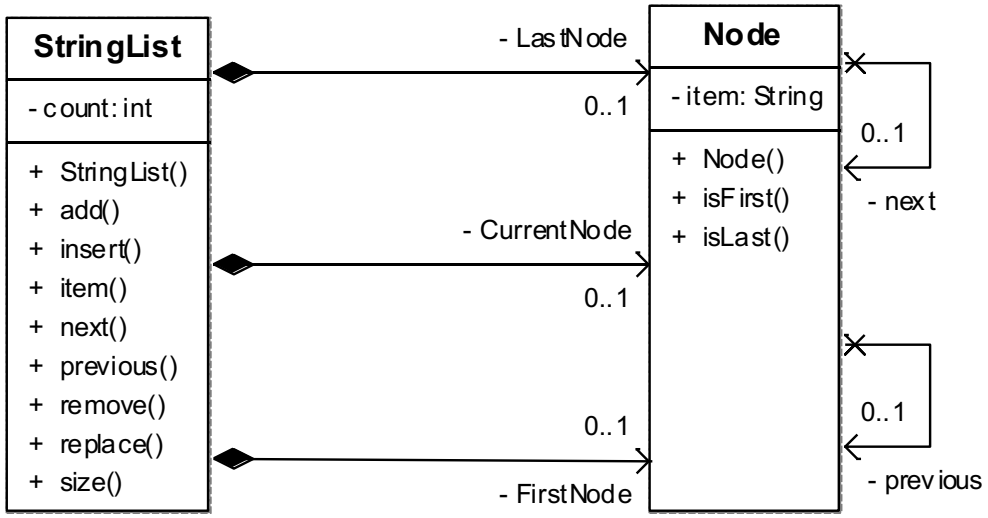


Exemple : test de StringList



- Créer une classe de test qui manipule des instances de la classe StringList
- Au moins 9 cas de test (1 par méthode publique)
- Pas accès aux attributs privés : count, LastNode, CurrentNode, FirstNode

Exemple : insertion dans une liste



spécification du cas de test

initialisation

appel avec donnée de test

oracle

```
//first test for insert: call insert
//and see if current element is the
//one that's been inserted
public void testInsert1(){
    list.add("first");
    list.add("second");
    list.insert("third");
    assertTrue(list.size()==3);
    assertTrue(list.item()=="third");
}
```

JUnit: codage

- Organisation du code des tests
  - cas de Test: TestCase
    - setUp() et tearDown()
    - les méthodes de test
  - suite de Test: TestSuite
    - Méthodes de test
    - Cas de test
    - Suite de Test

JUnit

- Une classe de test regroupe des cas de test pour une classe

```
import org.junit.Before;
import org.junit.Test;

public class TestStringList {
	//déclaration des instances
	private StringList list;

	//setUp()
	//tearDown()
	//méthodes de test
	//main()
}
```

JUnit

- la méthode setUp:

```
//appelée avant chaque cas de test
//permet de factoriser la construction de l'état initial
@Before
protected void setUp() throws Exception {
    list = new StringList();
}
```

- la méthode tearDown:

```
//appelée après chaque cas de test
//permet de défaire « l'état du monde »
@After
protected void tearDown() throws Exception {
    super.tearDown();
}
```

JUnit

- les méthodes de test:

```
//test add two elements
@Test
public void testAdd2(){
   list.add("first");
   list.add("second");
   assertTrue(list.size()==2);
   assertTrue(list.item()=="second");
}
```

Les assertions de JUnit

- fail() / fail(String message)
- assertTrue(...), assertFalse(...)
- assertEquals(Object, Object)
- assertSame(Object, Object)
- assertNull(...)
- assertEquals(double expected, double actual, double delta)

Voir la liste des méthodes de la classe `Assert`

JUnit v3 : détail d'implémentation

- Pour exécuter une suite de tests, JUnit utilise l'introspection

```
public TestSuite (final Class theClass){
        …
        Method[] = theClass.getDeclaredMethods
        …
}
private boolean isTestMethod(Method m) {
        String name= m.getName();
        Class[] parameters= m.getParameterTypes();
        Class returnType= m.getReturnType();
        return parameters.length == 0 && name.startsWith("test") &&
returnType.equals(Void.TYPE);
         }
```

JUnit version 4/5

- Fonctionne avec Java 5+
- Utilisation intensive des annotations
- Tests paramétrés, timeouts, etc

JUnit v4/5 : classe et méthode de test

- Classe de test :
  - `import org.junit.Test;`
  - `import static org.junit.Assert.*;`
- Méthodes de test :
  - Nom de méthode quelconque
  - Annotation `@Test`
  - Publique, type de retour `void`
  - Pas de paramètre, peut lever une exception
  - Annotation `@Test(expected = Class)` pour indiquer l'exception attendue
  - Annotation `@Ignore` pour ignorer un test

JUnit

- Permet de structurer les cas de test
  - cas de test / suite de test
- Permet de sauvegarder les cas de test
  - important pour la non régression
  - quand une classe évolue on ré-exécute les cas de test

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 (IDE or external)
  - 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)

- Logger
  - Record events on a registry, to be used at execution time or later on (via some event handlers)
  - Tools
    - Apache Log4J, ObjectWeb MonoLog
    - Package java.util.logging since J2SE1.4
      - Logger, LogRecord, Handler

Validation

- Assertion
  - Pre-Condition, Post-Condition, Invariant
    - EIFFEL, CLU ... built-in
    - Java since SE 1.4
- Other tools
  - 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/
    - tool support with https://www.openjml.org/
  - iContract (Reliable Systems)
  - ...

Design by Contract with JML (by Gary T. Leavens and Yoonsik Cheon)

Performances

- Measure/Analyze
  - Benchmark
  - java.awt.Robot (to build clients for testing)
  - Accounting : http://abone.unige.ch/jraf/index.htm
  - JProfiler, OptimiszeIt …
  - JMH
- Optimization
- See books:
  - Steve Wilson, Jeff Kesselman, « Java Platform Performance: Strategies and Tactics (The Java Series) », 1 edition (May 25, 2000), Addison-Wesley Pub Co; ISBN: 0-201-70969-4
  - Jack Shirazi, “Java Performance Tuning”, Ed Oreilly, 2000, ISBN 0-596-00015-4

Choosing the JVM and JRE

- Some criteria
  - License, redistribution, supports, performances, contraintes (embedded, servers, réal-time, …), runtimes, …
- Examples (cf. https://en.wikipedia.org/wiki/List_of_Java_virtual_machines)
  - Azul Zulu
  - Bck2Brwsr
  - CACAO
  - Codename One
  - DoppioJVM
  - Eclipse OpenJ9
  - GraalVM
  - HaikuVM
  - HotSpot
  - Jamiga
  - JamVM
  - Jikes RVM (Jikes Research Virtual Machine)
  - JVM.go
  - leJOS
  - Maxine
  - Multi-OS Engine
  - RopeVM
  - …

Measurements and Analysis of Performances

- Java Profiler
  - Use to profile an application
    - CPU usage, Memory, Network, time spent, and Garbage collection
    - In Total or by threads, or called methods

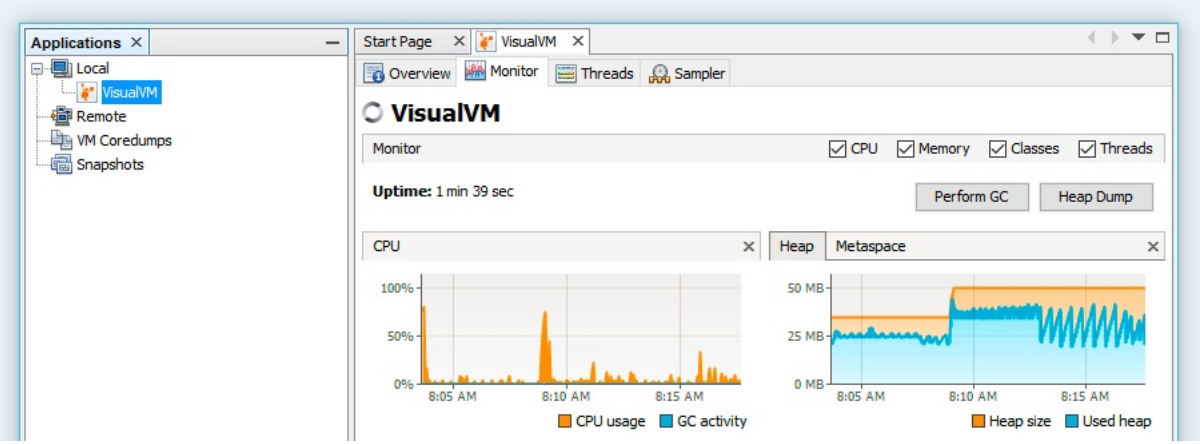
JProfiler

YourKit

VisualVM

https://visualvm.github.io/

- « VisualVM is a visual tool that integrates several existing JDK software tools and lightweight memory and CPU profiling capabilities. This tool is designed for both production and development time use and further enhances the capability of monitoring and performance analysis for the Java SE platform.»
- **VisualVM includes the JConsole.**



Performance Optimizations

- Java's Script Engine
  - Jython (jython.org), …
- Bytecode interpreter, JIT and compiler
  - GraalVM, OpenJ9, Azul…
- Native compiler (static)
  - .class to .c to .s to .exe
- On-the-fly compiler (dynamic)
  - Compilation JIT (Just-In-Time) de Symantec
- HotSpot™ Optimizer
  - garbage collector
  - « method inlining »
    - with load-time verification (dynamic) of class bytecode
- Benchmark de JVM (e.g., JMH, Krun)

Code Quality Metrics

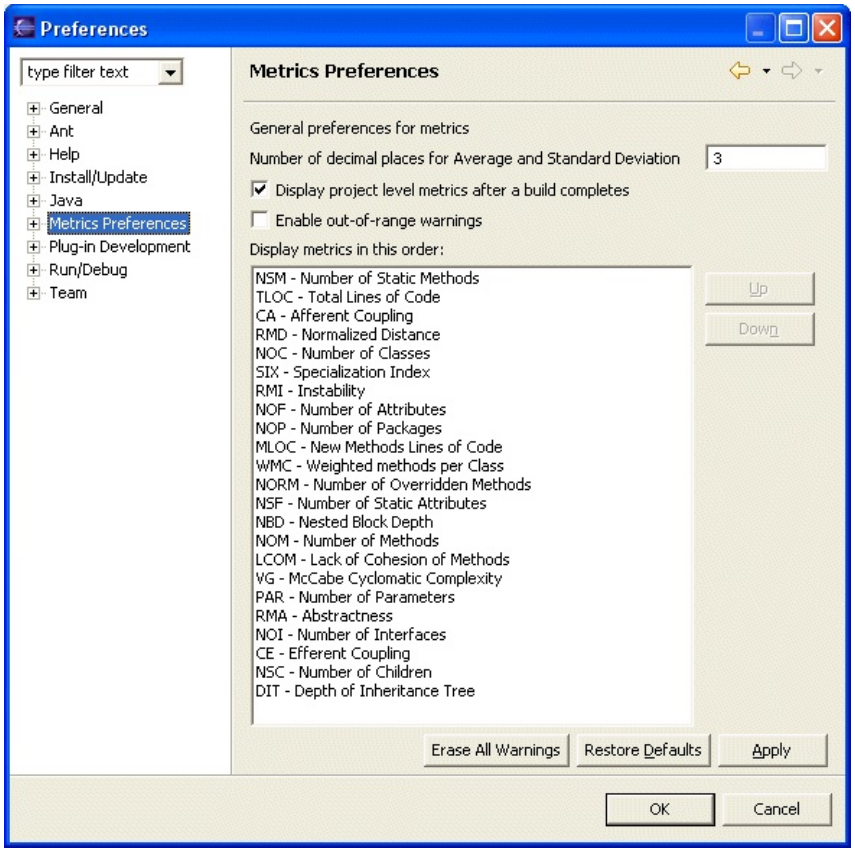
- Metrics on project source code to evaluate its quality (maintenance,reverse-engineering,evolution …)
  - and how good the development team is :-)
- Metrics
  - LOC, LOCC, McCabe Cyclomatic Complexity, …

Lectures

- Brian Henderson-Sellers , "Object-Oriented Metrics, measures of Complexity", Ed Prentice Hall, 1996
- Robert Martin, "OO Design Quality Metrics, An Analysis of Dependencies", 1994, http://www.objectmentor.com/resources/articles/oodmetrc.pdf
- Chidamber and Kemerer, A Metrics Suite for Object Oriented Design, http://www.pitt.edu/~ckemerer/CK%20research%20papers/MetricForOOD_ChidamberKemerer94.pdf
- Mariano Ceccato and Paolo Tonella, Measuring the Effects of Software Aspectization, http://homepages.cwi.nl/~tourwe/ware/ceccato.pdf
- Robert Martin, "Agile Software Development, Principles, Patterns and Practices", Prentice Hall, 1st edition, 2002, ISBN: 978-0135974445

Code Quality Metrics

- Example: Metrics (Ant/Maven/Sonar + Eclipse plugin)



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:

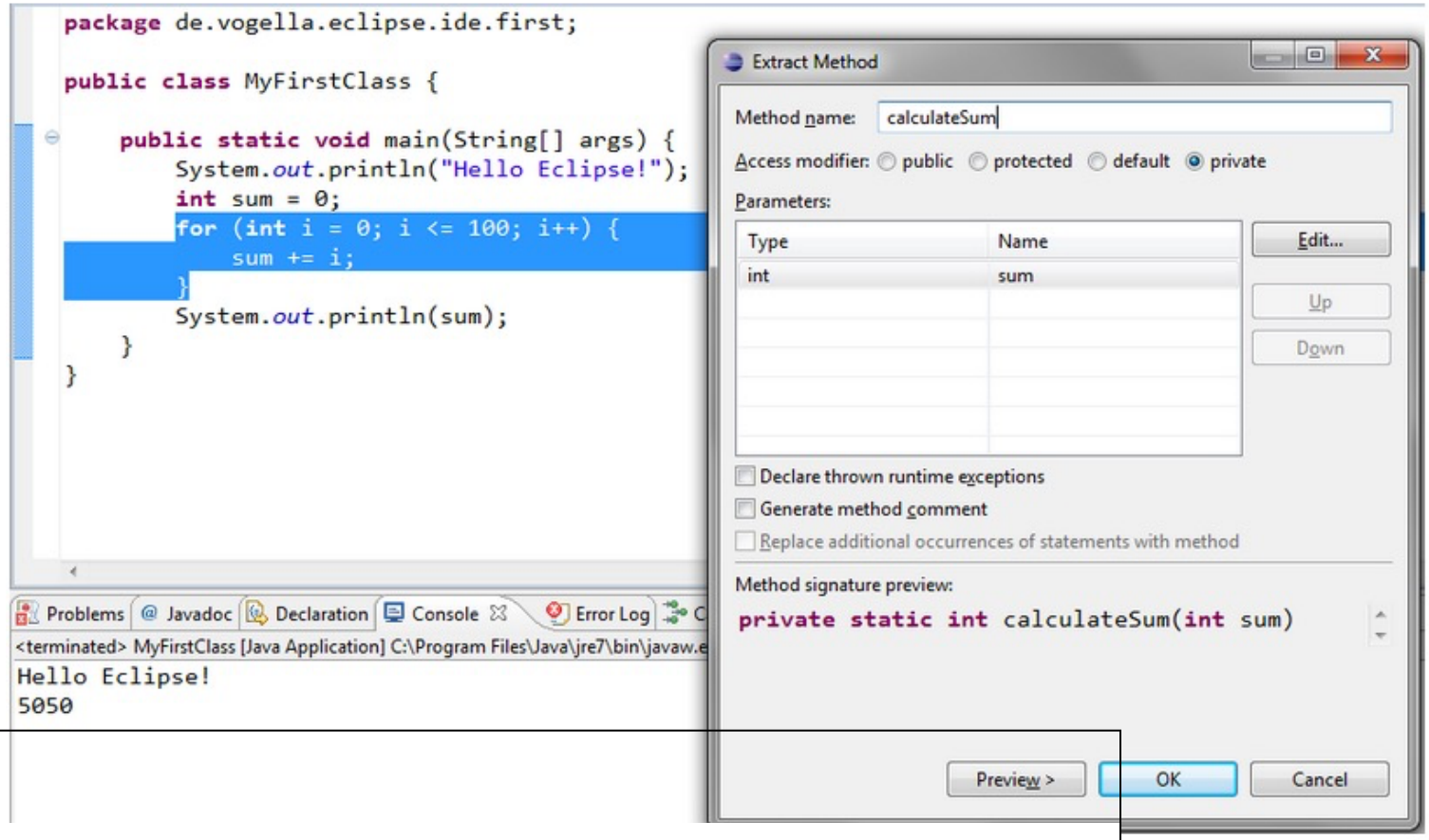
```
if (isValid(markT) && isValid(markL)){
        float markAvg = (markT + markL)/2;
        System.out.println("Your mark: " + mark);
}
```

Why do we Refactor?

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

How do we Refactor?

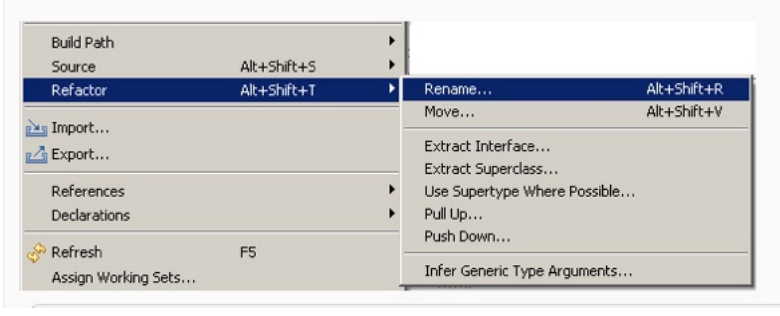
- Manual Refactoring
  - Code Smells
- Automated/Assisted Refactoring
  - Refactoring by hand is time consuming and prone to error
  - Tools (IDE)
- In either case, **test your changes**

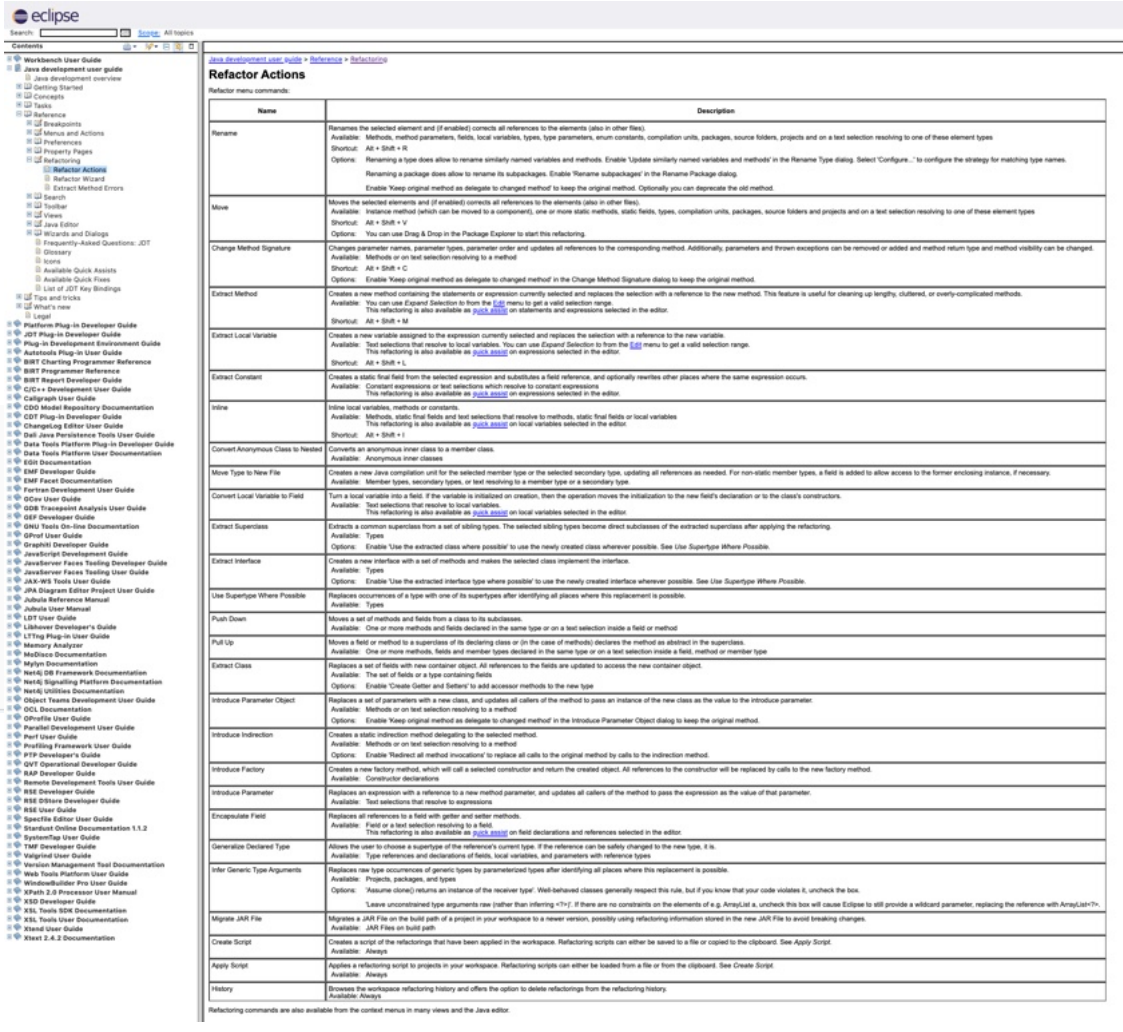



```
package de.vogella.eclipse.ide.first;

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

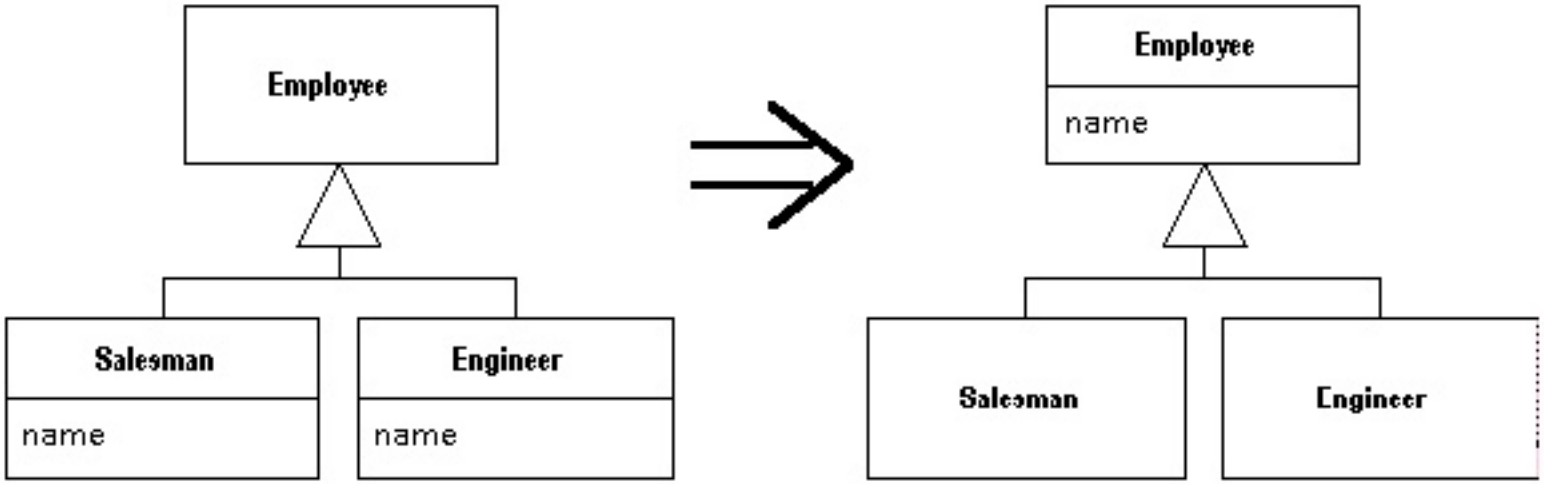
- Rename variable / class / method / member
- Extract method
- Extract constant
- Extract interface
- Encapsulate field





Two subclasses have the same field.

Move the field to the superclass.



You have a complicated expression.

Put the result of the expression, or parts of the expression, in a temporary variable with a name that explains the purpose.

```
if ( (platform.toUpperCase().indexOf("MAC") > -1) &&
      (browser.toUpperCase().indexOf("IE") > -1) &&
       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
        }
```

Build automation

Compilation chain

- Tools (*aka.* build automation systems)
  - make, gmake, nmake (Win),
  - Apache ANT, MAVEN, Gradle, NPM, FinalBuilder, Grunt
- To automate:
  - pre-compilation, obfuscation, verification
  - generation of .class and .jar
    - normal, tracing, debug, …
  - documentation generation
  - « stubs » generation (rmic, idl2java, javacard …)
  - test
  - …

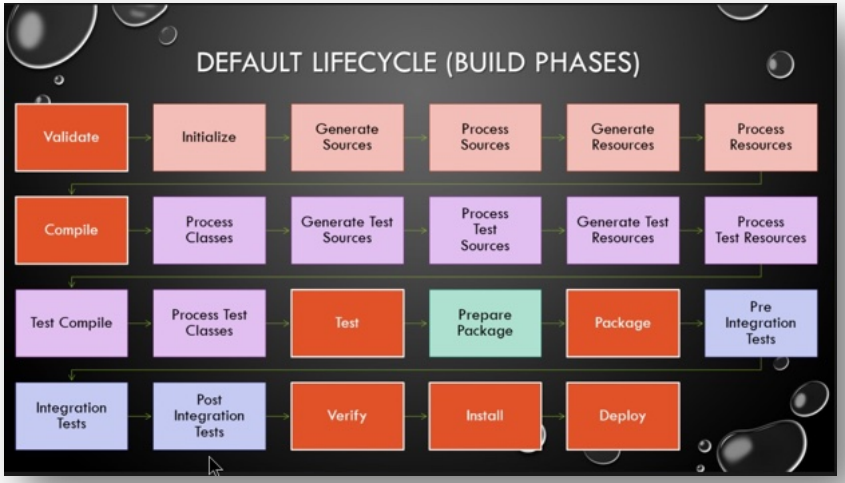
Maven

- 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
  - Automatic updates
- Project repositories
  - public or private, local or remotes
  - caching and proxy
- Extensible via external plugins



Common goals

- **clean** – clean the current project
- **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
- **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

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 …

Creating project website

- Run: mvn site
- Let the build run, it’ll start downloading and creating things left and right
- Eventually in the target dir you end up with a \sivnte dir, with an apache-style project website
- Javadoc, various reports, and custom content can be added

More stuff

- Automatically generate reports, diagrams, and so on through Maven / the project site
- Internationalization – create different language project websites
- Create projects within projects (more pom.xml files inside sub dirs\jars), with different build stats and so on
- Maven can make .war files, EJBs, etc.

Using Maven Plugins

- Whenever you want to customise the build for a Maven project, this is done by adding or reconfiguring plugins
- For example, configure the Java compiler to allow JDK 5.0 sources
- Plugins in Maven 3.0 look much like a dependency

```
...
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <source>1.5</source>
        <target>1.5</target>
      </configuration>
    </plugin>
  </plugins>
</build>
...
```

Maven Plugins

- **AlmostPlainText**
- **Maven Axis**
- **Maven Cobertura**
- **Maven DB2**
- **Dbunit**
- **Debian Package**
- **Maven DotUml**
- **Doxygen**
- **Maven Files**
- **FindBugs**
- **Maven flash**
- **Help**
- **Maven IzPack**
- **Java Application**
- **Maven JAVANCSS**
- **Maven JAXB**
- **JUNITPP**
- **Kodo**
- **Maven Macker**
- **SDocBook**
- **Sourceforge**
- **Maven SpringGraph**
- **RPM Plugin**
- **Runtime Builder**
- **Strutsdoc**
- **Tasks**
- **Maven Transform**
- **Maven UberDist**
- **Maven Vignette**
- **WebSphere 4.0**
- **WebSphere 5 (5.0/5.1)**
- **Maven WebLogic**
- **Canoo WebTest**
- **Wiki**
- **Word to HTML**
- **XML Resume**
- **Maven DotUml**
- **Middlegen**
- **Maven News**

Archetypes

- For reuse, create archetypes that work as project templates with build settings, etc
- An archetype is a project, with its own pom.xml
- An archetype has a descriptor called archetype.xml
- Allows easy generation of Maven projects

Maven plugin for JAVA IDE

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

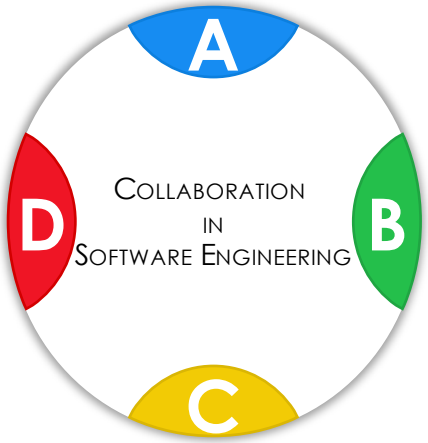
Good things about Maven

- Standardization
- Reuse
- Dependency management
- Build lifecycle management
- Large existing repository
- IDE aware
- One directory layout
- A single way to define dependencies
- Setting up a project is really fast
- Transitive dependencies
- Common build structure
- Use of remote repository
- Web site generation
- Build best practices enforcement
- Automated build of application
- Works well with distributed teams
- All artifacts are versioned and are stored in a repository
- Build process is standardized for all projects
- A lot of goals are available
- It provides quality project information with generated site
- Easy to learn and use
- Makes the build process much easier at the project level
- Promotes modular design of code

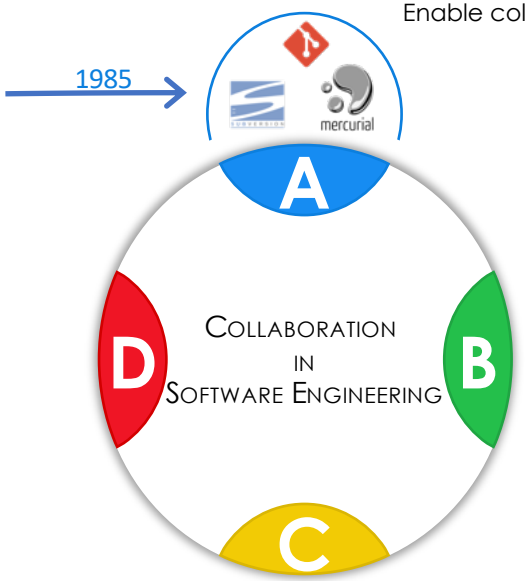
Collaborative Methods in Software Engineering:

a state of the art and practices

Collaboration Landscape in Software Engineering



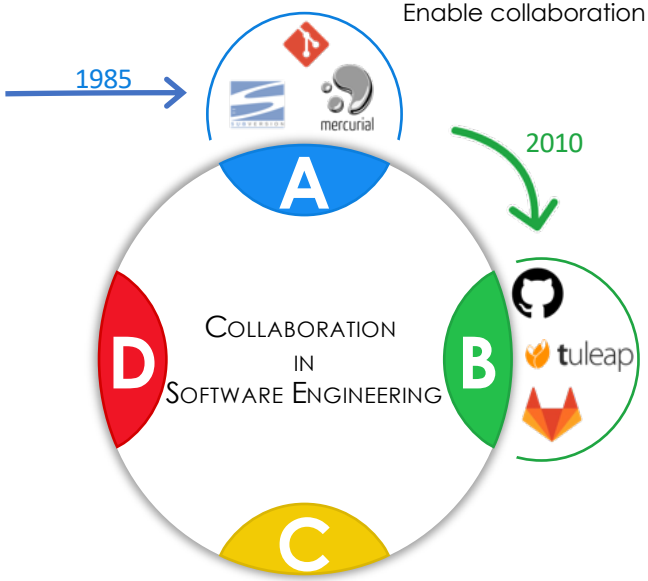
Collaboration Landscape in Software Engineering



Version Control Systems (VCS)

Enable collaboration at the **artifact-level**.

Collaboration Landscape in Software Engineering



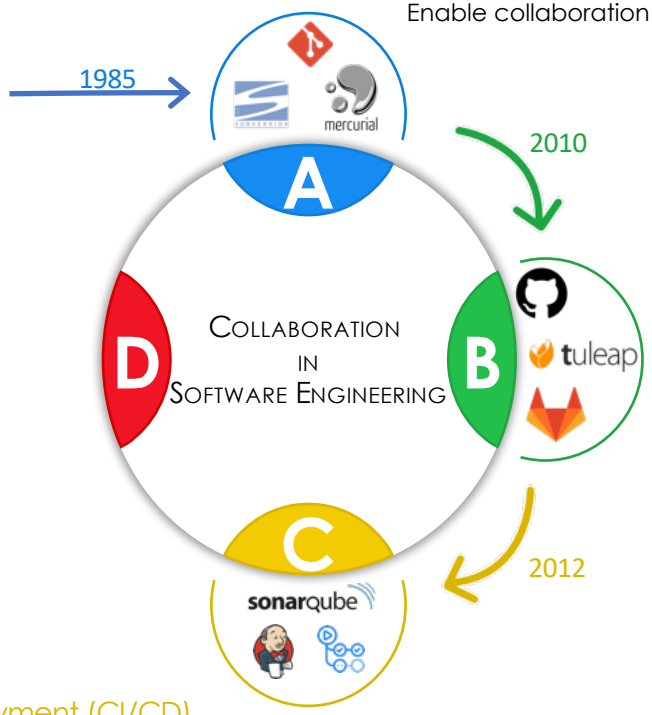
Version Control Systems (VCS)

Enable collaboration at the **artifact-level**.

Software Forges

Enable collaboration at the **social-level**.

Collaboration Landscape in Software Engineering



Version Control Systems (VCS)

Enable collaboration at the **artifact-level**.

Software Forges

Enable collaboration at the **social-level**.

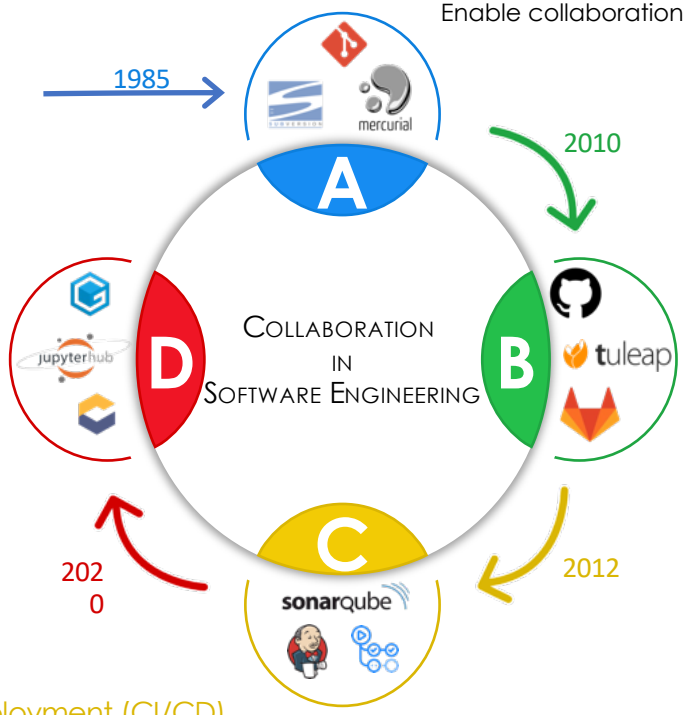
Continuous Integration/Deployment (CI/CD)

Support collaboration at the social-level
by providing insights about artifacts and **automating tasks**.

Collaboration Landscape in Software Engineering

Version Control Systems (VCS)

Enable collaboration at the **artifact-level**.



Web/Cloud IDEs

Enable collaboration at the **environment-level**.

Software Forges

Enable collaboration at the **social-level**.

Continuous Integration/Deployment (CI/CD)

Support collaboration at the social-level
by providing insights about artifacts and **automating tasks**.

Collaboration Landscape in Software Engineering

A. Version Control System (VCS)

Collaboration Landscape in Software Engineering

A. Version Control System (VCS)

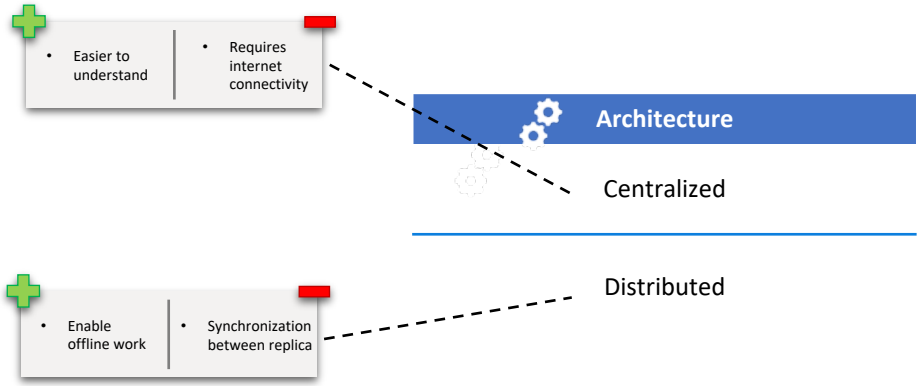
- Enable both **sharing** and **versioning** of **artifacts** [5,6] (source files, models, etc.)

[5] Rochkind, M. J. (1975). The source code control system. IEEE transactions on Software Engineering, (4), 364-370.
[6] Uzunbayir, S., & Kurtel, K. (2018, September). A review of source code management tools for continuous software development. In 2018 3rd International Conference on Computer Science and Engineering (UBMK) (pp. 414-419). IEEE.

Collaboration Landscape in Software Engineering

A. Version Control System (VCS)

- Enable both **sharing** and **versioning** of **artifacts** [5,6] (source files, models, etc.)

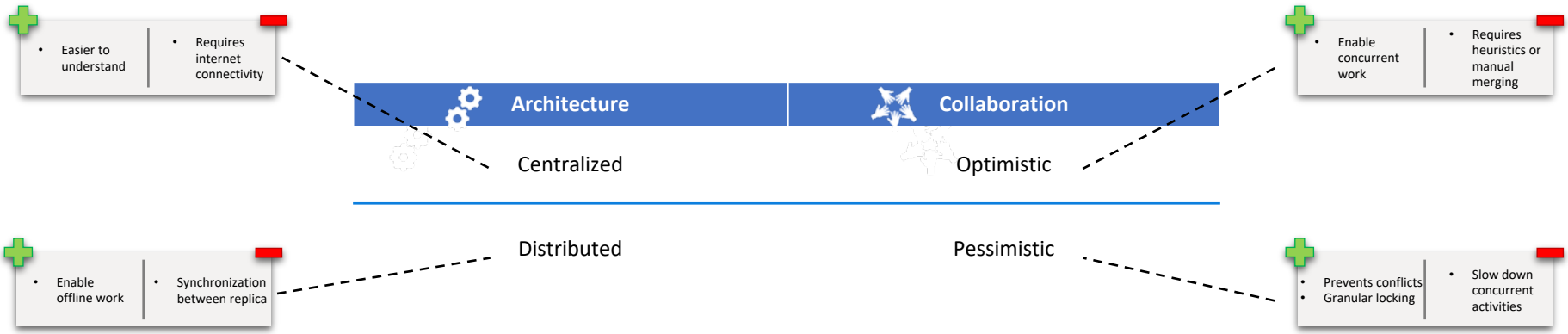


[5] Rochkind, M. J. (1975). The source code control system. IEEE transactions on Software Engineering, (4), 364-370.
[6] Uzunbayir, S., & Kurtel, K. (2018, September). A review of source code management tools for continuous software development. In 2018 3rd International Conference on Computer Science and Engineering (UBMK) (pp. 414-419). IEEE.

Collaboration Landscape in Software Engineering

A. Version Control System (VCS)

- Enable both **sharing** and **versioning** of **artifacts** [5,6] (source files, models, etc.)



[5] Rochkind, M. J. (1975). The source code control system. IEEE transactions on Software Engineering, (4), 364-370.
[6] Uzunbayir, S., & Kurtel, K. (2018, September). A review of source code management tools for continuous software development. In 2018 3rd International Conference on Computer Science and Engineering (UBMK) (pp. 414-419). IEEE.

Collaboration Landscape in Software Engineering

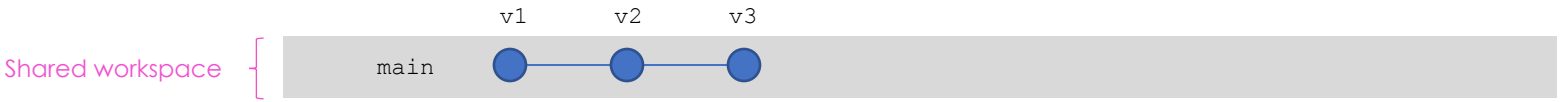
A. Version Control System (VCS)

- Enable both **sharing** and **versioning** of **artifacts** (source files, models, etc.)
- Ease collaboration through **branch engineering** (branch = isolated workspace)

Collaboration Landscape in Software Engineering

A. Version Control System (VCS)

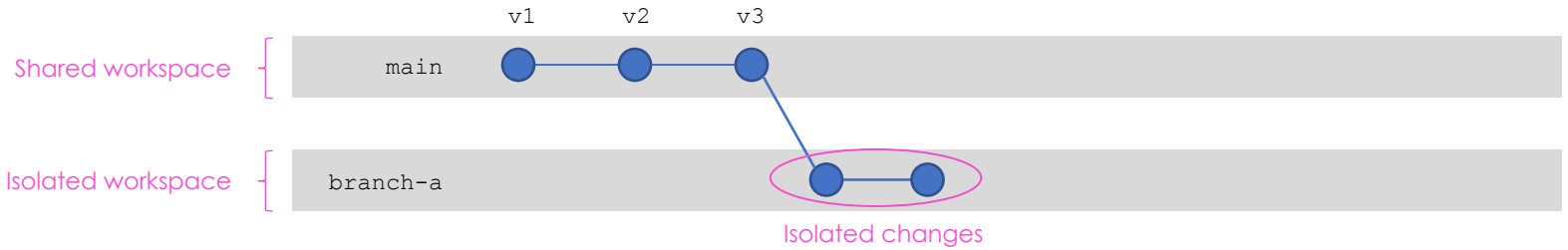
- Enable both **sharing** and **versioning** of **artifacts** (source files, models, etc.)
- Ease collaboration through **branch engineering** (branch = isolated workspace)



Collaboration Landscape in Software Engineering

A. Version Control System (VCS)

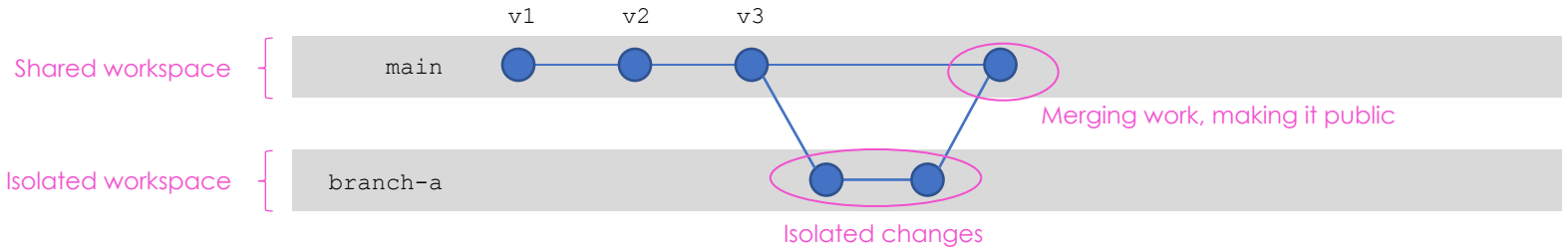
- Enable both **sharing** and **versioning** of **artifacts** (source files, models, etc.)
- Ease collaboration through **branch engineering** (branch = isolated workspace)



Collaboration Landscape in Software Engineering

A. Version Control System (VCS)

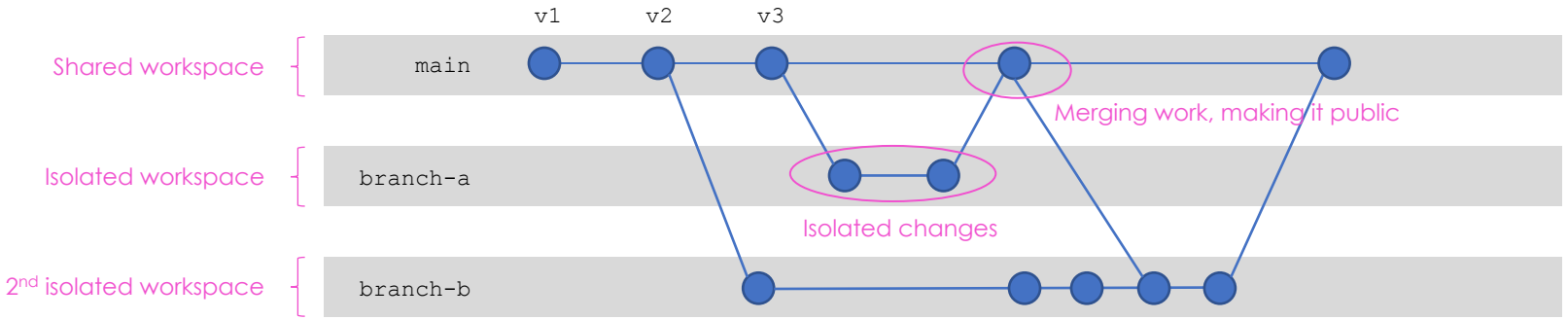
- Enable both **sharing** and **versioning** of **artifacts** (source files, models, etc.)
- Ease collaboration through **branch engineering** (branch = isolated workspace)



Collaboration Landscape in Software Engineering

A. Version Control System (VCS)

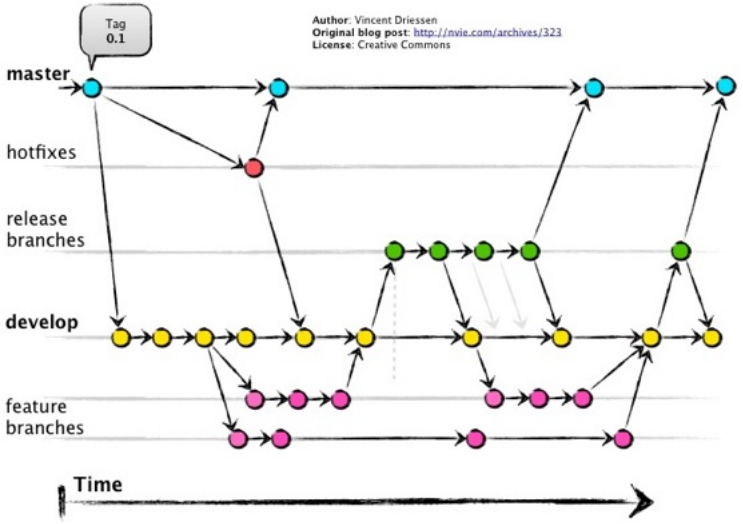
- Enable both **sharing** and **versioning** of **artifacts** (source files, models, etc.)
- Ease collaboration through **branch engineering** (branch = isolated workspace)



Collaboration Landscape in Software Engineering

A. Version Control System (VCS)

- Enable both **sharing** and **versioning** of **artifacts** (source files, models, etc.)
- Ease collaboration through **branch engineering** (branch = isolated workspace)



Gitflow [7]

A well-spread **method** for branch engineering that uses branches for:

- individual features,
- releases,
- hotfix,
- ongoing development,
- production.

[7] https://nvie.com/posts/a-successful-git-branching-model/

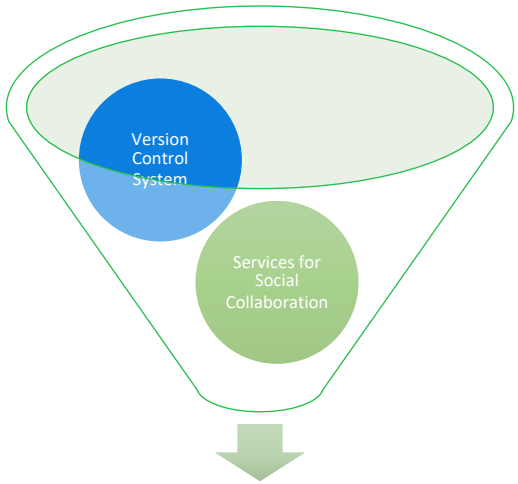
Collaboration Landscape in Software Engineering

B. Software Forges


Collaboration Landscape in Software Engineering

B. Software Forges

- Provide **social collaboration** on top of VCS [8] (planification, discussions, reviews, etc.)
- Together they enable **socio-technical coordination**.



[8] Riehle, D., Ellenberger, J., Menahem, T., Mikhailovski, B., Natchetoi, Y., Naveh, B., & Odenwald, T. (2009). Open collaboration within corporations using software forges. *IEEE software*, *26*(2), 52-58.

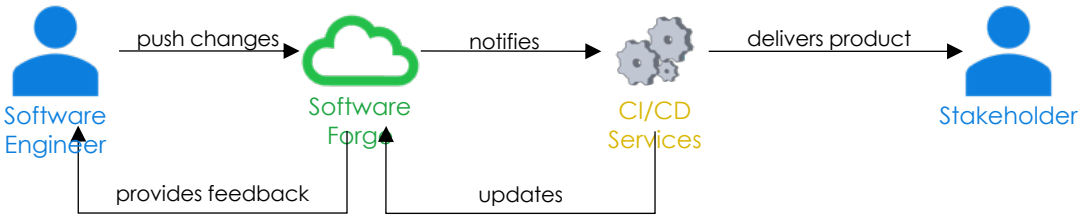
Collaboration Landscape in Software Engineering

C. Continuous Integration/Deployment (CI/CD)

Collaboration Landscape in Software Engineering

C. Continuous Integration/Deployment (CI/CD)

- **Automate recurrent tasks** relative to the **integration of changes** (build, quality/validity checks) and **product deployment** [4,9].
- Support **early feedback** to engineers regarding their contributions.
- Support **continuous product delivery** from software implementation.
- Optional yet more and more prevalent.



[4] Leite, L., Rocha, C., Kon, F., Milojicic, D., & Meirelles, P. (2019). A survey of DevOps concepts and challenges. ACM Computing Surveys (CSUR), 52(6), 1-35.
[9] Shahin, M., Babar, M. A., & Zhu, L. (2017). Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. IEEE Access, 5, 3909-3943.

Collaboration Landscape in Software Engineering

D. Web/Cloud IDEs

Collaboration Landscape in Software Engineering

D. Web/Cloud IDEs

- Enable the **easy setup** of a **pre-configured environment** [10,11,12].

[10] Aho, T., Ashraf, A., Englund, M., Katajamäki, J., Koskinen, J., Lautamäki, J., ... & Turunen, I. (2011). Designing IDE as a service. *Communications of Cloud Software*, *1*(1), 1-10.
[11] Gadhikar, L. M., Mohan, L., Chaudhari, M., Sawant, P., & Bhusara, Y. (2013). Browser based IDE to code in the cloud. In *New Paradigms in Internet Computing* (pp. 59-69). Springer, Berlin, Heidelberg.
[12] Devadiga, A., Kumar, S., Rachhadiya, M., & Sahitya, A. (2021). Integrated Development Environment on Cloud. *Available at SSRN 3867647*.

Collaboration Landscape in Software Engineering

D. Web/Cloud IDEs

- Enable the **easy setup** of a **pre-configured environment** [10,11,12].

Local IDE

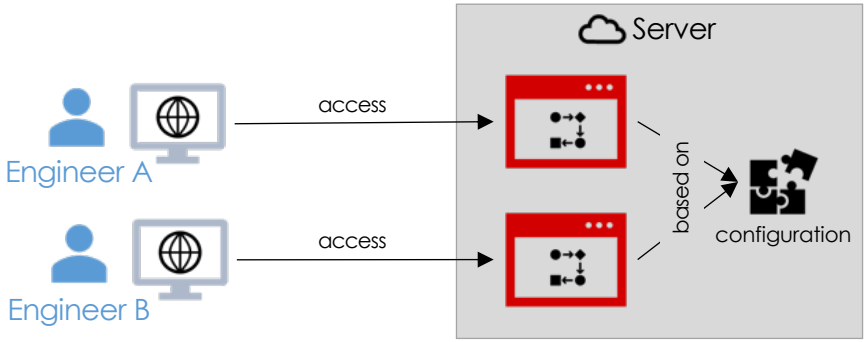
- takes time to configure
- may be OS-dependent
- developer-dependent configuration
- changes to the environment are hard to propagate

[10] Aho, T., Ashraf, A., Englund, M., Katajamäki, J., Koskinen, J., Lautamäki, J., ... & Turunen, I. (2011). Designing IDE as a service. *Communications of Cloud Software*, *1*(1), 1-10.
[11] Gadhikar, L. M., Mohan, L., Chaudhari, M., Sawant, P., & Bhusara, Y. (2013). Browser based IDE to code in the cloud. In *New Paradigms in Internet Computing* (pp. 59-69). Springer, Berlin, Heidelberg.
[12] Devadiga, A., Kumar, S., Rachhadiya, M., & Sahitya, A. (2021). Integrated Development Environment on Cloud. *Available at SSRN 3867647*.

Collaboration Landscape in Software Engineering

D. Web/Cloud IDEs

- Enable the **easy setup** of a **pre-configured environment** [10,11,12].



Local IDE

- takes time to configure
- may be OS-dependent
- developer-dependent configuration
- changes to the environment are hard to propagate

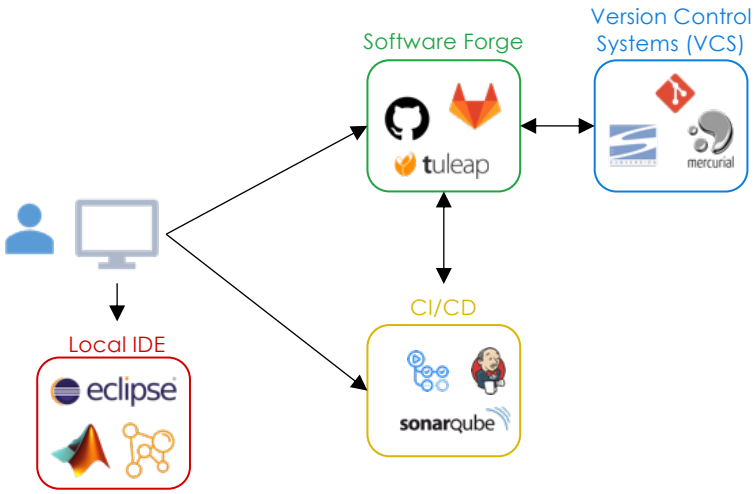
Web/Cloud IDEs

- can be accessed from a simple URL and ready in a few seconds
- cross-platform
- shared across (possibly geographically distributed) team members
- changes to the environment are automatically propagated

[10] Aho, T., Ashraf, A., Englund, M., Katajamäki, J., Koskinen, J., Lautamäki, J., ... & Turunen, I. (2011). Designing IDE as a service. *Communications of Cloud Software*, *1*(1), 1-10.
[11] Gadhikar, L. M., Mohan, L., Chaudhari, M., Sawant, P., & Bhusara, Y. (2013). Browser based IDE to code in the cloud. In *New Paradigms in Internet Computing* (pp. 59-69). Springer, Berlin, Heidelberg.
[12] Devadiga, A., Kumar, S., Rachhadiya, M., & Sahitya, A. (2021). Integrated Development Environment on Cloud. *Available at SSRN 3867647*.

Current Trends

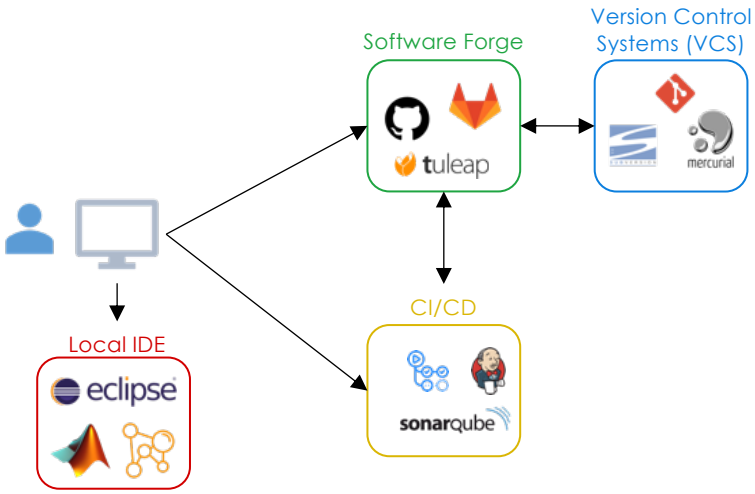
Today



A **constellation** of distinct tools

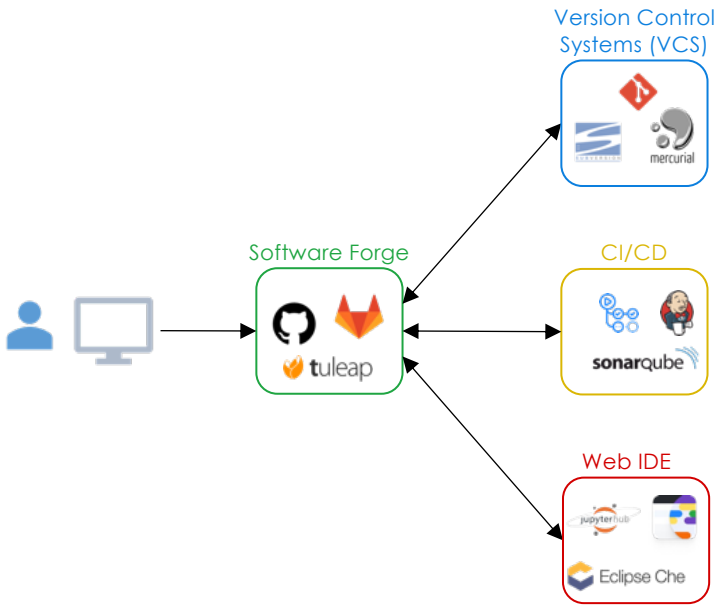
Current Trends

Today



A **constellation** of distinct tools

Tomorrow



A **unification** of tools as parts of software forges

Versioning

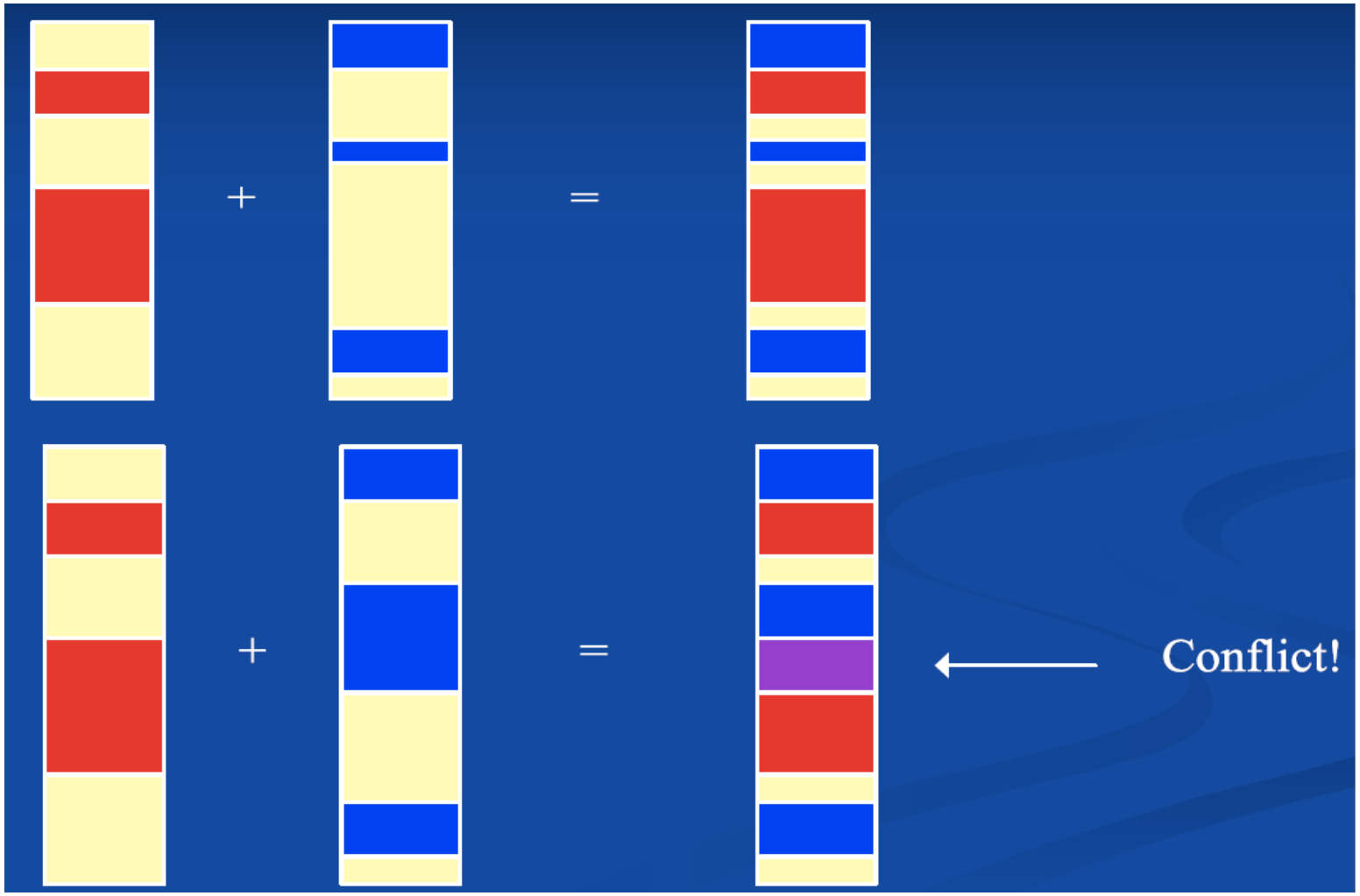
Versioning of source code

- Collaborative software engineering
- To master
  - revisions over the time
  - concurrent activities by large developer teams
  - parallel implementations (experiments, vendors, environment)
- Goals
  - Increase productivity of developers and software robustness
  - Low-down development costs
- Manage software system configuration
  - to control software system's 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



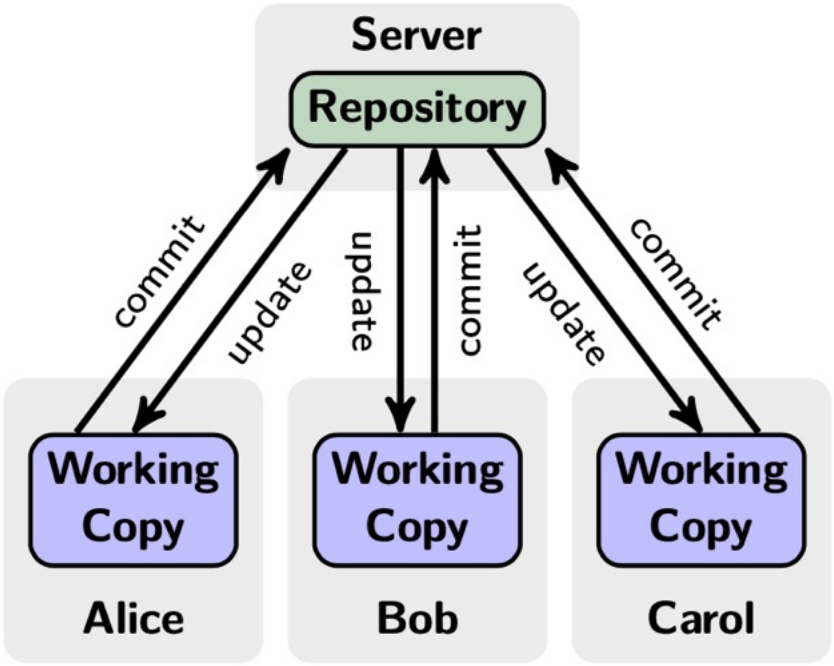
Concurrency control

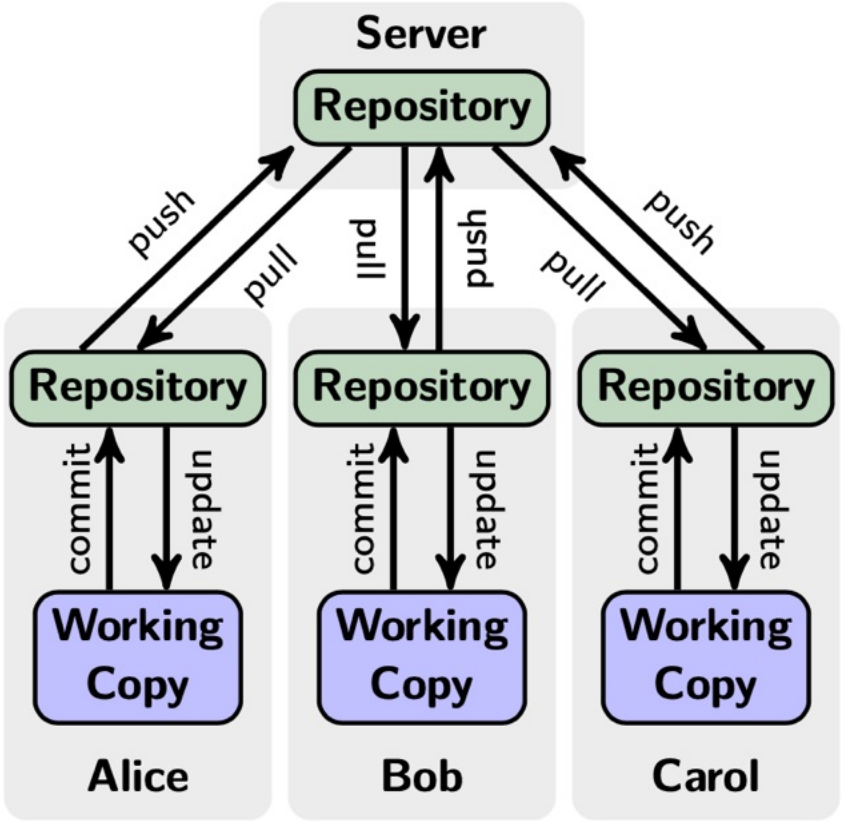
- Doing nothing!
- Lock-Modify-Unlock (Pessimistic)
  - SCCS, RCS
  - Decrease productivity
- Copy-Modify-Merge (Optimistic)
  - Conflicts resolution when concurrent modifications (which are actually rare)
    - Merge, Selection, …
  - CVS, SVN, Git : Client level resolution
- Policy-based
  - Merging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

Centralized vs. Distributed





- **Repository**: hold the whole history of the project (as a data-base of code)
- **Working Copy**: a snapshot of the code taken from the repository on which you can work

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

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

Tools

	Gestion locale	Client-Serveur	Décentralisé
Libre	1972 - SCCS 1982 - RCS	1990 - **CVS** 1992 - CVSNT 2000 - **Subversion**	2000 - Bitkeeper 2001 - GNU arch 2002 - Darcs, DCVS 2003 - ArX, SVK, Monotone 2005 - Bazaar, Codeville, **Git**, Mercurial 2007 - Fossil 2011 - Veracity 2015 - Pijul
Propriétaire	1985 - PVCS 1991 - QVCS	1992 - Rational ClearCase 1994 - Visual SourceSafe 1995 – Perforce, CCC/Harvest, Starteam 2002 - AccuRev 2003 – Sourceanywhere, Vault	2006 - Plastic SCM

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.
- https://git-scm.com (http://git.or.cz)

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

Modèle d'utilisation

"On crée une histoire en local et on partage ensuite"

Pull

Récupère les changements d'une base distante dans une base locale et un répertoire de travail local

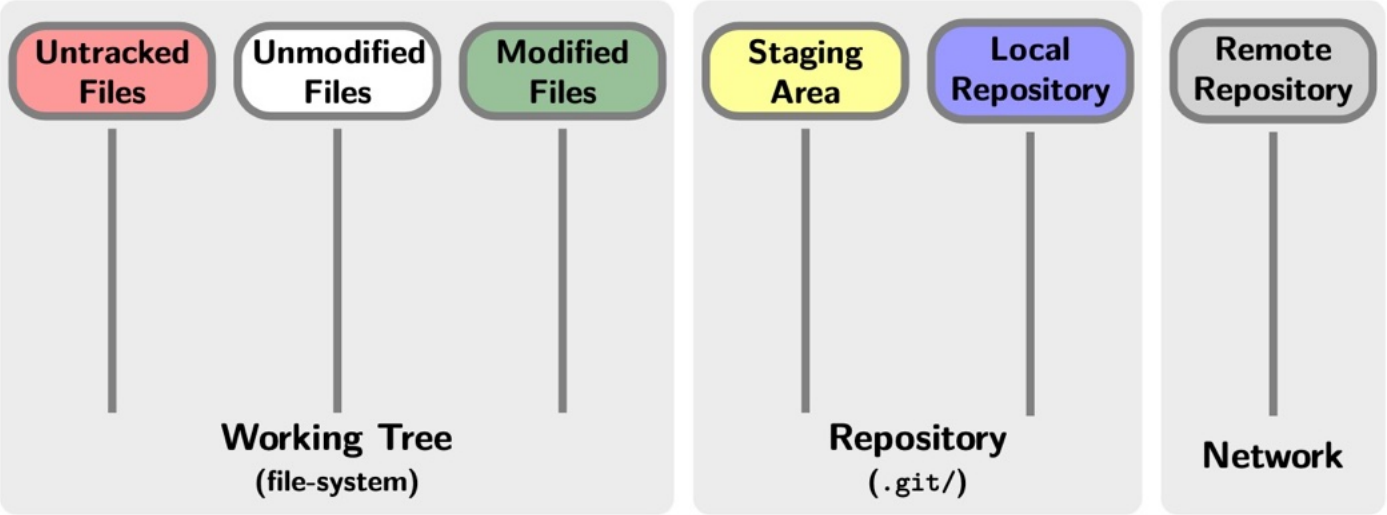
Commit

Valide les changements de la copie de travail pour les ajouter à la base locale

Push

Envoie les changement vers une base distance

Files States

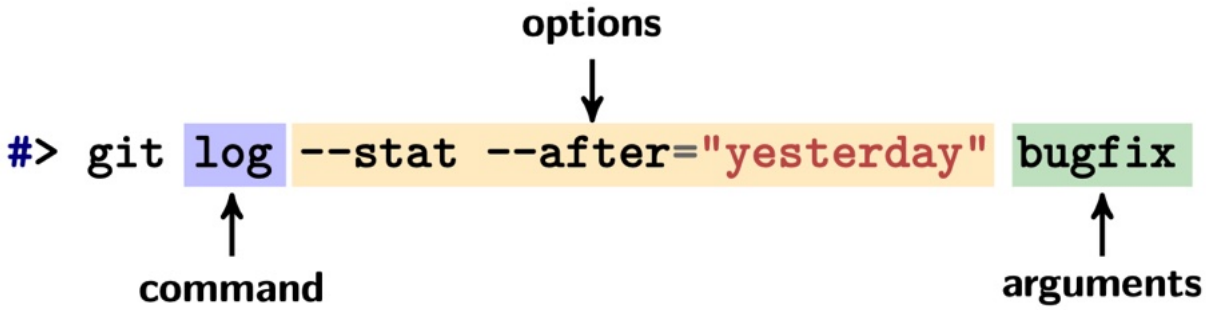


- Working Tree
  - Untracked Files: Files that are not tracked by git in the working copy.
  - Unmodified Files: Tracked files that are not modified.
  - Modified Files: Tracked files that are modified and ready to get staged.
- Repository
  - Staging Area (.git/index): Stores all the temporary modifications before committing to the repository.
  - Local Repository (.git/objects/): Hold the whole history of the project with all the modifications.
- Network
  - Remote Repository: A place used to synchronize your repository with other developpers.

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
  - git stash
- Get changes with
  - git fetch (fetches and merges)
  - git pull
- Propagate changes with
  - git push

.gitignore: Ignoring Untracked Files

- Why Ignore Files?
  - You do not want to see these files in ’git status’ to have a better understanding of the modifications you did on the code.
  - Building the project and editing files produce a lot of intermediate files that are not relevant to be cared upon. These files must be banned from the history because:
    - It blurs the ’status’ command a lot;
    - Versionning binary files is inefficient;
    - It breaks Makefile timestamps on files;
    - It breaks merge algorithm;
    - It eats disk space for nothing.
- How to Ignore Files?
  - Ignore files: ’~/.config/git/ignore’ (global) or ’project/.gitignore’ (local)
  - Basic Ignore Format:

```
# This is a comment
# Ignore all files ended by '~'
*~
# Ignore all file named 'core'
core
# Ignore the directory 'images/' and its content
images/
```

Configuration

- Set your identity to tag properly commits and get your own tools (e.g. editor).
- Two options: ’global’ or ’local’
- Global Configuration (’~/.gitconfig’ or ’~/.config/git/config’)

```
#> git config --global user.name "John Doe"
#> git config --global user.email john.doe@esir.fr
#> git config --global core.editor emacs
```

- Local Configuration (’project/.git/config’)

```
#> git config --local user.name "John Doe"
#> git config --local user.email john.doe@esir.fr
#> git config --local core.editor emacs
```

- Local > Global

Git Workflows

- Centralized Workflow

  Easy setup

- Feature Branch Workflow

  Focus on features

- Gitflow Workflow

  Focus on release cycle

- Forking Workflow

  Support for untrusted contributors

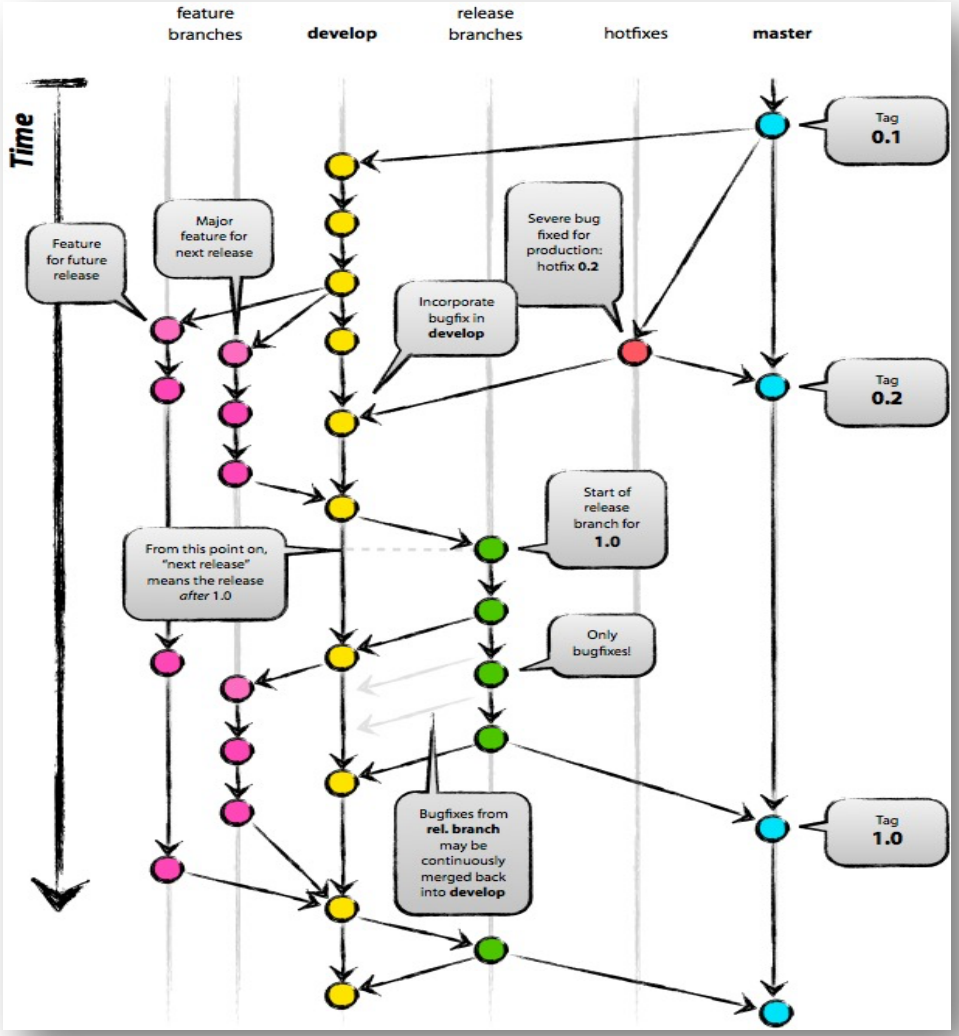
Recommanded for

Small projects

Large projects

https://www.atlassian.com/git/workflows

GitFlow



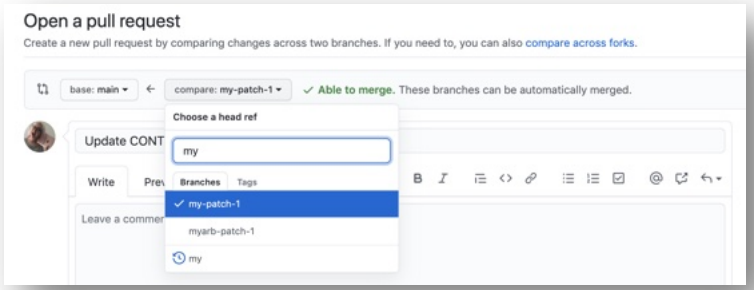
https://nvie.com/posts/a-successful-git-branching-model/

Tools to use Git

- Plugins in your IDE
- Standalone tools: SmartGit, TortoiseGit, CyberDuck, Etc.
- Integrated in collaborative platforms, e.g. forges:
  - GitHub (https://github.com)
    - Now Microsoft
    - Rich client: https://desktop.github.com
  - GitLab (https://gitlab.com)
    - Open-source platform

Github/gitlab/bitbucket

- Forges: Web-based collaborative platforms
  - VCS (git)
  - Social network
  - Git workflow
  - Continuous integration tools
  - Etc.
- Github/gitlab fork/Pull Request: make easier external contributions



Fork / Pull Request

Les commandes fork, merge (une pull request) sont possibles avec des commandes git standard mais requière d'associer et gérer plusieurs "remote" sur le dépôt

```
git remote add name url
git push –u destination ref
```

Idem pour le Squash à coup de rebase et cherry pick

Quelques différences entre les plateforme:

- Ex: bitbucket: Mise en avant dans l'IHM des modèles de type gitflow en encourageant des type de branches: release, hotfix, feature, bugfix

Best Practices

- Don’t panic!
- Know, practice and read about git!
- On doubt, back-up your repository before trying!
- Avoid conflicts at any price!
- Commit early and commit often!
- One commit for one thing, no more!
- Be meaningful on your commit messages!
- Branch as much as possible!
- Choose one workflow and stick to it!
- Keep up with remote project and others!

Common traps

- *git diff* without arguments shows the differences with the index

-> run *git diff HEAD* to show the differences with the last commit

- *git reset* reverts the index, but keeps the working copy unchanged !

Do *git reset –hard* if you need to revert the working copy too

- GIT is not forgiving, do not ignore its warnings and do not use --force

unless you have a clear idea of what you are doing

- GIT's history is not immutable

- *git checkout* on an arbitrary commit or a tag (anything that is not a branch) puts your in \detached HEAD" state.
  You can commit, but your history be lost if you don't create any branch (or tag) to reference them.

Documenting,
Testing,
Design Patterns, bad smells
Refactoring,
Debugging, monitoring, logging

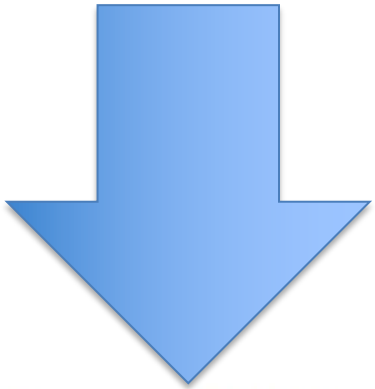
#1 What is the link?

- Documenting
  - Understanding (readability, maintainability)
- Refactoring
  - Improving the design (readability, maintainability, extensibility)
- The activity of documenting can somehow be replaced/automated 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
 }
```



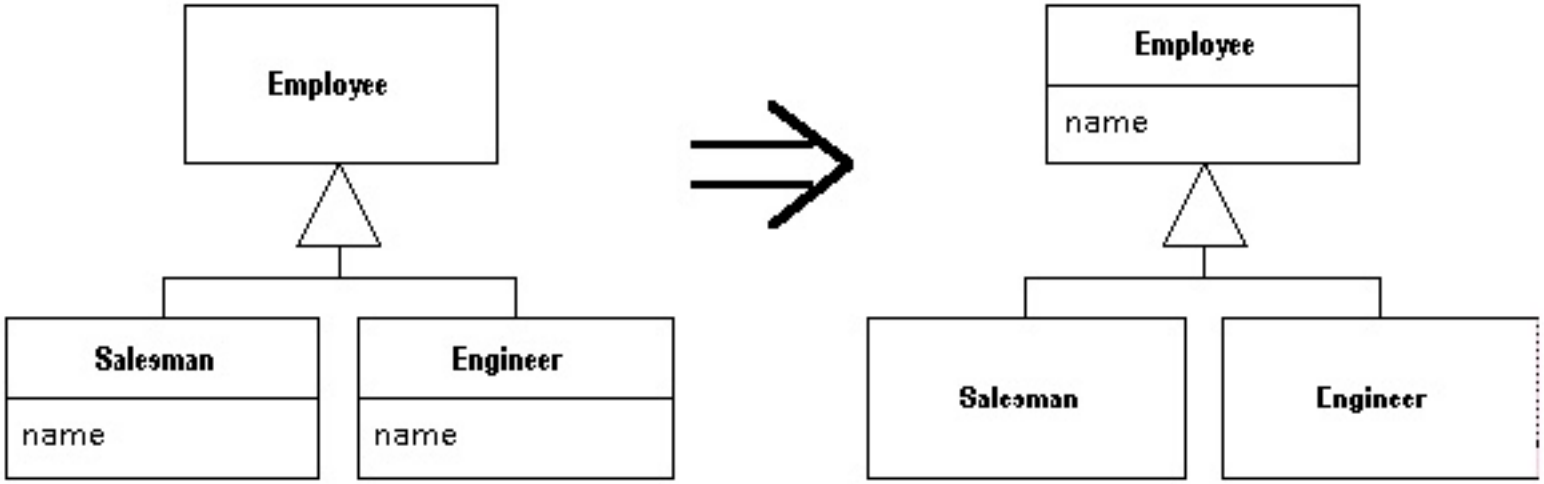
```
final boolean isMacOs     = platform.toUpperCase().indexOf("MAC") > -1;
final boolean isIEBrowser = browser.toUpperCase().indexOf("IE")  > -1;
final boolean wasResized  = resize > 0;

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

#2 What is the link?

Design patterns: there are refactorings

Two subclasses have the same field.

Move the field to the superclass.

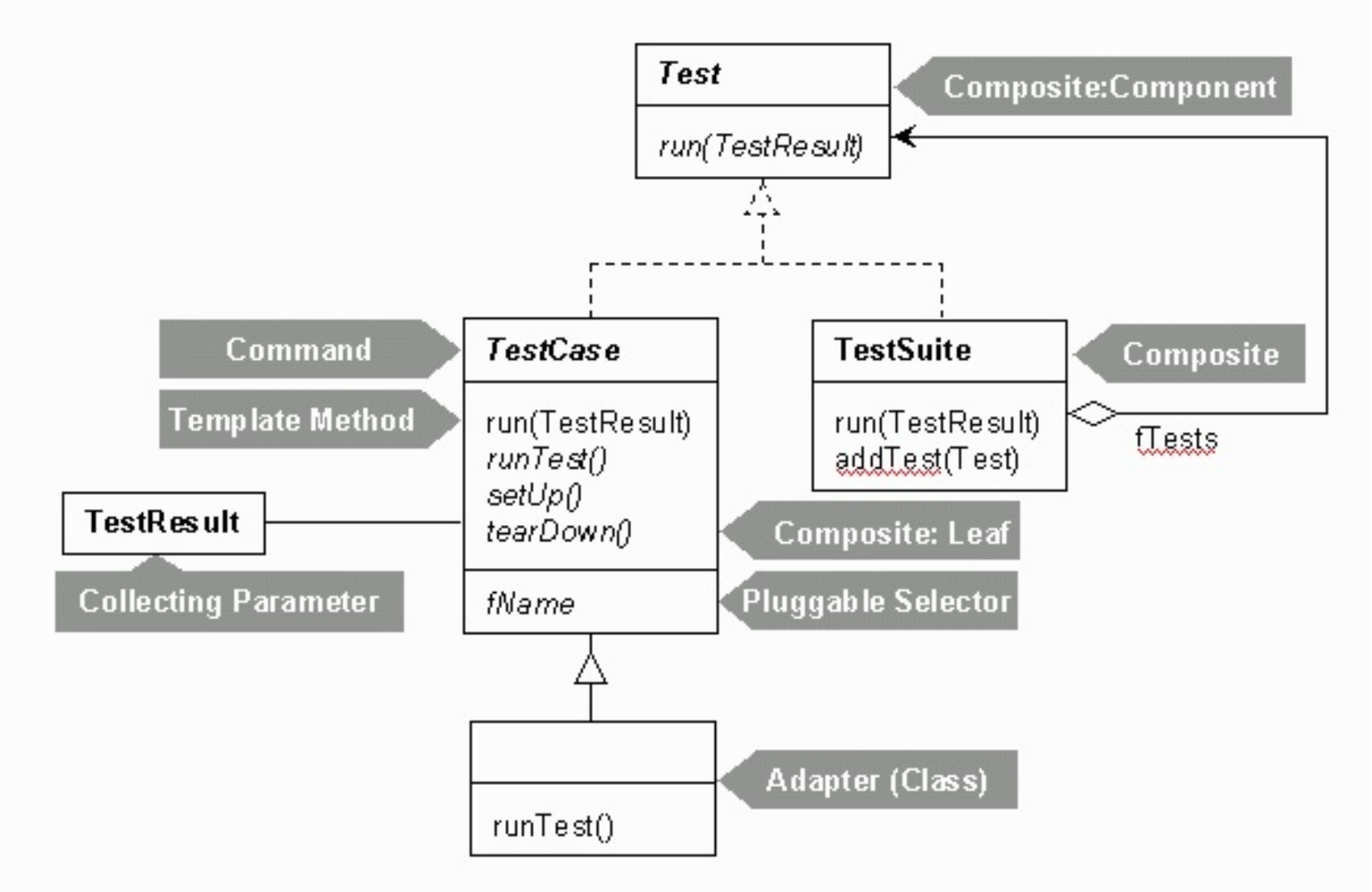


#3 What is the link?

- 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.

JUnit and… Design patterns



Worth reading!

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

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
- 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

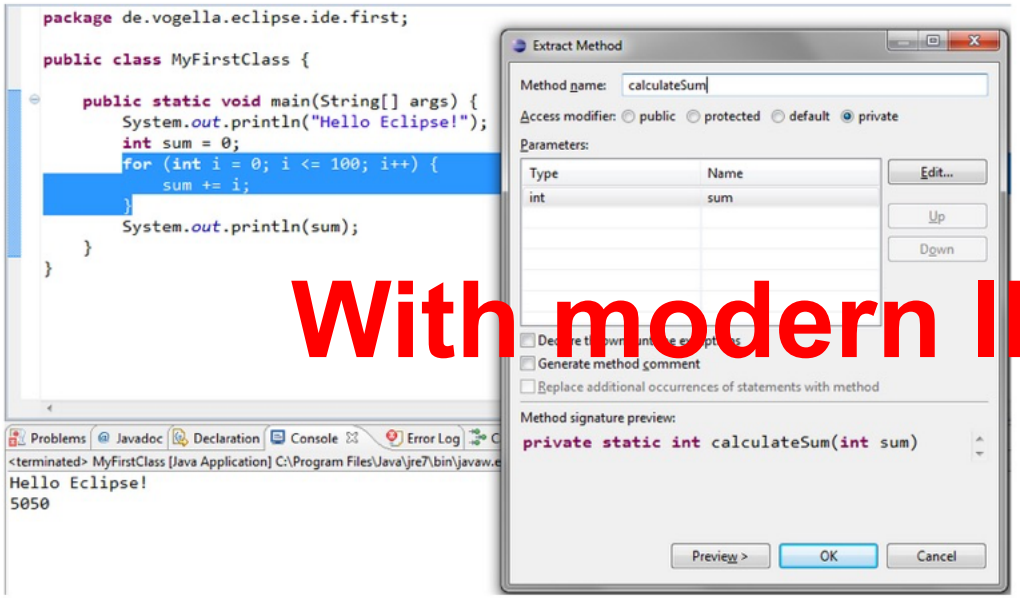
Document, refactor… Execute your tests… Debug.. Write test.. And so on!

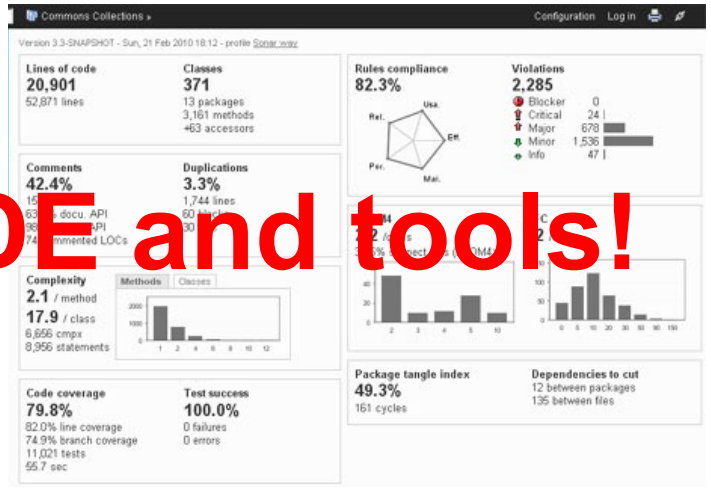
Documenting

Debugging

Refactoring

Testing





With modern IDE and tools!

INTEGRATION CONTINUE

INTRODUCTION

- Qu’est ce que l’intégration continue ?
- Pourquoi automatiser ?
- Par où commencer ?
- Le cycle vertueux de l’intégration continue

Définitions

"L'intégration continue est un ensemble de pratiques utilisées en génie logiciel. Elles consistent à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression de l'application en cours de développement."

Wikipedia

"Une pratique considérant différemment l'intégration, habituellement connue comme pénible et peu fréquente, pour en faire une tâche simple faisant partie intégrante de l'activité quotidienne d'un développeur."

Documentation
CruiseControl.NET

Qu’est ce que l’intégration continue ?

- Technique puissante permettant dans le cadre du développement d’un logiciel en équipes de:
  - Garder en phase les équipes de dév
  - Limiter risques de dérive
  - Limiter la complexité

- A intervalles réguliers, vous allez construire (build) et tester la dernière version de votre logiciel.

- Parrallèlement, chaque développeur teste et valide (commit) son travail en ajoutant son code dans un lieu de stockage unique.

Pourquoi automatiser ?

- Gagner du temps
  - Vous ne faites pas de taches répétitives
- Gagner en confiance
  - Indépendant de votre efficacité du moment
  - Procédures répétables
- Diminue le besoin de documentation
  - Pour nouveaux entrants projet, utiliser scripts !
  - … et + en analysant le script.

Par où commencer ?

- 1) Outil gestion versions code sources
  - Lieu unique de partage
  - Retour arrières, snapshots, branches…

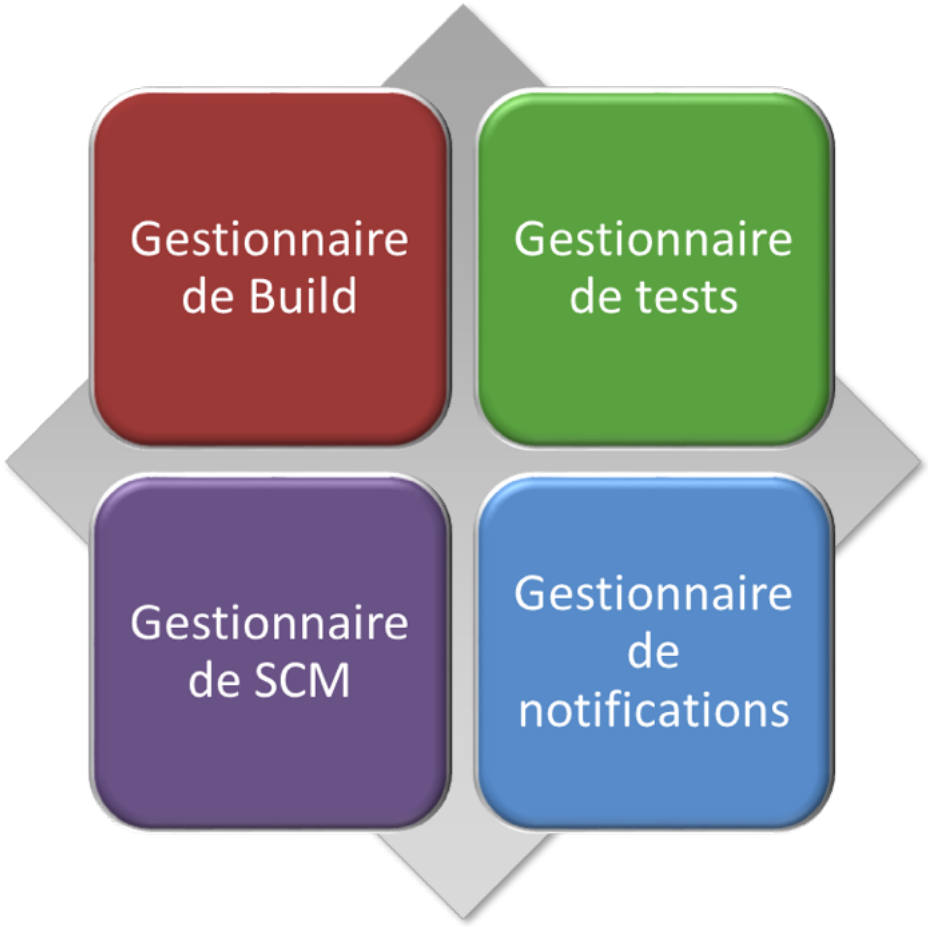
- 2) Tests automatisés
  - Chaque développeur

JUnit

- 3) Scripts
  - Coté serveur pour automatiser (Ex : crontab)

- 4)Outils de communication
  - Mail, Tél, Rss…

Architecture d’un logiciel d’intégration

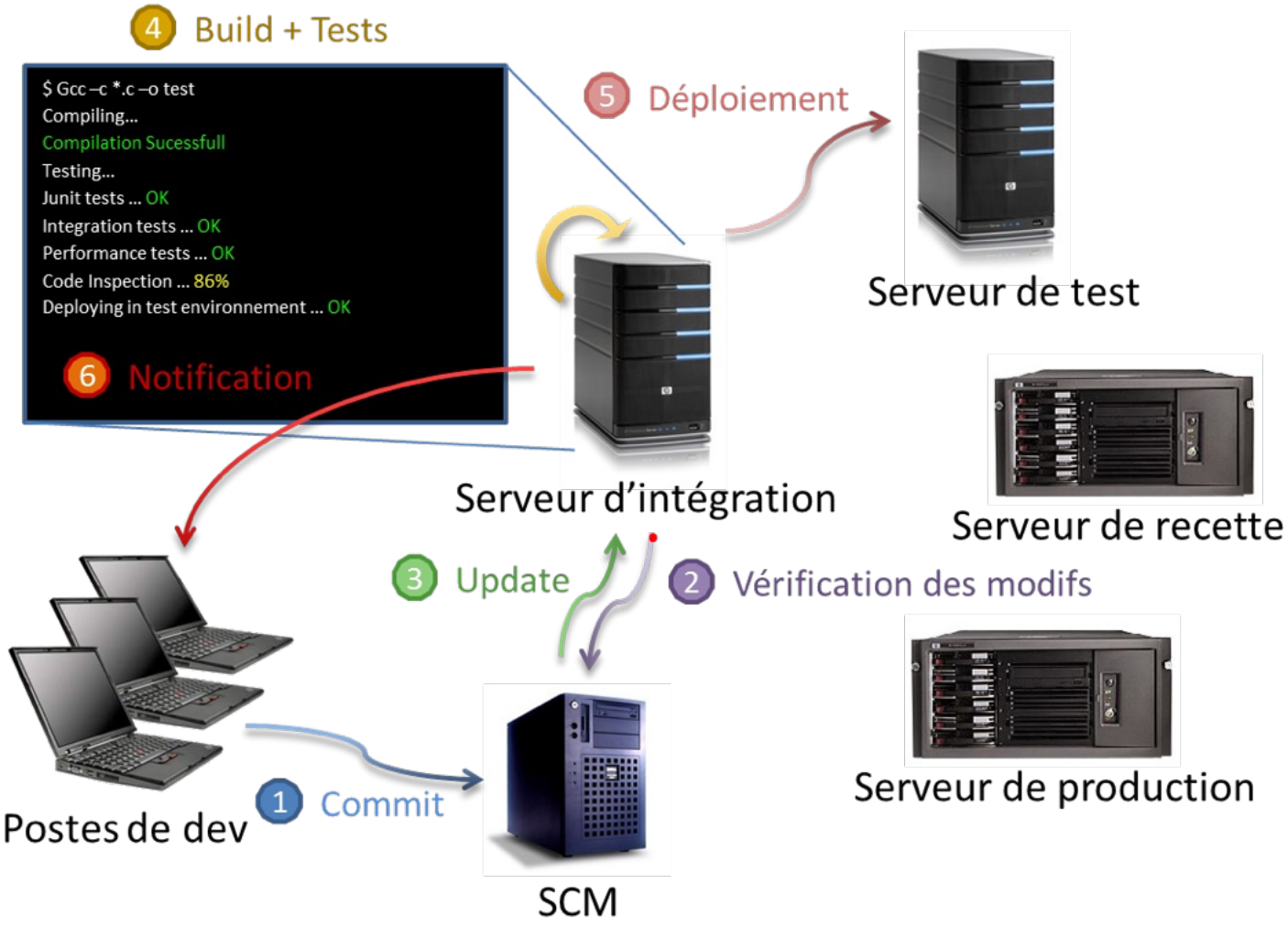


Un fonctionnement actif

- Les développeurs « committent »
- Le serveur d'intégration surveille le serveur SCM (Cron)

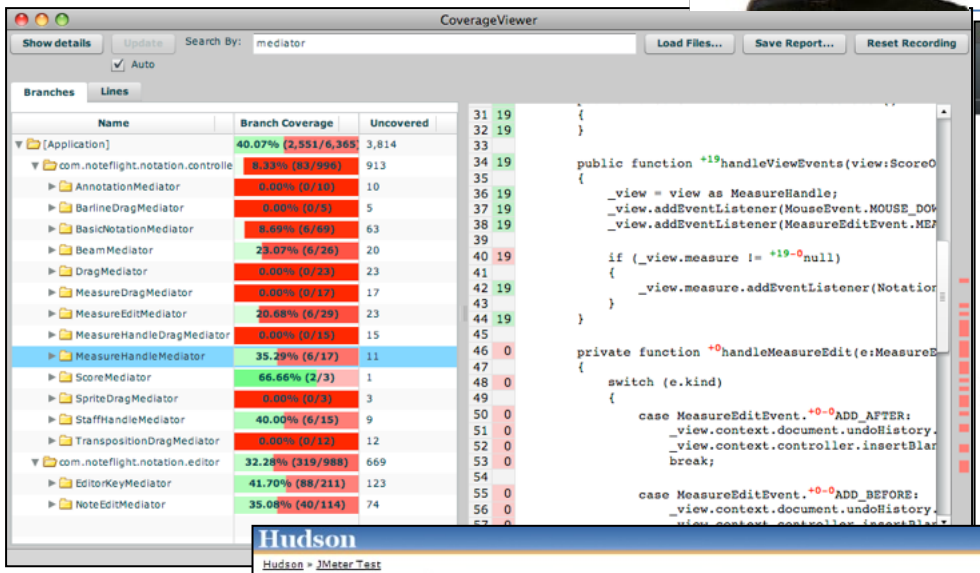
Cas d'utilisation

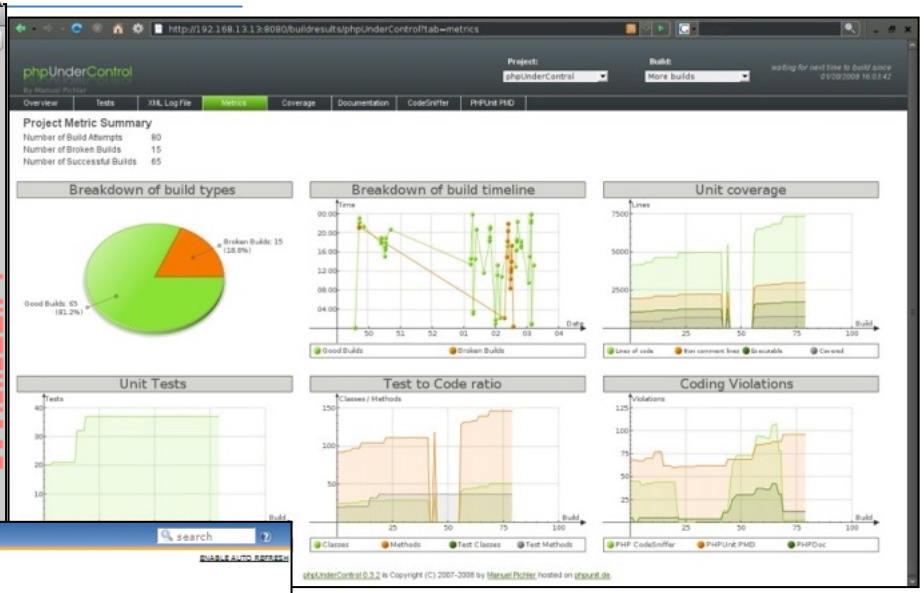
Le développeur soumet une modification

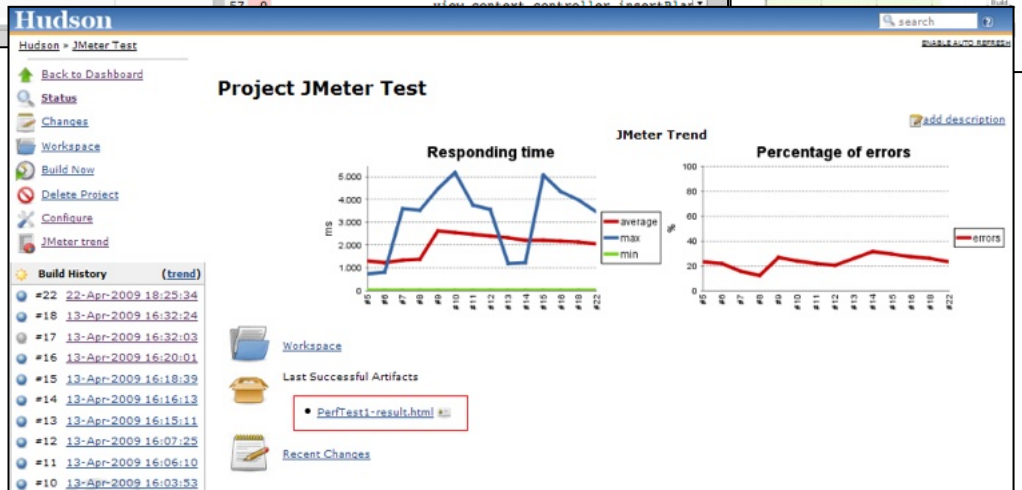


Cas d'utilisation

Le chef de projet analyse le reporting







Les technologies existantes

- Hudson, jenkins
- Github Actions
- Gitlab CI
- Travis CI
- CruiseControl / CruiseControl.NET
- Apache Continuum
- QuickBuild (open-source: LuntBuild)
- Et beaucoup d’autres …

Improving Your Productivity

- Continuous integration can help you go faster
  - Detect build breaks sooner
  - Report failing tests more clearly
  - Make progress more visible

Jenkins for Continuous Integration

- Jenkins – open source continuous integration server
- Jenkins (http://jenkins-ci.org/) is
  - Easy to install
  - Easy to use
  - Multi-technology
  - Multi-platform
  - Widely used
  - Extensible
  - Free

Jenkins for a Developer

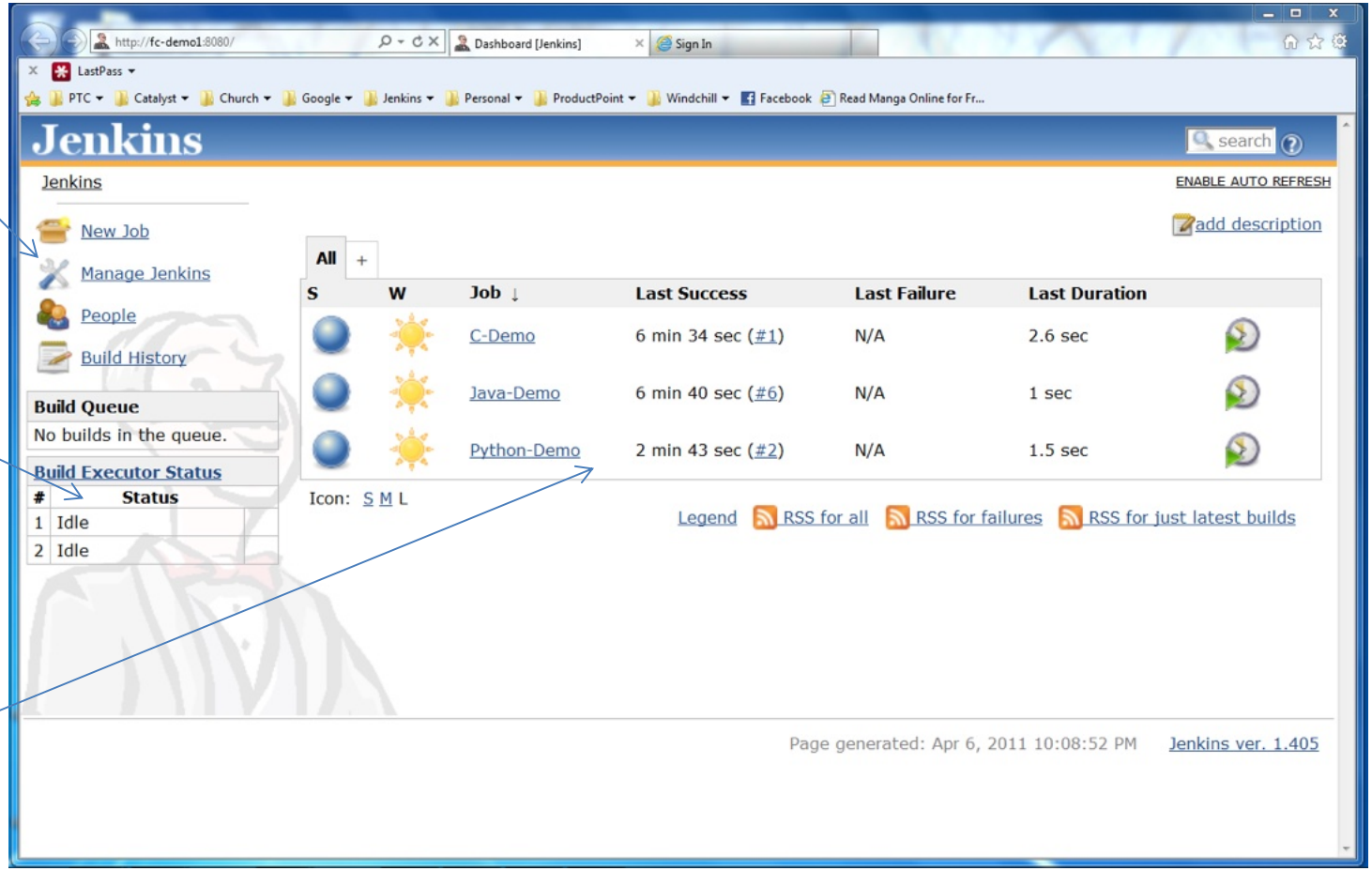
- Easy to install
  - Download one file – jenkins.war
  - Run one command – java –jar jenkins.war
- Easy to use
  - Create a new job – checkout and build a small project
  - Checkin a change – watch it build
  - Create a test – watch it build and run
  - Fix a test – checkin and watch it pass
- Multi-technology
  - Build C, Java, C#, Python, Perl, SQL, etc.
  - Test with Junit, Nunit, MSTest, etc.

Jenkins User Interface

Actions

Nodes

Jobs



Jenkins Plugins - SCM

- Version Control Systems
  - Accurev
  - Bazaar
  - BitKeeper
  - ClearCase
  - Darcs
  - Dimensions
  - Git
  - Harvest
  - MKS Integrity
  - PVCS
  - StarTeam
  - Subversion
  - Team Foundation Server
  - Visual SourceSafe






Jenkins Plugins – Build & Test

- Build Tools
  - Ant
  - Maven
  - MSBuild
  - Cmake
  - Gradle
  - Grails
  - Scons
  - Groovy
- Test Frameworks
  - Junit
  - Nunit
  - MSTest
  - Selenium
  - Fitnesse

Jenkins Plugins – Analyzers

- Static Analysis
  - Checkstyle
  - CodeScanner
  - DRY
  - Crap4j
  - Findbugs
  - PMD
  - Fortify
  - Sonar
  - FXCop
- Code Coverage
  - Emma
  - Cobertura
  - Clover
  - GCC/GCOV

Jenkins Plugins – Other Tools

- Notification
  - Twitter
  - Campfire
  - Google Calendar
  - IM
  - IRC
  - Lava Lamp
  - Sounds
  - Speak
- Authorization
  - Active Directory
  - LDAP
- Virtual Machines
  - Amazon EC2
  - VMWare
  - VirtualBox
  - Xen
  - Libvirt

What about GitLab CI?

What about GitHub Actions?

Our journey ;)

- Documentation
- (software) Logging
- Testing and Static Analysis
- Refactoring
- Build/configuration/release automation
- Continuous integration
- *Continuous delivery*
- *Continuous deployment*
- *(runtime) monitoring*
- *Continuous improvement*