Continuous Integration

Christophe Demarey
SED Lille Nord-Europe

March 2013, 20th

Outline

Introduction

1. Continuous Integration: Principles and practices

- identify key concepts of continuous integration
- reduce risks using continuous integration

Outline

2. Setting up a Continuous Integration system

- Build software at every change
- Test continuously
- Inspect the code continuously
- Deploy continuously
- Get a continuous feedback

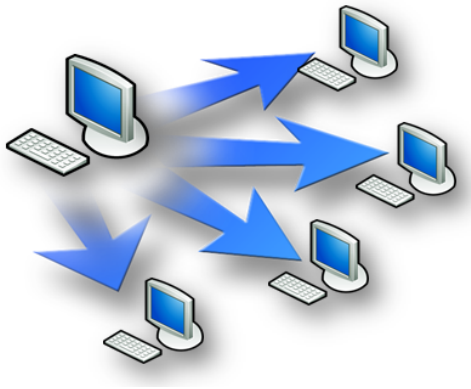
Conclusion

INTRODUCTION

« It's hard enough for software developers to write code that works on their machine. But even when that's done, there's a long journey from there to software that's producing value - since software only produces value when it's in production. »

Martin Fowler

Deployable software



Reduce time …

Increase the code quality

1

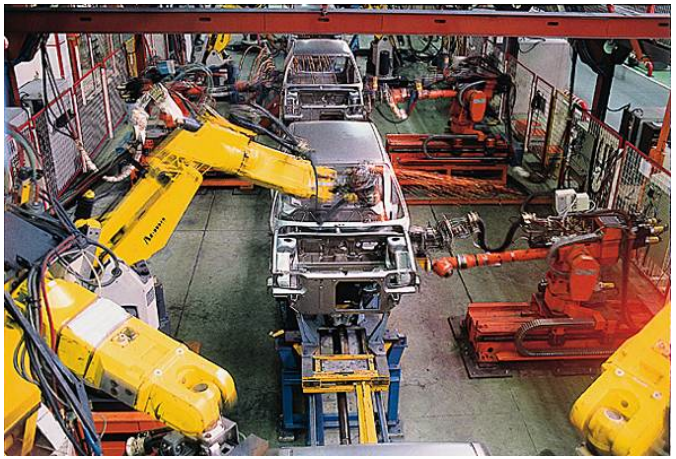
Continuous Integration

Principles and Practices

Increasing user requirements

Software development nowadays

Where we want to go



A

Continuous Integration

Key concepts

CI is a process

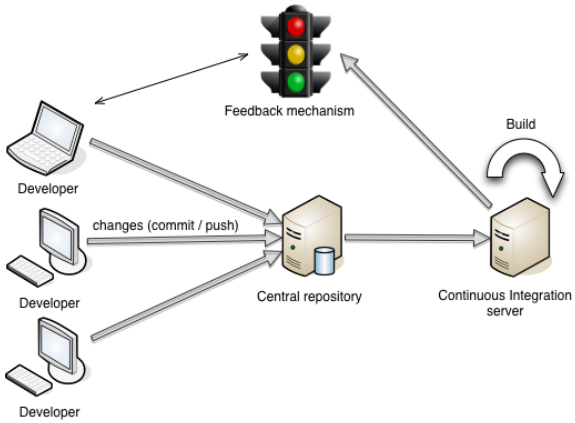
Continuous Integration

Software development strategy

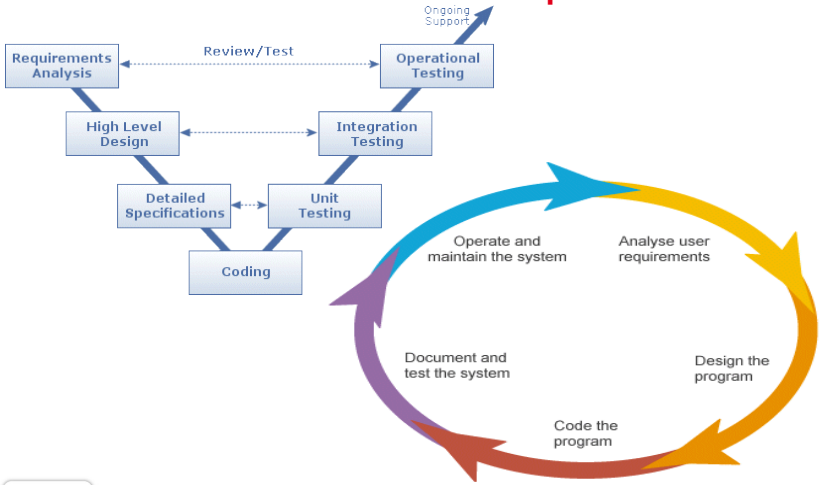
CI is the process of integrating work

and appling quality control frequently.

CI big picture



Incremental software development

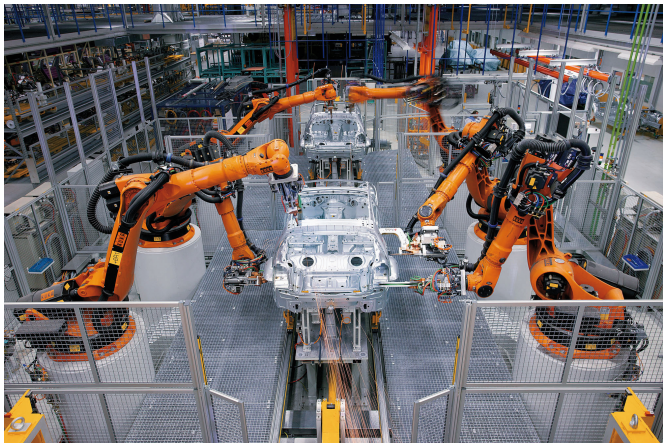


What is integration?

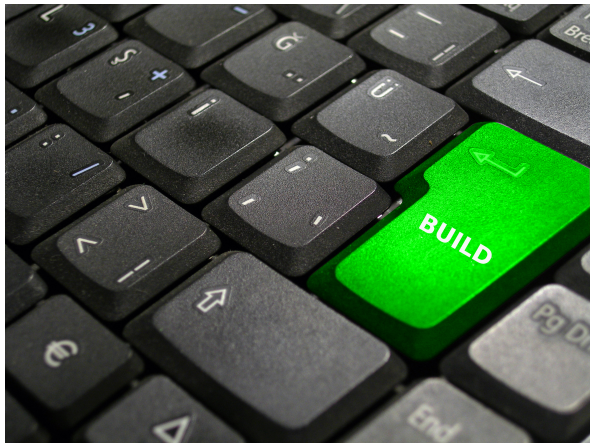
Why is it difficult?

What is a build?

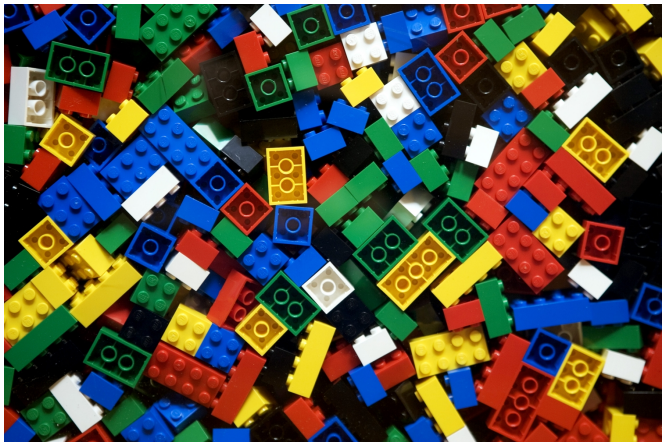
Automation is the key



Automation is the key



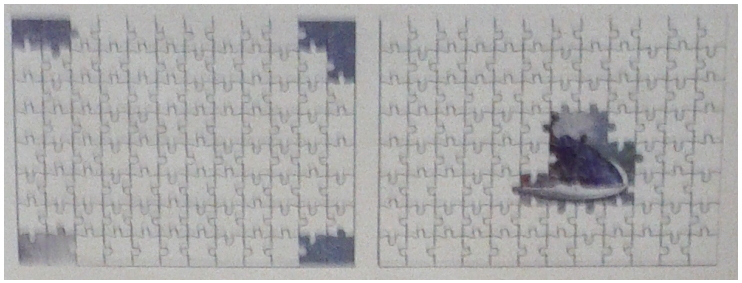
Small increments



Software decomposition

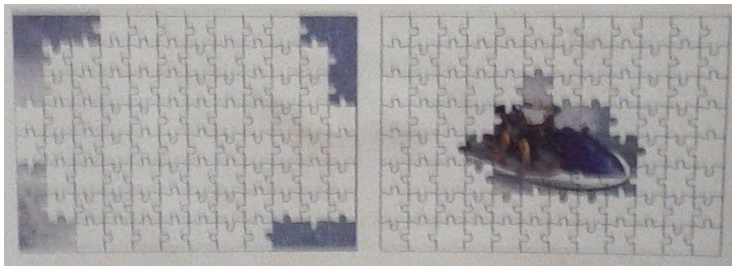
Software decomposition

Valuable elements first #1



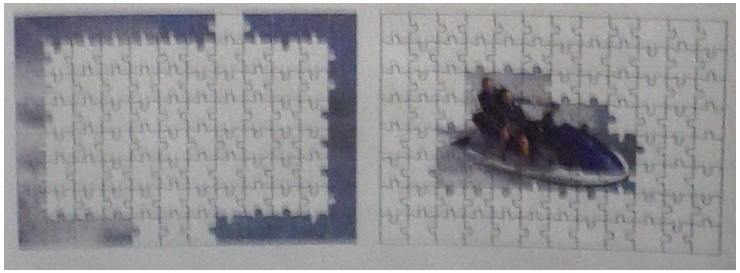
Incremental development: classic and value driven

Valuable elements first #2



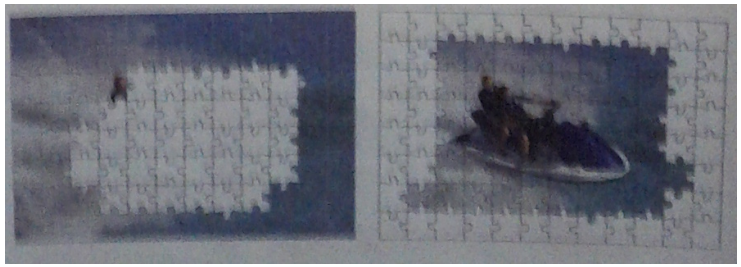
Incremental development: classic and value driven

Valuable elements first #3



Incremental development: classic and value driven

Valuable elements first #4



Incremental development: classic and value driven

« Continuously is more often than you think »

Commit

CI practices

1. Commit often to the central source code repository
2. Commit one change at a time (small increments)
3. Avoid getting broken code
4. Run private builds
5. Don’t commit broken code
6. Fix broken builds immediately
7. Write automated developer tests
8. All tests and inspections must pass

More time to focus on added value tasks

Exercise

Are you using a Version Control System (VCS, i.e. svn, git, etc.)?

Is your VCS shared by all developers?

Is your project's build process fully automated? Is it repeatable?

Are you writing tests?

Are you running automated tests?

Is tests execution part of your build process?

How do you check coding / design rules?

Do you have an automated feedback?

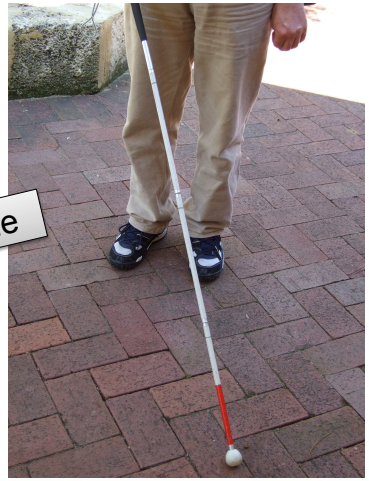
Are you using a dedicated integration machine to build software?

B

Continuous Integration

Reduce risks using continuous integration

Unusable software in the development stage



Exercise: Reducing risks

Lack of deployable software

Late discovery of defects

Lack of project visibility

Low quality software

Design / code smells

Exercise: Rapid feedback

2

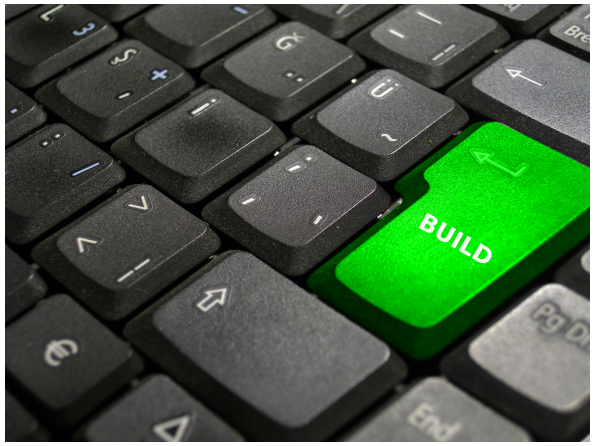
Implementing Continuous Integration

A

Implementing Continuous Integration

Build software at every change

Automate builds

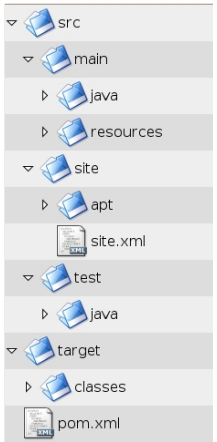


Single command builds

```
INFO]
INFO] Reactor Summary:
INFO]
INFO] jakubiak-xuggle-pom ................................ SUCCESS [3.277s]
INFO] jakubiak-xuggle-utils .............................. SUCCESS [3.135s]
INFO] jakubiak-xuggle-xuggler ............................ SUCCESS [7.156s]
INFO] jakubiak-xuggle-xuggler-red5 ....................... SUCCESS [2.729s]
INFO] jakubiak-xuggle-xuggler-red5-videotranscoder ....... SUCCESS [1.781s]
INFO] ------------------------------------------------------------------------
INFO] ------------------------------------------------------------------------
INFO] BUILD SUCCESSFUL
INFO] ------------------------------------------------------------------------
INFO] Total time: 18 seconds
INFO] Finished at: Wed Feb 11 19:10:26 CET 2009
INFO] Final Memory: 54M/400M
INFO] ------------------------------------------------------------------------
```


Centralize software assets

Project layout



Fail build fast

Build for any environment

Manage your environment

Store configuration with source code

- test data, database scripts,
- build script, deployment script

Third-party libraries

- use a dependency manager (ex: maven)
- or store them into the version control system

Dedicated integration build machine

Use a CI server

Jenkins

ATLASSIAN BAMBOO

TeamCity

cruisecontrol

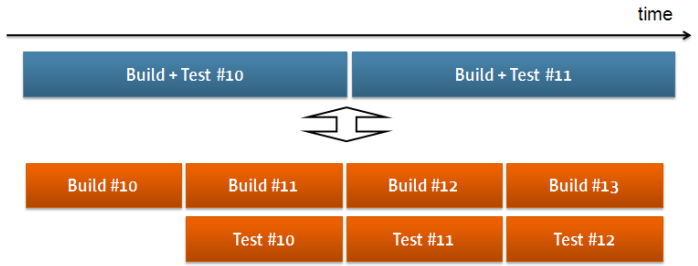
Using a CI server

- Long-running process which can execute a simple workflow at regular intervals
- View of the results of the processes, notifications, access to outputs
- Manage build distribution across a grid
- Integration with a lot of tools

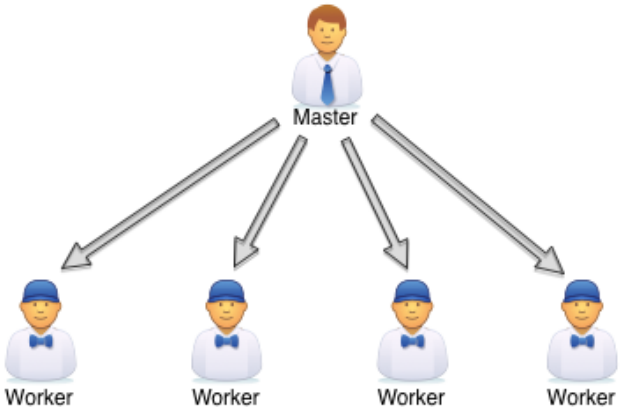
Unique opportunity to get all steps (installation, configuration) needed to have a running application.

Keep your build fast!

Staged builds

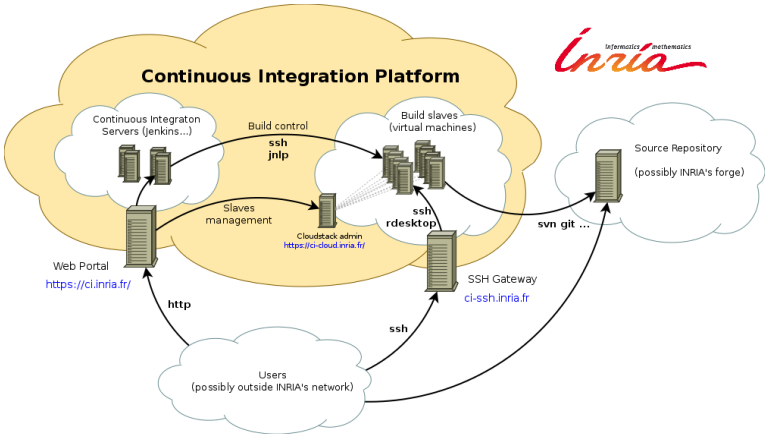


Distributed builds



Jenkins dashboard

Inria CI platform https://ci.inria.fr



B

Implementing Continuous Integration

Test continuously

Test continuously

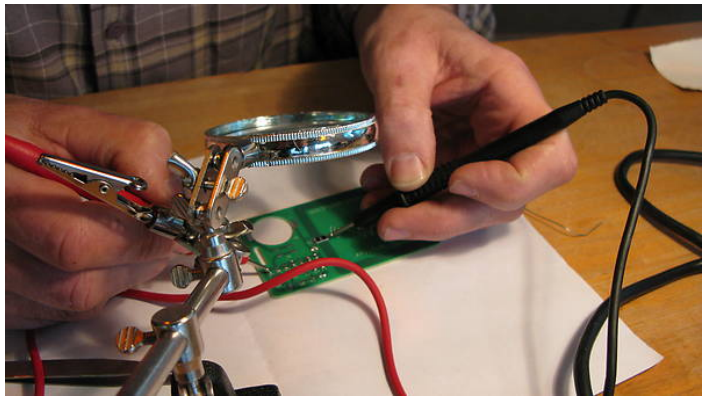
“Our acceptance tests validate that we built the right thing, while our unit and functional tests verify that we built the thing right.”

Confidence

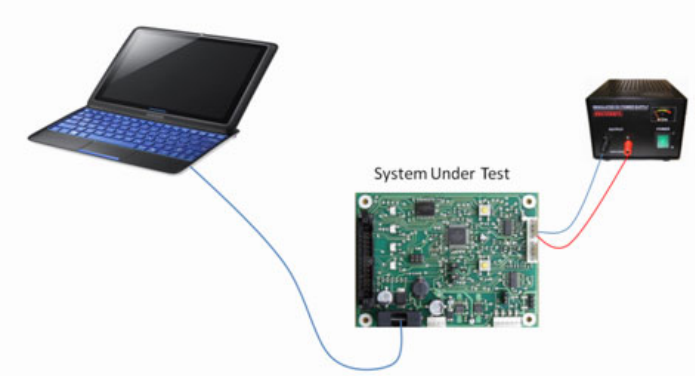
Unit testing

```java
public class TestAdder {
    public void testSum() {
        Adder adder = new AdderImpl();
        // can it add positive numbers?
        assert(adder.add(1, 1) == 2);
        assert(adder.add(1, 2) == 3);
        assert(adder.add(2, 2) == 4);
        // is zero neutral?
        assert(adder.add(0, 0) == 0);
        // can it add negative numbers?
        assert(adder.add(-1, -2) == -3);
        // can it add a positive and a negative?
        assert(adder.add(-1, 1) == 0);
        // how about larger numbers?
        assert(adder.add(1234, 988) == 2222);
    }
}
```


Component testing



System testing



Functional testing

Demo!

Test doubles

Dummy objects

Fake objects

Stubs

Mocks

Test doubles
> Fake objects

```
class MemberDAO {
  private Connection connection;

  public Member find(String id) {
    return connection.createQuery("..").findOne();
  }
}
```

```
class FakeMemberDAO {
 private Map<Long, Member> members;

 public Member find(String id) {
  return this.members.get(id);
 }
}
```

Test doubles
> Stubs

```
public interface MailService {
  public void send (Message msg);
}
```

```
public class MailServiceStub implements MailService {
 private List<Message> messages = new ArrayList<Message>();

 public void send(Message msg) {
   messages.add(msg);
 }

 public int numberSent() {
   return messages.size();
 }
}
```


Test doubles
> Stubs

```
public void testMemberMailSentWhenSubscribed() {
 // Création d'un membre
 Member member = new Member("login", "password");

 // On insère le stub à la place du mailer par défaut
 MyApp.setMailer(new MailServiceStub());

 // On enregistre le membre
 MyApp.register(member);

 // Un mail doit être envoyé
 assertEquals(1, mailer.numberSent());
}
```


Test doubles
> Mocks

```
public class OrderInteractionTester extends MockObjectTestCase {
  private static String TALISKER = "Talisker";

  public void testFillingRemovesInventoryIfInStock() {
    //setup - data
    Order order = new Order(TALISKER, 50);
    Mock warehouseMock = new Mock(Warehouse.class);

    //setup - expectations
    warehouseMock.expects(once()).method("hasInventory")
      .with(eq(TALISKER),eq(50))
      .will(returnValue(true));
    warehouseMock.expects(once()).method("remove")
      .with(eq(TALISKER), eq(50))
      .after("hasInventory");

    //exercise
    order.fill((Warehouse) warehouseMock.proxy());

    //verify
    warehouseMock.verify();
    assertTrue(order.isFilled());
  }
```


Test doubles
> Mocks

```
public void testFillingDoesNotRemoveIfNotEnoughInStock() {
  Order order = new Order(TALISKER, 51);
  Mock warehouse = mock(Warehouse.class);

  warehouse.expects(once()).method("hasInventory")
    .withAnyArguments()
    .will(returnValue(false));

  order.fill((Warehouse) warehouse.proxy());

  assertFalse(order.isFilled());
}
```

CI and tests

Automate tests
Categorize tests to be able to run slower tests at different intervals than faster tests.
Write tests for defects : regression tests to ensure that the defect will not surface again. Use a bug tracker.
Run faster tests first
Make component tests repeatable : ensure that the data is in a “known state” (ex;: use a database test framework)
Limit the number of asserts in a test case to spend less time tracking down the cause of a test failure.

C

Implementing Continuous Integration

Inspect the code continuously

Reduce code complexity

Complexité
52 055

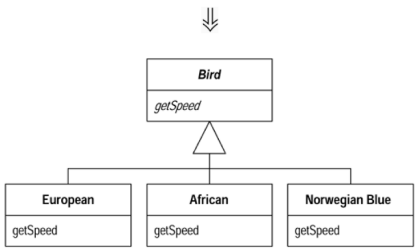
OpenEJB :: Container	31 251
OpenEJB :: Container :: Core	19 502
OpenEJB :: iTests	11 644
OpenEJB :: Container :: Java EE	10 636
OpenEJB :: Server	6 773
OpenEJB :: iTests Client	6 284

org.apache.openejb.config	4 859
org.apache.openejb.jee	3 776
org.apache.openejb.util	2 327
org.apache.openejb.client	2 271
org.apache.openejb.assembler.classic	1 374
org.apache.openejb.test.stateless	1 136

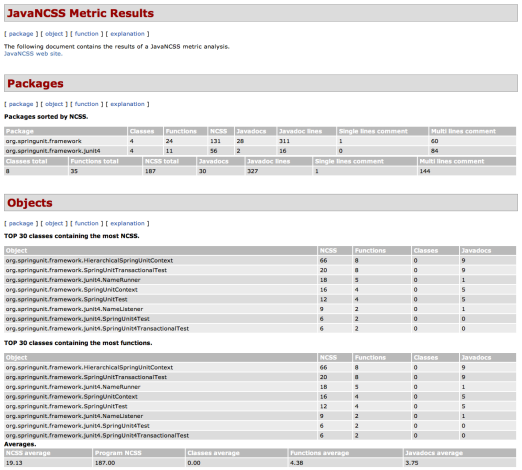
AnnotationDeployer	983
AutoConfig	413
MathUtils	333
DeploymentLoader	323
EJBCronTrigger	320
SuperProperties	308

Reduce code complexity
> cyclomatic complexity

```
double getSpeed() {
    switch (_type) {
        case EUROPEAN:
            return getBaseSpeed();
        case AFRICAN:
            return getBaseSpeed() - getLoadFactor() * _numberOfCoconuts;
        case NORWEGIAN_BLUE:
            return (_isNailed) ? 0 : getBaseSpeed(_voltage);
    }
    throw new RuntimeException ("Should be unreachable");
}
```



Perform design reviews



JavaNCSS Metric Results

[package] [object] [function] [explanation]

The following document contains the results of a JavaNCSS metric analysis.
JavaNCSS web site.

Packages

[package] [object] [function] [explanation]

Packages sorted by NCSS.

Package	Classes	Functions	NCSS	Javadocs	Javadoc lines	Single lines comment	Multi lines comment
org.springunit.framework	4	24	131	28	311	1	60
org.springunit.framework.junit4	4	11	56	2	16	0	84
Classes total	**Functions total**	**NCSS total**	**Javadocs**	**Javadoc lines**	**Single lines comment**	**Multi lines comment**	
8	35	187	30	327	1	144	

Objects

[package] [object] [function] [explanation]

TOP 30 classes containing the most NCSS.

Object	NCSS	Functions	Classes	Javadocs
org.springunit.framework.HierarchicalSpringUnitContext	66	8	0	9
org.springunit.framework.SpringUnitTransactionalTest	20	8	0	9
org.springunit.framework.junit4.NameRunner	18	5	0	1
org.springunit.framework.SpringUnitContext	16	4	0	5
org.springunit.framework.SpringUnitTest	12	4	0	5
org.springunit.framework.junit4.NameListener	9	2	0	1
org.springunit.framework.junit4.SpringUnit4Test	6	2	0	0
org.springunit.framework.junit4.SpringUnit4TransactionalTest	6	2	0	0

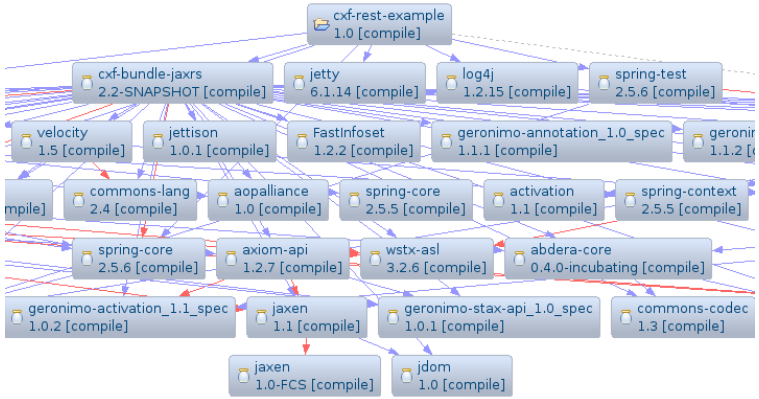
TOP 30 classes containing the most functions.

Object	NCSS	Functions	Classes	Javadocs
org.springunit.framework.HierarchicalSpringUnitContext	66	8	0	9
org.springunit.framework.SpringUnitTransactionalTest	20	8	0	9
org.springunit.framework.junit4.NameRunner	18	5	0	1
org.springunit.framework.SpringUnitContext	16	4	0	5
org.springunit.framework.SpringUnitTest	12	4	0	5
org.springunit.framework.junit4.NameListener	9	2	0	1
org.springunit.framework.junit4.SpringUnit4Test	6	2	0	0
org.springunit.framework.junit4.SpringUnit4TransactionalTest	6	2	0	0

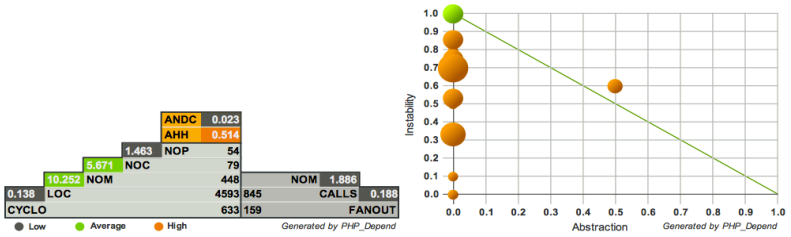
Averages.

Class average	Program NCSS	Classes average	Functions average	Javadocs average
19.13	187.00	0.00	4.38	3.75

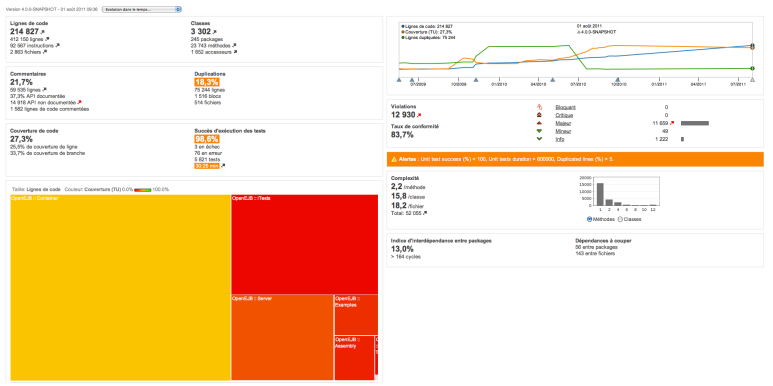
Perform design reviews



Maintain organizational standards with code audit



Maintain organizational standards with code audit



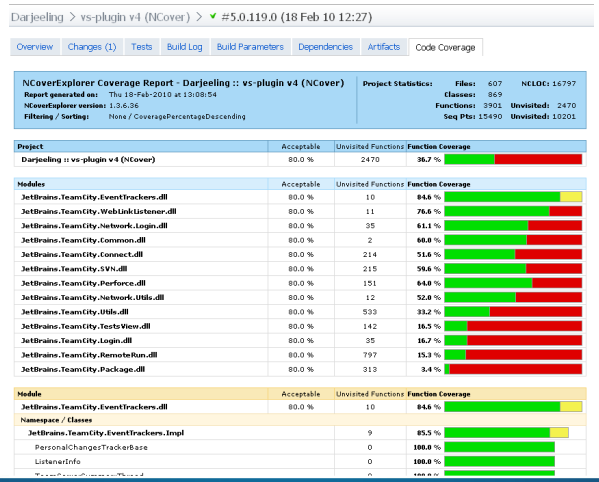
Reduce duplicate code
> Extract method pattern

```
void printOwing(double amount) {
    printBanner();

    //print details
    System.out.println ("name:" + _name);
    System.out.println ("amount" + amount);
}
```

⇓

```
void printOwing(double amount) {
    printBanner();
    printDetails(amount);
}

void printDetails (double amount) {
    System.out.println ("name:" + _name);
    System.out.println ("amount" + amount);
}
```


Assess Code coverage



D

Implementing Continuous Integration

Continuous feedback

Communication !

“As a general rule, the most successful man in life is the man who has the best information.”

Benjamin Disraeli (1804-1881)

Mechanisms to enable continuous feedback

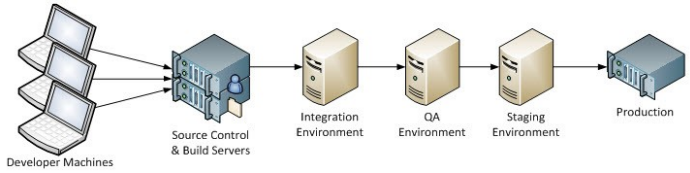
E

Implementing Continuous Integration

Deploy continuously

Continuous deployment

Continuous deployment



Continuous deployment

1. Tag releases put inti production
2. Produce a clean environment,
   free of assumptions
3. Generate and label builds
4. Successfully run tests at all levels in a clone
   of the production environment
5. Create build feedback reports
6. Be able to rollback quickly if needed

F

Implementing Continuous Integration

Common practices

Always run tests locally before committing

Don’t check in on a broken build

Time-box fixing before reverting

Never go home on a broken build

Don’t comment out failing tests

Test Driven Development

Some suggested practices

eXtreme Programming development practices (refactoring)

Failing a build

- for slow tests,
- for warnings,
- for code style breaches

Some suggested practices

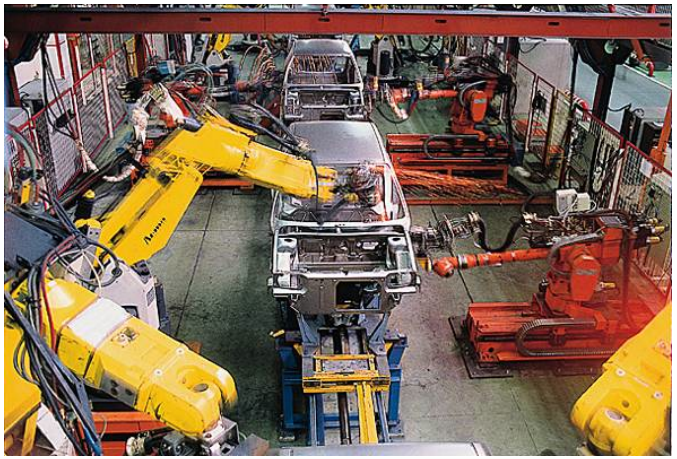
Get the latest version here !

3

Conclusion

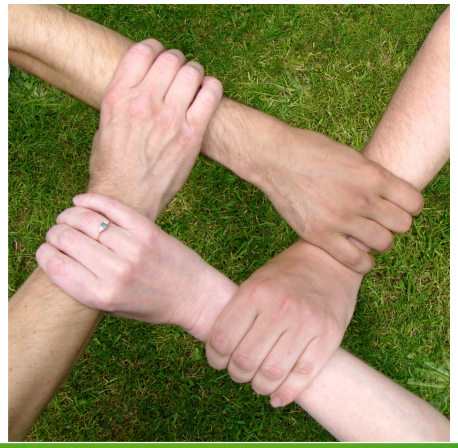
What you should keep in mind

Automation is the key



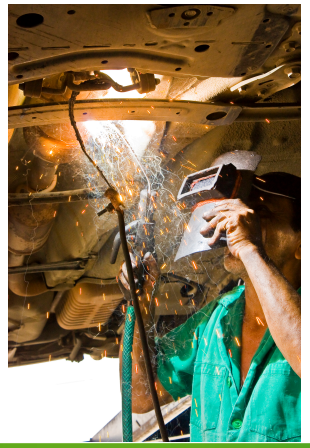
You need tests !

Team adherence, discipline

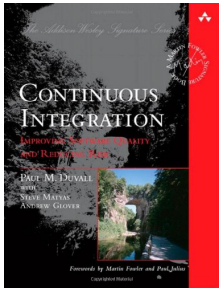


Agreement of the team

> Highest priority : Fix changes that breaks the application



References



Continuous Integration article
by Martin Fowler
http://martinfowler.com/articles/continuousIntegration.html

Continuous Integration softwares (non exhaustive list) :

- **Jenkins** - http://jenkins-ci.org/
- Bamboo - http://www.atlassian.com/software/bamboo/overview
- Travis - http://about.travis-ci.org/
- Hydra - http://nixos.org/hydra/

Wikipedia
http://sourcemaking.com

Thank you!

INVENTEURS DU MONDE NUMÉRIQUE