



Introduction au test logiciel

Fabrice Ambert, Fabrice Bouquet

prenom.nom@femto-st.fr

Préambule aux exercices



Exemple fil rouge

Le Robot

- L'ensemble des exercices fera référence à un exemple fil rouge : Le Robot
- Une implémentation java servira de support aux exercices sur les tests structurels, les tests unitaires et une partie des tests fonctionnels
- Une implémentation Web sera utilisée pour illustrer les outils de test d'interface Web ainsi que pour le suivi des exigences
- Une description textuelle du Robot et de ses fonctionnalités est jointe

2. Mise en œuvre de la Couverture structurelle

1. Sur papier, production de graphes de contrôle et de données de tests pour assurer un niveau de couverture
2. Sur machine, écriture et exécution de tests unitaires

Graphe de flot de contrôle

Production de graphe de contrôle

Production de données de test pour couvrir le graphe de contrôle



Exercice 1

Méthode nextForwardPosition

- Produire le graphe de contrôle de la fonction ci-dessous
- Produire des cas de test pour couvrir tous les nœuds du graphe

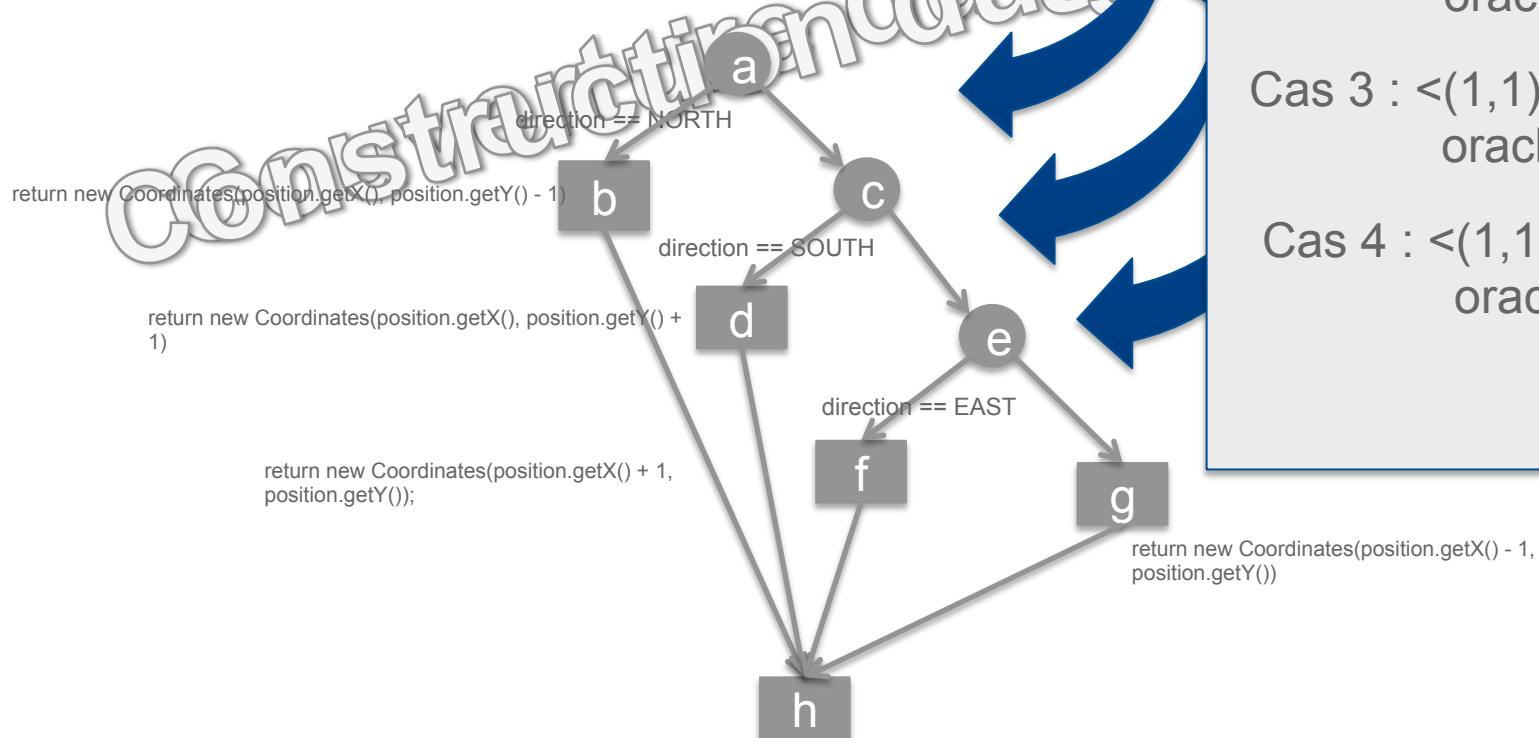
```
public static Coordinates nextForwardPosition(Coordinates position, Direction direction) {  
    if (direction == NORTH)  
        return new Coordinates(position.getX(), position.getY() - 1);  
    if (direction == SOUTH)  
        return new Coordinates(position.getX(), position.getY() + 1);  
    if (direction == EAST)  
        return new Coordinates(position.getX() + 1, position.getY());  
    return new Coordinates(position.getX() - 1, position.getY());  
}
```



```
public class Coordinates {  
  
    private int x;  
    private int y;  
  
    public Coordinates(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    ...
```

Correction

```
public static Coordinates nextForwardPosition(Coordinates position, Direction direction) {  
    if (direction == NORTH)  
        return new Coordinates(position.getX(), position.getY() - 1);  
    if (direction == SOUTH)  
        return new Coordinates(position.getX(), position.getY() + 1);  
    if (direction == EAST)  
        return new Coordinates(position.getX() + 1, position.getY());  
    return new Coordinates(position.getX() - 1, position.getY());  
}
```



Cas 1 : <(1,1), NORTH>
oracle : (1,0)

Cas 2 : <(1,1), SOUTH>
oracle : (1,2)

Cas 3 : <(1,1), EAST>
oracle : (2,1)

Cas 4 : <(1,1), WEST>
oracle : (0,1)



Exercice 2

Méthode letsGo

- Produire le graphe de contrôle de la fonction ci-dessous
- Produire des cas de test pour couvrir tous les arcs du graphe

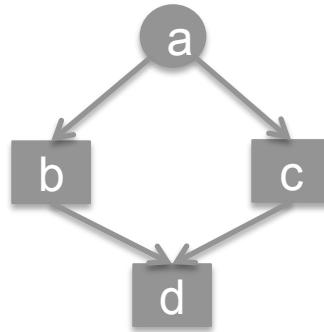
```
public List<CheckPoint> letsGo() throws UnlandedRobotException, UndefinedRoadbookException,  
    InsufficientChargeException, LandSensorDefaillance, InaccessibleCoordinate {  
  
    if (roadBook == null) throw new UndefinedRoadbookException();  
    List<CheckPoint> mouchard = new ArrayList<CheckPoint>();  
    while (roadBook.hasNextInstruction()) {  
        Instruction nextInstruction = roadBook.next();  
        if (nextInstruction == FORWARD) moveForward();  
        else if (nextInstruction == BACKWARD) moveBackward();  
        else if (nextInstruction == TURNLEFT) turnLeft();  
        else if (nextInstruction == TURNRIGHT) turnRight();  
        CheckPoint checkPoint = new CheckPoint(position, direction, false);  
        mouchard.add(checkPoint);  
        blackBox.addCheckPoint(checkPoint);  
    }  
    return mouchard;  
}
```



```
class CheckPoint {  
  
    public final Coordinates position;  
    public final Direction direction;  
    public final boolean manualDirective;  
  
    public CheckPoint(Coordinates position, Direction direction, boolean manualDirective) {  
        this.position = position;  
        this.direction = direction;  
        this.manualDirective = manualDirective;  
    }  
  
}  
  
public enum Instruction {  
    TURNLEFT,  
    BACKWARD,  
    TURNRIGHT,  
    FORWARD  
}
```

Choisir la couverture

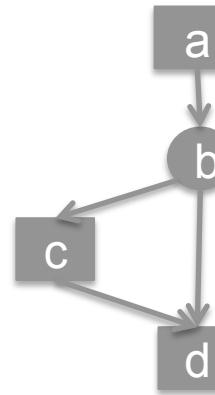
```
if (x>y)
    max=x;
else
    max=y;
return max
```



$x=3, y=2$
Oracle 3

tous nœuds

```
read(x, y);
if (x<0)
    y=-y;
return y/x;
```

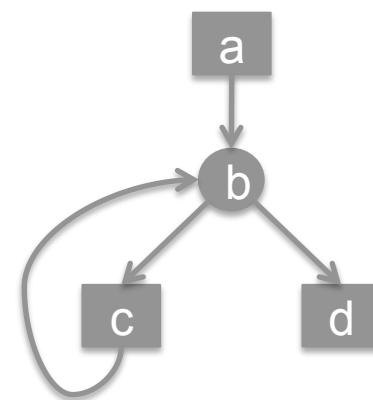


$x=-2, y=4$
Oracle 2

tous arcs



```
read(x, y);
i=0;
while(i<x) {
    y=2*y;
    i++;
}
return y/x;
```



$x=-2, y=4$
Oracle 2

chemins



Exercice 3

Méthode lireCoordonnee

- Produire le graphe de contrôle de la fonction ci-dessous
- Produire des cas de test pour couvrir tous chemins indépendants

```
static Coordinates lireCoordonnee(Scanner scanner) {
    boolean conforme;
    int x = 0;
    int y = 0;
    do {
        conforme = true;
        String line = scanner.nextLine();
        String[] tokens = line.replace("(", " ").replace(")", " ").split(",");
        if (tokens.length != 2) {
            conforme = false;
            System.out.println("Format incorrect. c, l ou (c, l)");
        }
    } else
        try {
            x = Integer.valueOf(tokens[0].trim());
            y = Integer.valueOf(tokens[1].trim());
        } catch (NumberFormatException e) {
            conforme = false;
        }
    } while (!conforme);
    return new Coordinates(x, y);
}
```



Exercice 4

Méthode compacte

- Produire le graphe de contrôle de la fonction ci-dessous
- Produire des cas de test pour couvrir tous les arcs et FPC

```
static List<Instruction> compacte(List<Instruction> instructions) {  
    List<Instruction> copieCompacte = new ArrayList<Instruction>();  
    List<Instruction> instructionsEnAttente = new ArrayList<Instruction>();  
    for (int i = 0; i < instructions.size(); i++) {  
        if (instructions.get(i) == TURNRIGHT && instructionsEnAttente.size() == 2) {  
            instructionsEnAttente.clear();  
            copieCompacte.add(TURNLEFT);  
        } else if (instructions.get(i) == TURNRIGHT)  
            instructionsEnAttente.add(TURNRIGHT);  
        else {  
            copieCompacte.addAll(instructionsEnAttente);  
            instructionsEnAttente.clear();  
            copieCompacte.add(instructions.get(i));  
        }  
    }  
    copieCompacte.addAll(instructionsEnAttente);  
    return copieCompacte;  
}
```

Tests unitaires – travaux pratiques

- Présentation des éléments de syntaxe de Junit
- Sur RobotSimple, exécuter et compléter des tests fournis.
Mesurer la couverture avec les outils intégrés
- Les tests en isolation : présentation et mise en œuvre avec Mockito sur RobotComplet
- Utilisation d'un outils d'analyse statique de la qualité du code



Outils mis en œuvre



Tests avec JUnit

Organisation

- Les tests sont rassemblés dans des classes de tests
- Une classe de test = une suite de test
- Une classe de test est associée à une seule classe de source
- Classe de test et classe sous test partagent le même package
 - mais pas le même répertoire
- Avec maven
 - /src/main/java
 - /src/test/java



Anatomie d'un test unitaire



```
@Test      Annotation désignant la méthode comme un test
public void testXXX() {
    //Define
        Instructions de mise en contexte
    //When
        Instruction sous test
    //Then
        Observation et vérification de l'oracle
}
```

Exprimer un résultat attendu

La classe org.junit.Assert

```
@Test  
public void testXXX() {  
    //Define  
    //When  
    //Then  
    Assert.assertEquals(Oracle, SUT.methodeSousTest(...));  
}
```



Tester l'apparition d'une exception

Forme simple

```
@Test (expected = ClasseException.class)
public void testXXX() {
    //Define

    //When

}
```



Tester l'apparition d'une exception

Forme avancée

```
@Rule  
Public ExpectedException thrown = ExpectedException.none();  
@Test  
public void testXXX() {  
    // ici l'exception fait échouer le test  
    thrown.expect(ExceptionAttendue.class);  
    // ici l'exception est attendue  
}
```

Retirer momentanément un test d'une suite



@Ignore

```
@Ignore  
@Test(expected = ClasseException.class)  
public void testXXX() {  
    //Define  
  
    //When  
  
}
```

Lors de l'exécution de la suite, le test est ignoré mais est mentionné dans le rapport d'exécution

Indépendance des tests



Le résultat d'un test ne doit pas dépendre de l'exécution des tests précédents

- Pas d'attributs dans la classe de test
- Création et initialisation des instances utiles au test dans chacune des méthodes de test
- OK pour des classes simples ne nécessitant pas de mise en contexte compliquée
- Attributs dans la classe de test
- Utilisation des annotations `@Before`, `@BeforeClass`, `@After`, `@AfterClass`
- Allège l'écriture des tests lors de mises en contexte similaires

Indépendance des tests

Une exécution par test



```
@Before  
public void setUp() {  
}
```

Cette méthode est exécuté avant chaque test de la suite

```
@After  
public void tearDown() {  
}
```

Celle ci après chaque test de la suite

Indépendance des tests



Une seule exécution pour la suite (classe) de test

```
@BeforeClass  
public void suiteSetUp() {  
}
```

Cette méthode est activée avant l'exécution du setUp du 1^{er} test de la suite

```
@AfterClass  
public void suiteTearDown() {  
}
```

Celle ci après l'exécution du tearDown du dernier test de la suite

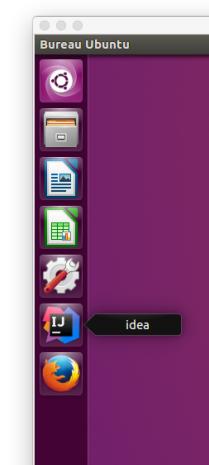
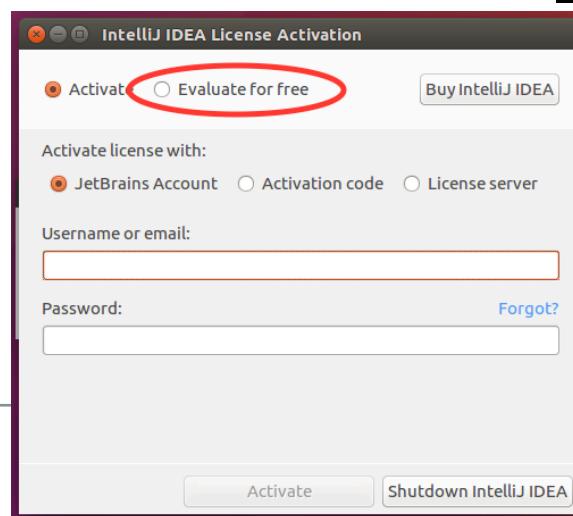
Séance de mise en œuvre

Lancement de la machine virtuelle

- Lancer VirtualBox et choisir la machine Formation TestAutom
- Sélectionnez l'utilisateur **Part Icipant** (mot de passe : partici)

IDE installé : IntelliJ Idea

- Double cliquez sur l'icône
- License : evaluation



Séance de mise en œuvre



Ouverture du projet

Le projet RobotSimple sur lequel vous allez travailler est ouvert au lancement de l'IDE.



Description du projet

- les sources modélisent un Robot capable de se déplacer. Le robot dispose de ses coordonnées et de sa direction (points cardinaux). Il peut accomplir 2 actions modifiant ses coordonnées (déplacement avant et déplacement arrière) et 2 actions pour modifier sa direction (tourner à droite et tourner à gauche). Dans l'état actuel de l'implémentation, certaines classes n'ont pas encore d'utilité.
- la classe de test RobotUnitTest contient les premiers tests associés à la classe Robot. Ces 2 classes sont dans le même package *robot* mais dans des répertoires différents
- la classe BatteryUnitTest testera la classe Battery

À réaliser



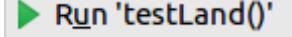
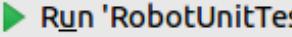
Exécuter et compléter

- ouvrir dans l'éditeur de l'IDE la classe RobotUnitTest
 - Des erreurs apparaissent (Test en rouge)
 - Placer le curseur d'insertion dans le mot `@Test`
 - Appuyer simultanément Alt + Entrée (propose contextuellement des corrections)
 - Choisir « Add 'JUnit4' to classpath » puis dans la fenêtre qui s'ouvre « Use 'JUnit4' from IntelliJ IDEA distribution »
 - Les erreurs de compilation disparaissent
- **comprendre et exécuter les tests un à un, compléter les tests lorsque cela est nécessaire**

À réaliser



Exécuter les tests

- Pour exécuter un test
 - placer le curseur de souris sur la méthode de test
 - clic droit et sélection  **Run 'testLand()'** **Ctrl+Maj+F10**
 - le test est alors exécuté et le verdict affiché
- Pour exécuter une suite de test (classe)
 - placer le curseur de souris sur le nom de la classe dans la fenêtre d'édition ou sur le nom de la classe dans la vue projet
 - clic droit et sélection  **Run 'RobotUnitTest'** **Ctrl+Maj+F10**
 - tous les tests de la classe sont exécutés
- Pour exécuter tous les tests
 - placer le curseur de souris sur répertoire java de test dans la vue projet
 - clic droit et sélection  **Run 'All Tests'** **Ctrl+Maj+F10**

À réaliser



Utiliser les outils de couverture intégrés

- de manière similaire à l'exécution des tests, il est possible de demander l'exécution des tests avec mesure de la couverture. Elle peut être activée pour un test, pour une classe ou pour tous les tests.  [Run 'All Tests' with Coverage](#)
- IDEA propose plusieurs outils de couverture et pour celui de l'outil le mode Tracing en plus du mode Sampling
 - le mode sampling permet de calculer la couverture des instructions
 - en mode tracing, la couverture des branche est assurée
- **Complétez les tests pour atteindre 100% de couverture sur Robot et sur Battery**

Bilan



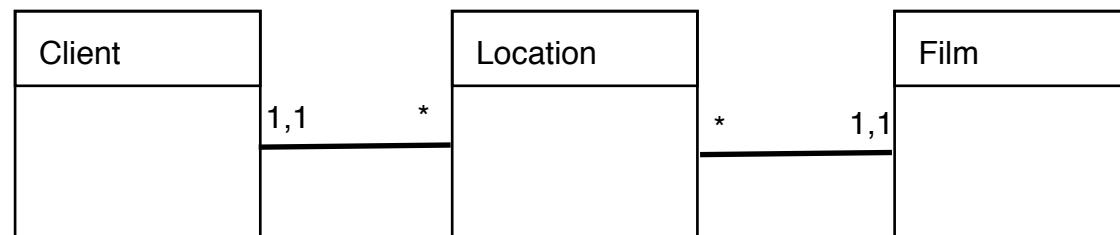
Pendant cette étape, nous avons abordé :

- le test unitaire de méthodes simples en utilisant la classe org.junit.Assert
- le test d'apparition d'exceptions
- le test de méthodes privée
- l'utilisation des outils intégrés de couverture



Tester en isolation

Le résultat d'un test ne doit dépendre que des méthodes sous test



Tester en isolation



```
public class Location {  
private Film film ;  
private Client client ;  
...  
public float montant(int duree) {  
    if (client.getCat() == PRIVILEGE)  
        return  
    film.prixJour()*(duree-1) ;  
    else ...  
}
```

```
public class Film {  
private Categorie categorie ;  
private String titre ;  
...  
public float prixJour() {  
    switch (categorie) {  
        case Categorie.NOUVEAUTE :  
            return categorie.prixBase()  
        * ... ; ...  
    }  
}
```



Tester en isolation

```
@Test  
public void testMontant() {  
    Film film = new Film();  
    ...  
    Client client = new Client(PRIVILEGE);  
  
    Location loc = new Location(film, client);  
  
    Assert.assertEquals(3.5, loc.montant(2));  
}
```

Une erreur dans la méthode prixJour de la classe Film provoquera l'échec de ce test



Tester en isolation

Utiliser des mock et stub

- Remplacer un objet réel par un simulacre
- Remplacer l'exécution d'une méthode par une réponse pré-déterminée
- Contrôler le flux d'exécution de la méthode sous test
- Toutes les classes autres que la classe sous test peuvent être « mockées »
- Jmock, Mockito....



Le package Mockito

<http://mockito.org>

```
@Test
```

```
public void testMontant() {
```

```
    Film film = Mockito.mock(Film.class) ;
```

```
    Mockito.when(film.prixJour()).thenReturn(3.5) ;
```

```
    Client client = Mockito.mock(Client.class) ;
```

```
    Mockito.when(client.getCat()).thenReturn(PRIVILEGE) ;
```

```
    Location loc = new Location(film, client) ;
```

```
    Assert.assertEquals(3.5, loc.montant(2)) ;
```

```
}
```



Mockito - spy

```
@Test
```

```
public void testMontant() {
```

```
    Film film = Mockito.mock(Film.class) ;
```

```
    Mockito.when(film.prixJour()).thenReturn(3.5) ;
```

```
    Client client = Mockito.mock(Client.class) ;
```

```
    Mockito.when(client.getCat()).thenReturn(PRIVILEGE) ;
```

```
    Location loc = new Location(film, client) ;
```

```
    loc.montant(2)) ;
```

```
    Mockito.verify(film).prixJour();
```

```
}
```

Le mock mémorise les appels qui lui sont fait. On peut ensuite l'interroger sur les invocations auxquelles il a répondu.



Mise en œuvre

Récupération des sources sur le repository svn

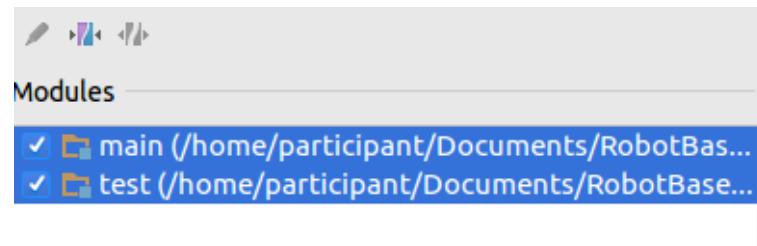
La mise en œuvre des tests en isolation sera réalisée sur une deuxième version du robot. Les sources sont dans le repository svn local à la machine virtuelle.

- En ligne de commande :
`svn checkout svn://localhost/webRobot/RobotComplet`
- Depuis l'IDE :
 - Lancez IntelliJ Idea
 - Sélectionnez Check out from Version Control – Subversion
 - Ajoutez un repository +
 - URL : `svn://localhost/webRobot/RobotComplet`
 - Choisissez le répertoire de destination :
 - `participant/Documents`
 - `/home/participant/Documents/webRobot/RobotComplet`

Séance de mise en œuvre

Ouverture du projet sous IntelliJ

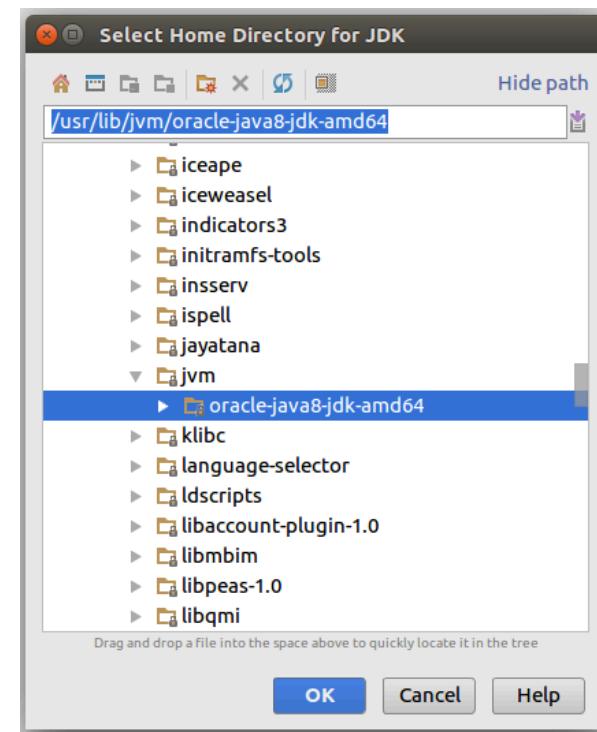
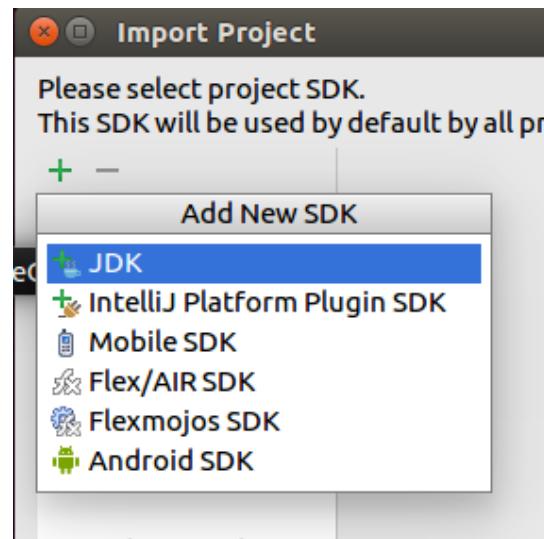
- Après le checkout, IntelliJ propose de créer un projet idea à partir des sources
- il propose « create project forme existing sources » Next
- un nom de projet et une localisation sont proposés (RobotComplet et ~/Documents/RobotComplet) Next
- Idea détecte des fichiers sources dans 2 répertoires Next
- Idea identifie qu'aucune librairie n'est associée au projet Next
- Idea détecte 2 modules (un pour les sources et un pour les test). Il faut à cette étape les fusionner.
 - sélectionner les 2 modules
 - cliquer sur 
 - nommer le module : robot



Séance de mise en œuvre

Ouverture du projet sous IntelliJ

- La dernière étape concerne la configuration du SDK
- Cliquer sur + et choisir JDK



A réaliser



L'objet de l'étape 2 est d'introduire la prise en compte de la consommation d'énergie qui intègre la consommation de base modulé par les aspérités du terrain. Pour ce faire, il faut reprendre le code de Robot et intégrer dans les méthodes en charge des déplacements la prise en compte de la consommation d'énergie. Le source du Robot a été modifié en conséquence, il est dans **RobotComplet**.

Par contre les tests n'ont pas suivi, vous devez les mettre à niveau.

Les consignes pour cette étape sont :

- les tests doivent être réalisés en isolation
- quelques erreurs se sont glissées dans le code, trouvez les !



Qualité du code

L'IDE peut améliorer le code

- L'ajout de plugins ou l'utilisation de règles de l'inspecteur intégré peuvent participer à l'amélioration de la qualité du code
- Plugin SonarLint : analyse du code dans l'IDE et affichage des règles de codage



Bilan

Cette étape a été l'occasion d'aborder

- le test en isolation
- l'utilisation de bouchons (mock, stubs) avec Mockito
- l'utilisation de SonarLint

Utilisation d'un outil d'intégration continue

1. Utilisation d'un constructeur de build : Maven
2. Utilisation d'un outils d'intégration continue : Jenkins
3. Utilisation d'un outils d'analyse du code : SonarQube



Jenkins



- Interface dans un navigateur
- Permet de définir des **Jobs**
- Donne une vue synthétique de l'état des jobs sur sa page d'accueil
- Plusieurs jobs peuvent s'enchaîner pour un même projet

Pour l'installation de Jenkins (non abordée dans cette formation)
<http://jenkins-le-guide-complet.github.io/continuous-integration-with-hudson.pdf>

Tutorial de création de job

Intégrer le projet RobotComplet à Jenkins

- Ce document présente les étapes de création et configuration d'un job Jenkins
- Les étapes d'installation et de configuration de l'outil ont déjà été réalisée.
- Jenkins est accessible à l'adresse :
<http://localhost:9090/jenkins>



Étape préalable

Mavenisation du projet

- Pour être intégré au processus de build continu, le projet doit disposer de scripts de construction qui seront activés par Jenkins
- En Java, une solution est d'utiliser Maven
- Maven utilise des fichiers pom.xml pour spécifier les éléments utiles à la création du projet
- Maven impose une architecture des fichiers (mais peut aussi être utilisé en dehors de cette architecture)
 - src/main/java
 - src/main/resources
 - src/test/java



fichier pom.xml

Version minimale

```
<?xml version="1.0" encoding="UTF-8"?>
<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>fr.test.formation</groupId>
    <artifactId>RobotComplet</artifactId>
    <version>1.0-SNAPSHOT</version>
</project>
```



fichier pom.xml

ajout d'une dépendance sur JUnit

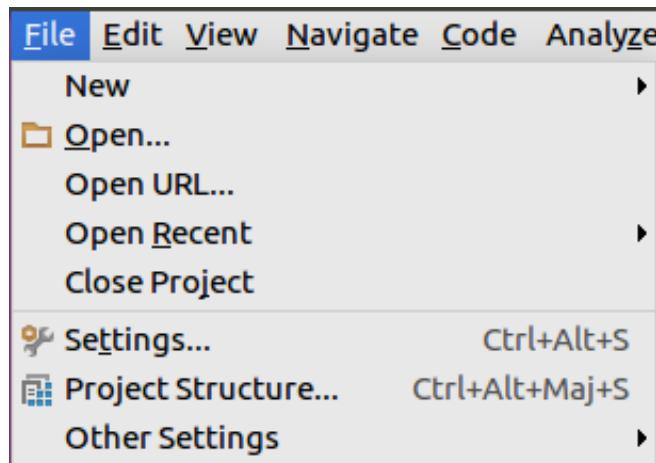
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
...
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

Mettre le projet sous Maven

Adaptation de la structure du projet

- À partir du projet IDEA, ouvrir « Project Structure »

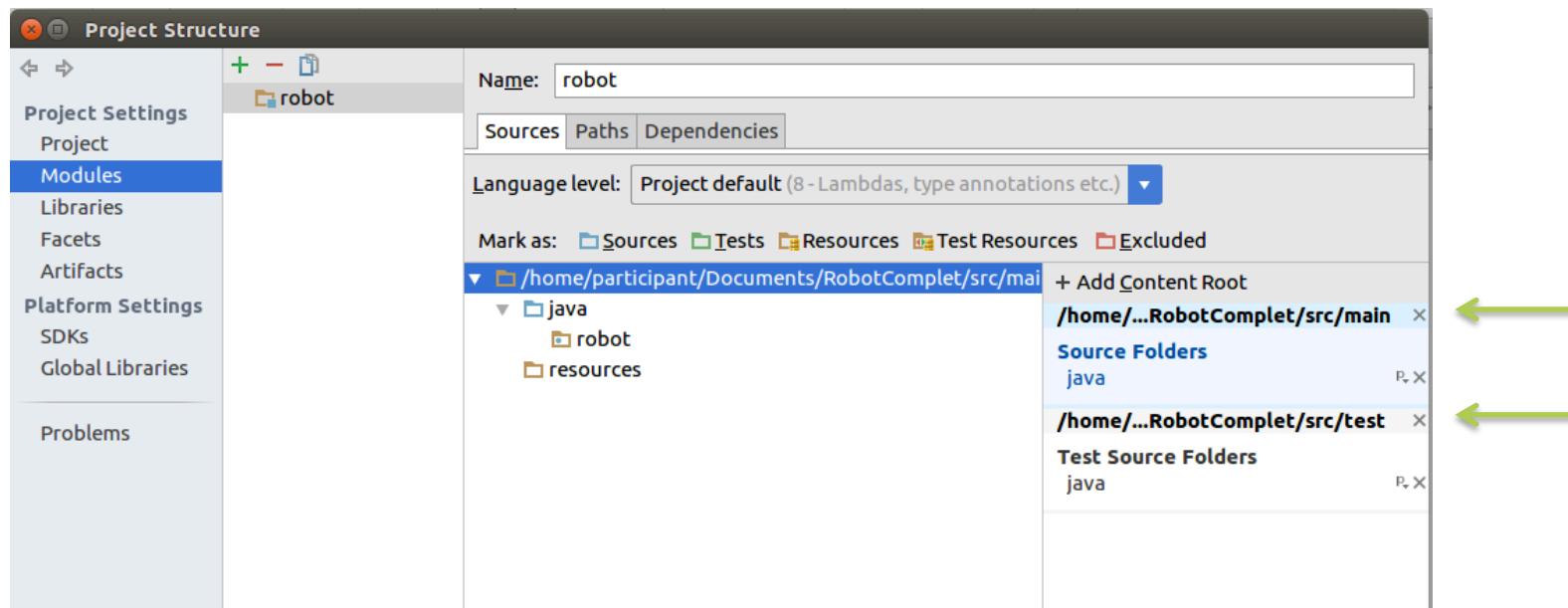


- Puis choisir « Modules »

Mettre le projet sous Maven

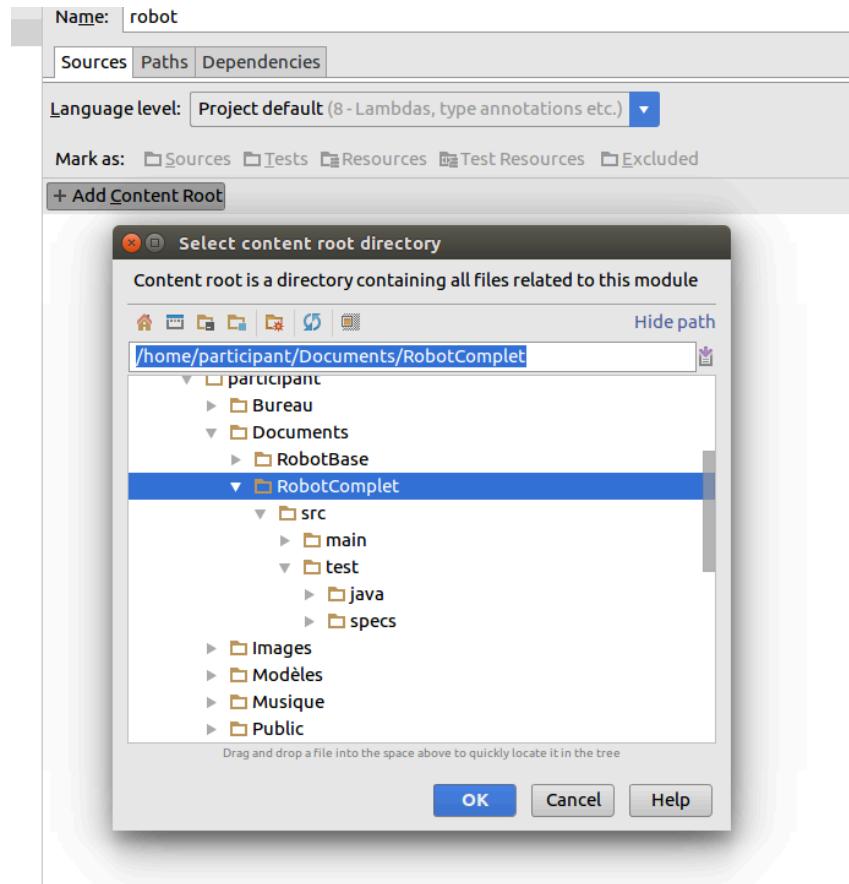
Adaptation de la structure du projet

- Le module comporte 2 « Content Root »
- Les supprimer puis en recréer un seul sur RobotComplet



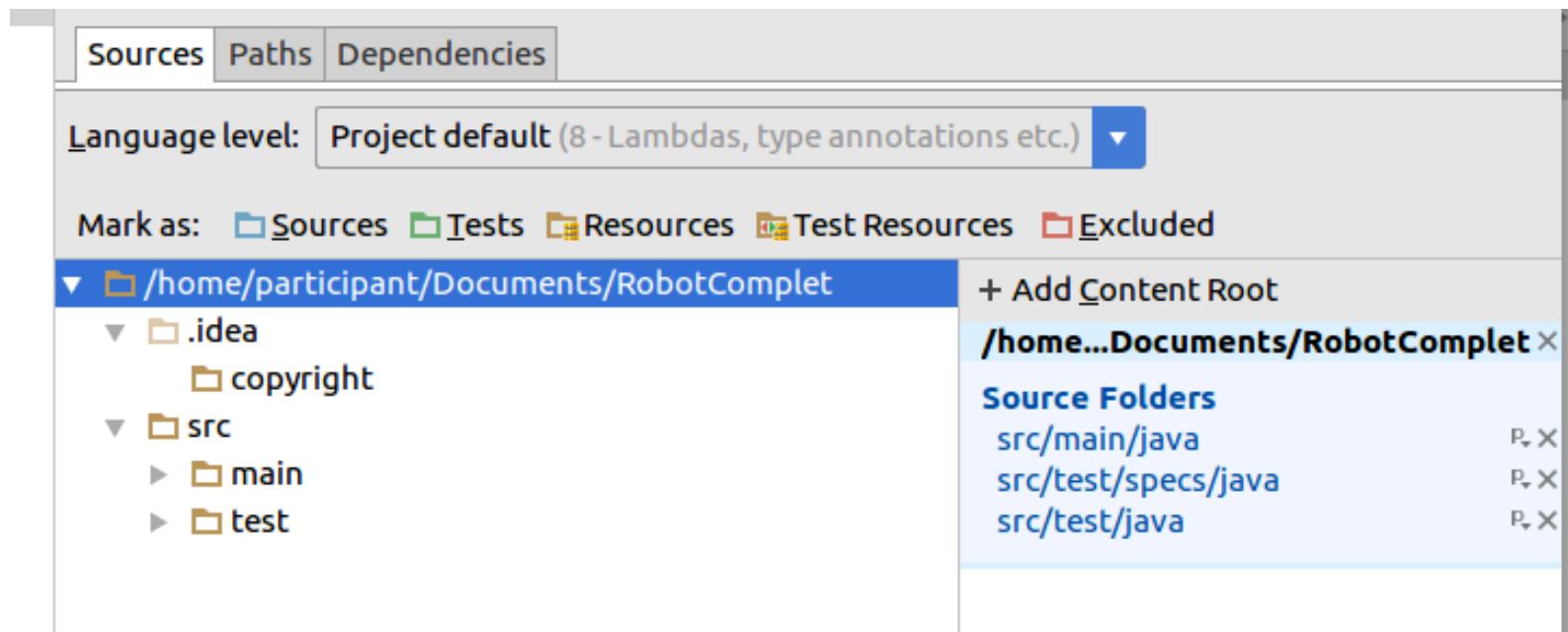
Mettre le projet sous Maven

Adaptation de la structure du projet



Mettre le projet sous Maven

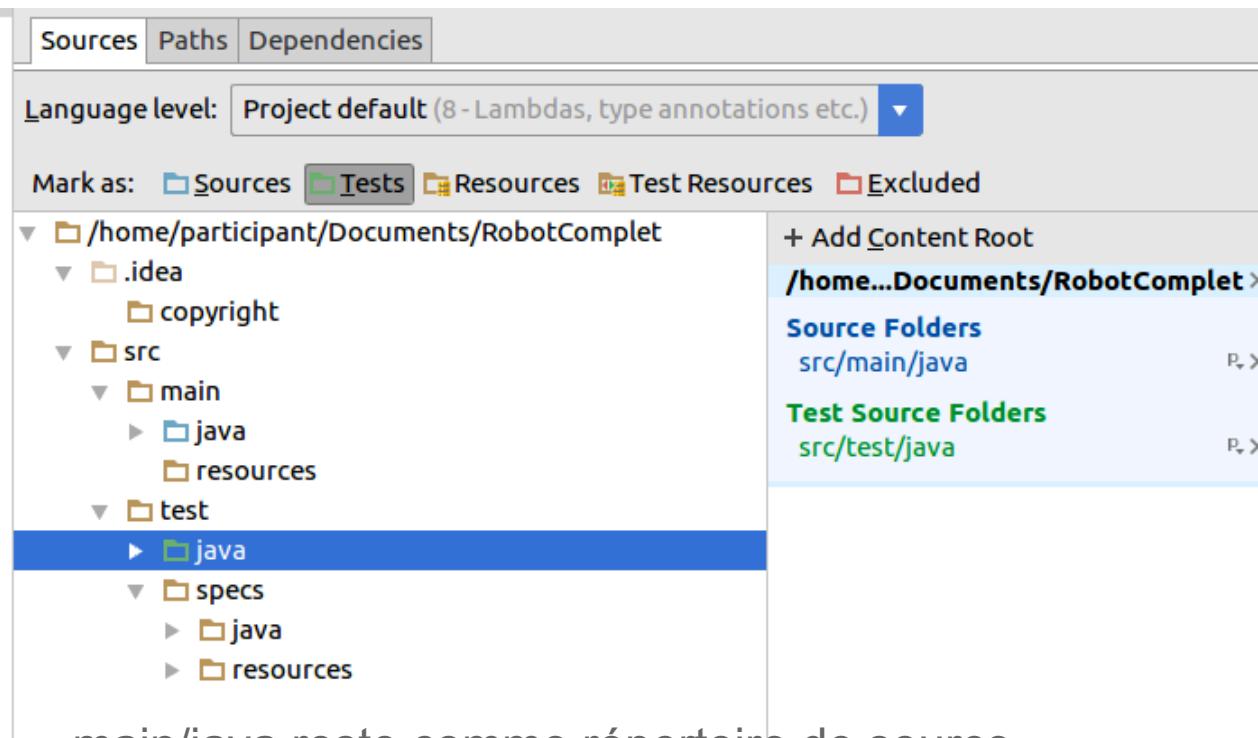
Adaptation de la structure du projet



IDEA trouve 3 répertoires contenant du Java qu'il associe comme répertoire de sources du projet

Mettre le projet sous Maven

Adaptation de la structure du projet

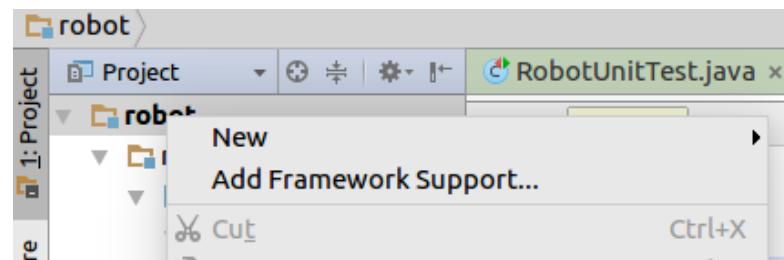


main/java reste comme répertoire de source
test/java devient répertoire de source de test
test/specs/java est pour l'instant retiré des sources

Mettre le projet sous Maven

Générer pom.xml en utilisant IDEA

- La structure est alors conforme à Maven
- Clic droit sur la racine du projet
- Choisir « Add Framework Support... »

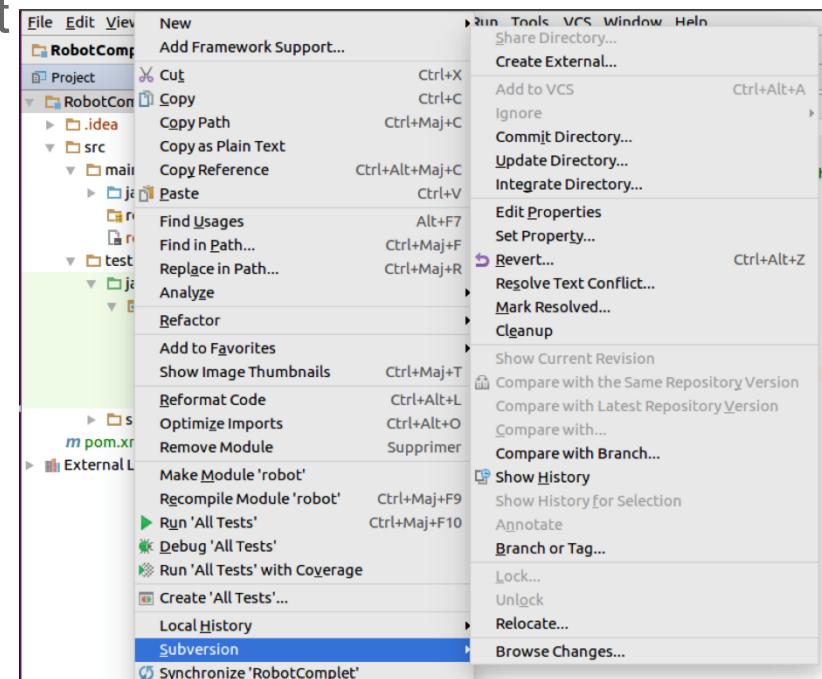


- Sélectionner Maven
- Le fichier pom.xml est généré, il faut reporter les informations de groupId et artifactId telles que données précédemment.

Mettre le projet sous Maven

Commit de la nouvelle structure

- Clic droit sur la racine du projet
- Choisir « Subversion »
- Commit Directory





Création du job Jenkins

Le projet est prêt à être intégré au build continu

Les étapes suivante permette de créer un job jenkins

en rouge : les actions à réaliser pour créer votre job sur le
Robot

Formation TestAutom [Running]

Jenkins - Mozilla Firefox

Jenkins Page d'accueil d'Ubuntu

localhost:9090/login?from=%2F

Rechercher

Jenkins

Utilisateur:

Mot de passe:

Conserver mes informations sur cet ordinateur

S'identifier

L'identification se fait par login mot de passe, les mêmes que pour accéder au repository svn

Page générée: 12 oct. 2016 21:51:05 CEST REST API Jenkins ver. 2.25

Dashboard

Le dashboard comporte 2 zones

The screenshot shows the Jenkins dashboard running in Mozilla Firefox. The main area displays a list of projects, with 'TotoJob' being the only one currently listed. The sidebar on the left contains various links and a search bar. A large watermark in the center of the screen reads 'État des jobs en cours'.

Tableau de bord [Jenkins] - Mozilla Firefox

Formation TestAutom [Running]

Tableau de bord [Je... x Page d'accueil d'Ubuntu x + localhost:9090

Rechercher

Jenkins

Nouveau Item

Utilisateurs

Historique des

Administrat

Mes

d'attente des constructions

File d'attente des constructions vide

État du lanceur de compilations

1 Au repos

2 Au repos

Tous + Nom du projet Dernier succès Dernier échec Dernière durée Ajouter une description

Icône: S M L

Légende RSS pour tout RSS de tous les échecs RSS juste pour les dernières compilations

État des jobs en cours

Création d'un nouveau job

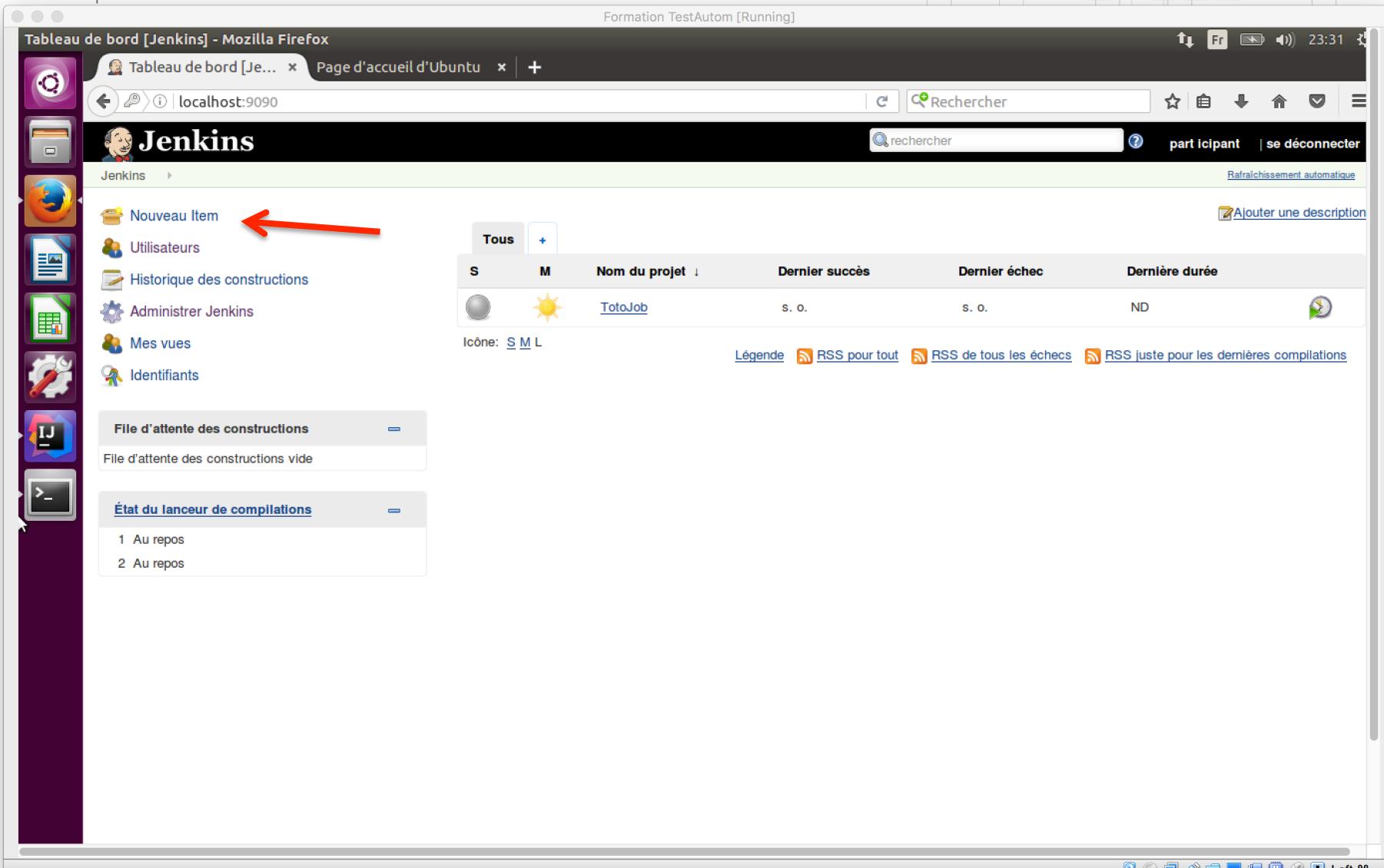


Tableau de bord [Jenkins] - Mozilla Firefox

Formation TestAutom [Running]

localhost:9090

Rechercher

Jenkins

Nouveau Item

Utilisateurs

Historique des constructions

Administrer Jenkins

Mes vues

Identifiants

Tous +

S M Nom du projet Dernier succès Dernier échec Dernière durée

S	M	Nom du projet	Dernier succès	Dernier échec	Dernière durée
		TotoJob	S. O.	S. O.	ND

Icône: S M L

Légende RSS pour tout RSS de tous les échecs RSS juste pour les dernières compilations

File d'attente des constructions

File d'attente des constructions vide

État du lanceur de compilations

1 Au repos

2 Au repos

Création d'un nouveau job

New Item [Jenkins] - Mozilla Firefox
Formation TestAutom [Running]

localhost:9090/view/Tous/newJob

Jenkins

Enter an item name

RobotUnitTest

» Required field

1. Nommer le job

Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Maven project

Build a maven project. Jenkins takes advantage of your POM files and automatically reduces the configuration.

2. Choisir le type de job

External Job

This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system.

Multi-configuration project

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

3. Valider

if OK Create a new item from other existing, you can use this option:

SCiences & Technologies

cfnrs

Configuration du job

1 - Onglet Général



The screenshot shows the Jenkins configuration interface for a job named 'RobotUnitTest'. The 'General' tab is selected. The 'Maven project name' field contains 'RobotUnitTest'. The 'Description' field is empty. Under 'Advanced...' settings, several checkboxes are present:

- Discard old builds
- Disable Automated Maven Repository Cleanup
- This project is parameterized
- Disable this project
- Execute concurrent builds if necessary

At the bottom, there are 'Save' and 'Apply' buttons.

- Nommage et description du job
- Gestion de la conservation des « build » Jenkins
- Désactivation rapide du job
- Exécution concurrente

Configuration du job

gestion de la conservation des builds



The screenshot shows the Jenkins configuration page for a job named "RobotUnitTest". The "General" tab is selected. A red circle highlights the "Description" field, which contains the text: "Job en charge de la surveillance de non-régression au niveau composant - passage des tests unitaires du projet après chaque commit" and "Conservation des 3 derniers builds". Another red circle highlights the "Max # of builds to keep" input field, which is set to "3".

La conservation des éléments construits par Jenkins peut s'exprimer en durée et/ou en nombre de builds conservés

Configuration du job

gestion du code source



RobotUnitTest Config [Jenkins] - Mozilla Firefox

RobotUnitTest Conf... × Page d'accueil d'Ubuntu × +

localhost:9090/job/RobotUnitTest/configure

Rechercher

Jenkins > RobotUnitTest >

General Source Code Management Build Triggers Build Environment Pre Steps Build Post Steps

Build Settings Post-build Actions

Source Code Management

None
 Subversion

Modules

Repository URL: svn://localhost/webRobot/RobotComplet
Credentials: participant/*****
Local module directory: .
Repository depth: infinity
Ignore externals:

Add module...

Additional Credentials: Add additional credentials...

Check-out Strategy: Use 'svn update' as much as possible

Use 'svn update' whenever possible, making the build faster. But this causes the artifacts from the previous build to remain when a new build starts.

Repository browser: (Auto)

Save Apply Advanced...

SCiences & Technologies

Configuration des accès au serveur svn

MinIVOL 2016



Configuration du job

déclenchement du job



RobotUnitTest Config [Jenkins] - Mozilla Firefox

Formation TestAutom [Running]

localhost:9090/job/RobotUnitTest/configure

Jenkins > RobotUnitTest >

General Source Code Management Build Triggers Build Environment Pre Steps Build Post Steps

Build Settings Post-build Actions

Build Triggers

Build whenever a SNAPSHOT dependency is built

Schedule build when some upstream has no successful builds

Trigger builds remotely (e.g., from scripts)

Build after other projects are built

Build periodically

Poll SCM

Schedule: H/10 7-21 * * *

Toutes les 10 mn entre 7h et 21h, tous les jours, tous les mois

No schedules so will never run

Ignore post-commit hooks

Build Environment

Save Apply

build starts

A red arrow points to the 'Poll SCM' checkbox. Another red arrow points to the schedule field, with the text 'Toutes les 10 mn entre 7h et 21h, tous les jours, tous les mois' overlaid.

Le job peut être lancé sur :

- un changement dans les librairies du projet
- à distance
- après un autre job
- périodiquement
- sur modification des sources du projet (commit)

Configuration du job

Build



The screenshot shows the Jenkins configuration interface for a job named 'RobotUnitTest'. The 'Build' tab is selected. In the 'Build' section, the 'Root POM' field contains 'pom.xml'. The 'Goals and options' field contains 'clean test', which is circled in red. At the bottom of the form, there are two buttons: 'Save' (circled in red) and 'Apply'.

Maven intègre un cycle de production dans lequel les étapes s'enchainent dans un ordre préétablit. Ici test indique qu'il faut réaliser toutes les étapes jusqu'à test

Lancer le job manuellement



Formation TestAutom [Running]

RobotUnitTest [Jenkins] - Mozilla Firefox

RobotUnitTest [Jen... x Page d'accueil d'Ubuntu x +

localhost:9090/job/RobotUnitTest/

Rechercher

Jenkins

Participant | log out

ENABLE AUTO REFRESH

Back to Dashboard

Status

Changes

Workspace

Build Now ←

Delete Maven project

Configure

Modules

Subversion Polling Log

Move

edit description

Disable Project

Maven project RobotUnitTest

Job en charge de la surveillance de non-régression au niveau composant - passage des tests unitaires du projet après chaque commit
Conservation des 3 derniers builds

Workspace

Recent Changes

Permalinks

- Last build (#2), 5 min 4 sec ago
- Last failed build (#2), 5 min 4 sec ago
- Last unsuccessful build (#2), 5 min 4 sec ago
- Last completed build (#2), 5 min 4 sec ago

Build History trend →

#2 Oct 13, 2016 12:13 PM

#1 Oct 13, 2016 11:23 AM

RSS for all RSS for failures

Left 36

Consulter le résultat

The image displays three screenshots of a Jenkins interface, illustrating the process of monitoring and reviewing build results.

- Screenshot 1: Build History**
This screenshot shows the Jenkins dashboard for the "RobotUnitTest" project. A red arrow points to the "Build History" link in the left sidebar. The main content area displays the latest four builds, with the most recent one being successful (green icon) and the previous one failing (red icon).

Build #	Result	Timestamp
Oct 13, 2016 4:12 PM	Success	Oct 13, 2016 4:12:46 PM
Oct 13, 2016 12:20 PM	Failure	Oct 13, 2016 12:20:00 PM
Oct 13, 2016 12:18 PM	Success	Oct 13, 2016 12:18:00 PM
Oct 13, 2016 12:16 PM	Failure	Oct 13, 2016 12:16:00 PM
- Screenshot 2: Build #4 Details**
This screenshot shows the details of the fourth build. A red arrow points to the "Test Result" link in the left sidebar. The page lists the test results for the build, showing three failures.
 - robot.BatteryUnitTest.testCharge
 - robot.BatteryUnitTest.testUseMax
 - robot.RobotUnitTest.testLetsGo
- Screenshot 3: Detailed Test Result**
This screenshot shows a detailed view of the failed tests from the fourth build. A red arrow points to the "Test Result" link in the left sidebar. The "Failed Tests" section lists:

Test Name	Duration	Age
robot.BatteryUnitTest.testCharge	27 ms	1
robot.BatteryUnitTest.testUseMax	1 ms	1
robot.RobotUnitTest.testLetsGo	2 ms	1

Ajouter la publication des rapports de test



fichier pom.xml

ajout d'un plugin d'édition de rapport

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
...
    <version>1.0-SNAPSHOT</version>
...
    </dependencies>
    <reporting>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-report-plugin</artifactId>
                <version>2.19.1</version>
            </plugin>
        </plugins>
    </reporting>
</project>
```

Publication des rapports de test en html

Configurer le job

The screenshot shows a Mozilla Firefox browser window with the title "Formation TestAutom [Running]". The address bar displays "localhost:9090/job/RobotUnitTest/". The main content is the Jenkins dashboard for the "RobotUnitTest" project.

Project Information:

- Name:** Maven project RobotUnitTest
- Description:** Job en charge de la surveillance de la non-régression au niveau composant - passage des tests unitaires du projet après chaque commit
Conservation des 3 derniers builds
- Actions:** edit description, Disable Project

Project Configuration:

- Workspace
- Recent Changes
- Latest Test Result (3 failures)
- Latest Test Result (3 failures)

Build History:

Build #	Date
#4	Oct 13, 2016 4:12 PM
#3	Oct 13, 2016 12:20 PM
#2	Oct 13, 2016 12:13 PM

Permalinks:

- Last build (#4), 1 min 14 sec ago
- Last successful build (#4), 1 min 14 sec ago
- Last failed build (#3), 3 hr 53 min ago
- Last unstable build (#4), 1 min 14 sec ago
- Last unsuccessful build (#4), 1 min 14 sec ago
- Last completed build (#4), 1 min 14 sec ago

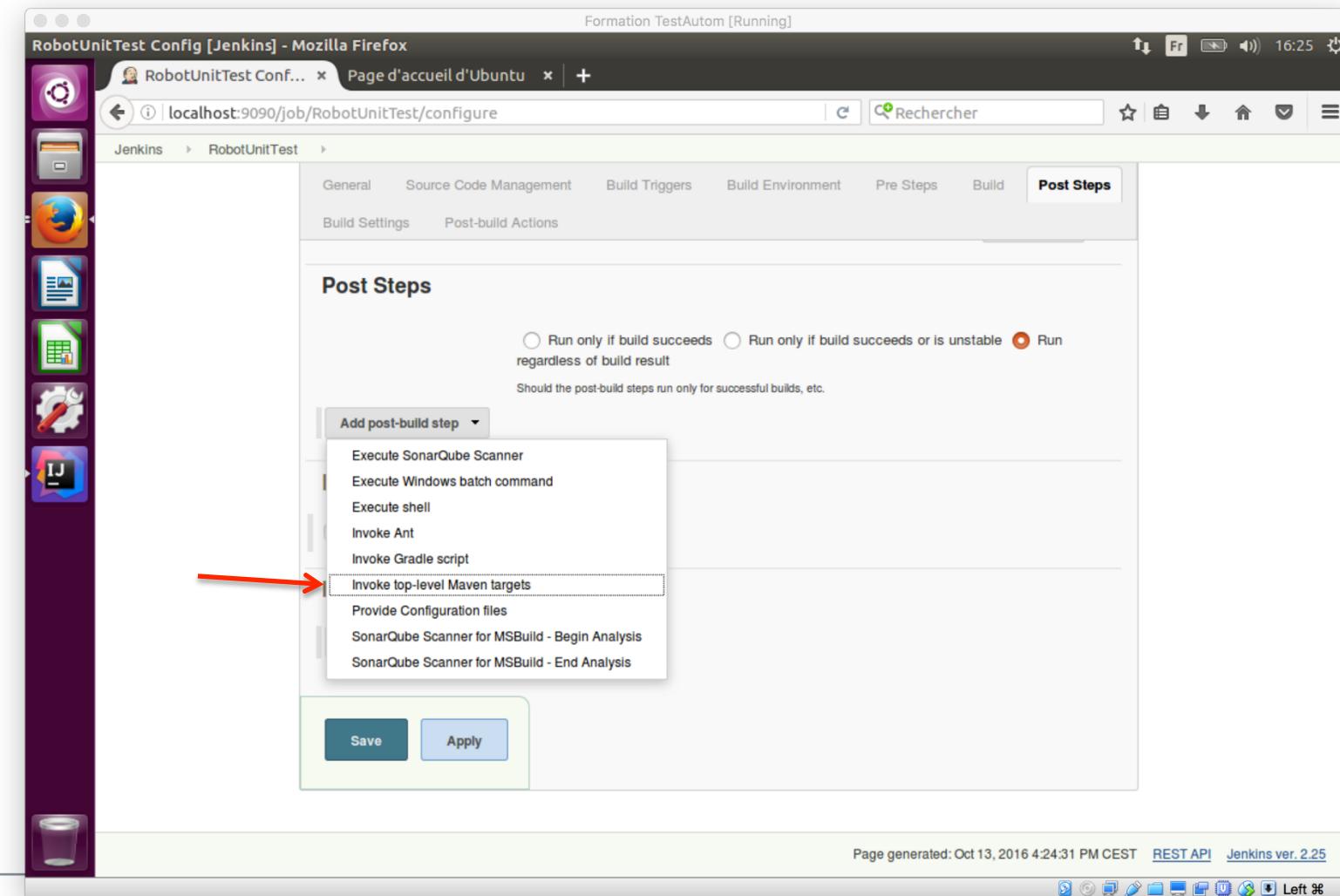
Bottom Navigation:

- RSS for all
- RSS for failures

Address Bar: localhost:9090/job/RobotUnitTest/modules

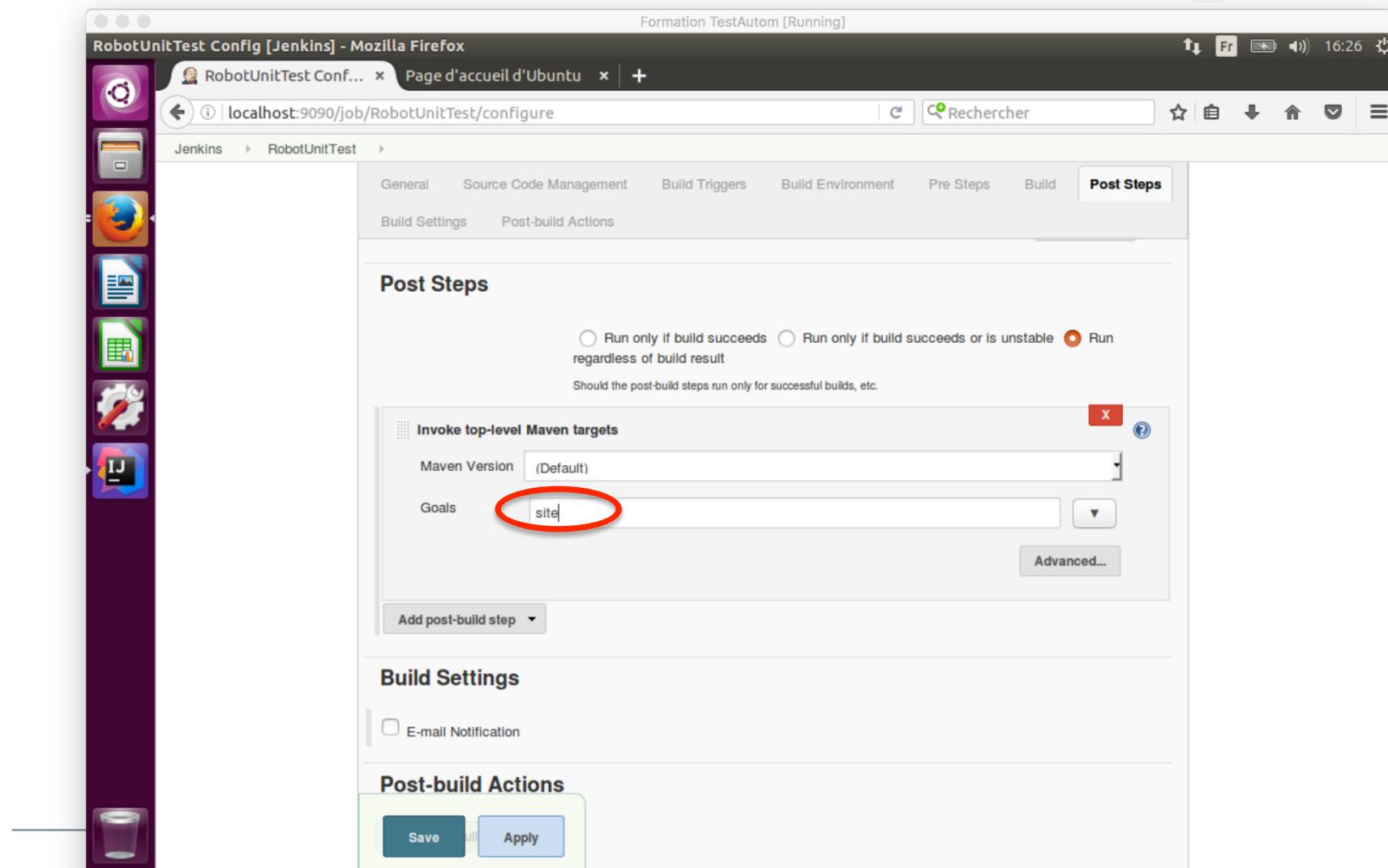
Publication des rapports de test en html

Ajouter une cible maven après le build



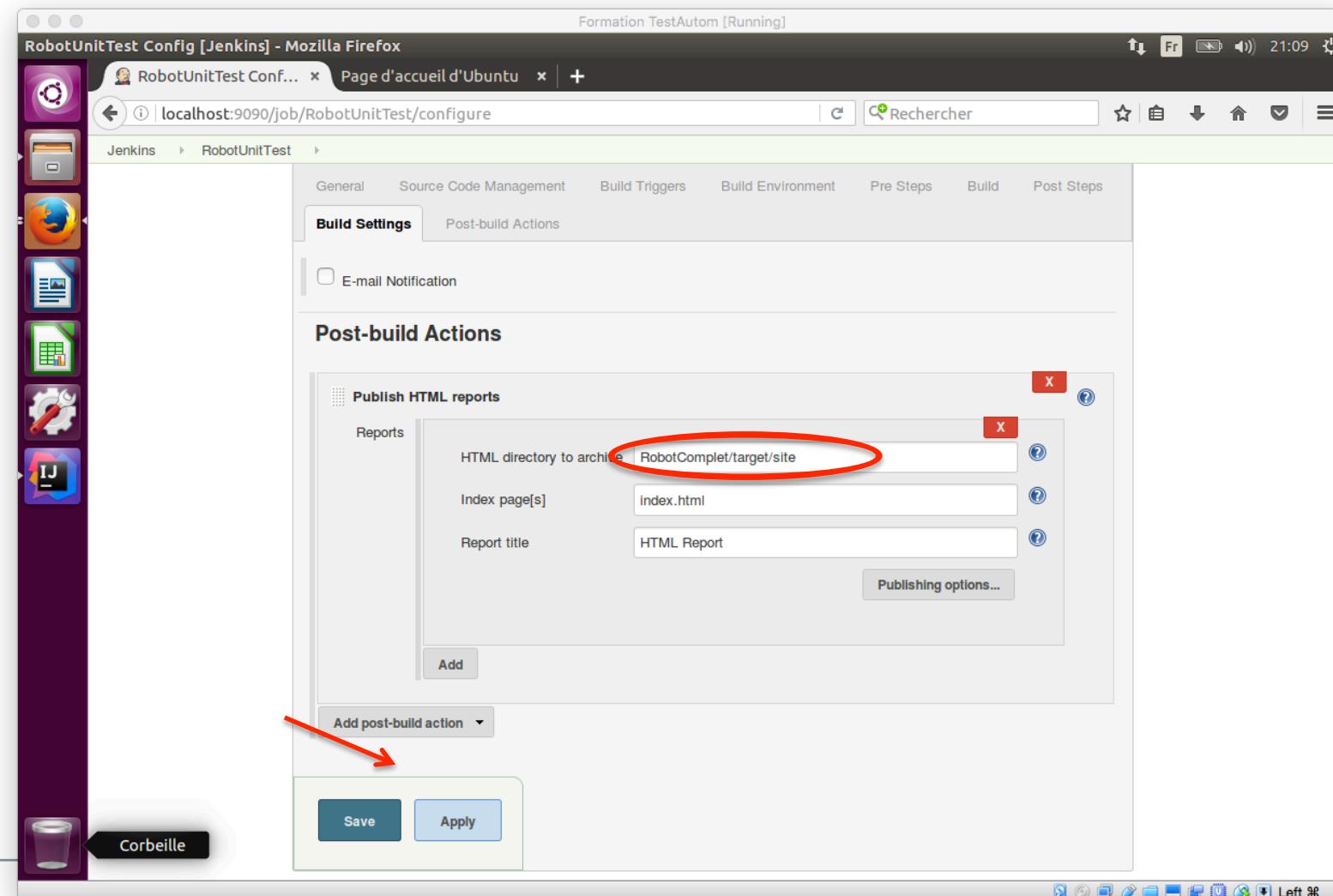
Publication des rapports de test en html

Ajouter une cible maven site après le build



Publication des rapports de test en html

Ajouter l'action « Publish HTML reports



Accès aux rapports html



Après avoir lancé un build

RobotUnitTest [Jenkins] - Mozilla Firefox

Formation TestAutom [Running]

RobotUnitTest [Jen... Page d'accueil d'Ubuntu RobotComplet - Projec... +

localhost:9090/job/RobotUnitTest/ Rechercher part icipant log out ENABLE AUTO REFRESH

Jenkins > RobotUnitTest >

Back to Dashboard Status Changes Workspace Build Now Delete Maven project Configure Modules Subversion Polling Log HTML Report Move

[edit description](#) [Disable Project](#)

Maven project RobotUnitTest

Job en charge de la surveillance de non-régression au niveau composant - passage des tests unitaires du projet après chaque commit
Conservation des 3 derniers builds

[HTML Report](#) [Workspace](#) [Recent Changes](#) [Latest Test Result \(3 failures / ±0\)](#)

[Latest Test Result \(3 failures / ±0\)](#)

Test Result Trend

count

#7 #8 #9

(just show failures) enlarge

Build History trend

find

#9 Oct 13, 2016 9:05 PM
#8 Oct 13, 2016 5:27 PM
#7 Oct 13, 2016 5:24 PM

[RSS for all](#) [RSS for failures](#)

Permalinks

- [Last build \(#9\), 14 min ago](#)
- [Last successful build \(#9\), 14 min ago](#)
- [Last failed build \(#7\), 3 hr 54 min ago](#)
- [Last unstable build \(#9\), 14 min ago](#)
- [Last unsuccessful build \(#9\), 14 min ago](#)
- [Last completed build \(#9\), 14 min ago](#)

SCIENCES & TECHNOLOGIES

Left 96 MiniVOL 201

Ajouter une analyse statique du code avec Sonar



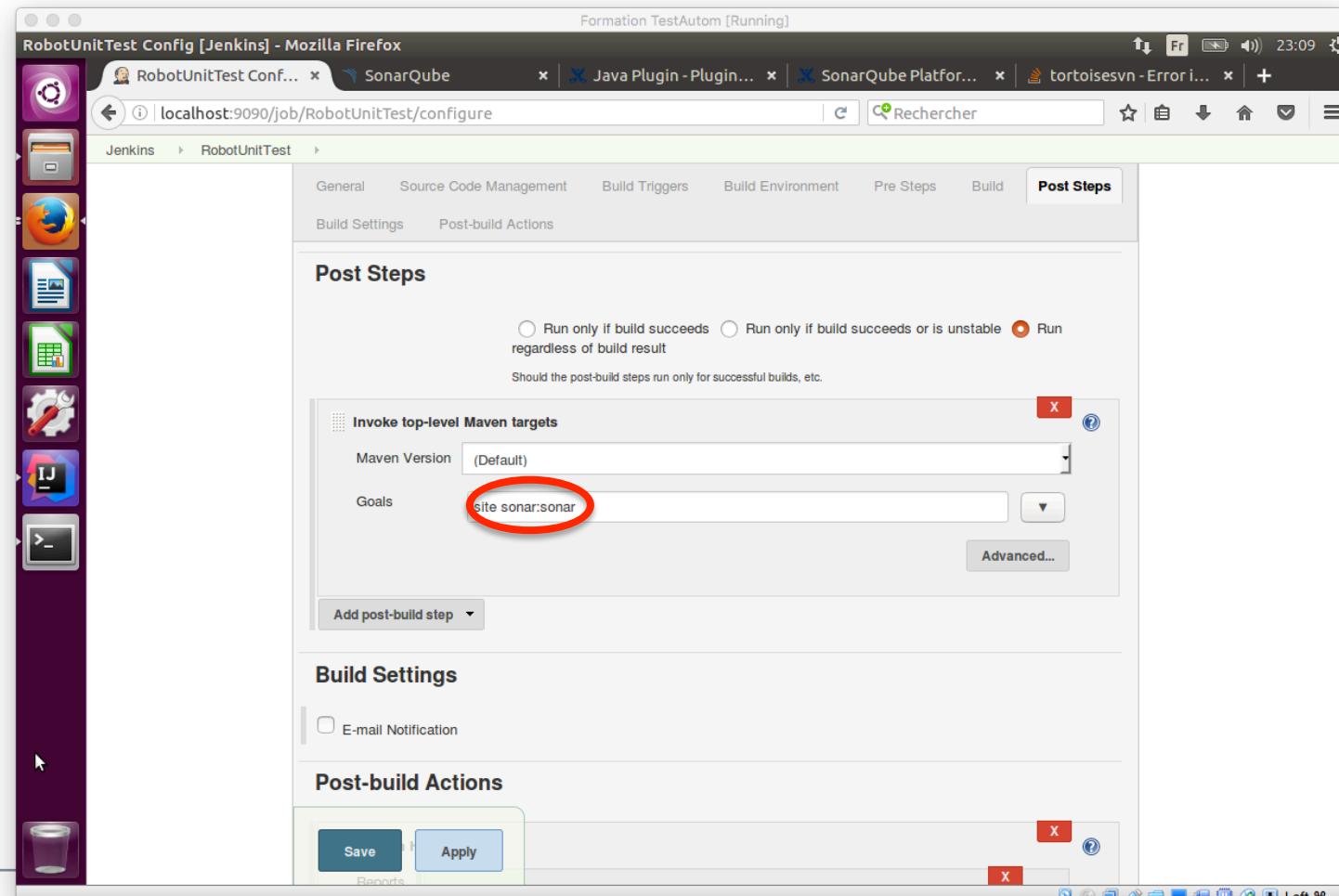
SonarQube Serveur

Outil d'analyse de qualité du code

- Permet une analyse statique du code
- Vérifie le respect de règles de codages pré-établies
- Donne des métriques sur la qualité du code
- serveur accessible sur <http://localhost:9000>
- L'analyse peut être lancée depuis le répertoire du projet par :
 - mvn sonar:sonar
- L'analyse peut être lancée depuis un job Jenkins

Analyse déclenchée depuis un job

Ajout d'une cible maven sonar après le build



Récupérer l'analyse Sonar dans l'IDE

Qualité du code



Feedback Sonar

- L'analyse Sonar déclenchée par un job Jenkins est silencieuse
- Le développeur doit consulter le serveur SonarQube pour prendre connaissance des rapports
- L'utilisation du Plugin SonarQube dans Idea permet d'obtenir cette boucle de feedback
- Les éléments d'analyse sont alors disponibles dans l'IDE

4. Test fonctionnel et son automatisation

Test d'acceptation en java : Concordion

Test fonctionnel de site Web : Sélénum

Gestion des exigences : Squash TM/TA

Test d'Acceptation en Java



Définition

Un test d'acceptation est un test métier permettant de valider tout ou partie d'une fonctionnalité.

Les tests d'acceptation permettent au client de vérifier qu'une fonctionnalité a été implémentée. Si l'ensemble des tests d'acceptation d'une fonctionnalité sont verts, le client peut accepter la fonctionnalité.

Par nature se sont des **tests fonctionnels**.

Test d'acceptation & Application

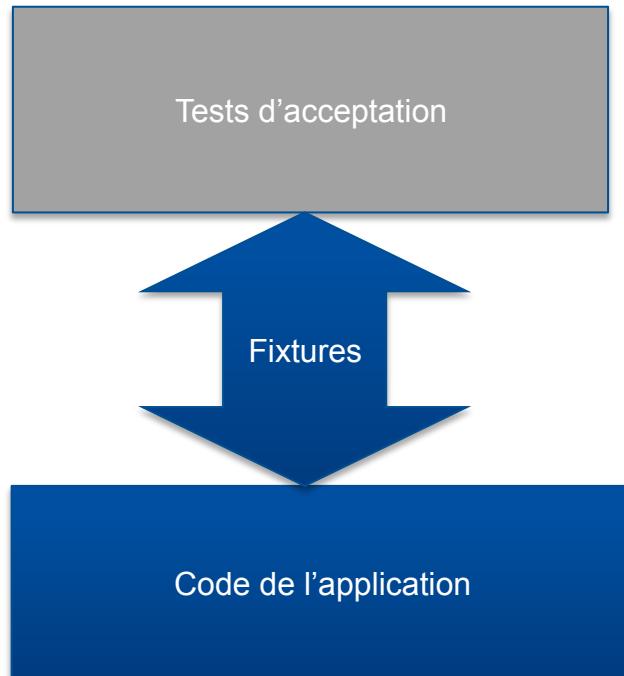


Tests d'acceptation
Langage métier, html/wiki

Fixture
java

Code de l'application
Langage java

Acteurs du test d'acceptation



Le **client** définit la fonctionnalité à implémenter et les tests d'acceptation associés

Le **développeur** code l'application et les fixtures permettant de réaliser le lien entre les tests d'acceptation et le code

Outils du Test d'Acceptation



wiki – accessible

Nécessite un serveur
d'interprétation

Visualisation des résultats
dans le wiki

<http://fitnesse.org/>



HTML – difficile

Similaire à JUnit –
intégration aux suites de
test

Résultats dans une page
différente

<http://www.concordion.org/>

Utilisation de Concordion



Distribution

- Soit en utilisant maven

```
<dependency>
  <groupId>org.concordion</groupId>
  <artifactId>concordion</artifactId>
  <version>2.0.3</version>
  <scope>test</scope>
</dependency>
```

- Soit en téléchargeant l'archive sur le site <http://concordion.org>
 - <http://dl.bintray.com/concordion/downloads/concordion-2.0.3.zip>

Utilisation de concordion



Principes de fonctionnement

- Appareiller des pages html avec des instructions concordion
 - pour extraire les valeurs d'entrée
 - pour appeler les fixtures
 - pour comparer les résultats aux oracles
- Écrire les fixtures java qui vont :
 - mettre en forme les entrées recueillies dans les pages html
 - créer les instances de SUT utiles à l'interprétation de la page
 - mettre en forme les résultats pour les retourner dans la page



Exemple

Page html à appareiller avec Concordion

Le Robot

Ceci est la page principale de description des tests d'acceptations sur le robot.

Caractéristiques du robot

Le robot possède certaines spécificités :

- Pour la gestion des [déplacements](#)
- Pour la gestion de l'[énergie](#)
- Pour la gestion de la [cartographie](#)

Au départ, le robot se trouve en vol, il atterrit à des coordonnées qui lui sont spécifiées. Ces coordonnées correspondent ensuite à son point de départ pour toute action qu'il souhaite effectuer. Lorsqu'il se pose, le robot est systématiquement orienté vers le nord.

Si l'on donne l'ordre au robot de se poser en coordonnées (3, 2), sa direction sera obligatoirement nord.

l'exemple mentionné dans la page sera le support du test d'acceptation

Fonctionnement du robot en condition réelle

Lors du fonctionnement en condition réelle du robot, il est important de prendre en considération la surface sur laquelle il doit se déplacer mais également le point qu'il doit atteindre. En effet, cela va intervenir dans le calcul de l'itinéraire à suivre. Ainsi, un autre module qui compose le robot est le [calculateur d'itinéraire](#). Celui-ci couple à la fois la gestion de l'énergie et le déplacement du robot afin de déterminer quel itinéraire le robot doit suivre pour consommer le moins d'énergie possible (le plus court chemin n'est pas forcément le moins coûteux).



Exemple

Appareillage du source html

```
<div class="example">  
Si l'on donne l'ordre au robot de se poser en coordonnées <span concordion:set="#coordonnee">(3, 2)</span>,  
sa direction sera obligatoirement <span concordion:assertEquals="directionAfterLanding(#coordonnee)">nord</span>.  
</div>  
| Attribute concordion:assertEquals is not allowed here more... (⌘F1)
```

- Les commandes concordion prennent place dans des balises qui peuvent être soit déjà dans la page soit ajoutées spécialement
- <balise concordion:set="#variable">valeur</balise> permet de capturer valeur dans la page html pour l'affecter à #variable
- <balise concordion:assertEquals="methodeDeFixture(#variable)">valeur</balise> permet d'appeler methodeDeFixture avec #variable en paramètre et de comparer le résultat avec valeur

Exemple

Fixture

```
@RunWith(ConcordionRunner.class)
public class RobotFixture {

    private Robot walle = new Robot(0, new Battery());

    public String directionAfterLanding(String coordonnees) throws LandSensorDefaillance {
        String trim = coordonnees.replace('(', ' ').replace(')', ' ').trim();
        String[] split = trim.split(" ");
        walle.land(new Coordinates(Integer.valueOf(split[0]), Integer.valueOf(split[1])), new LandSensor(new Random()));
        return directionAsString();
    }

    private String directionAsString() {
        Direction direction;
        try {
            direction = walle.getDirection();
        } catch (UnlandedRobotException e) {
            return e.getMessage();
        }
        switch (direction) {
            case NORTH:
                return "nord";
            case EAST:
                return "est";
            case SOUTH:
                return "sud";
            case WEST:
                return "ouest";
        }
        return "";
    }
}
```

- La méthode de la fixture extrait les éléments de la chaîne de caractères quelle reçoit en argument
- La fixture construit un objet sous test
- La méthode appelle la méthode à tester
- La méthode retourne, sous forme d'une chaîne de caractères, le résultat de l'exécution de la méthode sous test

Exemple

Résultat

Le Robot

Ceci est la page principale de description des tests d'acceptations sur le robot.

Caractéristiques du robot

Le robot possède certaines spécificités :

- Pour la gestion des [déplacements](#)
- Pour la gestion de l'[énergie](#)
- Pour la gestion de la [cartographie](#)

Au départ, le robot se trouve en vol, il atterrira à des coordonnées qui lui sont spécifiées. Ces coordonnées correspondent ensuite à son point de départ pour toute action qu'il souhaite effectuer. Lorsqu'il se pose, le robot est systématiquement orienté vers le nord.

Si l'on donne l'ordre au robot de se poser en coordonnées (3, 2), sa direction sera obligatoirement [nord](#).

Fonctionnement du robot en condition réelle

Lors du fonctionnement en condition réelle du robot, il est important de prendre en considération la surface sur laquelle il doit se déplacer mais également le point qu'il doit atteindre. En effet, cela va intervenir dans le calcul de l'itinéraire à suivre. Ainsi, un autre module qui compose le robot est le [calculateur d'itinéraire](#). Celui-ci couple à la fois la gestion de l'énergie et le déplacement du robot afin de déterminer quel itinéraire le robot doit suivre pour consommer le moins d'énergie possible (le plus court chemin n'est pas forcément le moins coûteux).



La page produite intègre les verdicts des tests sous forme d'une coloration verte ou rouge des valeurs des oracles

Results generated by [Concordion](#)
in 7 ms on 26-nov.-2015 at 23:41:03 CET

Agilité et Tests d'Acceptation



Les méthodes agiles utilisent des cycles de développement courts pendant lesquels sont pris en charge la réalisation de "stories". La définition et la "mise en page" des tests d'acceptation prennent naturellement place avant de débuter l'implémentation relative à une story.

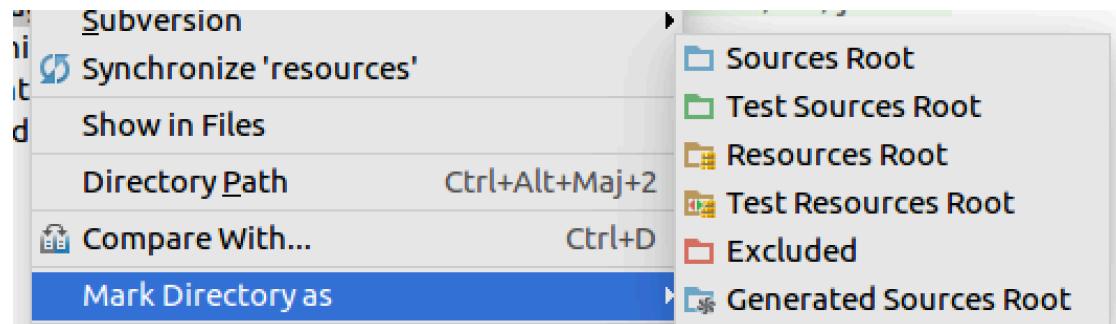
ATDD : Acceptance Test Driven Development



Exercice

Spécifications sur le Robot

- Dans l'IDE, reprenez le projet RobotComplet
- Considérez le répertoire spec fournit
- Intégrez test/specs/java comme répertoire de source de tests
- Marquez test/specs/resources comme répertoire de ressources de test





Exercice

Spécifications sur le Robot

- Modifier le pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<dependencies>
    <dependency>
        ...
    </dependency>
    <dependency>
        <groupId>org.concordion</groupId>
        <artifactId>concordion</artifactId>
        <version>2.0.3</version>
        <scope>test</scope>
    </dependency>
</dependencies>
</project>
```



Exercice

Spécifications sur le Robot

- Exécutez les tests (comme des tests Junit)
- Les résultats de l'exécution sont disponibles dans des fichier html dans le répertoire /tmp/concordion
- La page Mouvement.html est appareillée et la fixture MouvementFixture.java est disponible.
- **Appareillez la gestion de l'Energie**



Intégration à Jenkins

Définition d'un profil dans pom.xml

- Dans le fichier alnclureDansPom.xml vous trouverez la définition d'un profile adapté à l'exécution des tests d'acceptations
- Ce profile permet de :
 - spécifier des répertoires non usuels comme répertoires de tests ou de ressource de tests
 - effectuer la copie des fichiers images vers les répertoires de production de concordion (les pages résultats intègrent alors les images)
 - spécifier le répertoire de sortie de concordion



Intégration à Jenkins

Création d'un job dédié aux tests d'acceptation

- Reprenez les étapes de création d'un nouveau job Jenkins
- Pour ce qui déclenche le build choisir « Construire à la suite d'autres projets » et utiliser le job de tests unitaires comme projet amont
- Pour les cibles de build : site –Pacceptance-test
- Pour les Actions à la suite du build (cf slide suivant)
 - Publish Concordion test report
 - Publish HTML reports



Formation TestAutom [Running]

RobotAcceptanceTest Config [Jenkins] - Mozilla Firefox

localhost:9090/job/RobotAcceptanceTest/configure

Rechercher

Jenkins > RobotAcceptanceTest >

General Source Code Management Build Triggers Build Environment Pre Steps Build Post Steps

Build Settings Post-build Actions

Post-build Actions

Publish Concordion test report

Concordion report location: RobotComplet/target/concordion

Publish HTML reports

Reports

HTML directory to archive: RobotComplet/target/site
Index page[s]: index.html
Report title: HTML Report
Publishing options...

HTML directory to archive: RobotComplet/target/concordion
Index page[s]: Index.html
Report title: Acceptance Test Report
Publishing options...

Save Apply

This screenshot shows the Jenkins configuration interface for the 'RobotAcceptanceTest' job. The 'Post-build Actions' section is active, displaying two configured actions: 'Publish Concordion test report' and 'Publish HTML reports'. The 'Publish HTML reports' action is expanded, showing settings for archiving the 'RobotComplet/target/site' directory, using 'index.html' as the index page, and naming the report as 'HTML Report'. Below this, another instance of the 'Publish HTML reports' action is shown with similar settings but a different report title: 'Acceptance Test Report'. At the bottom of the configuration page are 'Save' and 'Apply' buttons.

Test Fonctionnel de site Web

Processus d'automatisation

D

1.

```
sudo apt-get remove firefox
```

2.

```
sudo apt-get install firefox=45.0.2+build1-0ubuntu1
```

3.



Test Selenium / Java

- L'objectif de cette étape est de simuler le comportement du navigateur pour :
 - . Tester le comportement de la page
 - . Contrôler que le résultat obtenu est conforme au résultat escompté suite à une action utilisateur
- Les tests sont décrits en Java et s'appuient sur jUnit (utilisation d'assertions)
- Librairies existantes : Selenium, HtmlUnit...
- Crée un projet Java ou Maven, ajouter les 2 libs ou dépendances :

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>compile</scope>
    </dependency>
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>2.53.1</version>
    </dependency>
</dependencies>
```



Etape 1 – Configurer / Installer

Deux types : Serveur ou Add-on

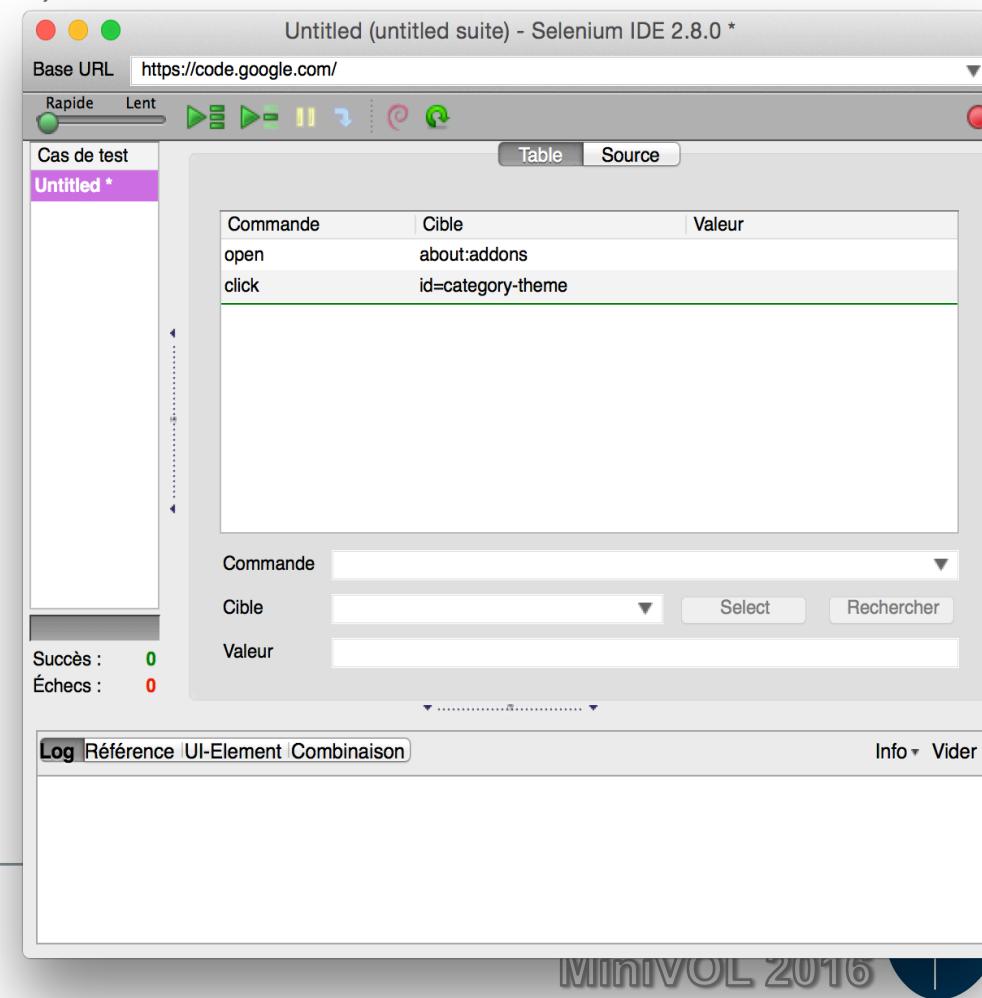
- Site <http://www.seleniumhq.org/>
- Installer Selenium IDE (Add-on Firefox) :

Menu : 'Outils'

- Modules Complémentaire
- Extensions'

Version 2.9.1.1 signed

- Lancer Selenium :
'Outils→ Selenium IDE'





Etape 2 – Jouer un test

- Aller sur l'url et enregistrer :

<http://localhost/webRobot/Accueil.php>

- Utiliser le bouton dans Selenium IDE : ▲ ≡
- Dans Selenium Fichier → Exporter le test sous ... → Java / Junit4 / WebDriver
- Sauver le fichier dans vos sources de projet IDEA
- Rafraîchir dans Idea vos sources, corriger les dépendances (Selenium-Java, Junit, libs...) et corriger nom du package et « / » dans variable baseUrl
- Exécuter vos premiers tests unitaires.

Ajouter des vérifications :

- Par exemple vérifier que la valeur de x vaut 0 Il faut utiliser :
 - Méthode de test : Assert.assertEquals(String 1, String 2)
 - Récupération de la valeur de "x" :

```
driver.findElement(By.id("x")).getText()
```

Exercice



- Créer vos tests fonctionnels sur l'Application webRobot
- Connecter vous à l'url :
`http://localhost/webRobot/Accueil.php`
- Vérifier que le comportement du robot est bien conforme aux attentes, tester :
 - Le déplacement du robot dans toutes les directions
 - Un déplacement avec retour au point de départ
 - La remise à l'état initial du robot après avoir fait 5 déplacements



Autre WebDriver : htmlUnit

- Pilote pour appel sans ouverture de navigateur
- Simplifie le test d'application web sans navigateur (build continue)
- Normalement, livré avec Selenium mais plus depuis la version 2.43
- Téléchargeable à <http://htmlUnit.sourceforge.net> ou mettre la dépendence maven :

```
<dependency><groupId>org.seleniumhq.selenium</groupId>
  <artifactId>htmlunit-driver</artifactId> <version>2.21</version></dependency>
```

- Une extension du framework jUnit et Selenium

```
import org.junit.*;
import org.openqa.selenium.htmlunit.HtmlUnitDriver;

public class RobotSeleniumTest {
    private HtmlUnitDriver driver; private String baseUrl;    private String valX;

    @Before
    public void setUp() throws Exception {
        driver = new HtmlUnitDriver(true);
        baseUrl = "http://localhost/webRobot"; }

    @Test
    public void testRobotSelenium() throws Exception {
        driver.get(baseUrl+"/Accueil.php");
        valX = driver.findElementById("x").getText();
        Assert.assertEquals("Erreur sur abscisse", valX, "0");
    }
}
```

Identification des exigences

Exemple de la cartographie du robot

Le Robot

Descriptif



- Caractéristiques du robot :
 - Gestion des déplacements
 - Gestion de l'énergie
 - Gestion de la cartographie
- Etat initial :
 - Le robot se trouve en vol.
 - Il atterrit à des coordonnées qui lui sont spécifiées et en direction du nord.
 - Ces coordonnées correspondent ensuite à son point de départ pour toute action qu'il souhaite effectuer.
 - La batterie a initialement 20 unités de charges



Cartographie

La Cartographie est réalisé avec une caméra

- Zone de cartographie
 - Ne peut être réalisée qu'à l'arrêt
 - Un carré de 9x9
 - Chaque activation de la caméra permet d'agrger des éléments de cartographie et d'augmenter la partie connue de la carte.
 - Une première cartographie est systématiquement effectuée lorsque le robot atterrit.
- Type de terrain identifié :
 - Terre
 - Roche
 - Boue
 - Sable
 - Infranchissable



Le déplacement

- Type de déplacement (une fois qu'il a atterri) :
 - Mouvement par mouvement
- Déplacements mouvement par mouvement :
 - Le robot peut se déplacer soit en avant soit en arrière.
 - Il peut également faire une rotation sur lui-même d'un quart de tour soit dans le sens des aiguilles d'une montre, soit dans le sens inverse.

L'énergie



Chaque mouvement réalisé par le robot coûte de l'énergie

- Consommation en fonction du terrain
 - Terre : 1 unité
 - Roche : 2 unités
 - Boue : 3 unités
 - Sable : 4 unités
- En cas d'insuffisance énergétique :
 - Le robot s'arrête.
 - Il ne répond plus tant que la charge n'est pas redevenue suffisante.
- Récupération d'énergie (capteur solaire) :
 - Il récupère N unités d'énergie chaque M temps avec les capteur déployés
 - Aucune autre action n'est possible pendant la charge

Organisation de l'atelier



Objectifs fonctionnels

- Cartographie
- Déplacements mouvement par mouvement
- Gestion de l'énergie
- Méthode :
 - Identification des caractéristiques fonctionnelles (exigences)
 - Identification des points de contrôle et d'observation
 - Pour chaque exigence, définition des cas (passant et non passant) à couvrir :
 - Définition du contexte de test
 - Définition des données d'entrée
 - Définition des attendus



Exemple : Exigences Cartographie

Définition des exigences

- **Ex0** : La cartographie ne peut être réalisée qu'à l'arrêt.
- **Ex1** : Une première cartographie est systématiquement effectuée lorsque le robot atterrit.
- **Ex2** : A l'état initial, aucune case n'est connue.
- **Ex3** : La cartographie renseigne chaque case par la nature de son terrain :
 - Terre,
 - Roche,
 - Boue,
 - Sable,
 - Infranchissable.
- **Ex4** : La cartographie permet de couvrir un carré de 9x9.
- **Ex5** : La cartographie permet d'agréger les éléments des cases couvertes aux éléments de cartographie déjà connus.



Exemple : Exigences Cartographie

Définition des points de contrôle et d'observation

- Points de contrôle :
 - Faire atterrir le robot
 - Déplacer le robot :
 - Aller en avant
 - Aller en arrière
 - Tourner à droite
 - Tourner à gauche
 - Faire une cartographie autour du robot
- Points d'observation :
 - La carte du terrain connue



Exemple : Exigences Cartographie

Exemple de scénario

- **Ex0** : La cartographie ne peut être réalisée qu'à l'arrêt.
 - **Ex1** : Une première cartographie est systématiquement effectuée lorsque le robot atterrit.
 - **Ex2** : A l'état initial, aucune case n'est connue.
 - **Ex3** : La cartographie renseigne chaque case par la nature de son terrain.
 - **Ex4** : La cartographie permet de couvrir un carré de 9x9.
 - **Ex5** : La cartographie permet d'agréger les éléments des cases couvertes aux éléments de cartographie déjà connus.
-
- **TEST 1** : Aucune cartographie en vol (Ex0)
 1. Action : Faire une cartographie
 2. Observation :
 - La cartographie n'est pas réalisée (Ex0)



Exemple : Exigences Cartographie

Exemple de scénario

- **Ex0** : La cartographie ne peut être réalisée qu'à l'arrêt.
 - **Ex1** : Une première cartographie est systématiquement effectuée lorsque le robot atterrit.
 - **Ex2** : A l'état initial, aucune case n'est connue.
 - **Ex3** : La cartographie renseigne chaque case par la nature de son terrain.
 - **Ex4** : La cartographie permet de couvrir un carré de 9x9.
 - **Ex5** : La cartographie permet d'agréger les éléments des cases couvertes aux éléments de cartographie déjà connus.
-
- **TEST 2** : Une première cartographie est réalisée à l'atterrisage (Ex1)
 1. **Action** : Faire atterrir le robot
 2. **Observation** :
 - Une cartographie est réalisée (Ex1)
 - Chaque case du carré de 9x9 est renseignée (Ex3,Ex4)
 - Aucune case en dehors du carré de 9x9 n'est renseignée (Ex2,Ex4)



Exemple : Exigences Cartographie

Exemple de scénario

- **Ex0** : La cartographie ne peut être réalisée qu'à l'arrêt.
 - **Ex1** : Une première cartographie est systématiquement effectuée lorsque le robot atterrit.
 - **Ex2** : A l'état initial, aucune case n'est connue.
 - **Ex3** : La cartographie renseigne chaque case par la nature de son terrain.
 - **Ex4** : La cartographie permet de couvrir un carré de 9x9.
 - **Ex5** : La cartographie permet d'agréger les éléments des cases couvertes aux éléments de cartographie déjà connus.
-
- **TEST 3** : Deux cartographies successives donnent le même résultat (Ex5)
 1. **Action** : Faire atterrir le robot, Faire une cartographie
 2. **Observation** :
 - La cartographie doit être identique à celle de l'atterrissement (Ex5)
 - Chaque case du carré de 9x9 est renseignée (Ex3,Ex4)
 - Aucune case en dehors du carré de 9x9 n'est renseignée (Ex2,Ex4,Ex5)



Exemple : Exigences Cartographie

Exemple de scénario

- **Ex0** : La cartographie ne peut être réalisée qu'à l'arrêt.
 - **Ex1** : Une première cartographie est systématiquement effectuée lorsque le robot atterrit.
 - **Ex2** : A l'état initial, aucune case n'est connue.
 - **Ex3** : La cartographie renseigne chaque case par la nature de son terrain.
 - **Ex4** : La cartographie permet de couvrir un carré de 9x9.
 - **Ex5** : La cartographie permet d'agréger les éléments des cases couvertes aux éléments de cartographie déjà connus.
-
- **TEST 4** : Cartographies identiques avec un mouvement d'aller-retour (Ex5)
 1. **Action** : Faire atterrir le robot, Avancer le robot, Reculer le robot, Faire une cartographie
 2. **Observation** :
 - La cartographie doit être identique à celle de l'atterrisage (Ex5)
 - Chaque case du carré de 9x9 est renseignée (Ex3,Ex4)
 - Aucune case en dehors du carré de 9x9 n'est renseignée (Ex2,Ex4,Ex5)



Exemple : Exigences Cartographie

Exemple de scénario

- **Ex0** : La cartographie ne peut être réalisée qu'à l'arrêt.
 - **Ex1** : Une première cartographie est systématiquement effectuée lorsque le robot atterrit.
 - **Ex2** : A l'état initial, aucune case n'est connue.
 - **Ex3** : La cartographie renseigne chaque case par la nature de son terrain.
 - **Ex4** : La cartographie permet de couvrir un carré de 9x9.
 - **Ex5** : La cartographie permet d'agréger les éléments des cases couvertes aux éléments de cartographie déjà connus.
-
- **TEST 5** : Agrégation des cartographies avec un mouvement (Ex5)
 1. **Action** : Faire atterrir le robot, Avancer le robot, Faire une cartographie
 2. **Observation** :
 - La cartographie obtenue est un rectangle de 10x9 (Ex5)
 - La cartographie contient celle de l'atterrissement plus une ligne (Ex5)
 - Chaque case du rectangle de 10x9 est renseignée (Ex3,Ex4)
 - Aucune case en dehors du rectangle de 10x9 n'est renseignée (Ex2,Ex4,Ex5)

Exemple : Exigences Cartographie



Variation des scénarios

- **TEST 1 :** Aucune cartographie en vol
- **TEST 2 :** Une première cartographie est réalisée à l'atterrissement
- **TEST 3 :** Deux cartographies successives donnent le même résultat
- **TEST 4 :** Cartographies identiques avec un mouvement d'aller-retour
- **TEST 5 :** Agrégation des cartographies avec un mouvement
- **TEST 2 :** - Nature du terrain à l'atterrissement ?
- **TEST 3 :** - Nature du terrain ?
 - Orientation du robot ?
 - Cartographie hors atterrissage ?
- **TEST 4 :** - Direction de l'aller-retour ?
 - Chemin plus complexe ?
 - Orientation du robot ?
- **TEST 5 :**
 - Nature du mouvement : variation en X et en Y, sur plusieurs lignes/colonnes, carrées 9x9 strictement adjacents, carrées 9x9 non adjacents
 - Confronter plus de 2 cartographies



A vous de jouer

En vous inspirant de l'exemple précédent

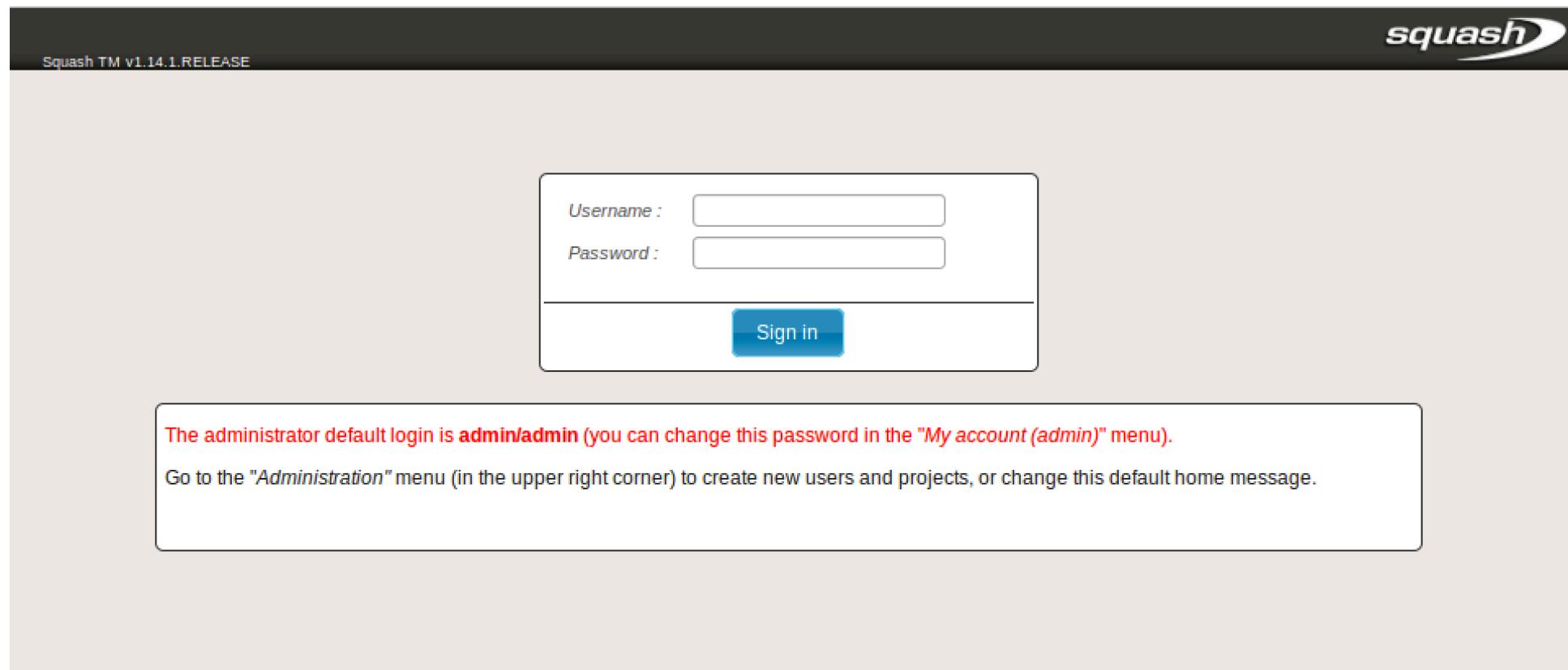
- Objectifs fonctionnels
 - Déplacements mouvement par mouvement
 - Gestion de l'énergie
- Méthode :
 - Identification des caractéristiques fonctionnelles (exigences)
 - Identification des points de contrôle et d'observation
 - Pour chaque exigence, définition des cas (passant et non passant) à couvrir :
 - Définition du contexte de test
 - Définition des données d'entrée
 - Définition des attendus

Gestion des exigences avec Squash

Squash TM - identification

Téléchargement : <http://www.squashtest.org>

Connexion serveur : <http://localhost:8080/squash>



Squash TM – page de démarrage

The administrator default login is **admin/admin** (you can change this password in the "My account (admin)" menu).
Go to the "Administration" menu (in the upper right corner) to create new users and projects, or change this default home message.

HOME Global filter [Administration](#) [My account \(admin\)](#) [Logout](#)

[Message](#) [Dashboard](#)

squash

Squash TM - Exigences



Requirements Workspace

Global filter Administration My account (admin) Logout

<< webRobot

Description

Attachments Upload Attachment Organize

Test Project-1

webRobot

The screenshot shows the Squash TM Requirements Workspace interface. On the left, there's a vertical sidebar with icons for home, document, and settings, and the word "squash". The main workspace has a title bar "Requirements Workspace" and a navigation bar with "Global filter", "Administration", "My account (admin)", and "Logout". Below the navigation is a breadcrumb trail "Test Project-1 / webRobot". The central area contains sections for "Description" and "Attachments", with "Upload Attachment" and "Organize" buttons. A toolbar with various icons is at the top of the workspace area.

Squash TM - Exigences



Requirements Workspace

Global filter Administration My account (admin) Logout

<< webRobot

Description

Attachments Upload Attachment Organize

New folder ...
New requirement...

The screenshot shows the Squash TM Requirements Workspace interface. On the left, there's a vertical sidebar with icons for home, document, and settings, and the word "squash". The main workspace has a toolbar with various buttons. A red box highlights the "New folder ..." and "New requirement..." buttons in the toolbar. To the right, a modal window titled "webRobot" is open, showing fields for "Description" and "Attachments", along with "Upload Attachment" and "Organize" buttons.

Squash TM - Exigences



Requirements Workspace Global filter Administration My account (admin) Logout

webRobot

Add a Requirement

Name : Avancer

Reference : av1

Criticality : 1-Major

Category : Functional

Functional
Non functional
Use case
Business
Test requirement
Undefined
Ergonomic
Performance
Technical
User story
Security

Description : avancer lorsque le terrain le permet.

Add another Add Close

The screenshot shows the Squash TM Requirements Management software interface. The main title is "Squash TM - Exigences". The top navigation bar includes links for "Global filter", "Administration", "My account (admin)", and "Logout". Below the title, there's a decorative hexagonal pattern. The main workspace is titled "Requirements Workspace" and shows a project structure with "Test Project-1" and "webRobot". A modal dialog box is open for "Add a Requirement". It has fields for "Name" (Avancer), "Reference" (av1), "Criticality" (1-Major), and "Category" (Functional). The "Category" dropdown is expanded, showing options like Functional, Non functional, Use case, etc. A description text area contains the text "avancer lorsque le terrain le permet.". At the bottom of the dialog are buttons for "Add another", "Add", and "Close". On the left side, there's a vertical toolbar with icons for home, file, and other functions, and the "squash" logo.

Squash TM - Exigences

Requirements Workspace

Global filter Administration My account (admin) Logout

Av1 - Avancer

Created on : 2016/10/08 18:16 (admin)
Updated on : never

Rename Create a new version Print

Information Attachments

General Informations [ID = 258]

Version nb : 1 [View version history](#)

Reference : Av1

Status : 1-Work in progress

Attributes

Criticality : 1-Major

Category : Functional

Description

Le robot doit pouvoir avancer lorsque le terrain le permet

Coverage indicators

Verification and validation rates perimeter : [Choose a perimeter](#)

Coverage rate : 0 %

Test Cases verifying this requirement

Project Reference Test Case Type

No matching records found

Actions: + -

Project Tree:

- Test Project-1
- webRobot
- Gestion des déplacements
 - Av1 - Avancer

Toolbar:

- Checkmark
- Search (highlighted with a red circle and arrow)
- New
- Open
- Save
- Print
- Delete

Squash TM – Cas de test



Test Cases Workspace

[Global filter](#) [Administration](#) [My account \(admin\)](#) [Logout](#)

webRobot

Dashboard [Refresh](#)

No dashboard was generated for this element. You can generate one by clicking on the button just above.

Description

Attachments [Upload Attachment](#) [Organize](#)

Actions:

- New folder ...
- New test case...

Squash

A screenshot of the Squash TM application interface. On the left, there's a sidebar with icons for Home, Test Cases, and Squash. The main workspace is titled 'Test Cases Workspace' and contains a 'Dashboard' section. The 'Dashboard' section has a message: 'No dashboard was generated for this element. You can generate one by clicking on the button just above.' Below this are fields for 'Description' and 'Attachments', each with a 'Upload Attachment' and 'Organize' button. On the far left of the workspace, there's a toolbar with icons for creating a new folder, creating a new test case, deleting, editing, and running. A red arrow points from the 'New folder ...' and 'New test case...' buttons in the toolbar to the corresponding items in the sidebar menu. The entire interface has a light beige background with dark grey header and footer bars.

Squash TM – Cas de test



Test Cases Workspace

Global filter Administration My account (admin) Logout

webRobot

Ajouter un cas de test

Nom : Test de déplacement

Référence : tf_av1

Description : Tester le déplacement vers le haut

Ajouter un autre Ajouter Fermer

Test Project-1 webRobot

refresh

Organize

The screenshot shows the Squash TM Test Cases Workspace interface. A modal dialog box titled "Ajouter un cas de test" is open. It contains fields for "Nom" (Name) with the value "Test de déplacement", "Référence" (Reference) with the value "tf_av1", and "Description" (Description) with the value "Tester le déplacement vers le haut". The "Ajouter" (Add) button at the bottom right of the modal is highlighted with a red border. The background workspace shows a project named "Test Project-1" containing a "webRobot" test case. The top navigation bar includes links for "Global filter", "Administration", "My account (admin)", and "Logout". The title bar says "webRobot". The left sidebar has icons for home, file, and squash. The bottom right corner of the screen has a decorative hexagonal grid pattern.

Squash TM – Cas de test



Test Cases Workspace

Global filter Administration My account (admin) Logout

<< tf_av1 - test de déplacement haut

Created on : 2016/10/08 18:26 (admin)
Updated on : 2016/10/08 18:44 (admin)

Rename Print

Information Script Parameters Attachments Executions

Description [ID = 240]

Reference :

Description :

Status : Confirm Cancel

Auto. script : 3-Approved 4-Obsolete 5-To be updated

Attributes

Weight : Confirm Cancel auto

Nature : Confirm Cancel

Type : Confirm Cancel

Undefined
Compliance

Prerequisite

Correction Evolution Regression

Requirement

End-to-end Partner Test case

Project

1 webRobot

Show 50 en

Choisir Annuler

Choisir un test automatisé

1-Work in progress Confirm Cancel

1-Work in progress
2-Under review
3-Approved
4-Obsolete
5-To be updated

4-Low Confirm Cancel auto

Functional Confirm Cancel

Undefined Confirm Cancel

Compliance

Correction Evolution Regression

End-to-end Partner Test case

Choisir Annuler

Squash TM – Cas de test



Test Cases Workspace

Global filter Administration My account (admin) Logout

<< tf_av1 - test de déplacement haut

Created on : 2016/10/08 18:26 (admin)
Updated on : 2016/10/08 18:44 (admin)

Rename Print

Information Script Parameters Attachments Executions

Parameters

#	Name	Description	Source Test Case
1	Nature_du_terrain	(Click to edit...)	
2	Valeur_graine	(Click to edit...)	

Datasets

#	Dataset	Nature_du_terrain	Valeur_graine
1	Boue	Boue	3
2	Infranchissable	Infranchissable	10
3	Roche	Roche	1
4	Sable	Sable	2
5	Terre	Terre	9

Show 50 entries: << < 1 > >>

The screenshot shows the Squash TM application interface. On the left, there's a sidebar with icons for home, file, and settings, and a 'squash' logo. The main workspace is titled 'Test Cases Workspace'. A red arrow points to the gear icon in the toolbar at the top left of the workspace. The central area displays a test case named 'tf_av1 - test de déplacement haut'. It shows details like creation and update times, and tabs for Information, Script, Parameters, Attachments, and Executions. The 'Parameters' tab is active, showing two entries: 'Nature_du_terrain' and 'Valeur_graine'. Below it, the 'Datasets' tab is active, displaying five rows of data with columns for Dataset name, Nature_du_terrain, and Valeur_graine.

Squash TM – Campagne de tests

Campaign Workspace

Global filter Administration My account (admin) Logout

<< webRobot

Description

Attachments Upload Attachment Organize

New folder ...

New Campaign...

Add Iteration...

The screenshot displays the 'Campaign Workspace' interface of the Squash TM application. At the top left, there's a vertical sidebar with icons for home, document, and settings. The main workspace has a title 'Campaign Workspace' and a toolbar with various icons. A red arrow points from the top-left towards a dropdown menu that appears when one of the toolbar icons is clicked. This menu contains three options: 'New folder ...', 'New Campaign...', and 'Add Iteration...'. The 'New folder ...' option is highlighted with a red box. The background features a purple header bar with the title 'webRobot' and sections for 'Description' and 'Attachments'. The top right of the screen shows navigation links for 'Global filter', 'Administration', 'My account (admin)', and 'Logout'.

Squash TM – Campagne de tests

Campaign Workspace

- + New
- Delete
- Edit
- Reorder
- Search
- Trash

- ★ Test Project-1
- ★ webRobot
- Déplacement
 - Validation
 - 1 - Itération 1

<< 1 - Itération 1

Created on : 2016/10/09 11:10 (admin)
Updated on : 2016/10/09 11:10 (admin)

Run automated tests Test suites Rename

Dashboard Information Execution Plan Attachments

Filter Reorder Test suites Status Assign Add Remove from execution plan

#	Location	Mode	Ref.	Test	Wt.	Dataset	Test suite	Status	% success	User	Last execution on	
1	webRobot		tf_av1	▶ test de déplacement haut	L	Boue	-	ready	0	-	-	▶ -
2	webRobot		tf_av1	▶ test de déplacement haut	L	Infranchissable	-	ready	0	-	-	▶ -
3	webRobot		tf_av1	▶ test de déplacement haut	L	Roche	-	ready	0	-	-	▶ -
4	webRobot		tf_av1	▶ test de déplacement haut	L	Sable	-	ready	0	-	-	▶ -
5	webRobot		tf_av1	▶ test de déplacement haut	L	Terre	-	ready	0	-	-	▶ -

Show 50 entries : << < 1 > >>

femto-st
SCIENCES &
TECHNOLOGIES

MinIVOL 2016 

Squash TM – Rapport



Report Workspace

Report : Requirement Coverage By Tests

Report Criteria

Dashboard for requirement coverage **Requirement list by Project**

Export as pdf **Export**

Projects

TOTAL		Global				By criticality			
		Total	Critical	Major	Minor	Undefined			
In progress	Coverage rate	2 50 % (1 / 2)	0 100 % (0 / 0)	2 50 % (1 / 2)	0 100 % (0 / 0)	0 100 % (0 / 0)			
For Approval	Coverage rate	0 100 % (0 / 0)	0 100 % (0 / 0)	0 100 % (0 / 0)	0 100 % (0 / 0)	0 100 % (0 / 0)			
Approved	Coverage rate	0 100 % (0 / 0)	0 100 % (0 / 0)	0 100 % (0 / 0)	0 100 % (0 / 0)	0 100 % (0 / 0)			
Obsolete	Coverage rate	0 100 % (0 / 0)	0 100 % (0 / 0)	0 100 % (0 / 0)	0 100 % (0 / 0)	0 100 % (0 / 0)			
Total Requirements		2	0	2	0	0			
Coverage rate (by at least one TC)		50 % (1 / 2)	100 % (0 / 0)	50 % (1 / 2)	100 % (0 / 0)	100 % (0 / 0)			
webRobot									
In progress	Coverage rate	2 50 % (1 / 2)	0 100 % (0 / 0)	2 50 % (1 / 2)	0 100 % (0 / 0)	0 100 % (0 / 0)			
For Approval	Coverage rate	0 100 % (0 / 0)	0 100 % (0 / 0)	0 100 % (0 / 0)	0 100 % (0 / 0)	0 100 % (0 / 0)			
Approved	Coverage rate	0 100 % (0 / 0)	0 100 % (0 / 0)	0 100 % (0 / 0)	0 100 % (0 / 0)	0 100 % (0 / 0)			
Obsolete	Coverage rate	0 100 % (0 / 0)	0 100 % (0 / 0)	0 100 % (0 / 0)	0 100 % (0 / 0)	0 100 % (0 / 0)			
Total Requirements		2	0	2	0	0			
Coverage rate (by at least one TC)		50 % (1 / 2)	100 % (0 / 0)	50 % (1 / 2)	100 % (0 / 0)	100 % (0 / 0)			

Execution Phase

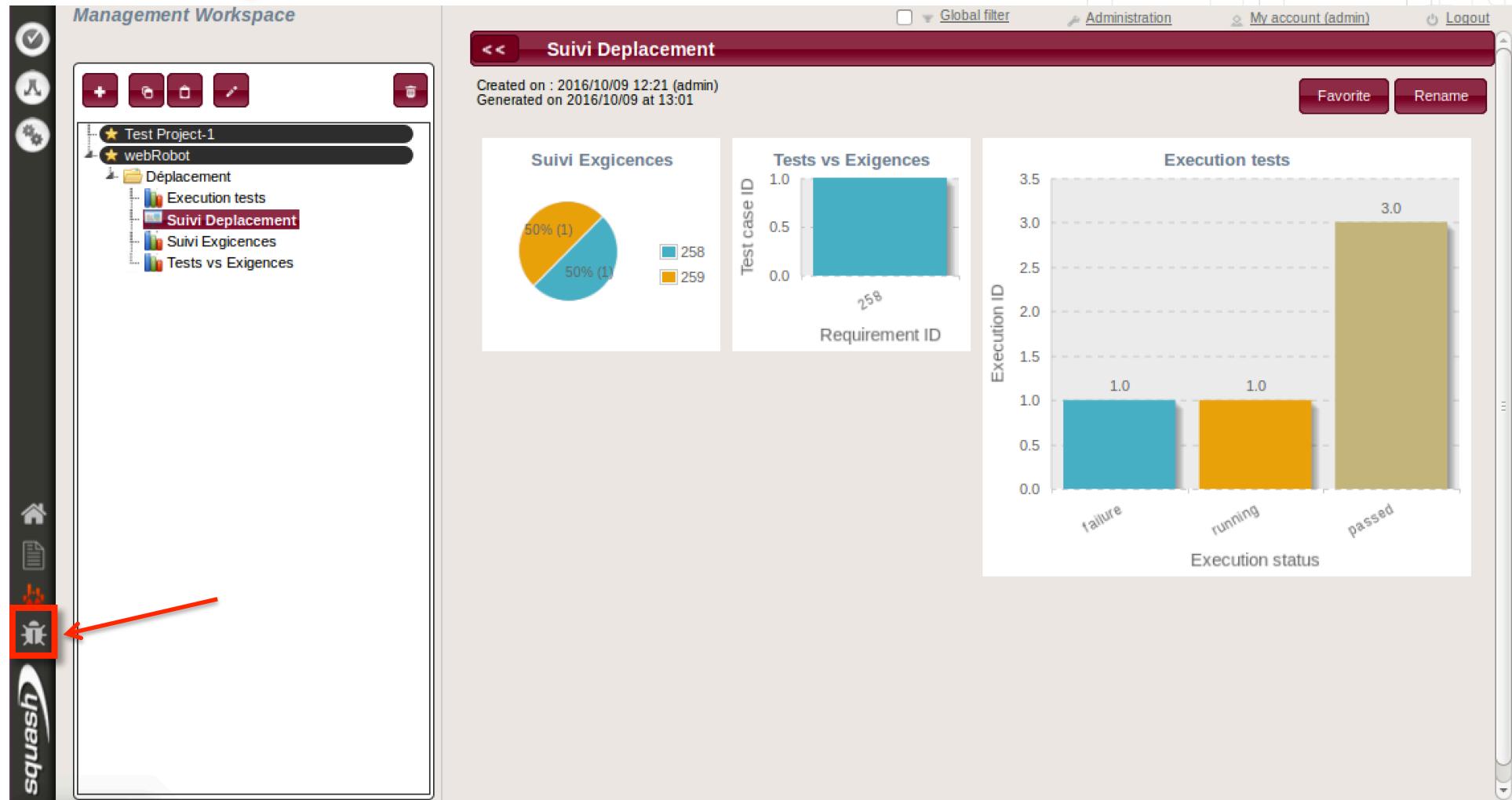
- Execution Followup :** Follow up executions
- Qualitative followup :** Generates a dashboard for test results and corresponding requirements

Preparation Phase

- Requirement Coverage By Tests :** Generates a report about functional coverage of requirements by tests.
- Requirements Report (editable format) :** Generates a list of requirement descriptions
- Test cases report (editable format) :** Generates a list of test case descriptions from selected test cases

SCIENTES & TECHNOLOGIES

Squash TM – Tableau de suivi



Configure Job Squash-TA



Jenkins

Jenkins > Tous >

[New Job](#)

[People](#)

[Build History](#)

[Manage Jenkins](#)

[Credentials](#)

Build Queue
No builds in the queue.

Build Executor Status

#	Status
1	Idle
2	Idle

Job name

Build a free-style software project
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

Build a maven2/3 project
Build a maven 2/3 project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

Build multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Monitor an external job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See [the documentation for more details](#).

Copy existing Job
Copy from templateSquashTAExecutionMavenJob

[Help us localize this page](#)

Page generated: Oct 12, 2016 8:28:32 PM [REST API](#) [Jenkins ver. 1.532.3](#)

Configure Job Squash-TA



Jenkins > FabTAJob > configuration

Execute concurrent builds if necessary

Advanced Project Options

Source Code Management

Advanced...

CVS

CVS Projectset

None

Subversion

Modules Repository URL: `svn:/localhost/webRobot/RobotSquash`

Unable to access `http://subversion.example.com/svn/repos/automation_sample/trunk`: svn: E175002: OPTIONS /svn/repos/automation_sample/trunk failed (show details)
(Maybe you need to [enter credential](#)?)

Local module directory (optional):

Repository depth option: infinity

Ignore externals option:

Add more locations...

Mettre le bon svn

Save Apply

Exercice



- Ecrire les exigences liées aux déplacements dans squash TM
- Ouvrir dans l'environnement le canevas proposé dans le svn robot
- Lancer les 2 tests dans votre environnement
- Développer les scripts « TA » correspondant aux tests associés aux exigences en utilisant le canevas proposé :
 - Les fichiers Junit sont à mettre dans :
`<projet>/src/squashTA/resources/selenium/java`
 - Les fichiers script TA sont à mettre dans :
`<projet>/src/squashTA/tests`
- Associer dans squash TM les scripts TA correspondant
- Lancer une campagne / itération pour valider
- Regarder le rapport lié à l'itération

Exercice



- Ecrire les exigences liées aux déplacements dans squash TM
- Ouvrir dans l'environnement le canevas proposé dans le svn robot
- Lancer les 2 tests dans votre environnement
- Développer les scripts « TA » correspondant aux tests associés aux exigences en utilisant le canevas proposé :
 - Les fichiers Junit sont à mettre dans :
`<projet>/src/squashTA/resources/selenium/java`
 - Les fichiers script TA sont à mettre dans :
`<projet>/src/squashTA/tests`
- Associer dans squash TM les scripts TA correspondant
- Lancer une campagne / itération pour valider
- Regarder le rapport lié à l'itération