Noël NOVELLI ; Université d'Aix-Marseille; LIF et Département d'Informatique – Case 901 ; 163 avenue de Luminy – 13 288 MARSEILLE cedex 9

Génie Logiciel

- Sommaire Développement et recette (interne)
  - La phase de développement :
    - *Gestion de configuration*
    - *Ateliers de développement*
    - *Normes de développement*
    - *Documentation applicative*
  - La phase de recette interne :
    - *Revue de code et tests unitaires*
    - *Tests fonctionnels et de non-régression*
    - *Tests techniques et de performance*
    - *Dossier de tests*

Ce support n'est qu'une adaptation des supports rédigés par **Isabelle VALEMBOIS** (Thales-services)

DEVELOPPEMENT

- Les **leviers** de productivité du développement :
  - Le prototype **technique**
  - L'application des **best-practices**
  - La gestion de **configuration**
  - L'utilisation d'un **atelier de développement**
  - Les **frameworks et autres composants** réutilisables
  - Le respect des normes
  - **La documentation du code**
  - La capitalisation

BEST PRACTICES (1/3)

- Best practices
  - Gestion de la complexité technologique. **Sous-estimer** cet aspect, c'est résoudre les **problèmes techniques au fil des développements avec des risques importants de remise en cause des réalisations** (conséquence : décalage de planning, démotivation des équipes…)
    - *Démarrer le plus tôt possible la phase d'architecture*
    - *Ne pas démarrer les développements tant que les normes de développement ne sont pas disponibles et l'équipe formée*
    - **Identifier et gérer les risques liées à l'utilisation de nouvelles technologies**
    - *Identifier* **les besoins en expertise technique**
    - *Isoler les solutions techniques dans des frameworks techniques*
    - *Valider* **les choix techniques via un prototype technique**
    - *Intégrer et faire respecter les normes de développement*
    - *Capitaliser sur les connaissances techniques*
    - *Capitaliser sur les best-practices et normes de développement*

BEST PRACTICES (2/3)

- Best practices (suite)
  - Organisation de l'équipe. Les projets à base de **nouvelles technologies** présentent un certain nombre de spécificités : équipe surdimensionnée ou sous dimensionnée en fonction des phases, insuffisance de ressources expertes…
    - *Respecter les phases de validation*
    - *Affecter les tâches en fonction du niveau d'expertise des ressources*
    - *Effectuer le suivi de planning de manière régulière*
    - *Spécialiser les équipes tout en laissant la possibilité à chacun de changer de spécialité*
    - *Favoriser des binômes de travail mélangeant des ressources juniors avec des ressources seniors*
    - *Profiter des creux d'activités pour former les équipes*
    - *Anticiper et éviter la surcharge de travail*

BEST PRACTICES (3/3)

- Best practices (fin)
  - **Communication**. La nécessité de **communiquer sur un projet est souvent sous estimée, qu'il s'agisse de communication au sein de l'équipe ou entre l'équipe et le client**
    - *Communiquer sur les enjeux et sur la stratégie d'un projet*
    - *Organiser* **régulièrement des réunions de travail** *au sein de l'équipe pour échanger sur les bonnes pratiques*
    - *Affecter* **des personnes différentes pour le développement** *d'un composant et* **pour la recette de celui-ci**
    - *Favoriser des livraisons au fil de l'eau pour faciliter le feed-back et favoriser l'implication du client*
    - *Expliquer l'intérêt et favoriser la remontée des alertes*
    - *Laisser* **s'exprimer les goûts et atouts de chacun**
    - *Profiter du bilan de projet pour faire le point sur les axes d'amélioration de la phase de réalisation*

GESTION DE CONFIGURATION (1/6)

- Objectifs
  - **Stocker les différentes versions d'un logiciel**, d'un document ou plus généralement d'un composant électronique
  - Permettre de revenir à une version précédente
  - Permettre la modification concurrente en gérant les conflits
- Extensions fonctionnelles
  - Gestion du **développement parallèle** (ramification, dérivation)
  - Gestion du **développement multi-sites**

GESTION DE CONFIGURATION (2/6)

- Normes
  - SCC (Source Change Control) est le protocole autrefois utilisé par Microsoft pour interfacer les outils de gestion de configuration aux outils de développement.
  - WebDAV (Distributed Authoring an Versioning) est une nouvelle norme générique du consortium W3C, à laquelle travaillent IBM, Microsoft, Oracle, Rational, Novel, Xerox… Il permet d'intégrer dans le protocole http et dans les serveurs web, les opérations de gestion de versions et de configuration.

GESTION DE CONFIGURATION (3/6)

- Principaux outils du marché
  - CVS (OpenSource)
  - PVCS (Merant) respecte les normes ISO 9000, SPICE et SEI CMM
  - ClearCase et ClearQuest (Rational)
  - Source Safe (Microsoft)
  - True Software (Mc Cabe & Associates)
  - Continuus (Telelogic)
  - Starteam (Starbase)
  - …

GESTION DE CONFIGURATION (4/6)

- CVS
  - Principes :
    - **Repository** : *répertoire (local ou sur un serveur distant) où CVS stocke l'ensemble de ses fichiers*
    - *CVS ne **stocke pas les différentes versions** de chaque fichier mais seulement l'historique des modifications qui y ont été effectuées*
    - *Lorsqu'un utilisateur souhaite travailler sur un fichier, **CVS lui fournit une copie** du fichier original. L'utilisateur peut alors travailler en local*
    - *Créer un repository :* **cvs init …**
    - *Importer un projet :* **cvs import …**
    - *Récupérer des sources depuis le repository :* **cvs checkout …**
    - *Ajouter un nouveau source dans le repository :* **cvs add …**
    - *Supprimer un source dans le repository :* **cvs remove …**
    - *Enregistrer des changements dans le repository :* **cvs commit …**

GESTION DE CONFIGURATION (5/6)

- CVS (suite)
  - Principes :
    - **Révision** *: une révision est la version d'un fichier. Toutes les révisions d'un fichier sont stockées par CVS. CVS gère **automatiquement** les numéros de révision, en partant de 1.1*
    - **Tag** *: on peut donner un nom symbolique à une révision d'un ensemble de fichiers. Les fichiers appartenant à une version d'une application peuvent avoir des numéros de révision individuels différents.*
    - *Assigner un numéro de révision :* **cvs commit …**
    - *Assigner un tag :* **cvs tag …**

GESTION DE CONFIGURATION (6/6)

- CVS (fin)
  - Principes :
    - *Un fichier peut être **modifié simultanément par deux développeurs**. Si un autre développeur effectue un commit, il est possible de fusionner localement les changements*
    - *Si un **conflit apparaît**, il faut résoudre le conflit avant de pouvoir réaliser le commit*
    - ***Fusionner les changements** d'un autre utilisateur :* **cvs update …**
    - *CVS permet également d'envoyer des notifications si plusieurs utilisateurs travaillent sur le même source*
    - *Ajouter une personne à une liste de notification :* **cvs watch …**
    - *CVS offre également un mécanisme de substitution pour remplacer des mots-clés (***$Author$**, **$Date$**, **$Header$**) par leur valeur*
    - *Enfin CVS ne permet pas seulement de travailler sur des fichiers textes mais aussi sur des fichiers binaires*

ATELIERS DE DEVELOPPEMENT (1/8)

- Objectifs
  - **Faciliter le travail du développeur et améliorer sa productivité**
  - Simplifier la complexité des architectures n-tier :
    - *Outils reposant sur la **programmation visuelle***
    - ***Plusieurs éditeurs**, chacun étant adapté à un type de composants logiciels (html, xml, servlet, jsp, ejb, applet…)*
    - *Outils de paramétrage des composants et des applications*
  - Homogénéiser la production d'applications :
    - *Interfaces (ou intégration) avec des outils de conception (XDE) facilitant ainsi la génération automatique de code*
    - *Interfaces (ou intégration) avec des outils de gestion de configuration facilitant ainsi le développement en équipe*
    - *Interfaces (ou intégration) avec des outils de tests favorisant la fiabilité des composants développés*

ATELIERS DE DEVELOPPEMENT (2/8)

- Objectifs (suite)
  - Capitaliser sur les best-practices en terme de développement :
    - ***Assistants*** *de développements (wizards)*
    - ***Aide*** *en ligne*
    - *Outil de recherche*
    - *Mécanisme de* ***complétion***
  - Favoriser la prise en compte des différents environnements d'exécution :
    - *Outils de déploiement sur différentes plates-formes*
    - *Outils de mapping avec les bases de données relationnelles*
  - Fiabiliser les composants développés :
    - *Fournir des environnements de tests*
    - *Fournir des outils de débug*
  - …

ATELIERS DE DEVELOPPEMENT (3/8)

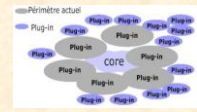
- Principaux outils du marché pour J2EE
  - Eclipse (OpenSource)
  - NetBeans (OpenSource développé par Sun)
  - JBuilder (Borland)
  - WSAD (IBM)
  - Studio Professional (WebGain)
  - Versata Studio (Versata)
  - JDeveloper (Oracle)
  - Forte (Sun)
- Outil DotNet
  - Visual Studio .Net (Microsoft)

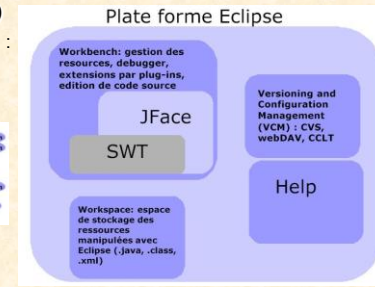
ATELIERS DE DEVELOPPEMENT (4/8)

- Eclipse
  - Eclipse est un environnement de développement
  - C'est une souche commune sur laquelle peuvent se greffer de nombreux plugins : Eclipse est en fait un IDE à la carte
  - Il est notamment le socle de l'offre IBM : WSAD
  - Il est entièrement écrit en java et sous licence free. Il se pose de ce fait comme un concurrent direct du produit développé par Sun : NetBeans
  - Le produit proposé au téléchargement par défaut (le SDK Eclipse) contient la couche générique ainsi que tous les plugins par défaut pour le développement en Java, un client CVS, JUnit pour les tests unitaires et Ant le langage de scripts multi plates-formes
  - Eclipse renforce la position de J2EE vis à vis de .Net puisqu'il peut tout aussi bien devenir un IDE C#, un atelier de génie logiciel UML (XDE), … ou tout cela en même temps
  - Eclipse offre de plus tous les outils pour développer de nouveaux plugins via notamment l'API SWT (Standard Widget Toolkit) qui remplace l'API Swing

ATELIERS DE DEVELOPPEMENT (5/8)

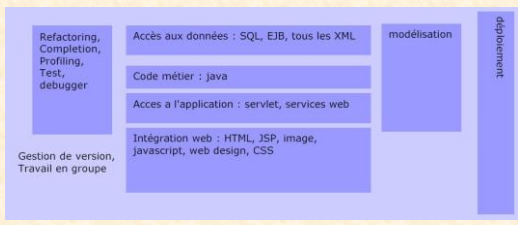
- Eclipse (suite)
  - Architecture :





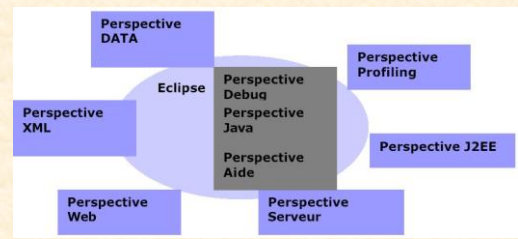
ATELIERS DE DEVELOPPEMENT (6/8)

- Eclipse (suite)
  - Besoins sur un projet J2EE :



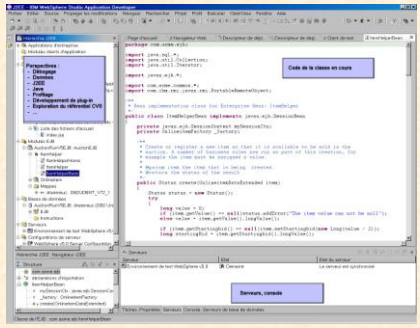
ATELIERS DE DEVELOPPEMENT (7/8)

- Eclipse (suite)
  - Les plugins apportés par WSAD à Eclipse :



ATELIERS DE DEVELOPPEMENT (8/8)

- Eclipse (fin)
  - L'interface WSAD :



NORMES DE DEVELOPPEMENT (1/6)

- Normes de développement
  - Cibles des normes de développement :
    - *Pages HTML*
    - *Feuilles de style CSS*
    - *Programmes Javascript*
    - *Composants XML, XSL…*
    - *Composants java : servlets, JSP, EJB…*
    - *Requêtes SQL*
    - *Composants ASP*
    - *…*
  - **Tous les composants sont susceptibles de faire l'objet de normes**.
  - Il existe des normes **reconnues par les communautés** de développeurs (normes SUN par exemple pour J2EE)

NORMES DE DEVELOPPEMENT (2/6)

- Normes de conception technique
  - Packaging :
    - *contenu et granularité des fichiers war*
    - *contenu et granularité des fichiers jar*
    - *contenu et granularité des fichiers ear*
    - *contenu et granularité des fichiers rar*
  - Hiérarchisation des composants :
    - *critères de **regroupement** par package ou répertoire*
    - *granularité des packages et répertoires*
  - Nomenclature des composants et des éléments d'un composant :
    - ***classes, interfaces, fichier HTML, javascript…***
    - ***attributs, méthodes…***
    - ***respect de la casse***

NORMES DE DEVELOPPEMENT (3/6)

- Normes de conception technique (fin)
  - Règles de portabilité et conditions d'utilisation des composants ou d'éléments :
    - *applets, plug-ins, cookies, balises html spécifiques à un navigateur*
    - *EJB Session stateless ou statefull, d'EJB Entity BMP ou CMP*
    - *triggers et procédures stockées*
  - Règles de réutilisabilité des composants :
    - *modèles de conception (ex : singleton, façade, factory…)*
    - *frameworks (ex : Struts, Log4J…)*
    - *librairies (ex : taglibs …)*
  - Version des normes à suivre (ejb 2.0, servlet 1.2, jdk 1.3…)

NORMES DE DEVELOPPEMENT (4/6)

- Normes de construction
  - Règles de lisibilité :
    - ***documentation** (utilisation de la javadoc, éléments à documenter, format de la documentation…)*
    - ***structure et indentation** du code. Les normes de construction peuvent le cas échéant proposer en annexes des squelettes de code*
    - *règles de **factorisation** (ex : package regroupant toutes les fonctions communes)*
    - *nombre **maximum de lignes** d'un code source*
  - Règles liées à la performance des applications :
    - ***optimisation** (ex : pool, cache, StringBuffer…)*
    - *gestion des sessions*
    - *gestion des threads*

NORMES DE DEVELOPPEMENT (5/6)

- Normes de construction (suite)
  - Programmation objet :
    - *visibilité des classes, des attributs, des méthodes*
    - *règles de dérivation*
  - Règles liées à l'évolutivité :
    - *découpage en couches (ex : pas d'accès direct entre un ejb et une page jsp)*
    - *échanges inter-applications (protocoles…)*
    - *chaînes de caractères (ex : pas de chaînes en dur, stockage des chaînes dans des fichiers properties)*
    - *internationalisation et localisation*
  - Règles liées à la fiabilité des applications :
    - *règles de gestion des exceptions et des erreurs*
    - *règles de gestion des transactions*

NORMES DE DEVELOPPEMENT (6/6)

- Normes de construction (fin)
  - Règles liées à la maintenabilité des applications :
    - *gestion des **traces***
    - *gestion des **tests unitaires** (ex : codage de la méthode main ou écriture de classes de tests…)*
  - Normes de représentation de certaines données récurrentes :
    - *dates et heures*
    - *montants monétaires*
    - *noms et prénoms*
    - *adresses, codes postaux, pays*
    - *n°SIRET, n°SIREN, n°INSEE*
    - *…*
  - …

TESTS (1/5)

- Objectifs des tests
  - Diminuer les **coûts dus à la correction** des erreurs
  - **Fiabiliser** le logiciel livré
  - **Accroître** la satisfaction client
  - Et pour cela : **formaliser et tracer**
- Origines des erreurs
  - **Définition erronée** des besoins utilisateurs
  - **Erreurs d'interprétation** des besoins utilisateurs
  - **Erreurs de conception**
  - **Erreurs de programmation**
  - **Insuffisance ou inexistence** des tests unitaires
  - **Insuffisance ou inexistence** des tests fonctionnels
  - Utilisation du logiciel dans un **contexte non prévu**…

TESTS (2/5)

- Best-practice n°1 – Le test est un état d'esprit qui implique de la rigueur
  - 2 constats :
    - *Il est impossible de programmer sans bugs*
    - *La majorité des logiciels ne sont pas suffisamment testés*
  - Le succès d'une campagne de tests dépend beaucoup plus de l'attitude psychologique des développeurs et des testeurs que des techniques mises en œuvre
  - Les tests sont des procédures formelles qui doivent être préparées, documentées et suivies

TESTS (3/5)

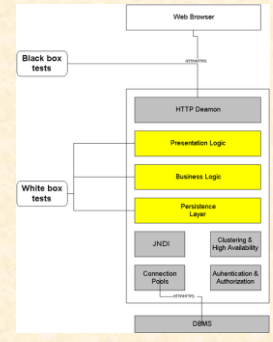
- Best-practice n°2 - Respecter le cycle de vie de l'activité de tests
  - A la fin de la phase d'**analyse** :
    - *définition du plan de tests*
  - A la fin de la phase de **conception** :
    - *définition des tests unitaires et d'intégration*
  - A la fin de la **réalisation d'un composant** :
    - *exécution des tests unitaires*
  - A la fin de la phase de **réalisation** :
    - *exécution des tests d'intégration*
- Le test doit avoir une place centrale dans le processus de développement

TESTS (4/5)

- Les principaux types d'outils de tests
  - **Outils de gestion de tests** qui permettent l'édition de plans de tests et l'intégration avec d'autres outils de tests
  - **Outils basés sur les spécifications** ou exigences : les exigences doivent alors être formulées dans un langage formel. Ce type d'outil n'est adapté qu'à des logiciels ayant un périmètre restreint et/ou de fortes contraintes de fiabilité
  - **Outils de tests unitaires et de revue de code**, basés sur le langage
  - **Outils de tests d'intégration**, basés sur des robots de tests d'IHM : des séquences de tests ainsi qu'une description des résultats attendus sont définis par l'utilisateur. Le robot peut alors enregistrer ces informations, rejouer les tests et fournir un bilan de cette exécution.
  - **Outils de tests de performance**, qui permettent de mesurer les performances d'un système dans ses conditions d'exploitation en simulant la charge via des stimuli envoyés sur le réseau ou dans l'environnement d'exécution

TESTS (5/5)

- 2 types de fonctionnement des outils de tests
  - boîtes blanches
  - boîtes noires



GESTION DES TESTS (1/2)

- Fonctionnalités des outils de gestion des tests :
  - Conception des cas de tests, avec définition d'étapes clés et de résultats attendus pour chaque cas de test
  - Interface avec un robot de tests qui permettra de créer les scripts de tests automatisés
  - La planification et l'organisation de l'exécution des tests automatisés et manuels
  - La saisie et le suivi d'anomalies comprenant :
    - *l'association des anomalies avec les tests*
    - *le suivi des corrections apportées (date, intervenant, description de la solution apportée)*
    - *le résumé statistique des activités de test et de débug*
  - Ces outils sont généralement un point de contrôle unique pour orchestrer toutes les phases de test

GESTION DES TESTS (2/2)

- Les principaux outils de gestion des tests
  - TestDirector (Mercury)
  - QADirector (Compuware)
  - TestManager (Rational)

REVUE DE CODE (1/3)

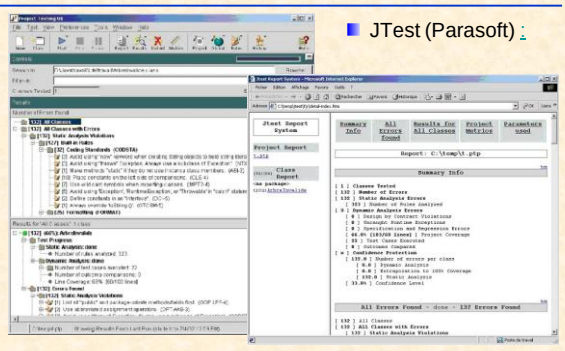
- Fonctionnalités des outils de revue de code :
  - Paramétrage des normes de l'entreprise
  - Contrôle du respect des normes de codage (standard et personnalisé)
  - Détection des exceptions non traitées
  - Contrôle à partir d'assertions (jdk 1.4)
  - Identification de code mort
  - Identification d'erreurs de régression
  - Mise à disposition d'indicateurs de couverture de tests
  - Mise à disposition d'indicateurs de complexité de l'application
  - Mise à disposition d'indicateurs de couverture de documentation

Ce type d'outil apporte de surcroît une dimension pédagogique

REVUE DE CODE (2/3)

- Principaux outils de revue de code
  - JTest (Parasoft) dédié au langage Java
  - JDepend (OpenSource)
  - PMD (OpenSource)
  - FindBugs (OpenSource)

REVUE DE CODE (3/3)



- JTest (Parasoft) :

TESTS UNITAIRES (1/8)

- Fonctionnalités des outils de tests unitaires :
  - Contrôle la bonne gestion des exceptions
  - Contrôle à partir de classes de tests
  - Contrôle à partir d'assertions (jdk 1.4)
  - Identification d'erreurs de régression
- Les principaux outils de tests unitaires :
  - JUnit (OpenSource) dédié au langage Java
  - NUnit (OpenSource) dédié à .Net
  - CUnit (OpenSource) dédié au langage C++
  - DUnit (OpenSource) dédié au langage Delphi
  - MockObjects (OpenSource)

TESTS UNITAIRES (2/8)

- Généralités sur les tests unitaires
  - Le test unitaire est celui de plus bas niveau fonctionnel car il concerne les méthodes de chacune des classes mises en jeu.
  - Il s'élabore à partir des documents de conception détaillée (le recensement des classes à tester est directement issu du diagramme de classes)
  - Il s'agit de tester le comportement attendu d'une méthode donnée en terme de valeurs retournées, de valorisation des variables utilisées, les scénarios permettant de fixer certaines valeurs d'entrée particulières
  - Trois approches :
    - *Ascendante : test d'une unité indépendamment de l'unité appelante, en partant du haut de la représentation hiérarchique*
    - *Descendante : approche inverse (évite l'utilisation de bouchon)*
    - *Combinaison des deux approches précédentes*

TESTS UNITAIRES (3/8)

- JUnit est utilisé dans un certain nombre de projets qui propose d'étendre les fonctionnalités de JUnit :
  - Cactus : framework OpenSource de tests pour composants serveur
  - JunitReport : tâche Ant générant un rapport des tests effectués avec JUnit sous Ant
  - JWebUnit : framework OpenSource de tests pour applications web
  - StrutsTestCase : extension de JUnit pour les tests d'application utilisant Struts 1.0.2 et 1.1
  - XMLUnit : extension de JUnit pour les tests sur des documents XML

TESTS UNITAIRES (4/8)

- JUnit
  - JUnit est un framework OpenSource pour réaliser des tests unitaires sur du code Java. Son but est d'automatiser les tests unitaires.
  - Ceux-ci sont exprimés dans des classes sous la forme de cas de tests avec leurs résultats attendus. JUnit exécute ces tests et les comparent avec ces résultats.
  - Avec JUnit, l'unité de test est une classe dédiée qui regroupe des cas de tests. Ces cas de tests exécutent les tâches suivantes :
    - *création d'une instance de la classe et de tout autre objet nécessaire aux tests*
    - *appel de la méthode à tester avec les paramètres du cas de test*
    - *comparaison du résultat obtenu avec le résultat attendu : en cas d'échec, une exception est levée*
  - Eclipse 2.1 intègre JUnit directement dans l'IDE.
  - 2 modes d'exécution : en ligne ou via une interface graphique

TESTS UNITAIRES (5/8)

- Best-practice n°3 – La bonne mesure
  - Il ne faut pas chercher à tester systématiquement toutes les méthodes d'une classe (par exemple : les tests d'accesseurs et de mutateurs n'apportent pas beaucoup de plus-value)
  - Il faut plutôt chercher à tester les services rendus par une classe. Ainsi, si elle est complexe, il peut être nécessaire de tester plusieurs fois une même méthode avec des arguments différents.
  - Il arrive également qu'un service particulier de l'application repose sur les interactions d'un ensemble de "petites" classes. JUnit permet de tester un groupe de classes.
  - Attention toutefois à utiliser cette dernière tactique avec discernement. En effet, plus le nombre de classes mises en jeu augmente, moins les tests sont "unitaires", et donc plus le temps passé à investiguer d'éventuels problèmes augmente.

TESTS UNITAIRES (6/8)

- Mise en place de tests avec JUnit sous Eclipse
  - Pour utiliser JUnit, ajouter tout d'abord le fichier junit.jar dans le classpath du projet (sélectionner les *propriétés* du projet puis dans l'onglet *librairies*, cliquer sur *Add External Jar* sous Eclipse)
  - Créer une classe qui va contenir les cas de test (créer une nouvelle entité de type *java/JUnit/TestCase* sous Eclipse)
  - Compléter ensuite la classe générée selon les besoins. Par exemple ajouter un attribut qui va contenir une instance de la classe à tester, instancier cette classe dans la méthode *setUp()* et libérer l'instance dans la méthode *tearDown()*.
  - Ajouter enfin les traitements nécessaires dans les méthodes *testXXX()* en utilisant l'API de JUnit (JUnit utilise l'instrospection pour exécuter les méthodes commençant par test)
  - Pour exécuter les tests, il faut exécuter la classe en tant que *JUnit Test*. Eclipse exécute les tests et affiche le résultat dans une vue dédiée.

TESTS UNITAIRES (7/8)

- Techniques de tests avec JUnit
  - Créer une classe de test par classe à tester.
  - Placer vos classes de tests dans le même package que vos classes à tester.
  - Par contre, ne les stocker pas dans le même répertoire, cela vous évitera de vous encombrer de ces classes lors du déploiement.
  - Pour tester une classe dépendantes d'autres classes, utiliser :
    - *la technique du **self shunt** : la classe de test joue le rôle de bouchon*
    - *la technique du **mock object** : une nouvelle classe spécifique est écrite pour jouer le rôle du bouchon*
  - Pour tester plusieurs classes :
    - *Utiliser l'objet **TestSuite***

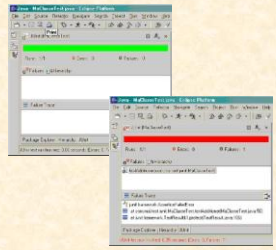
```
package org.xpday.junit;

import junit.framework.*;

public class AllTests {
  public static Test suite() {
    TestSuite suite = new TestSuite();
    suite.addTestSuite(MoneyTest.class);
    suite.addTestSuite(OtherTest.class);
    return suite;
  }
}
```

TESTS UNITAIRES (8/8)

- Mise en place de tests avec JUnit sous Eclipse (fin)
  - Le code de la classe de test

```
package com.moi.test.junit;
import junit.framework.TestCase;

public class MaClasseTest extends TestCase {
  private MaClasse maClasse = null;

  public MaClasseTest(String arg0) {
      super(arg0);
  }

  public static void main(String[] args) {
      junit.swingui.TestRunner.run(MaClasseTest.class);
  }

  protected void setUp() throws Exception {
      super.setUp();
      maClasse = new MaClasse();
  }

  protected void tearDown() throws Exception {
      super.tearDown();
      maClasse = null;
  }

  public void testAdditionner() {
      assertTrue(maClasse.additionner(2,2) == 4);
  }
}
```

  - Le résultat :



TESTS D'INTEGRATION (1/2)

- Fonctionnalités des outils de tests d'intégration :
  - Automatisation de tous les tests : intégration, fonctionnels, non-régression, robustesse, ergonomie
  - Utilisation à la fois côté maîtrise d'œuvre (tests d'intégration) et côté maîtrise d'ouvrage (tests utilisateurs)
  - Création des scénarios de tests soit sous la forme de scripts, soit en mémorisant les interactions d'un utilisateur
  - Enregistrement des scénarios dans un référentiel global
  - Exécution, analyse et comparaison des résultats des tests
  - Constitution de jeux d'essai (alimentation sous la forme de scripts d'une base de données) et comparaison des données avec les résultats attendus après les tests. Le plus souvent, cette fonctionnalité nécessite un outil complémentaire.
  - Générer un rapport de test.

TESTS D'INTEGRATION (2/2)

- Les principaux outils de tests fonctionnels :
  - QA Run (Compuware)
  - SQA Robot (Rational)
  - WinRunner (Mercury)
  - WebKing (Parasoft)
  - JFunc (OpenSource)
  - WatchDog (OpenSource)
  - Cactus (OpenSource)

TESTS DE PERFORMANCE (1/4)

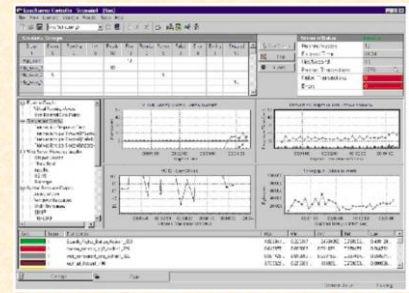
- Fonctionnalités des outils de tests de performance :
  - Création de profils de comportement utilisateur (enregistrement de scénarios d'interactions)
  - Création d'un scénario de test de charges en affectant à différents profils de comportement, un nombre d'utilisateurs en connexion simultanée
  - Assistant d'alimentation de bases de données (scripts)
  - Exécution du scénario
  - Mise à disposition via une console d'indicateurs sur toute l'infrastructure (serveurs, réseaux, base de données…)
  - Stockage des informations en base ou génération de rapports
  - Mise à disposition d'analyses et de diagnostics

TESTS DE PERFORMANCE (2/4)

- Les principaux outils de tests de performance :
  - QA Load (Compuware)
  - SQA Robot / SQA Load Test (Rational)
  - QA Partner et QA Performer (Segue Software)
  - Loadrunner et Astra LoadTest(Mercury)
  - WebKing (Parasoft)
  - AutoController (Auto Tester)
  - OpenSta (OpenSource)
  - JMeter (OpenSource)
  - HttpUnit (OpenSource)
  - JWebUnit (OpenSource)

TESTS DE PERFORMANCE (3/4)

- Loadrunner (Mercury)



TESTS DE PERFORMANCE (4/4)

- OpenSTA (Opensource)
  - Open System Testing Architecture permet de réaliser :
    - *des tests de montée en charge*
    - *des tests de stress d'applications*
    - *la mesure et le suivi de la performance des infrastructures*
  - L'utilisation des ressources des serveurs Web, des serveurs d'applications, des serveurs de base de données et des systèmes d'exploitation en test peut être surveillée, représentée graphiquement et analysée tout autant que le temps de réponse des composants de l'architecture WEB déployée. Les ingénieurs en charge des tests peuvent ainsi planifier efficacement les ressources nécessaires et réduire les risques liés au déploiement des applications web

TESTS DE ROBUSTESSE

- Les tests de robustesse
  - L'élaboration des cas de tests repose sur des actions critiques au niveau utilisateur :
    - *répétition des clics de souris,*
    - *appui aléatoire sur le clavier,*
    - *arrêt brutal de la machine*
    - *arrêt brutal d'un élément du système de communication…*

DOSSIER DE TESTS (1/2)

- Les différentes parties du Dossier de Tests
  - Introduction :
    - *Objectifs du document*
    - *Cible du document*
    - *Terminologie et acronymes*
    - *Structure du document*
    - *Documents de référence*
  - Plan de tests :
    - *Objectifs des tests (cartographie, couverture…)*
    - *Documents de références et normes à respecter*
    - *Stratégie de mise en œuvre des tests (types de tests…)*
    - *Description du processus : conception, validation, exécution des différents tests (qui, quand, but, pré-requis, tâches, sorties)*
    - *Organisation du travail (ordonnancement, planning, responsabilités)*
    - *Usage contractuel (coût des tests, sous-traitance…)*

DOSSIER DE TESTS (2/2)

- Les différentes parties du Dossier de Tests (fin)
  - Cahier des tests
    Modèles et nomenclature des documents à utiliser (TU + TI)
  - Référentiel des tests unitaires
  - Référentiel des tests d'intégration
  - Journal de tests et tableaux de bord
    Présentation et analyse de chaque exécution de test
    - *libellé et nature du test*
    - *machine système utilisée*
    - *date et heure d'exécution*
    - *condition particulière de test, jeu d'essai*
    - *résultat de l'exécution*
    - *mesures réalisées*
    - *liste des références des anomalies identifiées*
  - Bilan

CONCLUSION (1/2)

- 11 idées principales à retenir en terme de réalisation :
  - Garder à l'esprit les différents leviers de productivité :
    - *prototype technique*
    - *outils : gestion de configuration et atelier de développement*
    - *best-practices*
    - *frameworks et composants réutilisables*
    - *respect des normes et documentation*
  - Formation
  - Capitalisation
  - Communiquer
  - Tracer
  - Favoriser le feed-back
  - Respect des phases de validation et anticipation

CONCLUSION (2/2)

- 7 idées principales à retenir en terme de recette :
  - Tests et état d'esprit
  - Intégration des tests tout au long du cycle de vie
  - Formalisation, préparation, planification
  - Affectation judicieuse des ressources
  - Viser l'efficacité plutôt que l'exhaustivité
  - Suivi
  - Outils (revue de code, tests unitaires, tests d'intégration, tests de performance)