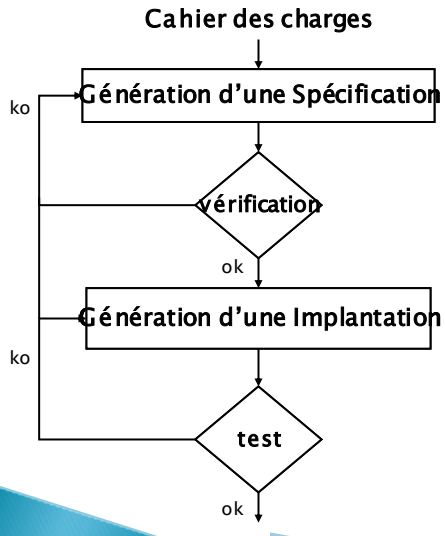
La vérification

- a pour but de vérifier la cohérence de la spécification.
- « La vérification formelle est la production d'une preuve démontrant que le produit respecte effectivement la spécification formelle dont il "prétend" être la réalisation. »

61

La vérification



62

La vérification

- Plusieurs types:
  - Vérification de propriétés (preuve): analyse automatique de la spécification pour rechercher si des propriétés sont satisfaites
    - Interblocage (Deadlock)
    - Bouclage (Livelock)
    - Réception non spécifiée
    - Code mort (état inaccessible,
    - transition infranchissable…)

63

La vérification

- Plusieurs types:
  - Model checking: vérification exhaustive
    - Exprimé plus formellement par:

étant donné une formule logique p et un modèle M ayant un état initial s, le model checking vérifie si M,s ⊨ p (satisfait).

64

La vérification

- Outils
- Spin basé sur langage promela (C++ + mots clés)
- Kronos (automates)

Le test

Méthodes formelles, test en entreprise, test unitaire

Les types de test

- **Phase de Validation de l'implantation :**
  - aussi appelée la phase de Test,
  - permet de vérifier que l'implantation satisfait un certain nombre de propriétés de la spécification. Elle permet donc la détection de défectuosités du fonctionnement de l'implantation du système.
- Différents types de test : conformité, robustesse, performance, interopérabilité, …

67

Définition du test

- According to the classic definition of Myers "*Software Testing is the process of executing a program or system with the intent of finding errors.*"
- According to the definition given by Hetzel, "*Testing involves any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results*"

68

Les boites

- Système ou logiciel vu sous forme d'une ou plusieurs boite:
- Boite blanche, noire grise
- Test en boite blanche: ceux-ci, aussi appelés tests structurels, sont effectués sur des systèmes dont la structure interne est connue et observable.
  - exemple le test de boucle qui vise à valider toutes les boucles d'un programme

69

Les boites

- Test en boite noire :
- cette catégorie rassemble les tests, aussi appelés tests fonctionnels, qui sont appliqués sur des systèmes où la structure interne est inconnue.
- Seules les interfaces reliant le système avec l'environnement extérieur sont connues. (PCO)

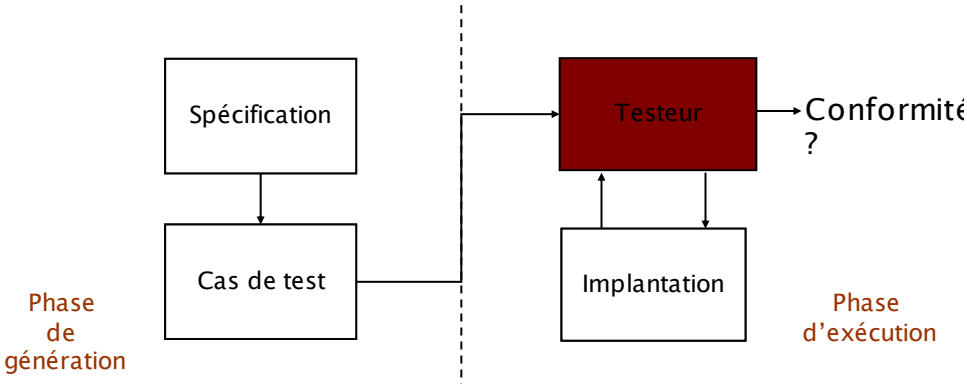
70

Les boites

- Test en boite grise
- Connaissance d'une partie du système, de l'environnement,…
  - Ex: accès au serveur web ou est ébergé une appli web
- Connaissance d'hypothèses sur l'application
  - Ex: l'appli web ne prends jamais de mails invalides

71

Phases de test



72

Cas de test

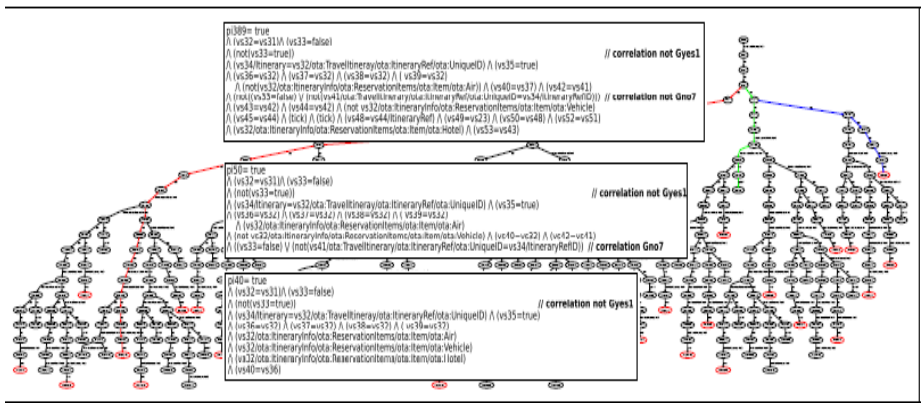
- Junit:

```
import static org.junit.Assert.assertEquals;
import org.junit.Test; public class

MyFirstJUnitTest { @Test public void simpleAdd() { int result =
  1; int expected = 1; assertEquals(result, expected); } }
```

73

Cas de test



74

Cas de test

- Verdict:
- PASS:
- FAIL:
- INCONCLUSIVE:

Types de test

- Tests de caractéristiques: conformité, robustesse, interopérabilité, etc.
- Tests de granularité: nous groupons les tests que l'on trouve souvent en entreprise et qui expriment la granularité utilisée par un test
- Test d'accessiblité: rassemble les approches en boite blanche, grise et noire.

Exemples de types de test

- Tests de l'usager
  - Ces tests sont effectués au niveau de l'usager qui manipule le système pour vérifier si ce dernier répond bien à ses besoins. Ces tests, communément appelés beta-tests, permettent de vérifier les services les plus demandés.
- Tests d'Interopérabilité
  - Ces tests vérifient si le système développé interagit d'une façon correcte avec d'autres systèmes extérieurs en observant les fonctionnements des différents systèmes et des communications engendrées. Ces tests permettent de vérifier un service plus global fourni aux utilisateurs.

77

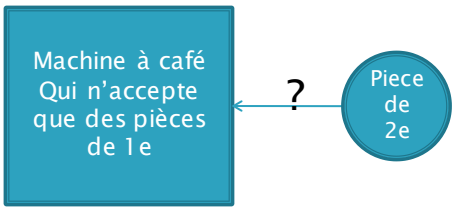
Exemples de types de test

- Tests de Robustesse
  - Ces tests consistent à vérifier la réaction d'un système dans des conditions d'utilisations extrêmes ou bien son exécution dans un environnement dit hostile.
  - Ces conditions ne sont généralement pas prévues dans la spécification, cette dernière référençant des conditions de fonctionnement normales.
  - Ces tests permettent ainsi de vérifier si d'autres erreurs existent telles que des fraudes ou des erreurs d'utilisation du système.
  - En java: Jcrasher

78

Exemples de types de test

- Tests de Robustesse
- Exemple:



79

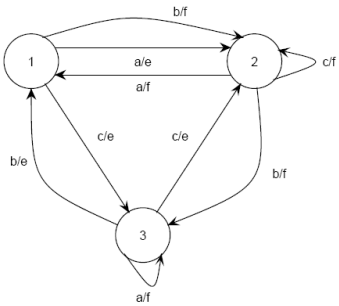
Exemples de types de test

- Test de conformité
  - Cas de test générés à partir de la spécification pour vérifier si le comportement de l'implantation est conforme à celui de la spécification
  - Test en boite noire ou grise
  - Pas d'équivalence: "on peut détecter la présence de fautes pas leur absences"
  - Relations d'implantatations: définition de la relation de conformité
  - Plusieurs relations plus ou moins fortes

80

Test de conformité

- Test de conformité
- sur automate:
- Tour de transition: consiste à trouver une séquence minimale, parcourant au moins une fois chaque transition de la FSM

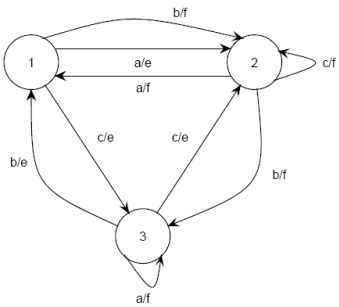


{ c/e a/f b/e a/e b/f c/e c/f a/f b/f }

81

Test de conformité

- Test de conformité
- sur automate:
- Caractérisation d'états: trouver une signature unique des états (suite d'entrées/sorties, puis test de chaque état de l'implantation



82

Test de conformité

- Beaucoup, beaucoup d'autres méthodes
  - Sur divers modèles
  - Par rapport à diverses architectures (syst. , syst. Distribué, temps réel, stockastique, services,…)
  - Test unitaire, de regression, … font partie du test de conformité (mais pas vraiment des méthodes formelles)

Test de conformité d'applications (en entreprise)

- Tests dits de granularité

- Test unitaire
- Test d'intégration
- Test du système
- Test d'acceptation du client
- Test de régression
- Test de sécurité

Test de conformité d'applications (en entreprise)

- Test unitaire
  - Méthode permettant de s'assurer que la plus petite partie d'une application fonctionne correctement "en isolation".
  - Plus petite partie = classe, une librairie, souvent à définir…
  - Exemple: JUNIT !

Test de conformité d'applications (en entreprise)

- Test de l'interaction entre plusieurs composants
- Parcours en largeur ou en profondeur
  - Largeur: on considère tous les modules et on rafine leur utilisation (on teste en raffinant tous les modules en ajoutant des détails petit à petit
  - Profondeur: on teste en détaillant d'abord en ajoutant petit à petit des modules

Test de conformité d'applications (en entreprise)

- Test du système:
  - = test d'intégration avec tous les modules avec détails
  - Après les tests unitaires
  - Souvent manuel ?
  - Outil: HP quality center, AGL objecteering (softeam)
- Test d'acceptation du client:
  - Vérification si l'application correspond à ce que le client demande (alpha test et beta test)
  - Avec CMMI 3, cette phase est revue pendant tout le cycle de vie

87

Test de conformité d'applications (en entreprise)

- Test de régression:
  - On teste si la modification de code n'a pas eu d'inpact sur les fonctionnalités existantes
  - On utilise pour cela les tests unitaires existants et on vérifie qu'aucune erreur n'est détectée
- Test de sécurité
  - Contrôle qu'il n'existe pas de vulnérabilités (de failles)
  - Sécurité réseau, d'accès aux BD, de paramètres de champs (web)
  - Test des log
  - Test de pénétration (buffer overflow, liens symboliques, race condition (pb de concurrence), troyens, …
  - Crack des mots de passe

88

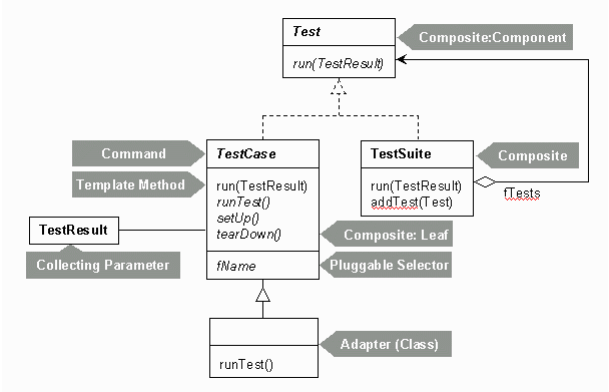
JUNIT

- (Extrait d'un doc de dvp.com)
- Framework JAVA de rédaction et d'exécution de cas de test (test de conformité unitaire)
- Assez basique (mieux ? HP quality center mais payant)
- Représentation de chaque test par un objet à part (1 test correspond souvent à une classe)
- 1 test composé de cas de test (test unitaire) permettant de valider les méthodes de la classe

89

JUNIT

- Construction manuelle des cas de test par rapport à une spécification
- architecture



90

JUNIT

- Programmeur doit manipuler les classes Testcase et Testsuite
- **TestCase** implémente l'interface **Test** et dérive en outre de la classe **Assert** dont les nombreuses méthodes vous permettront de valider votre code.

Méthode	Rôle
assertEquals	Vérifie que deux objets sont égaux
assertFalse	Vérifie que l'expression est fausse
assertNotNull	Vérifie que l'objet n'est pas nul
assertNotSame	Vérifie que deux références ne sont pas les mêmes
assertNull	Vérifie qu'un objet est nul
assertSame	Vérifie que deux références sont les mêmes
assertTrue	Vérifie que l'expression est vrai
fail	Provoque l'échec du test

91

JUNIT

Exemple:

```
public class VectorTest extends TestCase {
public void testClone() {
        Vector v1 = new Vector(2);
        v1.add("Test");
        v1.add("Case");
        Vector v2 = (Vector) v1.clone();
        assertNotSame(v1, v2);
        assertEquals(v1, v2); }
}
```

92

JUNIT

Exemple 2: utilisation d'objets dans les mêmes cas de test (fixture)

```
public class FileTest {
  private File dir;

  protected void setUp() {
       dir = new File("/etc"); }

public void testIsDirectory() {
  assertTrue(dir.isDirectory());  }
}
```

initialisation

protected void tearDown()

libération

93

JUNIT

- Exécution des tests:
- Exécution 1 par un 1 ou en utilisant une suite de test
- Exécution 1 par 1:
  - Classe TestCase possède une méthode runTest()
  - Code à placer dans la méthode *main()* de la classe à tester ou de la classe de test
    - TestCase tc = new VectorTest("testClone");
    - TestResult result = new TestResult();
    - tc.runTest(result);
    - TestSuite mysuite = suite();
    - TestResult result = new TestResult();
    - mysuite.run(result);

94

JUNIT

- Exécution des tests:
- Utilisation d'une suite:
  - Ex1:

```
public static Test suite() {
   TestSuite suite = new TestSuite();
   suite.addTest(new FileTest("testIsDirectory")); suite.addTest(new
   FileTest("testIsFile"));
return suite; }
```

  - Ex2:
  - public static Test suite() { return new TestSuite(FileTest.class); }
- Exécution d'une suite:

```
mysuite.run(result);
```


JUNIT

- Lecture manuelle des résultats:

```
Object error = null;
for (Enumeration e = result.errors();
        e.hasMoreElements();
        error = e.nextElement())          {
        System.out.println("ERROR:  " + error); }
```

- Mais heureusement Junit fournit des interfaces (test, awt ou swing)

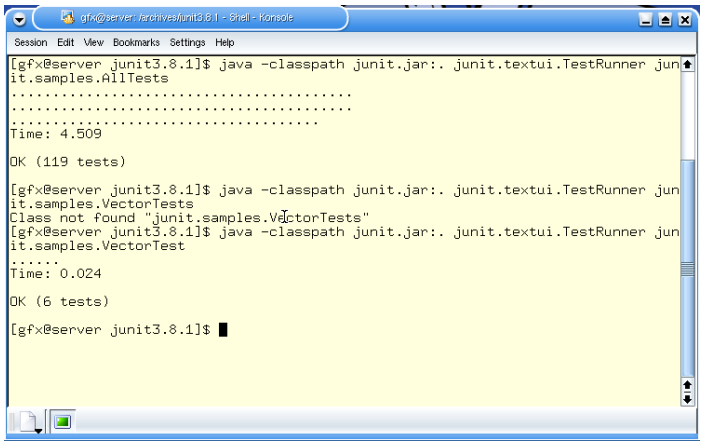
JUNIT

- Lecture des résultats par interface:
  - Lancement dans du code:
    junit.textui.TestRunner.run(FileTest.class);
  - Lancement en ligne de commande:
    java -classpath junit.jar:. junit.textui.TestRunner FileTest
- remplacer junit.textui par junit.awtui ou junit.swingui, pour obtenir une interface graphique

97

JUNIT

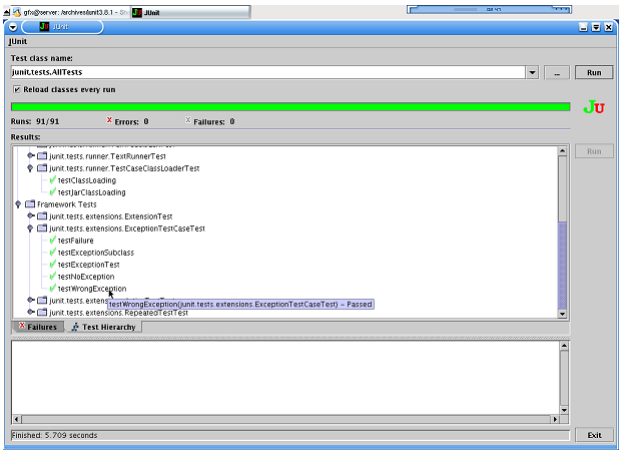
Interface texte:



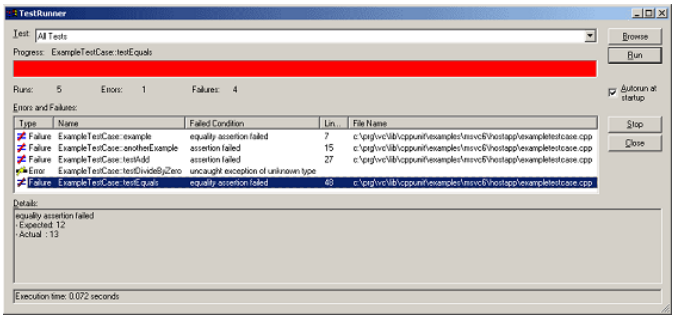
98

JUNIT

- Interface graphique



Junit pour c++: cppunit



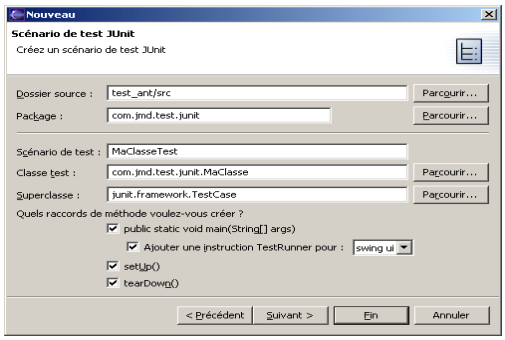
Intégration de JUNIT dans Eclipse

- Ajout de junit.jar dans classpath

- Création d'un test:

"nouveau cas de test Junit"

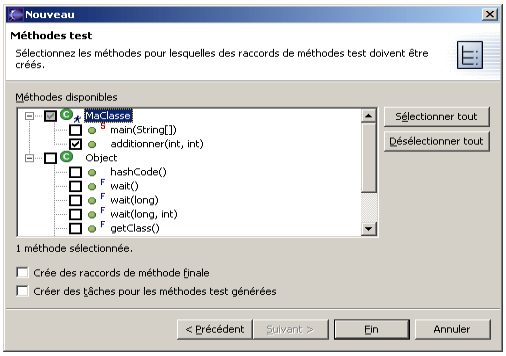
Il existe aussi des suites !!!



10
1

Intégration de JUNIT dans Eclipse

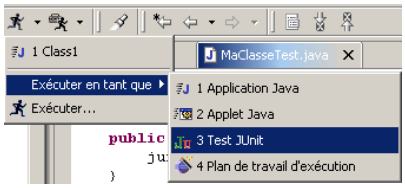
- sélection des méthodes

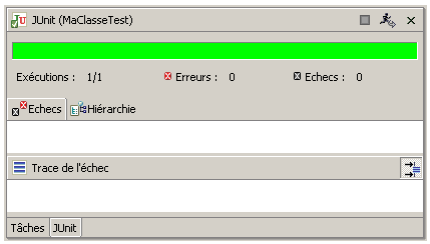


10
2

Intégration de JUNIT dans Eclipse

- Exécution de votre classe en tant que classe Junit:





Mais aussi...

- Qualité du code écrit
  - Format, boucles, commentaires,...
  - Outils: jdepend, jtest(parasoft)

Outils

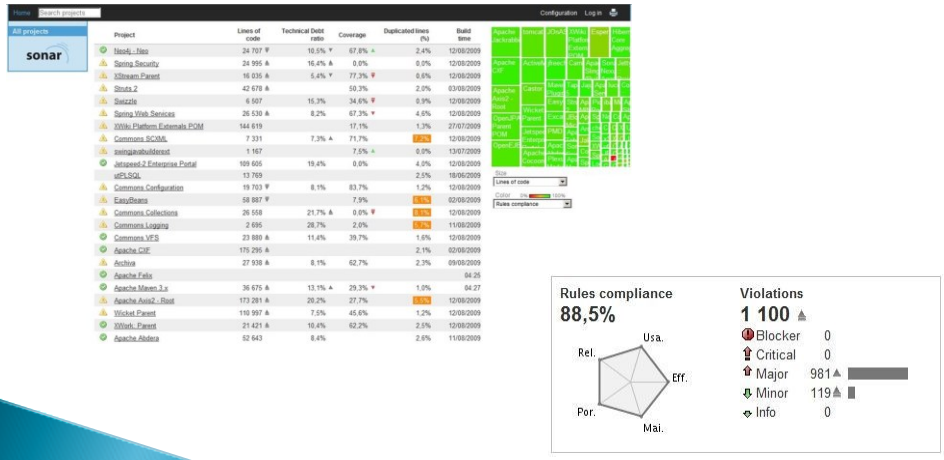
- Profiler
  - Netbeans
- Outils de travail colaboratif
  - Au moins svn (subversion, et tortoise sous windows)
- Outils de tests
  - Junit (et équivalents pour d'autres langages), HPQuality Center (test d'intégration, de regression),

Outils

- Outils de contrôle qualité: Sonar, Jdepend
  - Sonar
- Tableau de bord complet des différents projets suivis.
- Détection rapide du code à risque.
- Mesures quantitatives : nombre de classes, duplication de code, etc...
- Mesures qualitatives : couverture et taux de réussite des tests, complexité du code, respect des règles de codage...
- Historiques des statistiques, pour en voir l'évolution au cours du temps.
- Support de plus de 600 règles de qualité (suivant la norme ISO 9126-3).
- Gestion de profils pour les règles de codage.
- Visualisation du code source, surlignant les violations des règles de codage qui s'y trouvent.
- Fonction "Time machine" permettant de comparer plusieurs versions d'une même application.
- Identification des points faibles d'un projet.
- Support des plugins.

Outils

- Sonar



Outils

- Redmine : outil (app web) de gestion de projet (gantt, svn, etc.)