

Cycle de vie de logiciel

CMMI, spécifications, tests

plan

- ▶ CMMI
- ▶ Cycle de vie
- ▶ Tests (unitaires, de régression, fonctionnel, de conformité, de sécurité, de robustesse, etc.)

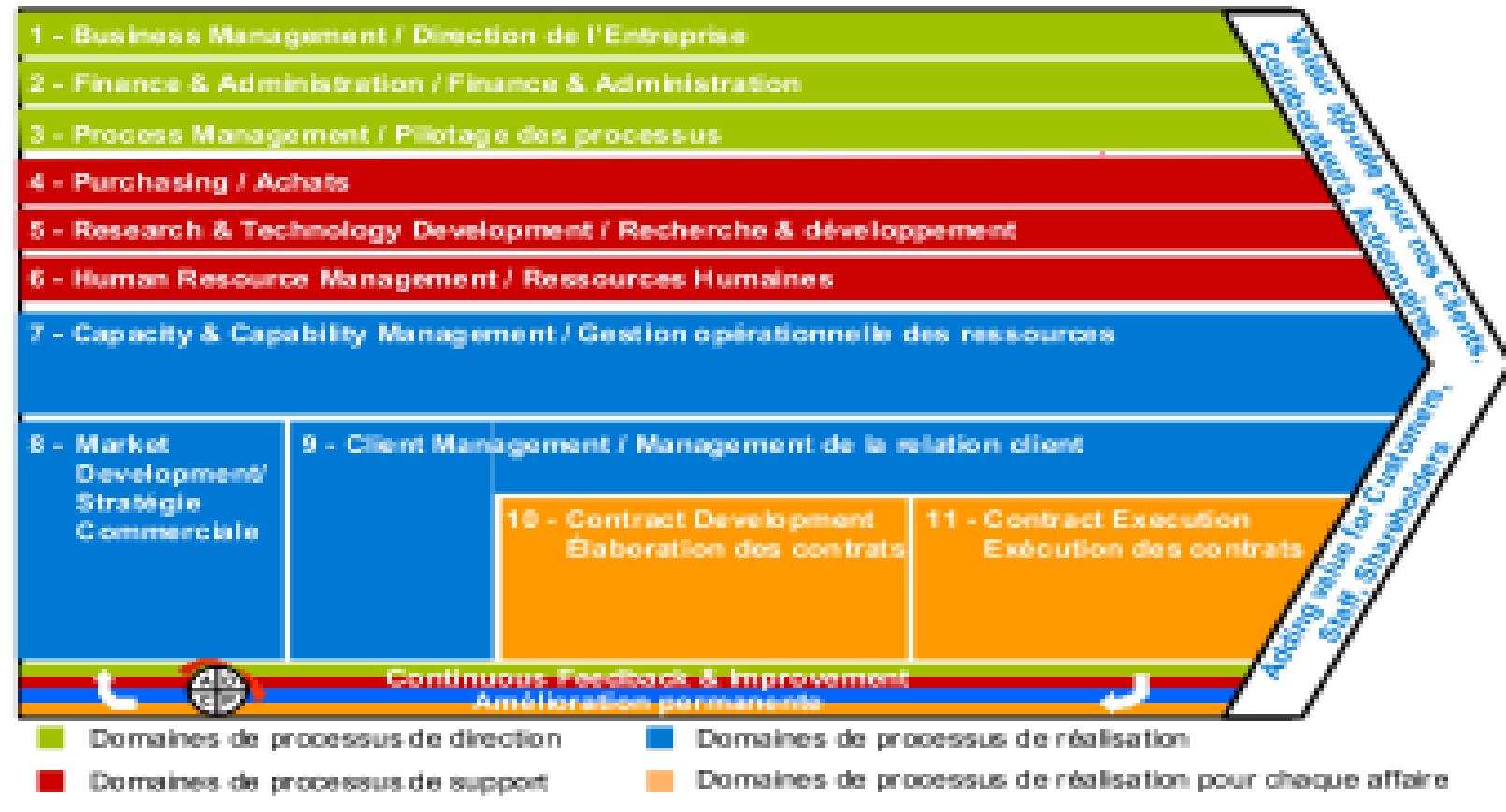
Capability Maturity Model Integration (CMMI)

- ▶ Wikipedia : Le Capability Maturity Model Integration est une approche interdisciplinaire d'ingénierie des systèmes couvrant les compétences et processus techniques et managériaux.
- ▶ C'est un modèle de référence qui permet d'évaluer le niveau d'organisation globale d'une entreprise IT relativement à une métrique d'utilisation de bonnes pratiques, en génie logiciel et management.
- ▶ Origine des US

Capability Maturity Model Integration (CMMI)

- ▶ Entreprises certifiées:
 - Demande de certification, effectuée par un organisme agréé CMMI
- ▶ CMMI modèle pas une norme ?
 - Souvent appelé norme mais c'est un modèle
 - Norme ISO 9001 = norme qualité au sens général
 - CMMI est un modèle satisfaisant la norme ISO 9001

Capability Maturity Model Integration (CMMI)



Capability Maturity Model Integration (CMMI)

- ▶ Pourquoi faire ?
 - Le développement “marchait” bien avant ? Pas tant que ça
 - => dvp, debuggage, dvp, debuggage, envoi au client => pb de fonctionnement, suivi de bug
 - Pour les systèmes critiques cette approche est des plus mauvaises ! => validation formelle depuis les années 70 ! Mais pas spécialement dans les entreprises
 - Aujourd’hui, les entreprises s’interressent de plus en plus à la qualité logicielle (ATOS-ORIGIN (CMMI3), impôts, boeing, intel, ...)
 - CMMI mais aussi d’autres modèles : ITIL Cobit,...

Capability Maturity Model Integration (CMMI)

- ▶ Normes qualité logicielle difficile à mettre en place
 - difficile de changer les habitudes
 - Lourd et coûteux (test de logiciel/système > 40% du coût pour les syst. Critiques)
 - Vers des nouveaux métiers informatiques ! Qualité logicielle
- ▶ Mais certaines entreprises d'Inde sont CMMI 5

Capability Maturity Model Integration (CMMI)

- ▶ CMMI proposé par le SEI (software engineering institute), version 1.3 aujourd'hui
- ▶ Plusieurs modèles CMMI
 - acquisition
 - Développement, doc de 573 pages
- ▶ CMMI est donc avant tout un référentiel d'évaluation
 - de la capacité à gérer et terminer un projet correctement,
 - proposant nombre de bonnes pratiques liées à la gestion, au développement et à la maintenance d'applications et de systèmes.

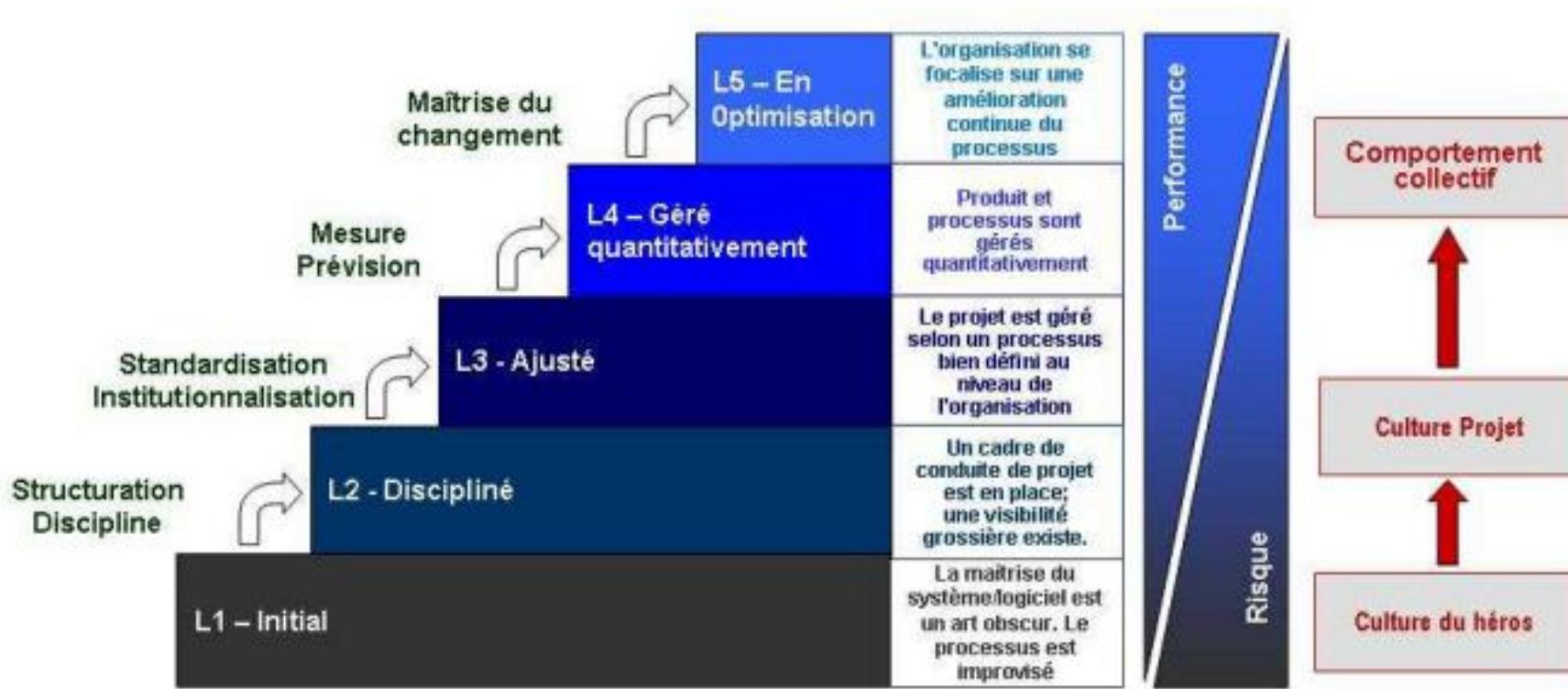
Capability Maturity Model Integration (CMMI)

- ▶ Ces bonnes pratiques sont regroupées en 24 processus, eux-mêmes regroupés en 4 types :
 - *Process Management*
 - (*définir, mesurer, contrôler les processus*) (*méthode de fonctionnement dans l'entreprise*)
 - *Project Management*
 - *Engineering*
 - *Support*
- et 5 niveaux de maturité.

Capability Maturity Model Integration (CMMI)

- ▶ 5 niveaux de maturation:
- ▶ un projet est supposé monter de niveau au fur et à mesure du temps. Plus le niveau est grand, moins il y a de risques et meilleure est la performance du projet.
- ▶ Pour instaurer dans les organisations et les projets l'importance de disposer de spécifications clairement établies et définies, même si elles peuvent être révisées en cours de route.
- ▶ CMMI a également établi l'intérêt de la relecture par un pair (*peer review*), qui est aujourd'hui l'un des points fondamentaux de l'Extreme Programming, ainsi que du contrôle de version, également très répandu aujourd'hui.

Capability Maturity Model Integration (CMMI)



Capability Maturity Model Integration (CMMI)

- ▶ Niveaux de maturité:
- ▶ Niveau 1:
 - ▶ Le niveau le plus bas montre que l'organisation n'est pas prête, et le projet pas stable.
 - ▶ Ce dernier dépend d'une poignée de personnes, qui ne font pas appel à des processus éprouvés.
 - ▶ Il se peut cependant que le projet aboutisse, mais en dépassant certainement le budget et le temps alloués. Le projet ne construit pas sur les succès passés.

Capability Maturity Model Integration (CMMI)

- ▶ Niveaux de maturité:
- ▶ Niveau 2:
- ▶ Le projet construit sur ce qui a été appris précédemment, en faisant appel à une certaine discipline et à une gestion de projet basique.
- ▶ De fait, le projet est géré selon les plans, avec étapes-clefs et vérification des coûts et des fonctionnalités.

Capability Maturity Model Integration (CMMI)

- ▶ Niveaux de maturité:
- ▶ Niveau 3:
- ▶ Ce n'est plus le projet qui dispose d'une bonne discipline, mais l'ensemble de l'organisation, de manière cohérente. Tous les projets s'en trouvent améliorés.
- ▶ Vérification ,test, gestion de risque,...

Capability Maturity Model Integration (CMMI)

- ▶ Niveaux de maturité:
- ▶ Niveau 4:
- ▶ Les efforts de mesure et de gestion autorisent un contrôle sans effort du développement, avec capacité d'ajuster et adapter des projets précis sans troubler les autres. Les performances des processus sont prévisibles en quantité comme en qualité.

Capability Maturity Model Integration (CMMI)

- ▶ Niveaux de maturité:
- ▶ Niveau 5:
- ▶ Les processus sont constamment améliorés de manière incrémentale et innovante. Les objectifs sont revus en permanence pour rester proches des besoins du marché. Les évolutions sont anticipées et gérées de bout en bout.

Capability Maturity Model Integration (CMMI)

Name	Abbr	ML	CL1	CL2	CL3	CL4	CL5
Requirements Management	REQM	2					
Project Planning	PP	2					
Project Monitoring and Control	PMC	2					
Supplier Agreement Management	SAM	2					
Measurement and Analysis	MA	2					
Process and Product Quality Assurance	PPQA	2					
Configuration Management	CM	2					
Requirements Development	RD	3					
Technical Solution	TS	3					
Product Integration	PI	3					
Verification	VER	3					
Validation	VAL	3					
Organizational Process Focus	OPF	3					
Organizational Process Definition +IPPD	OPD +IPPD	3					
Organizational Training	OT	3					
Integrated Project Management +IPPD	IPM +IPPD	3					
Risk Management	RSKM	3					
Decision Analysis and Resolution	DAR	3					
Organizational Process Performance	OPP	4					
Quantitative Project Management	QPM	4					
Organizational Innovation and Deployment	OID	5					
Causal Analysis and Resolution	CAR	5					

Figure 3.5: Target Profiles and Equivalent Staging

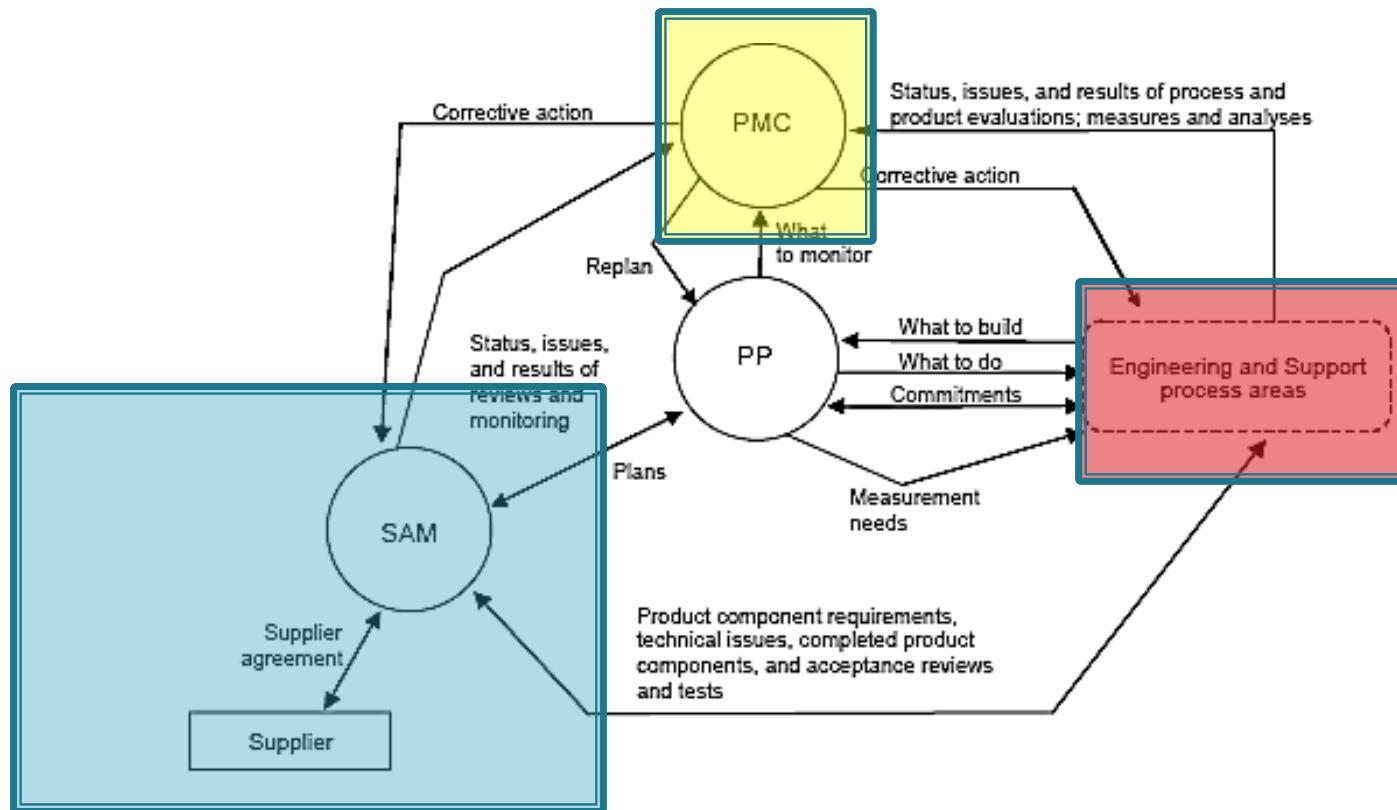
Capability Maturity Model Integration (CMMI)

- ▶ Dans CMMI, on trouve:
- ▶ Des gestions de planning, de projets, le monitoring et le contrôle de projet, la gestion des risques, la gestion du client,
- ▶ le cycle de vie du logiciel,....
- ▶ Voyons pour la partie gestion de projet:

Capability Maturity Model Integration (CMMI)

- ▶ Cf CMMI doc:
- ▶ The Basic Project Management process areas address the activities related to establishing and maintaining the project plan, establishing and maintaining commitments, monitoring progress against the plan, taking corrective action, and managing supplier agreements

Capability Maturity Model Integration (CMMI)



PMC = Project Monitoring and Control
PP = Project Planning
SAM = Supplier Agreement Management

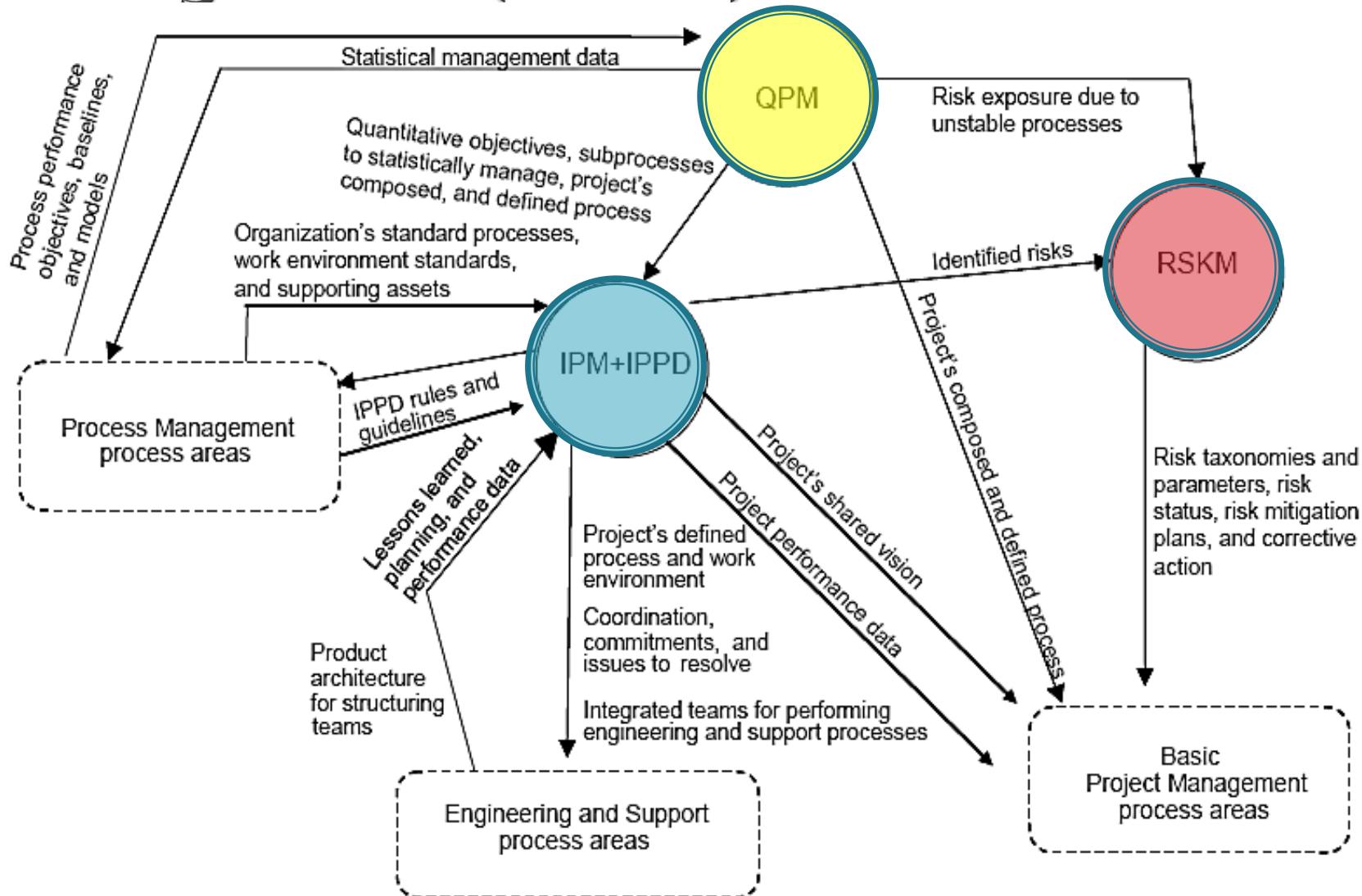
Figure 4.3: Basic Project Management Process Areas

Capability Maturity Model Integration (CMMI)

- ▶ CF CMMI doc:

The Advanced Project Management process areas address activities such as establishing a defined process that is tailored from the organization's set of standard processes, establishing the project work environment from the organization's work environment standards, coordinating and collaborating with relevant stakeholders, managing risk, forming and sustaining integrated teams for the conduct of projects, and quantitatively managing the project's defined process

Capability Maturity Model Integration (CMMI)



Capability Maturity Model Integration (CMMI)

IPM+IPPD = Integrated Project Management (with the IPPD addition)

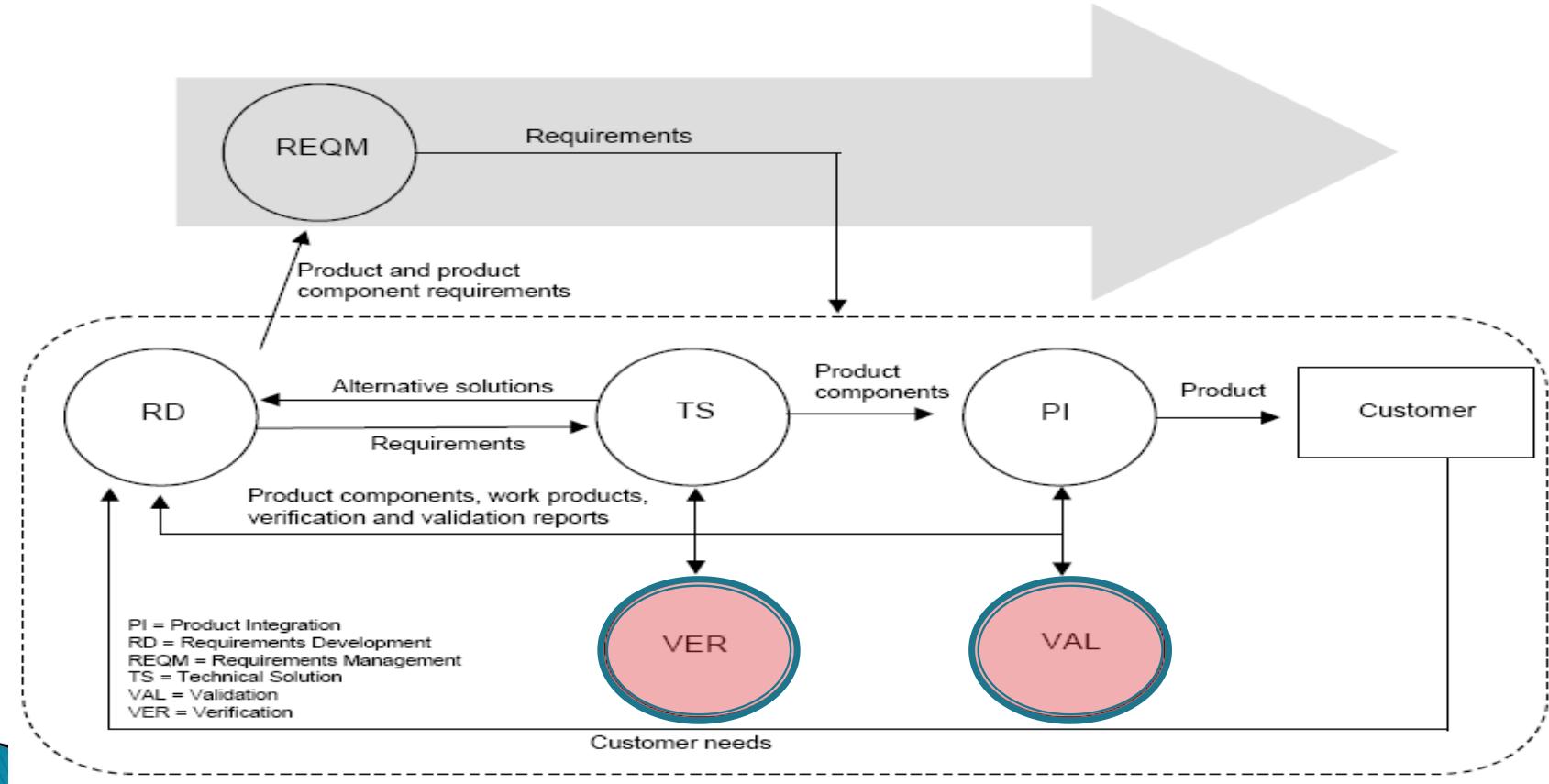
IPPD (integrated product and process development)=>

QPM = Quantitative Project Management

RSKM = Risk Management

Capability Maturity Model Integration (CMMI)

Engineering process area



Exemple de dossiers (atos??)

Dossier d'affaire

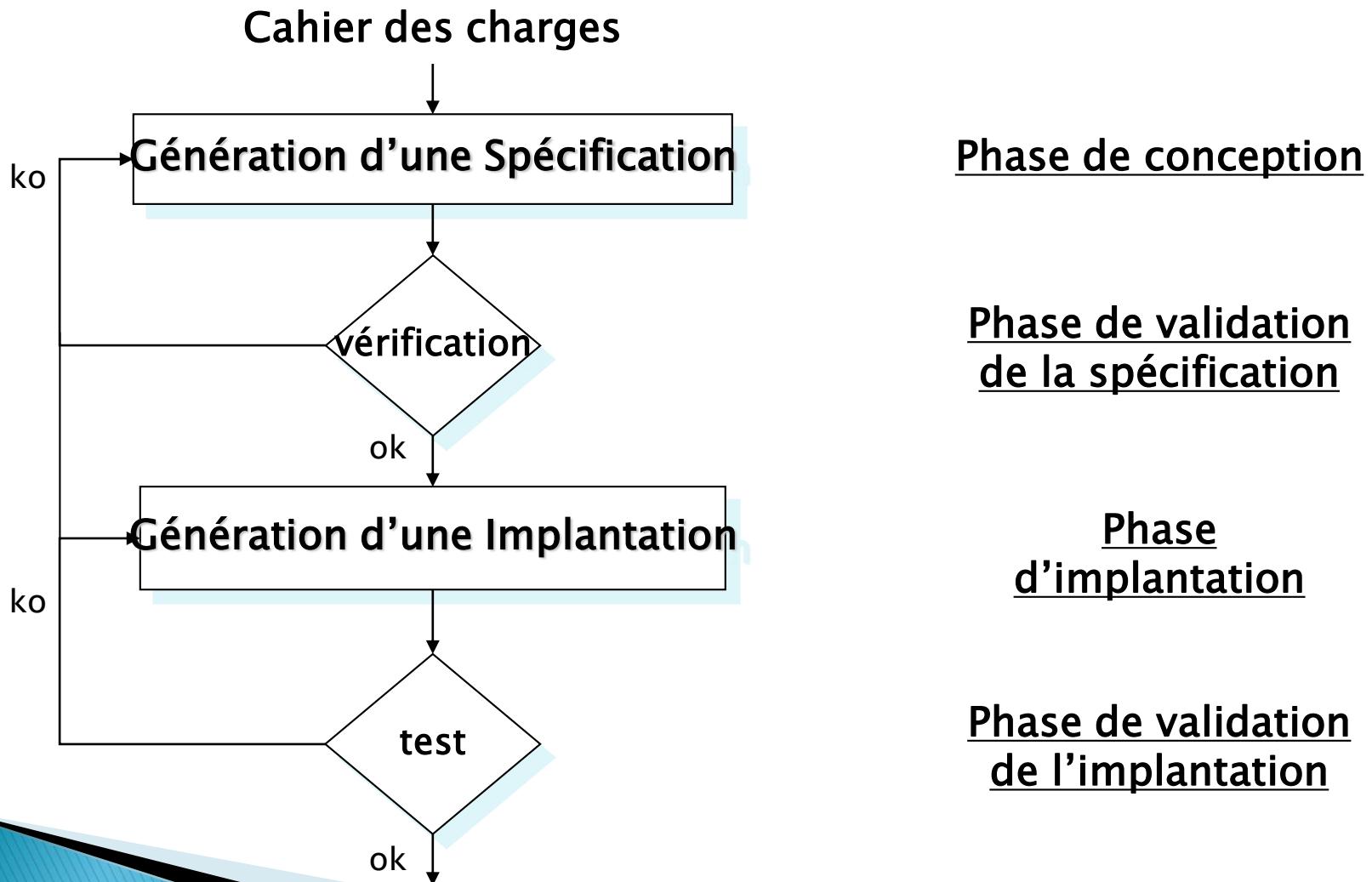


- PQP : plan qualité projet = référentiel (outils, ...)
- PdT : plan de test
- Plan d'avancement projet (remplis par chef de projet => indicateurs

Cycle de vie, specification, test

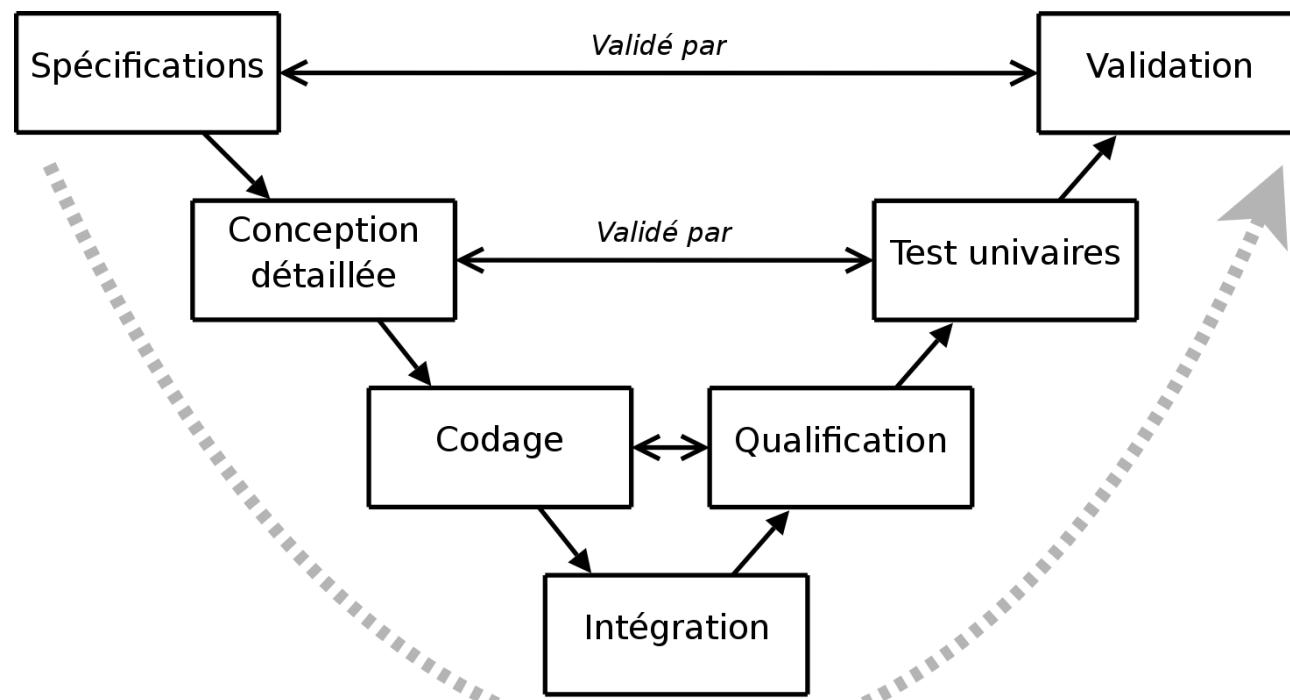
Cahier des charges

... engineering process area peut s'écrire aussi



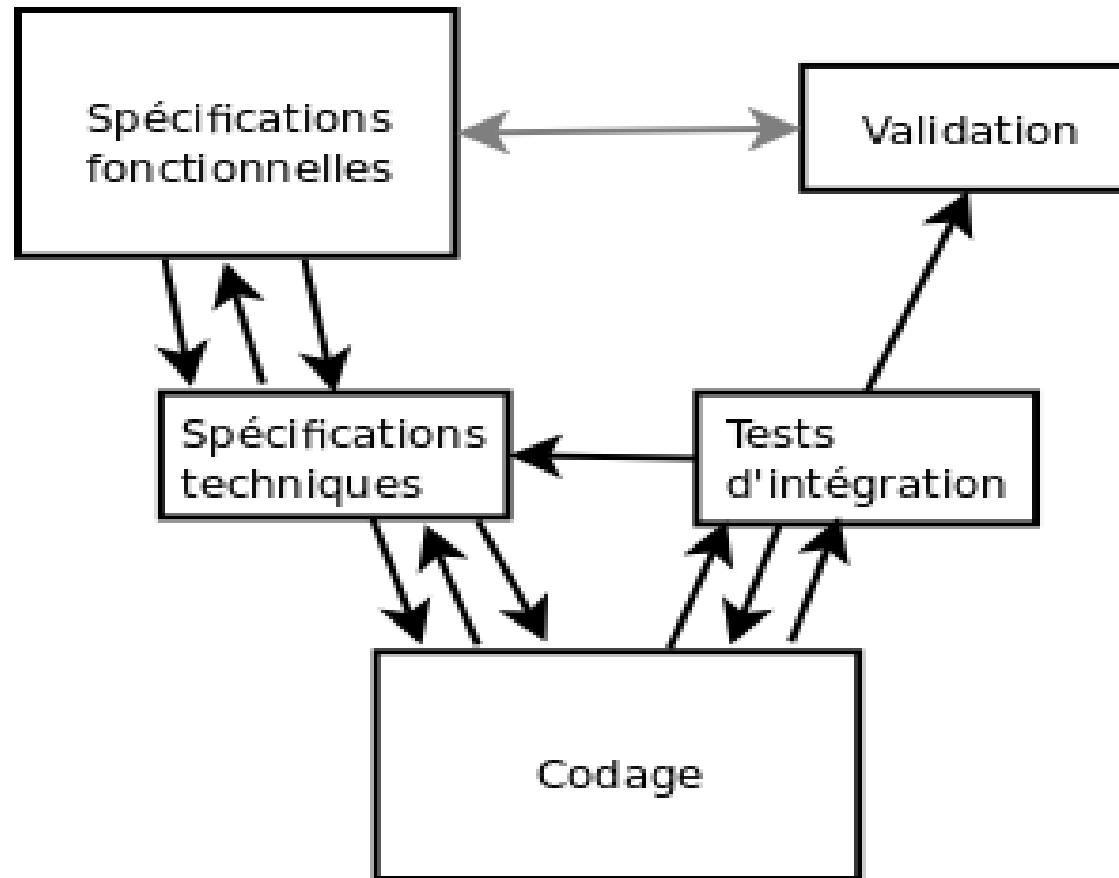
...Ou encore

Cycle en V



...Ou encore

Cycle en V pas assez souple, la réalité ?



...Ou encore

Méthodes agiles

- Cycles rapides de développement , recherche de scenarios, découpage en taches, attribution des taches, phases de tests et on recommence tant que le client a des scénarios à demander,
- Correspondance continue entre les dev. et le client,
- Réunions journalières, etc.

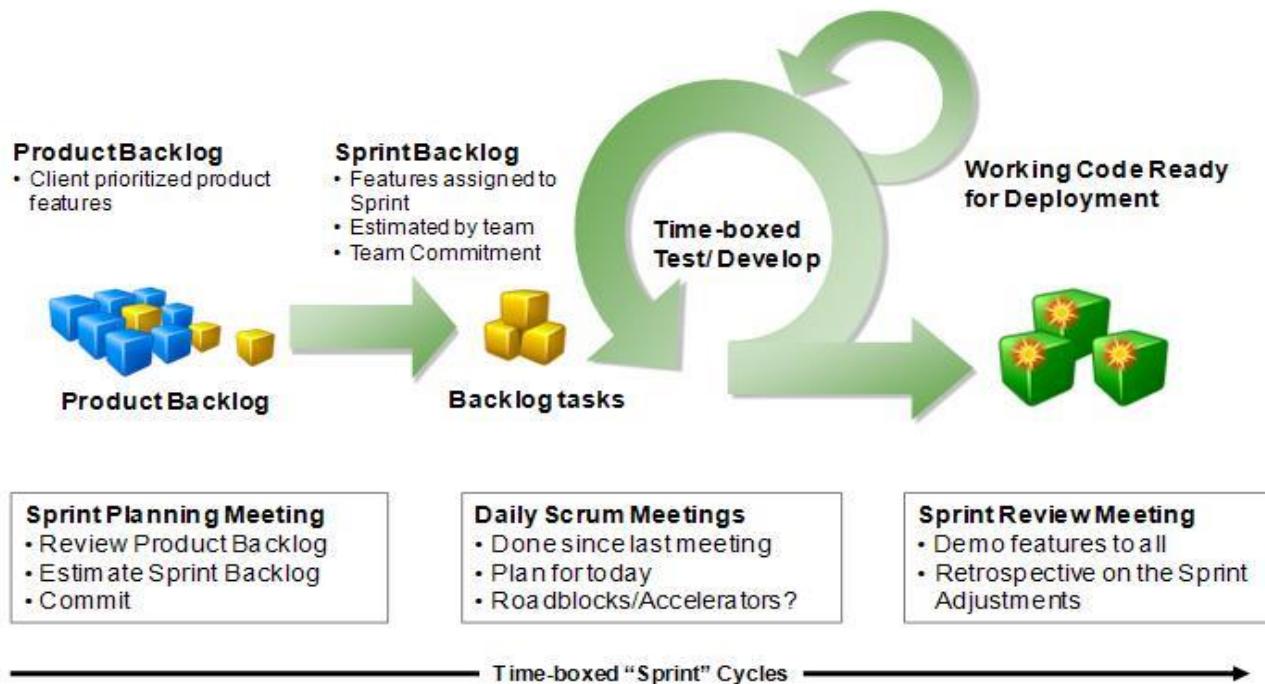
Ex de méthode:

- XP (extreme programming) :codage et revue de code en continue par un binôme !
- TDD (**Test-Driven Development**): Tests faits systématiquement avant chaque mise en œuvre,
- Refactoring important et continu,
- Cycles de dev. Rapides, etc.

...Ou encore

Méthodes agiles

SCRUM (transparence, inspection, adaptation):



Cahier des charges

- ▶ Un « **cahier des charges** » est un document contractuel décrivant ce qui est attendu du maître d'œuvre par le maître d'ouvrage.
- ▶ Il s'agit donc d'un document décrivant de la façon la plus précise possible, avec un vocabulaire simple, les besoins auxquels le maître d'œuvre doit répondre.
- ▶ Pas de technique
- ▶ Elements pour juger de la taille du projet et sa complexité
- ▶ Contenu peut être modifié sur la base d'un accord
- ▶ Doit aussi contenir un planning

Cahier des charges

► A) Introduction

► 1- Contexte

Décrire brièvement l'environnement dans lequel s'inscrit le projet (stratégie, enjeux, domaine, etc.)

► 2- Historique

Il est possible de donner un bref historique du contexte dans lequel s'inscrit le projet si cela est pertinent. Gardez à l'esprit que ceci n'est qu'un exemple de cahier de charge. Vous ne devez pas tout appliquer à la lettre.

Cahier des charges

B) Description de la demande

▶ 1– Les objectifs

Définir les résultats que le projet doit atteindre.

Méthode : Un énoncé d'objectif doit comporter un verbe d'action à l'infinitif et un objet.

Exemple : Diffuser un corpus de connaissance assimilable par toute personne de niveau Bac + 4.

▶ 2– Produit du projet

Proposer une description générale de ce produit.

▶ 3– Les fonctions du produit

Lister et justifier les principales fonctionnalités du produit.

▶ 4– Critères d'acceptabilité et de réception

Formuler des indicateurs précis qui permettent de mesurer si les objectifs de performance du produit sont atteints.

Exemple : Le produit doit répondre à la norme XX0.

Cahier des charges

C) Contraintes

- ▶ **1– Contraintes de coûts**
Spécifier le budget alloué au projet.
- ▶ **2– Contrainte de délais**
Spécifier la date de livraison du produit et les éventuelles échéances intermédiaires.
- ▶ **3– Autres contraintes**
Spécifier les éventuelles autres contraintes à prendre en compte dans le cadre du projet (normes techniques, clauses juridiques, etc.)

Cahier des charges

D) Déroulement du projet

- ▶ **1– Planification**

Représenter l'articulation des grandes phases du projet et des principaux jalons.

- ▶ **2– Ressources**

Lister les ressources humaines et matérielles que le client peut mettre à la disposition du prestataire.

Cahier des charges

E) Authentification

Date et signature du chef de projet et du maître d'ouvrage.

F) Annexes

Lister et joindre au cahier des charges les éventuels documents que le client peut mettre à disposition.

spécifications

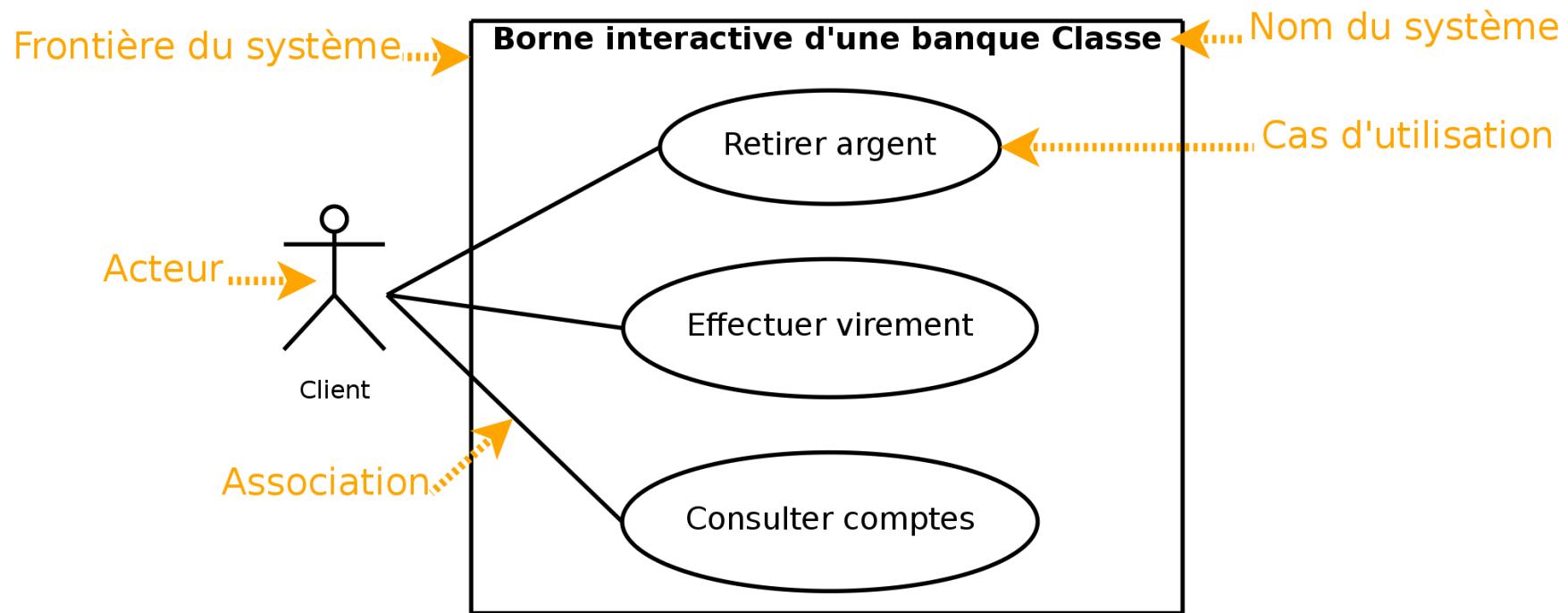
- ▶ Du cahier des charges => on en déduit une spécification détaillée (avec pl. étapes)
- ▶ “ce que vous appelez l’analyse”
- ▶ Plusieurs modèles de spécifications (tous sont formels)
 - Automates, automates temporisés
 - Réseau de petri
 - Lds, lotos et les dérivés stockastiques et temporisés
 - UML, SYSML et les dérivés, merise,...
 - BPEL, (services)

UML

- ▶ **La modélisation UML**
- ▶ UML fournit un moyen astucieux permettant de représenter diverses projections d'une même représentation grâce aux **vues**.
- ▶ **Les vues statiques**, c'est-à-dire représentant le système physiquement diagrammes d'objets
 - ▶ diagrammes de classes
 - ▶ diagrammes de cas d'utilisation
 - ▶ diagrammes de composants
 - ▶ diagrammes de déploiement
- ▶ **Les vues dynamiques**, montrant le fonctionnement du système
 - ▶ diagrammes de séquence
 - ▶ diagrammes de collaboration
 - ▶ diagrammes d'états-transitions
 - ▶ diagrammes d'activités

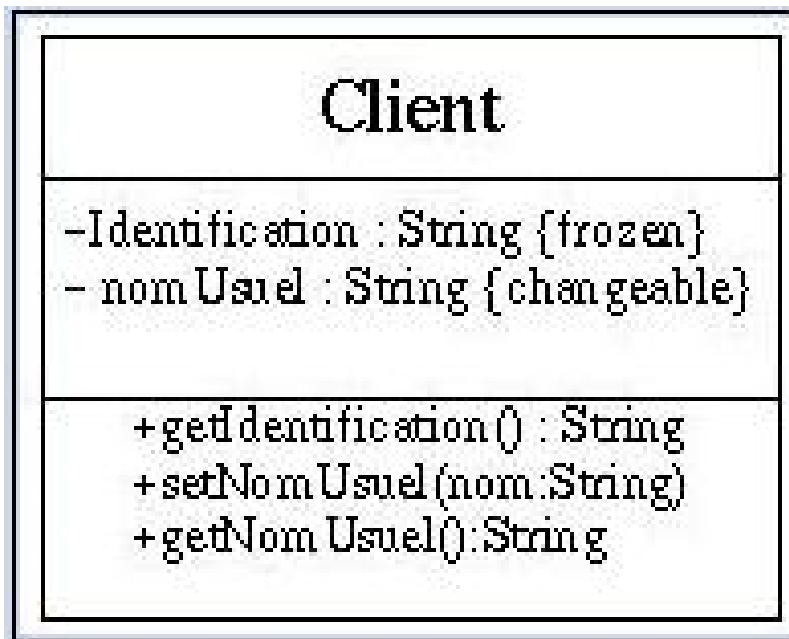
UML

1. Cas d'usages: décrit de façon fonctionnelle le système



UML

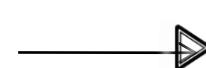
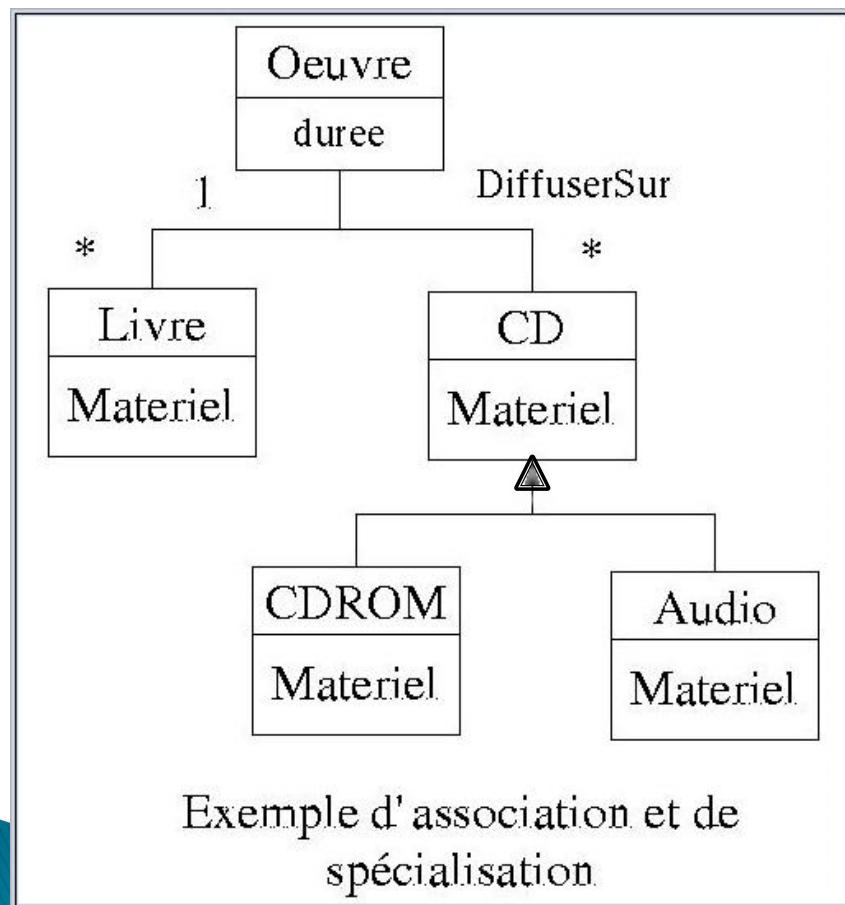
2. diagramme de classes



+ public
- Private
protected

UML

2. diagramme de classes



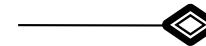
Héritage “est une sorte de “



Composition (classe ne peut pas exister par elle même)



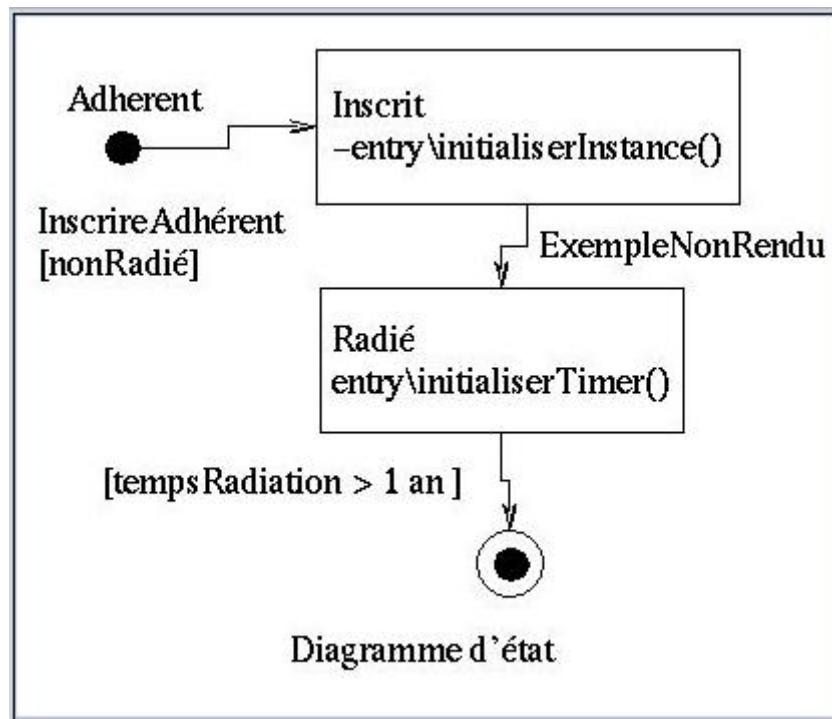
association



Aggrégation
Une classe regroupe d'autres (instance,...)

UML

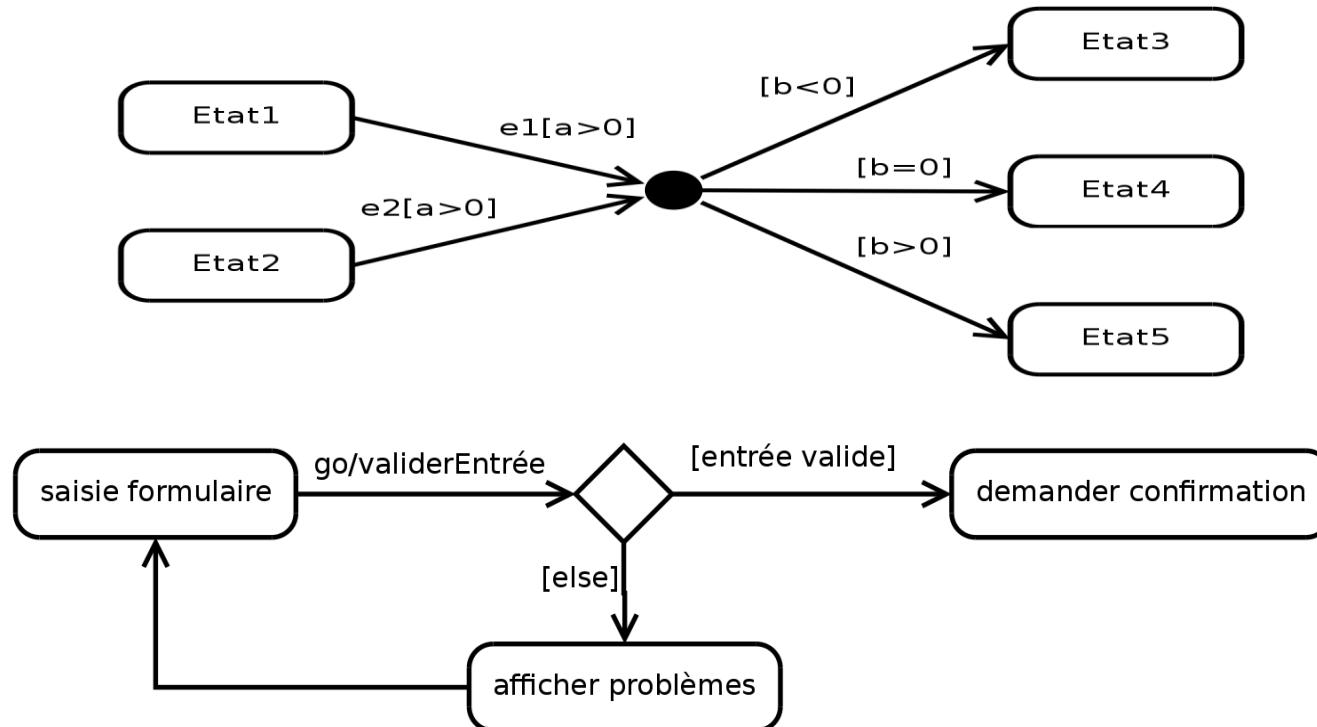
- 3. diagramme d'états: montre les états d'un objet au cours du temps



Point de choix
Decision (losange)

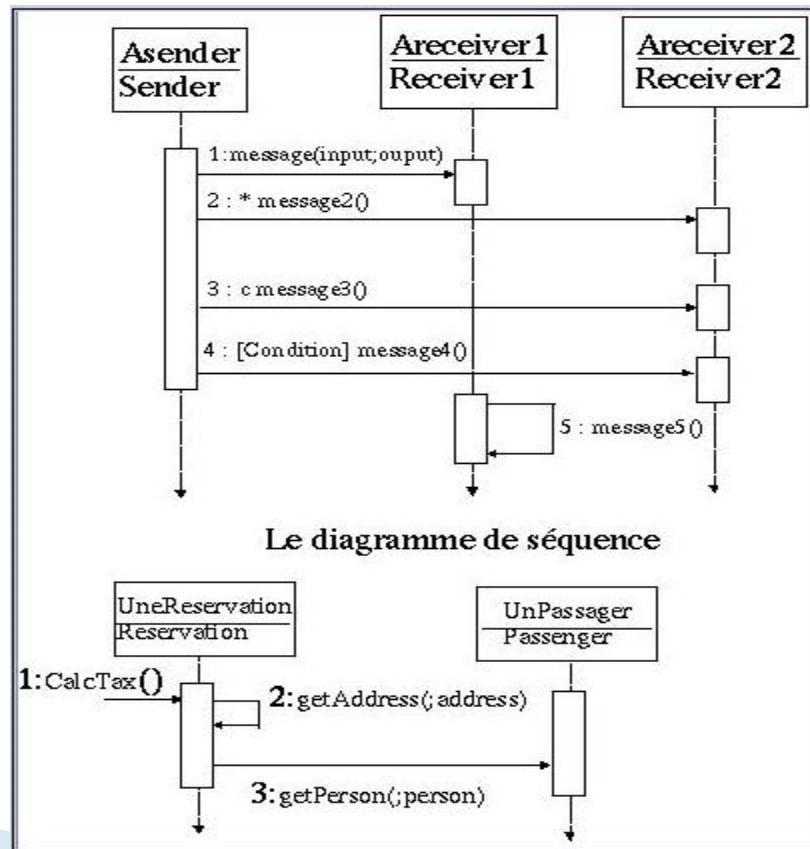
UML

- 3. diagramme d'états: montre les états d'un objet au cours du temps



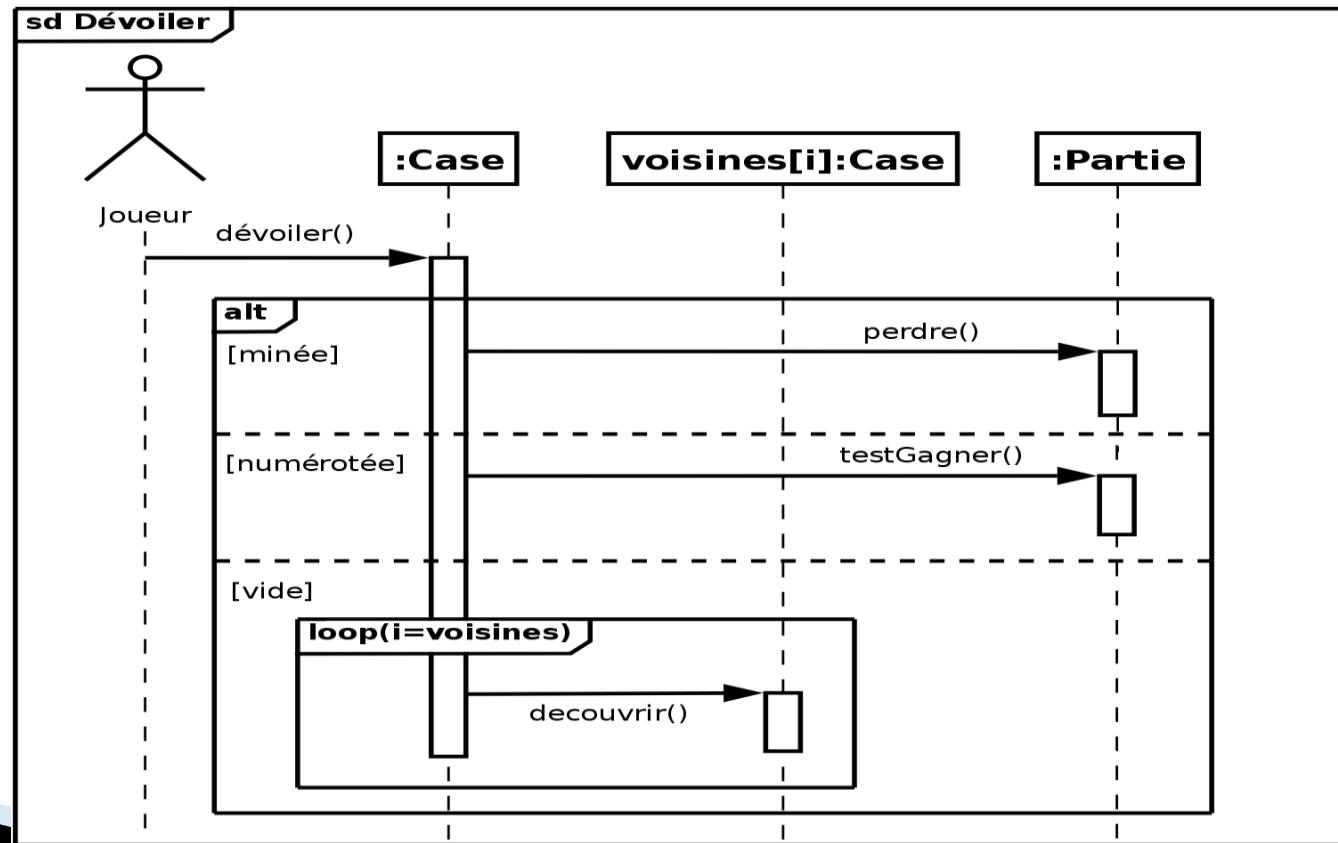
UML

- 4. diagramme de séquence : relations entre objets dans le temps



UML

► 4. diagramme de séquence : opérateurs (alt, loop, ...)

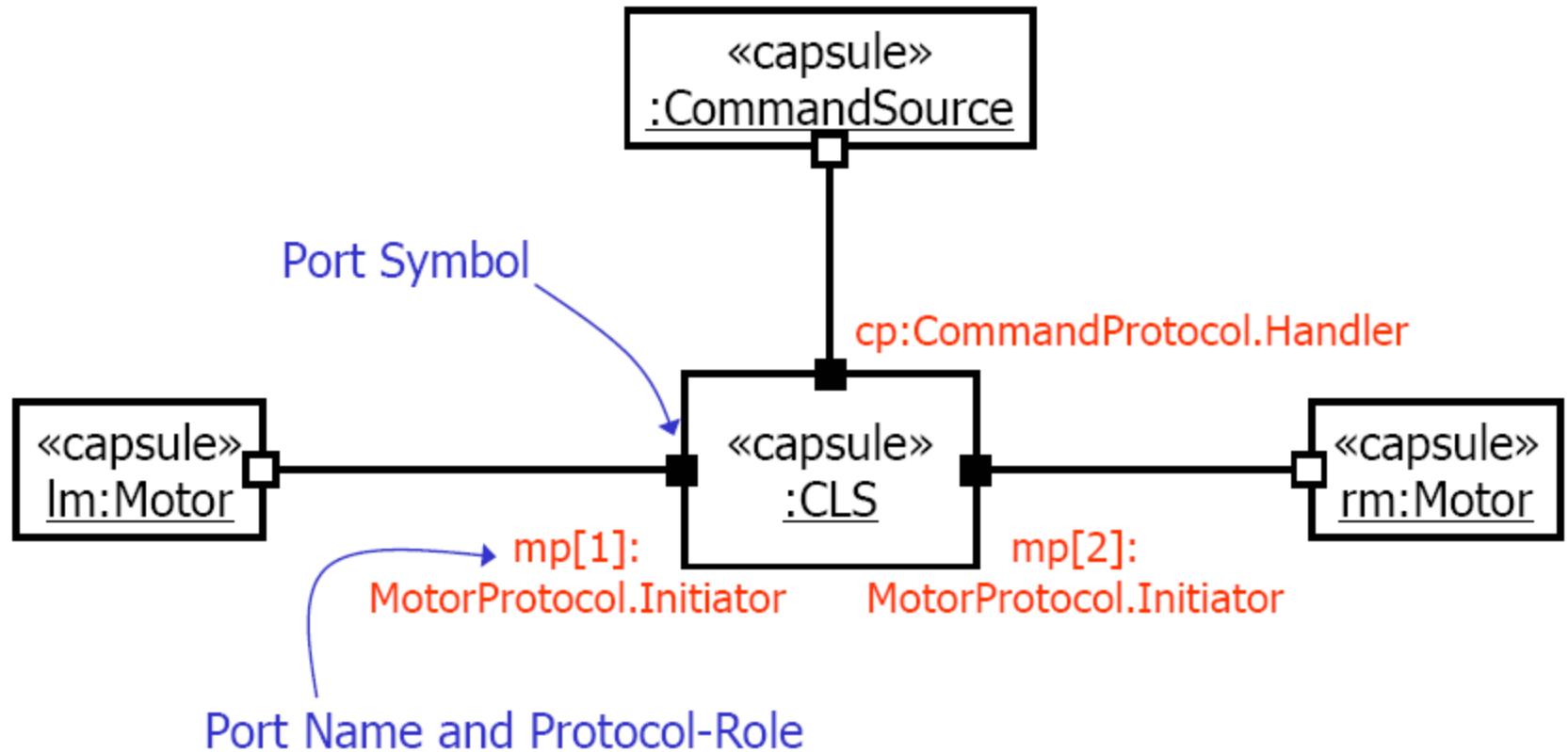


UML

► Beaucoup d'extensions:

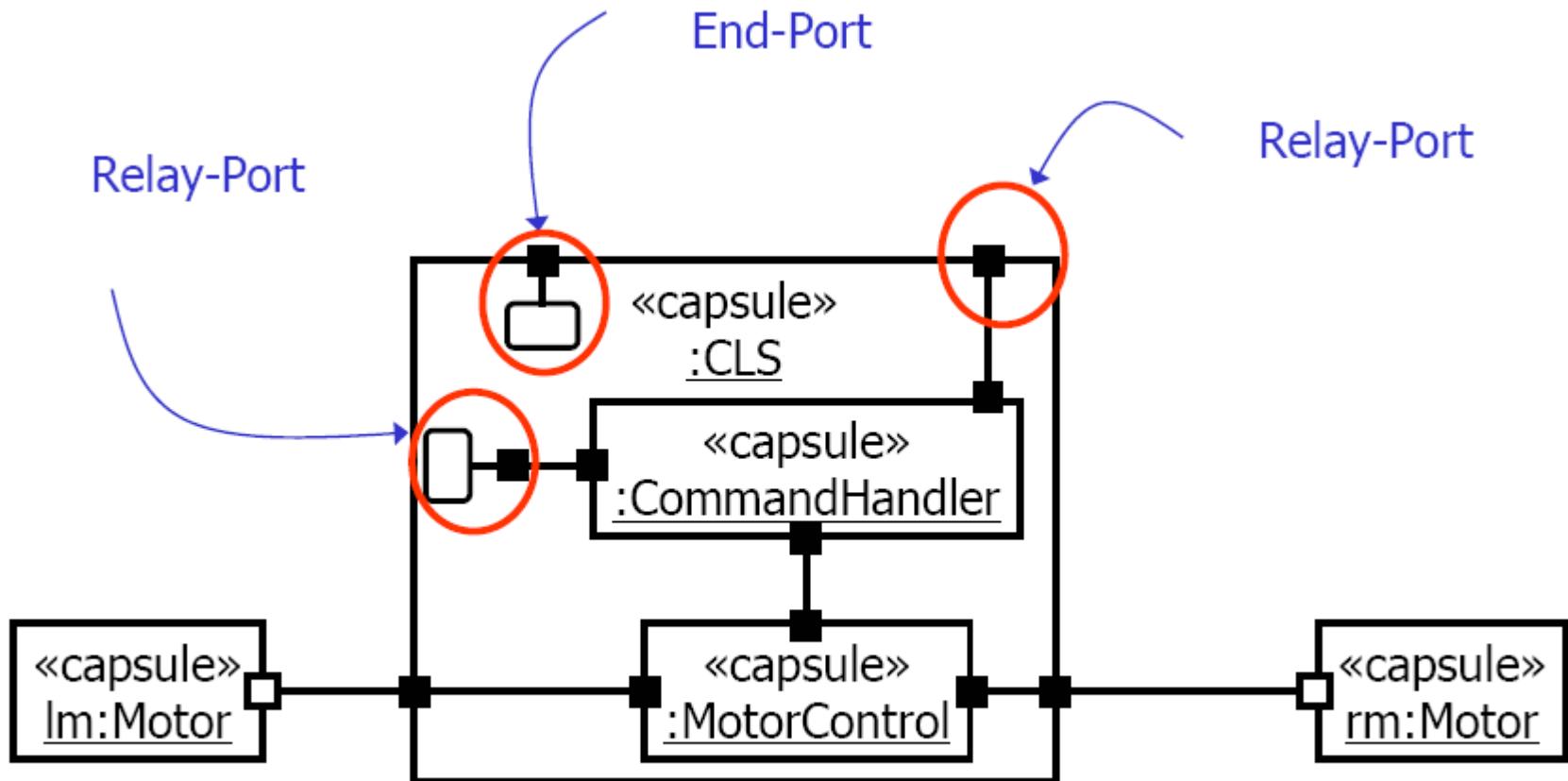
- SYML: Systems modeling langages
 - Langage plus léger
 - Permet de modéliser des performances, des contraintes mécaniques,...
- UML-RT (real time)
 - Ajout de concept comme la capsule (objet actif)
 - Ports
 - Collaboration de diagrammes,...

UML-RT



UML-RT

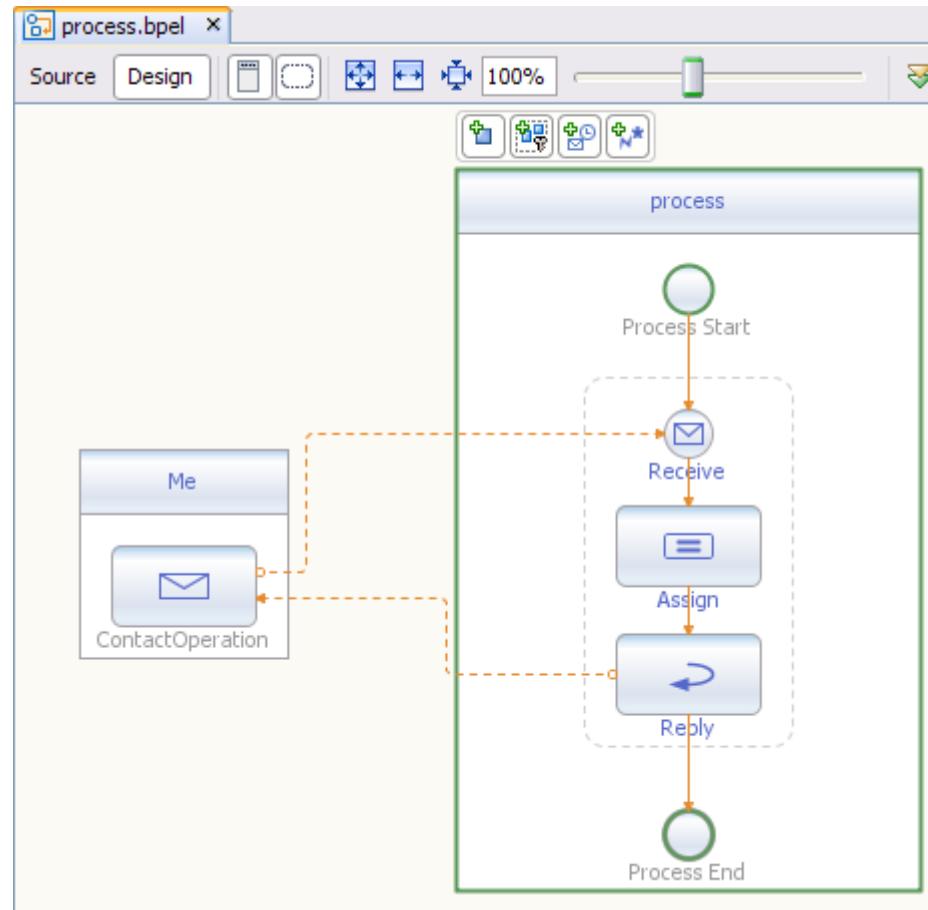
Ports et connecteurs



WS-BPEL

- ▶ Langage de modélisation d'architectures orientée services (SOA)
- ▶ Processus abstrait pour modéliser l'envoi/ l'attente de messages (orchestration)
- ▶ Utilisé avec les web services
- ▶ => donne un fichier XML directement utilisable par des moteurs de gestion des processus métiers, ces derniers se chargeant d'appliquer les règles du fichier en question.
 - On met en place les services web, on donne ce fichier qui indique comment interagissent les SW et ... c'est tout

WS-BPEL



Les automates

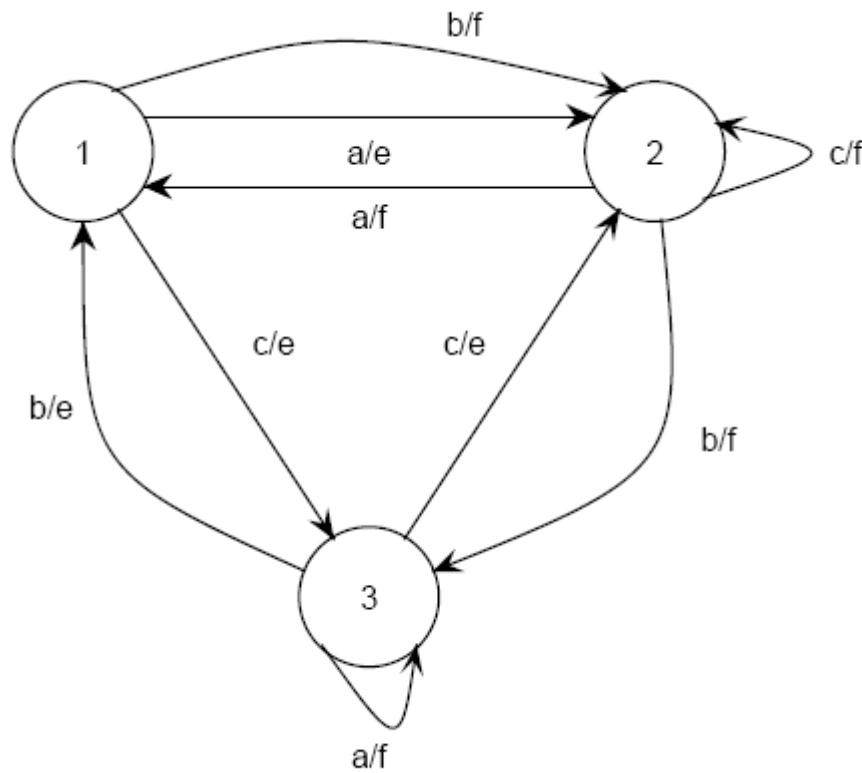
- ▶ Automates: très (très) utilisés pour modéliser les états d'un système d'un objet d'un composant.
- ▶ Utilisés dans la plupart des langages actuels (UML,...)

Les automates: FSM

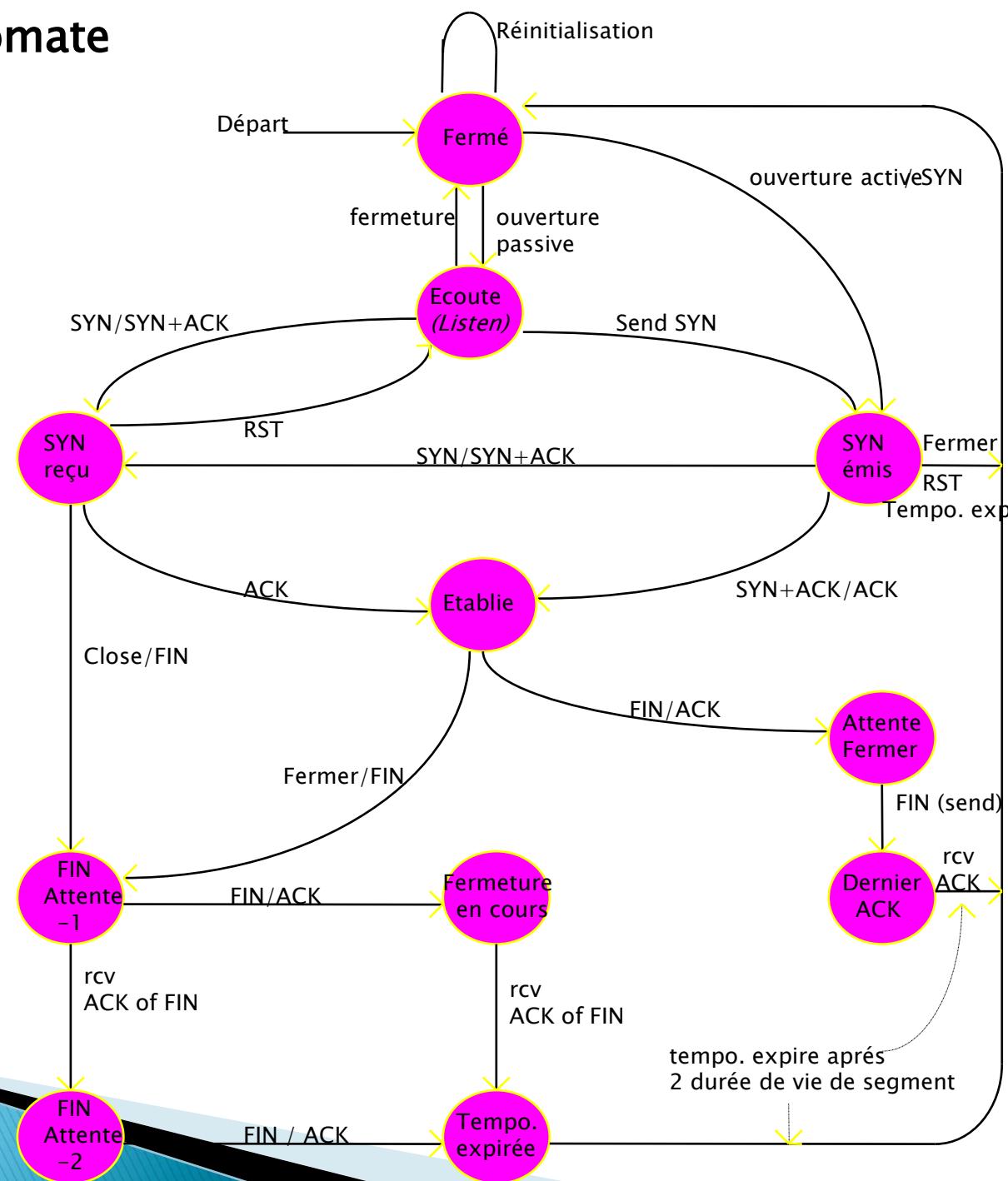
Définition 2.1.2 (FSM) Une *FSM* A est un quintuplet $\langle s_0, S, I, O, T \rangle$ où :

- $s_0 \in S$ est l'état initial,
- S est un ensemble non vide d'états,
- I est l'ensemble des entrées,
- O est l'ensemble des sorties,
- T est un ensemble non vide de transitions tel que $T \subseteq S \times I \times O \times S$. Le tuple $\langle s, i, o, s' \rangle$ représente une transition qui part de l'état s et va vers l'état s' . Celle-ci est étiquetée par une entrée et une sortie qui symbolisent le fait que pour passer à l'état s' , il est nécessaire d'envoyer l'interaction i et de recevoir l'interaction o . Une telle transition est communément notée par $s \xrightarrow{i/o} s'$.

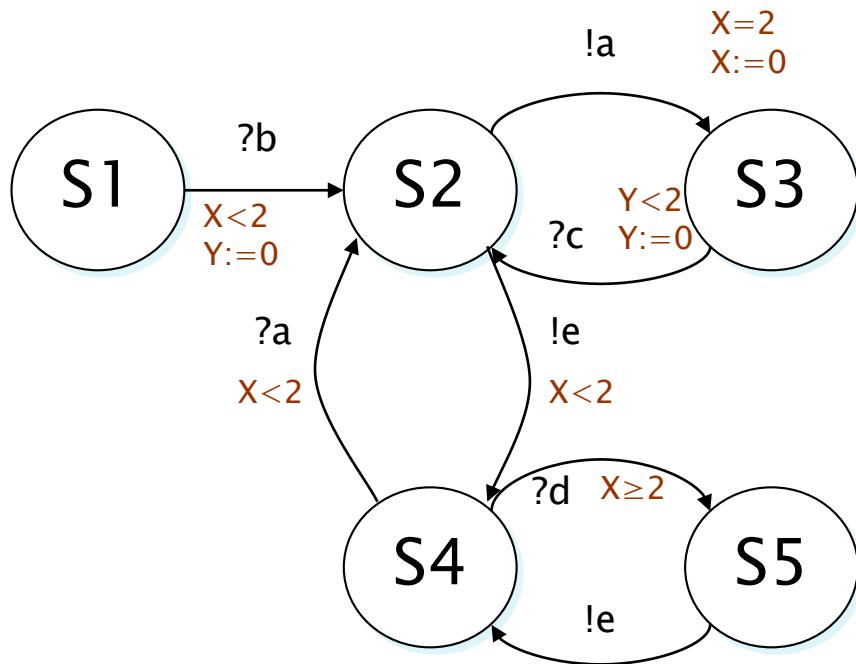
Les automates: FSM



TCP : L'automate



Les automates: automates temporisés

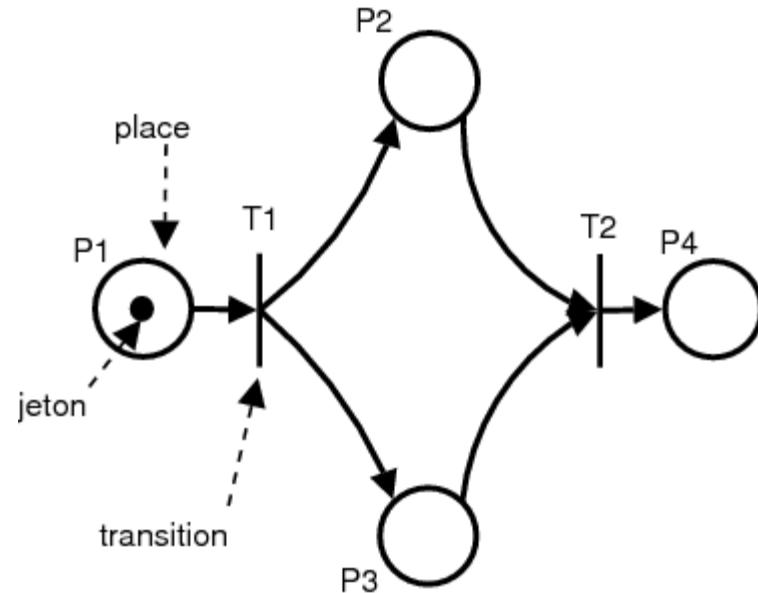


- Automate temporisé d'Alur et Dill composé :
- D'états, de transitions, de symboles,
- D'horloges réinitialisables,
- De contraintes temporelles

- Inconvénient : nombre d'états, du point de vue comportemental et temporel, infinis

Les RdP (réseaux de Petri)

- ▶ Un réseau de Pétri est un graphe représentant les relations entre trois ensembles d'éléments :
 - places
 - transitions
 - arcs



Beaucoup d'autres langages...

- ▶ SDL (specification and description language)
 - Proche de UML-RT par beaucoup d'aspects
- ▶ LOTOS
 - Pour syst. Distribués et paralleles
 - Notion de dendez vous et d'actions
 - Complexe
- ▶ Logiques,...
- ▶ Estelle, BPML, langage B, VRML (réalité virtuelle),

Et d'outils

- ▶ Beaucoup d'outils permettent de décrire des modèles
 - Uml: netbeans, eclipse, argouml,...
- ▶ Rappel:
- ▶ VOUS POUVEZ GENERER AUTOMATIQUEMENT LES CLASSES ET PARFOIS DU CODE
- ▶ Parfois vous pouvez pratiquer directement la phase de vérification et générer automatiquement les tests

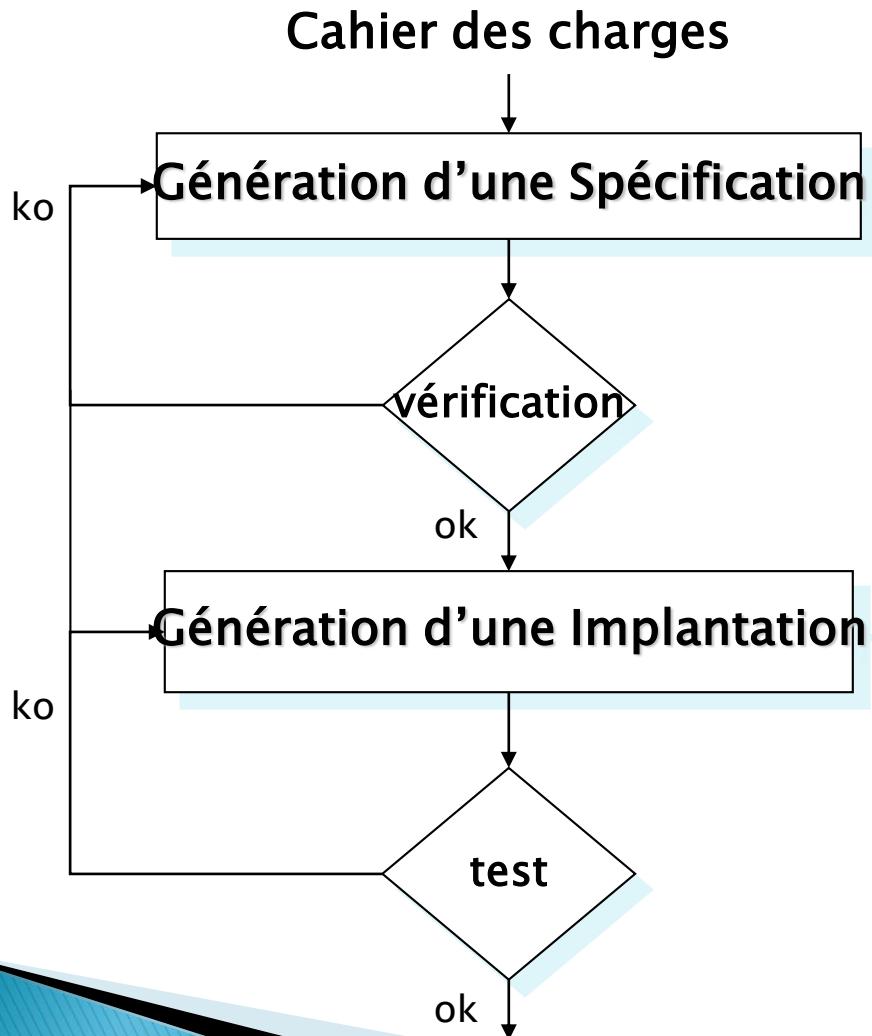
vérification

Preuve, model checking

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. »

La vérification



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...)

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 $M, s \models p$ (satisfait).

La vérification

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

Le test

Définitions, Types

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 défauts du fonctionnement de l'implantation du système.
- ▶ Différents types de test : conformité, robustesse, performance, interopérabilité, ...

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"

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

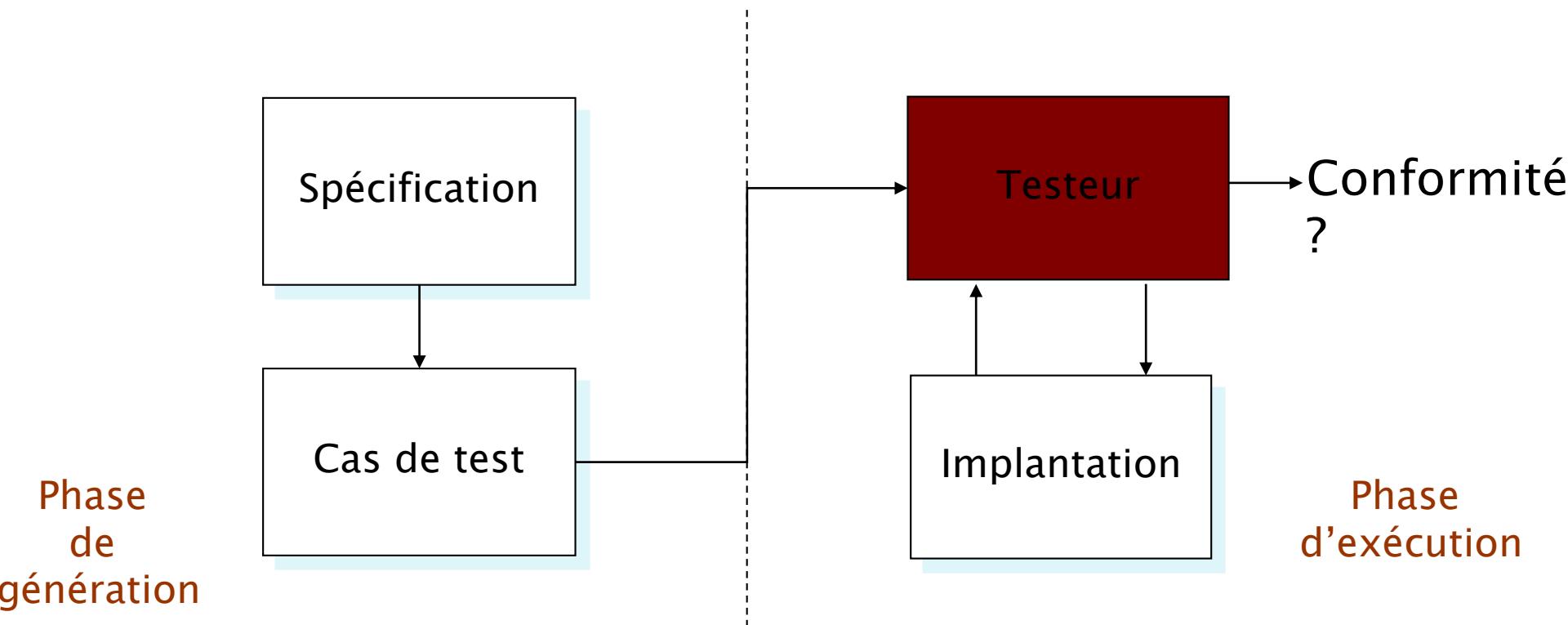
Les boites

- ▶ 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)

Les boites

- ▶ Test en boite grise
- ▶ Connaissance d'une partie du système, de l'environnement,...
 - Ex: accès au serveur web où est hébergé une application web
- ▶ Connaissance d'hypothèses sur l'application
 - Ex: l'application ne prend jamais de mails invalides

Phases de test classique (\neq TDD)



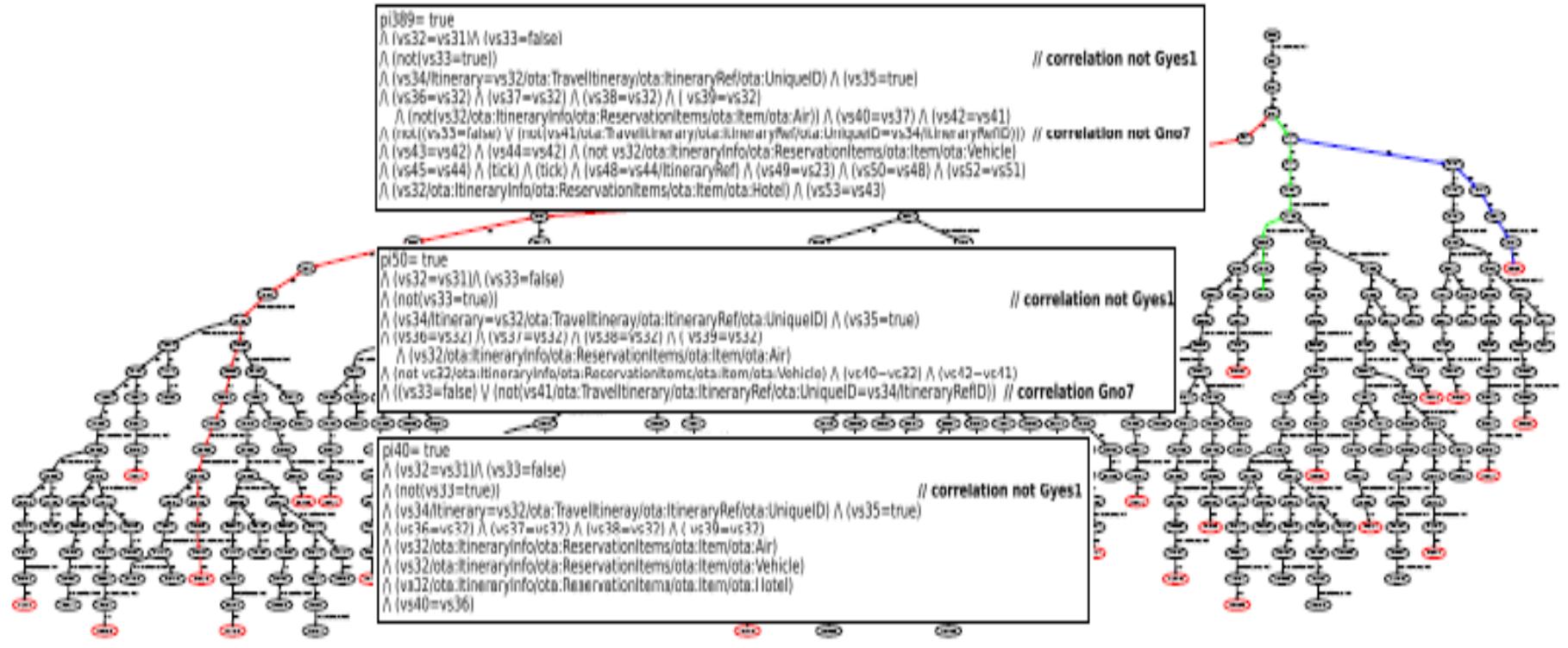
Cas de test

▶ 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); } }
```

Cas de test



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'accessibilité:** rassemble les approches en boîte 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.

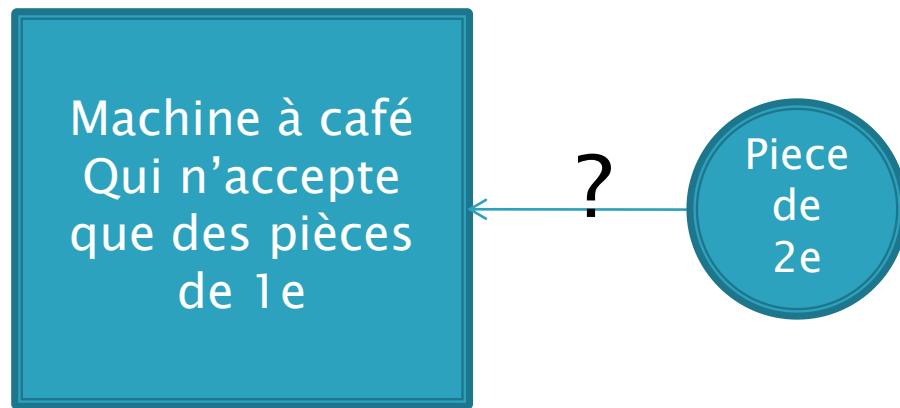
Exemples de types de test

▶ 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

Exemples de types de test

- ▶ Tests de Robustesse
- ▶ Exemple:



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”
- Et le TDD? Tests faits avant MAIS limité: tester des classes (pas des prog complets) => voir test unitaires

Test de conformité

- ▶ Beaucoup, beaucoup de méthodes
 - Sur divers modèles
 - Par rapport à diverses architectures (syst. , syst. Distribué, temps réel, stockastique, services,...)
 - Test unitaire, de regression, fonctionnel (entreprise), voir plus loin

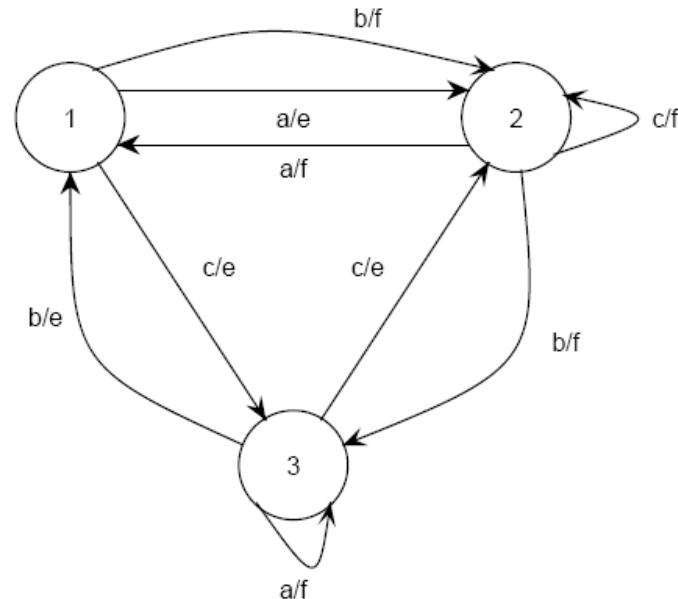
Test de conformité

▶ Test basé modèle:

- spécification décrite par langage formel (mathématique),
- automatisation de la gen. des tests)
- Relations d'implantations: on donne une équivalence entre une specif. et une boîte noire (qu'on ne connaît pas)
- Plusieurs relations plus ou moins fortes (isomorphisme, équivalence de trace, ioco, etc.)

Test de conformité

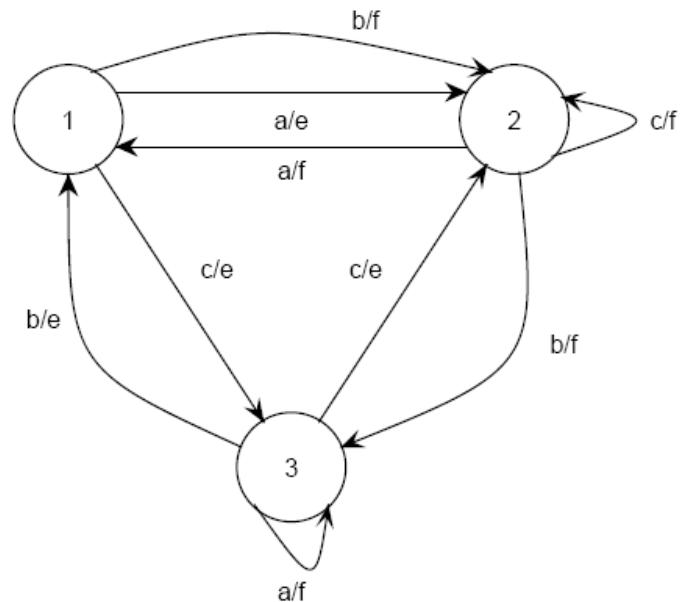
- ▶ Test basé modèle de conformité sur automate:
- ▶ Tour de transition: consiste à trouver une séquence minimale, parcourant au moins une fois chaque transition de la FSM



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

Test de conformité

- ▶ Test basé modèle 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)



Le test

En entreprise (unitaire, regression,
fonctionnel, outils, etc.)

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

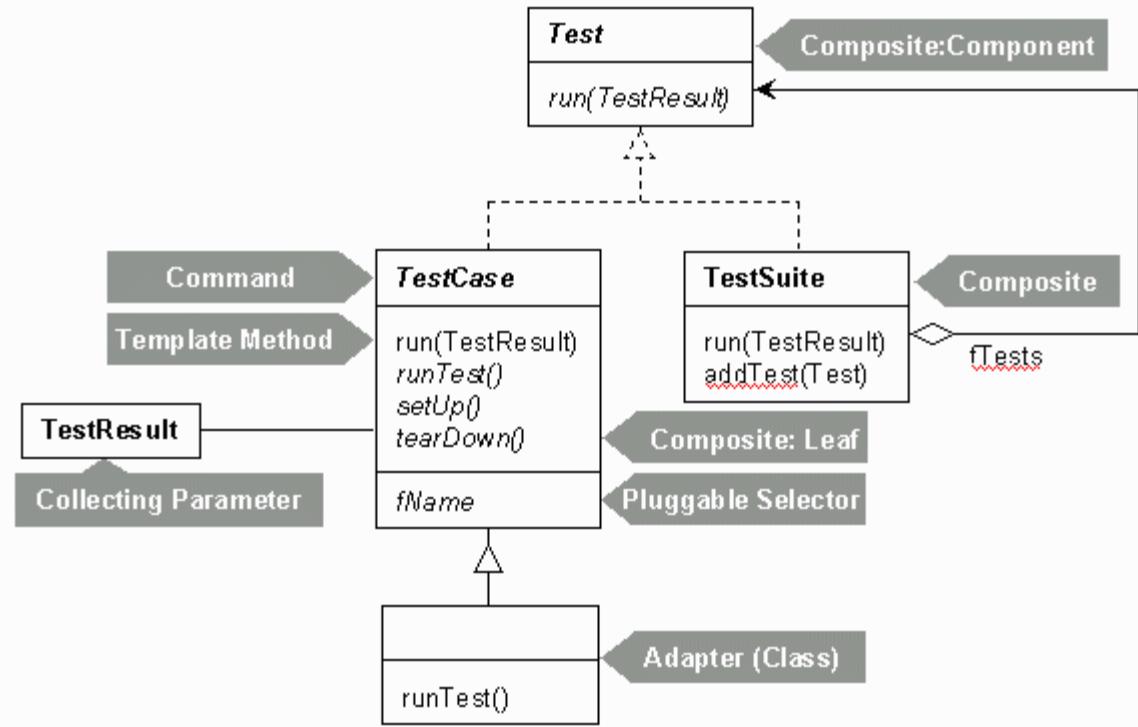
- Méthode permettant de s'assurer que la plus petite partie d'une application fonctionne correctement “en isolation”.
- Plus petite partie = classe, une librairie, souvent à définir...
- Exemple: JUNIT !

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

JUNIT

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



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

JUNIT

Assertion : expression booléenne supposée vraie.

Vérifier au fil du code que tout se passe comme attendu;

Ex:

```
AssertTrue(maClasse.isVisible());
```

JUNIT

- ▶ Programmeur doit manipuler les classes **Testcase** et **Testsuite** mais pas d'héritage nécessaire depuis Junit4
- ▶ Ajout d'annotations
 - @Test
 - //attente d'une exception:
 - @Test(expected=IllegalArgumentException.class)
 - //test d'une durée limitée
 - @Test(timeout=1000)

JUNIT

Exemple:

```
public class VectorTest « extends TestCase pas obligatoire » {  
  
    @Test  
    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); }  
}
```

JUNIT

```
public class TestFoobar extends TestCase{
```

initialisation

```
@BeforeClass  
    public static void setUpClass() throws Exception {  
        // Code exécuté avant l'exécution du premier test (et de la méthode  
        @Before)  
    }
```

Ex: Initialisation d'une base

@Before

```
public void setUp() throws Exception {  
    // Code exécuté avant chaque test
```

Ex: instantiation de classe ici

JUNIT

libération

```
@After  
public void tearDown() throws Exception {  
    // Code exécuté après chaque test  
}
```

```
@AfterClass  
public static void tearDownClass() throws Exception {  
    // Code exécuté après l'exécution de tous les tests  
on fait souvent appel à une méthode cleanup  
}
```

```
@Test //un cas de test  
public void test()  
{  
    assertTrue(true);  
}  
}
```

JUNIT

D'autres annotations:

- ▶ `@Ignore`, pour rendre indisponible un test

JUNIT assertions par contrats (hamcrest)

- ▶ Assertions classiques:
 - Pas de message explicite de l'échec
 - Pas de réelle combinaison d'assertion possible
- ▶ Assertions par contrats => assertThat
 - Facilite la lisibilité:
- ▶ Assertions de base:
 - Assert.assertThat("texte", IsSame.sameInstance("texte"));
 - Assert.assertThat("texte", IsEqual.equalTo("texte"));
 - Assert.assertThat("texte", IsInstanceOf.instanceOf(String.class));
 - Assert.assertThat(ITexte, IsNull.isNullValue());
 - **Assert.assertThat("texte", AnyOf.anyOf(
 IsInstanceOf.instanceOf(Integer.class),
 IsEqual.equalTo("Texte")
));**

JUNIT assertions par contrats

- ▶ Messages explicites

```
@Test  
public void  
testExplicite() {  
int  
x = 50;  
assertThat(x, OrderingComparisons.lessThan(10));  
}
```

- ▶ Message:

```
java.lang.AssertionError:  
Expected: a value less than <10>  
got: <50>  
at org.junit.Assert.assertThat(Assert.java:750)  
at org.junit.Assert.assertThat(Assert.java:709)  
at TestNonExplicite.testExplicite(TestNonExplicite.java:29)
```

JUNIT assertions par contrats

▶ Combinaison de contrats

- AllOf.allOf(les contrats)
- AnyOf.anyOf(les contrats)

▶ Exemple:

```
Assert.assertThat("texte", AnyOf.anyOf(  
    IsInstanceOf.instanceOf(Integer.class),  
    IsEqual.equalTo("Texte")  
));
```

JUNIT assertions par contrats

- ▶ Bibliothèque des contrats:

org.hamcrest.beans

org.hamcrest.collection

org.hamcrest.core

org.hamcrest.number

org.hamcrest.object

org.hamcrest.text

org.hamcrest.xml

JUNIT

Les suppositions:

Conditions qui doivent être remplies avant test.

Si non, pas d'erreur mais le test s'arrête.

```
final File lFile = new File("fichier.txt");
    Assume.assertTrue(lFile.exists());
```

Les suppositions:

- ▶ assumeNoException
- ▶ vérifie qu'une opération s'est déroulée sans lever de
- ▶ Throwable
- ▶ assumeNotNull
- ▶ vérifie qu'aucun paramètre n'est nul
- ▶ assumeThat
- ▶ vérifie qu'une condition par contrat est respectée
- ▶ assumeTrue
- ▶ vérifie que le paramètre est vrai

JUNIT Exécution

- ▶ Exécution des tests:
- ▶ Exécution 1 par un 1 ou en utilisant une suite de test
- ▶ Exécution 1 par 1:

```
public class TestRunner {  
    public static void main(String[] args) {  
        Result result = JUnitCore.runClasses(TestJUnit.class);  
        for (Failure failure : result.getFailures()) {  
            System.out.println(failure.toString());  
        }  
        System.out.println(result.wasSuccessful());  
    }  
}
```

JUNIT Exécution

- ▶ Suite de test:
- ▶ Classe vide annotée avec @RunWith
- ▶ possibilité de changer la classe d'exécution de la suite dans le @RunWith
- ▶ Suite.class par défaut
- ▶ @RunWith(Suite.class)
- ▶ @SuiteClasses({Tests1.class, Tests2.class})
- ▶ public class TousLesTests
- ▶ {}

JUNIT Exécution

- ▶ Mais aussi @RunWith(Parameterized.class)
- ▶ exécuteur de test acceptant les paramètres
- ▶ contient un constructeur avec les données et le résultat
- ▶ contient une méthode renvoyant une collection des paramètres : données et résultat
- ▶ annotée avec @Parameters

JUNIT Exécution

```
@RunWith(Parameterized.class)
public class TestsParametres {
    private int a,b, resultat;

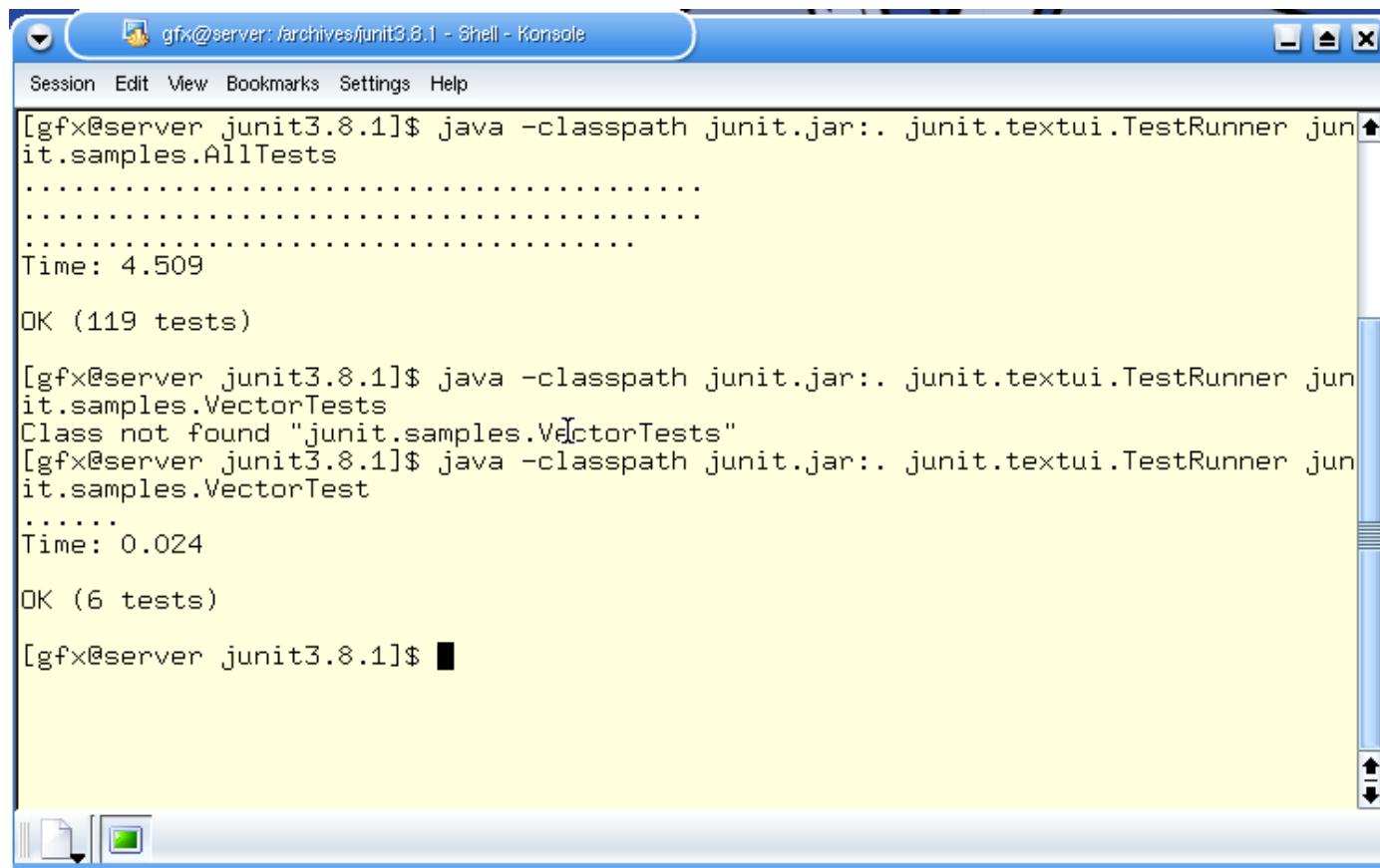
    public TestsParametres(int a, int b, int resultat){
        this.a = a; this.b = b; this.resultat = resultat; }

    @Parameters
    public static Collection getParametresAddition()
    {
        return Arrays.asList(new Object[][]{{1,2,3}, {3,5,8}, {9,6,15}});
    }

    @Test
    public void additionTest()
    {
        Calculatrice calc = new Calculatrice();
        Assert.assertEquals(resultat, calc.additionner(a, b));
    }
}
```

JUNIT

Interface texte:



The screenshot shows a terminal window titled "gfx@server: /archives/junit3.8.1 - Shell - Konsole". The window contains the following text output from JUnit tests:

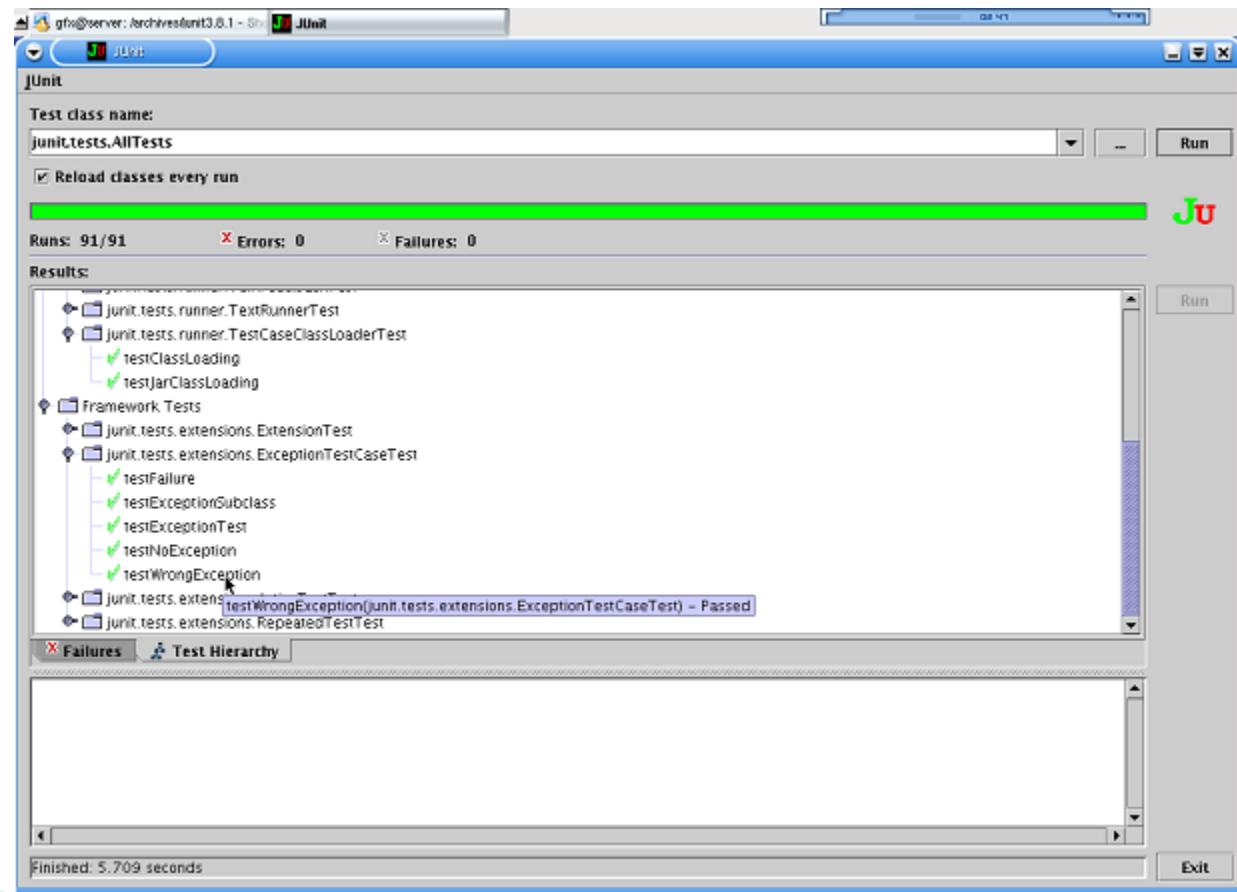
```
[gfx@server junit3.8.1]$ java -classpath junit.jar:. junit.textui.TestRunner junit.samples.AllTests
.
.
.
Time: 4.509
OK (119 tests)

[gfx@server junit3.8.1]$ java -classpath junit.jar:. junit.textui.TestRunner junit.samples.VectorTests
Class not found "junit.samples.VectorTests"
[gfx@server junit3.8.1]$ java -classpath junit.jar:. junit.textui.TestRunner junit.samples.VectorTest
.
.
.
Time: 0.024
OK (6 tests)

[gfx@server junit3.8.1]$
```

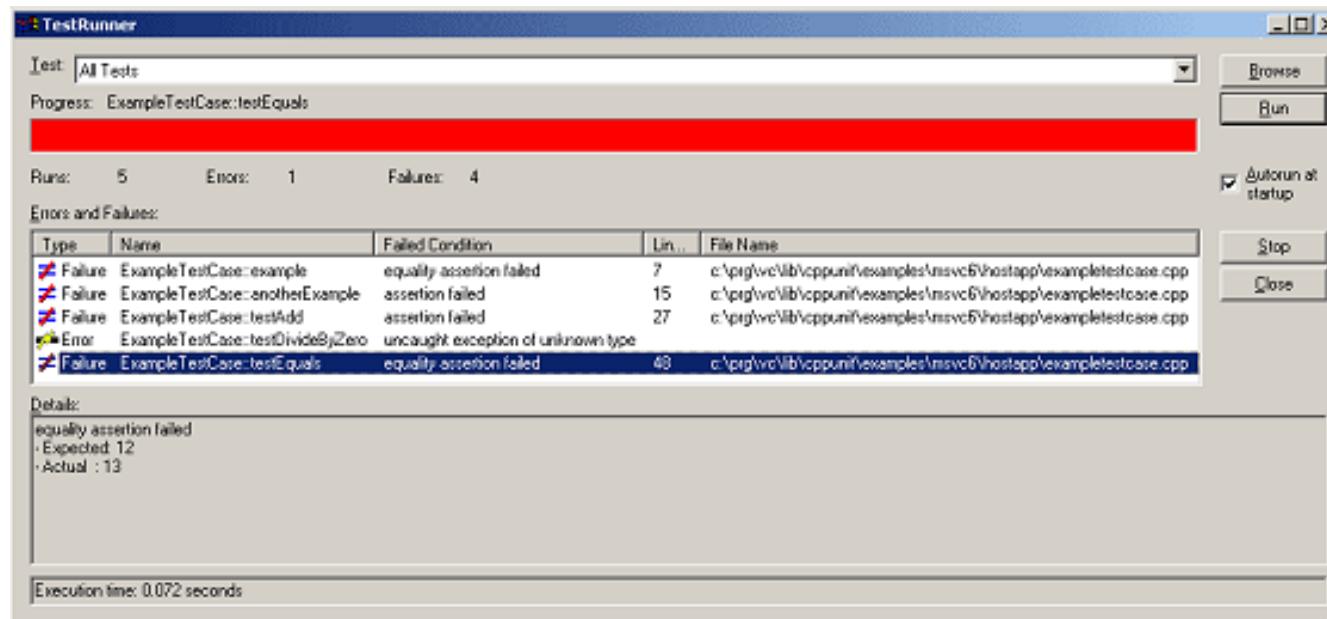
JUNIT

▶ Interface graphique



X-unit ?

En C++ : CPPUNIT



MAIS aussi en php, en c#, etc.

PHPUnit

- ▶ <https://phpunit.de/>
- ▶ Exemple:

```
<?php
class test extends TestCase {
    function test_pass(){
        $boolean = false;
        $this->assertFalse($boolean);
    }
} ?>
```

PHPUnit

- ▶ Notion de dependence:

```
/**  
 * @depends Previousmethod  
 */
```

- ▶ Test des sorties

```
$this->expectOutputString('foo');  
print 'foo';
```

- ▶ Exceptions:

```
/**  
 * @expectedException PHPUnit_Framework_Error  
 */
```

PHPUnit

- ▶ Fixture
 - setUp(), tearDown(), setUpBeforeClass(), tearDownAfterClass()
- ▶ Suites de test:

```
<phpunit>
  <testsuites>
    <testsuite name="money">
      <directory>tests</directory>
      // ou <file>tests/test1.php</file>
    </testsuite>
  </testsuites>
</phpunit>
```

Intégration de JUNIT dans Eclipse

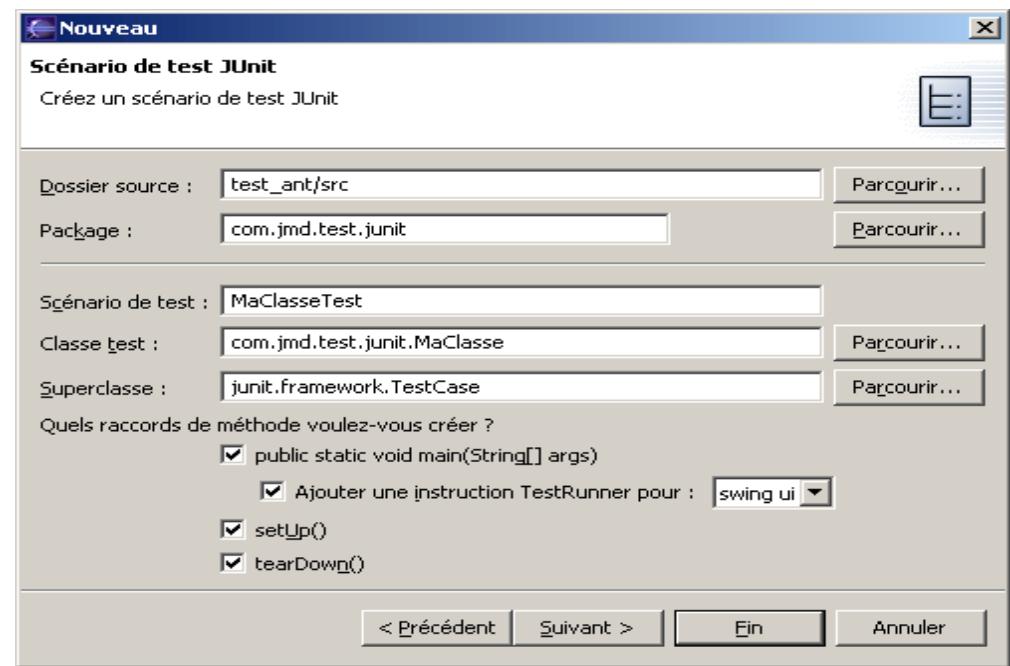
- ▶ Ajout de junit.jar dans classpath



- ▶ Crédit d'un test:

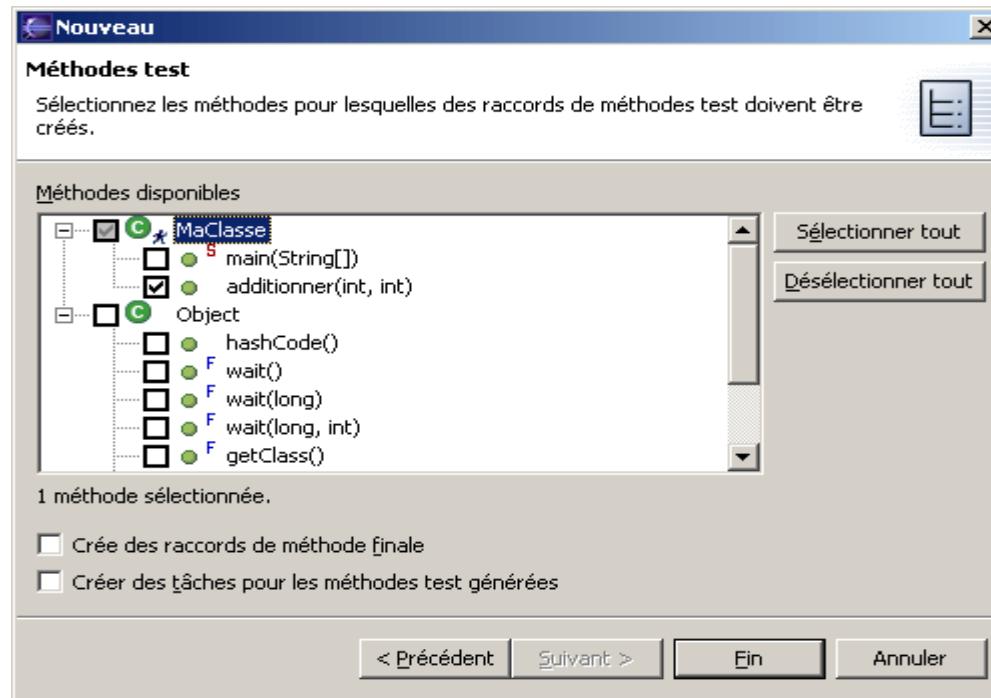
“nouveau cas de test Junit”

Il existe aussi des suites !!!



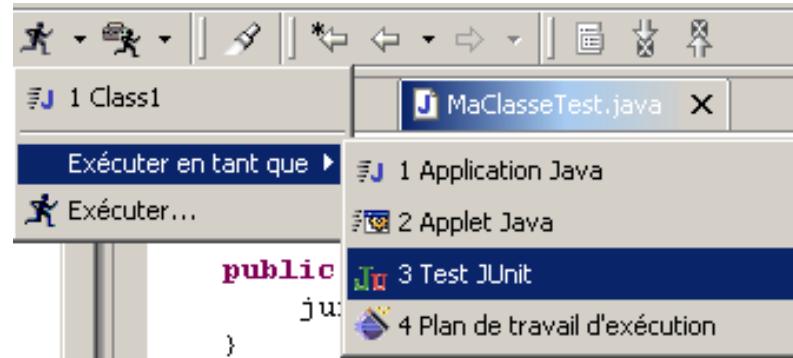
Intégration de JUNIT dans Eclipse

► sélection des méthodes



Intégration de JUNIT dans Eclipse

- ▶ Exécution de votre classe en tant que classe Junit:



TestNG (testng.org)

- ▶ Extension de JUnit
 - Permet de créer des groupes de tests, de méthodes
 - Peut // les tests, permettre de tester sans concurrence (Threadsafe)
 - définir un ordre d'exécution des tests ou pas, etc.
 - Configurations de tests,
 - Data driven testing
- ▶ avec
 - des annotations (extension de @before et @after)
 - Des nouvelles assertions
- ▶ Disponible comme plugin Eclipse

TestNG (testng.org)

- ▶ Exemple: dépendance de méthodes

```
import org.testng.annotations.Test;
```

```
public class App {
```

```
    @Test
```

```
    public void method1() {
```

```
        System.out.println("This is method 1");
```

```
        throw new RuntimeException();
```

```
}
```

```
    @Test(dependsOnMethods = { "method1" })
```

```
    public void method2() {
```

```
        System.out.println("This is method 2");
```

```
}
```

Méthode 2 est dépendante de Méthode 1, si le test de méthode 1 est FAIL -> le test de méthode 2 ne se fait pas

Outils associés aux tests unitaires

- ▶ Tests sur bases de données
 - DBUnit (en java) qui permet d'effectuer d'initialiser une Bd, de la stocker, d'effectuer des comparaisons entre ce qui est produit en base et ce qui est attendu, et de reinitialiser une BD avant chaque test
- ▶ Création d'objets simulés lorsque d'autres objets, en plus de ceux testés, sont nécessaires
 - Objets mockés
 - Mockito (google)

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 raffine 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)

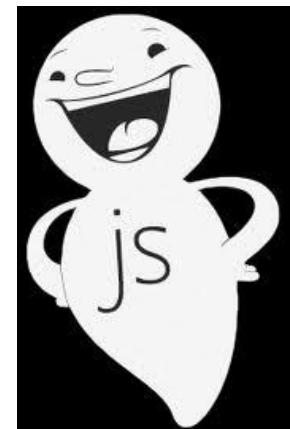
- ▶ Test fonctionnel
 - Test de l'application complète vue par l'utilisateur
- ▶ En Web, tests remplissent pages Web et vérification de résultats (Selenium-> création de suites de test, enregistrement, etc.)
- ▶ Ecriture de tests fonctionnels

```
public class temp script extends SeleneseTestCase {  
    public void setUp() throws Exception {  
        setUp("http://localhost:8080/", "*iexplore");  
    }  
    public void testTemp script() throws Exception {  
        selenium.open("/BrewBizWeb/");  
        selenium.click("link=Start The BrewBiz Example");  
        selenium.waitForPageToLoad("30000");  
        selenium.type("name=id", "bert");  
        selenium.type("name=Password", "biz");  
        selenium.click("name=dologin");  
        selenium.waitForPageToLoad("30000");
```



Test de conformité d'applications (en entreprise)

- ▶ Test fonctionnel
- ▶ CasperJS:
 - outil de script de navigation et de test
 - Scenario de navigation de haut niveau: simulation d'une personne manipulant une page Web
 - Test de Dom, clicks, téléchargements, captures d'écran, etc.



Test de conformité d'applications (en entreprise)

- ▶ Test fonctionnel
 - Avec android ? -> robotium, appium (récent, à voir car Android et ios)

Robotium (it's like selenium but for Android)



Exemple de test (basé sur JUNIT):

```
public void testPreferencesSaved() throws Exception {  
  
    solo.sendKey(Solo.MENU);  
    solo.clickOnText("More");  
    solo.clickOnText("Preferences");  
    solo.clickOnText("Edit File Extensions");  
    Assert.assertTrue(solo.searchText("rtf"));  
  
    solo.clickOnText("txt");  
    solo.clearEditText(2);  
    solo.enterText(2, "robotium");  
    solo.clickOnButton("Save");  
    solo.goBack();  
    solo.clickOnText("Edit File Extensions");  
    Assert.assertTrue(solo.searchText("application/robotium"));  
  
}
```

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

A voir également

- ▶ Arquillian (<http://arquillian.org/>, redhat): test unitaire, d'intégration, d'acceptation
- ▶ Plateforme de test (Java, J2EE) permettant:
 - Lancer des tests dans le vrai conteneur (serveur, BD etc.) et d'y injecter ce que l'on veut (BD, servlet, etc.)
 - De gérer le conteneur, de le lancer, de deployer,etc.
 - De construire un jar de test contenant uniquement les classes à tester via ShrinkWrap
- ▶ Extensions *drone* et *graphene* permettent de tester appli. Web avec l'envoi de requêtes (gestion client et serveur combinées)



Test de conformité d'applications (en entreprise)

- ▶ Test de non régression:
 - On teste si la modification de code n'a pas eu d'impact sur les fonctionnalités existantes
 - On utilise pour cela les tests unitaires existants et on vérifie qu'aucune nouvelle erreur n'est détectée

Test de conformité d'applications (en entreprise)

- ▶ Test de non régression:
 - Utilisation d'outils d'intégration continue:
- ▶ Ex: jenkins



Test de conformité d'applications (en entreprise)

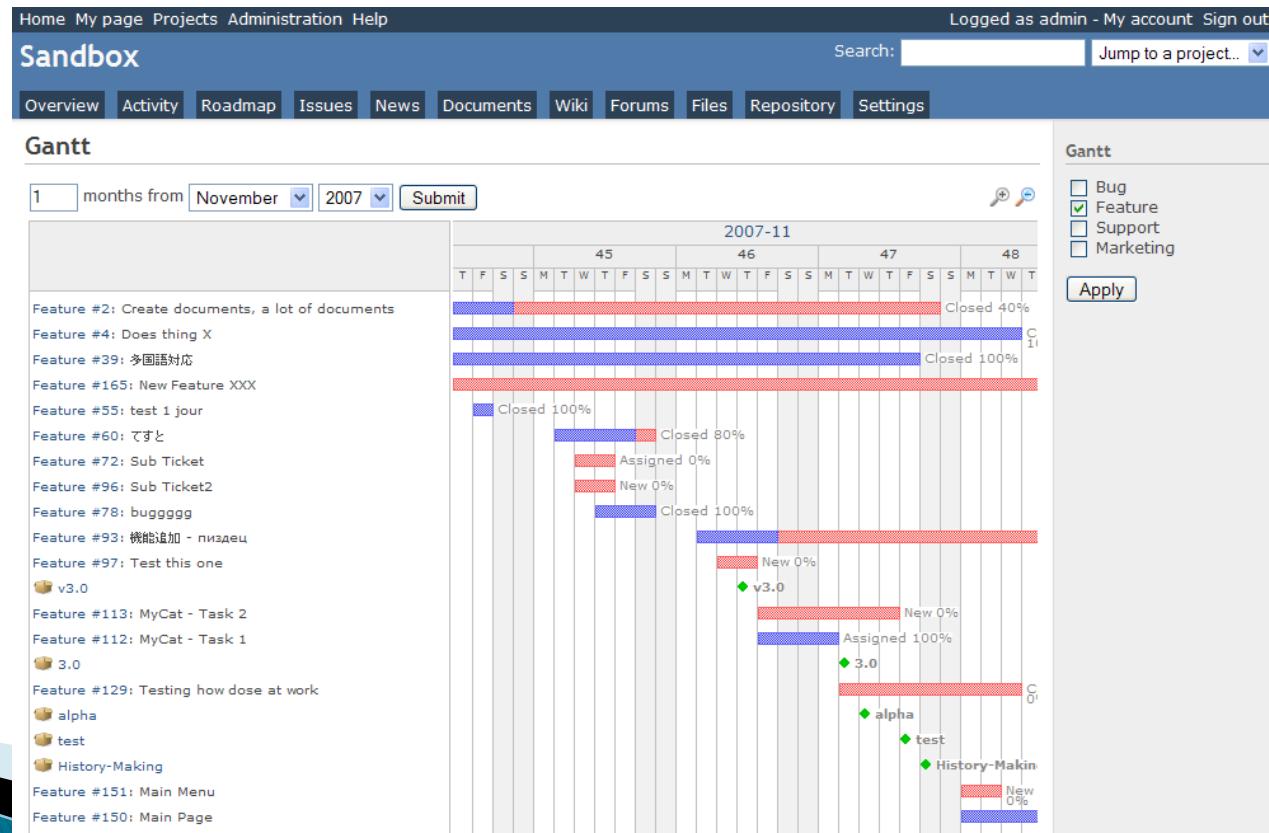
▶ Test de sécurité

- 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

Environnement de test

- ▶ Besoin de monter tout un environnement pour tests unitaires, d'intégration, etc.
 1. Environnement de gestion de projet (redmine ou gitlab mais moins complet)

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



Environnement de test

- ▶ Besoin de monter tout un environnement pour tests unitaires, d'intégration, etc.
2. Travail collaboratif, stockage: svn, GitLab, GitHub

GitLab / gitlabhq

Files Commits Network Issues 37 Merge Requests 1 Wiki Wall Settings

master > gitlabhq

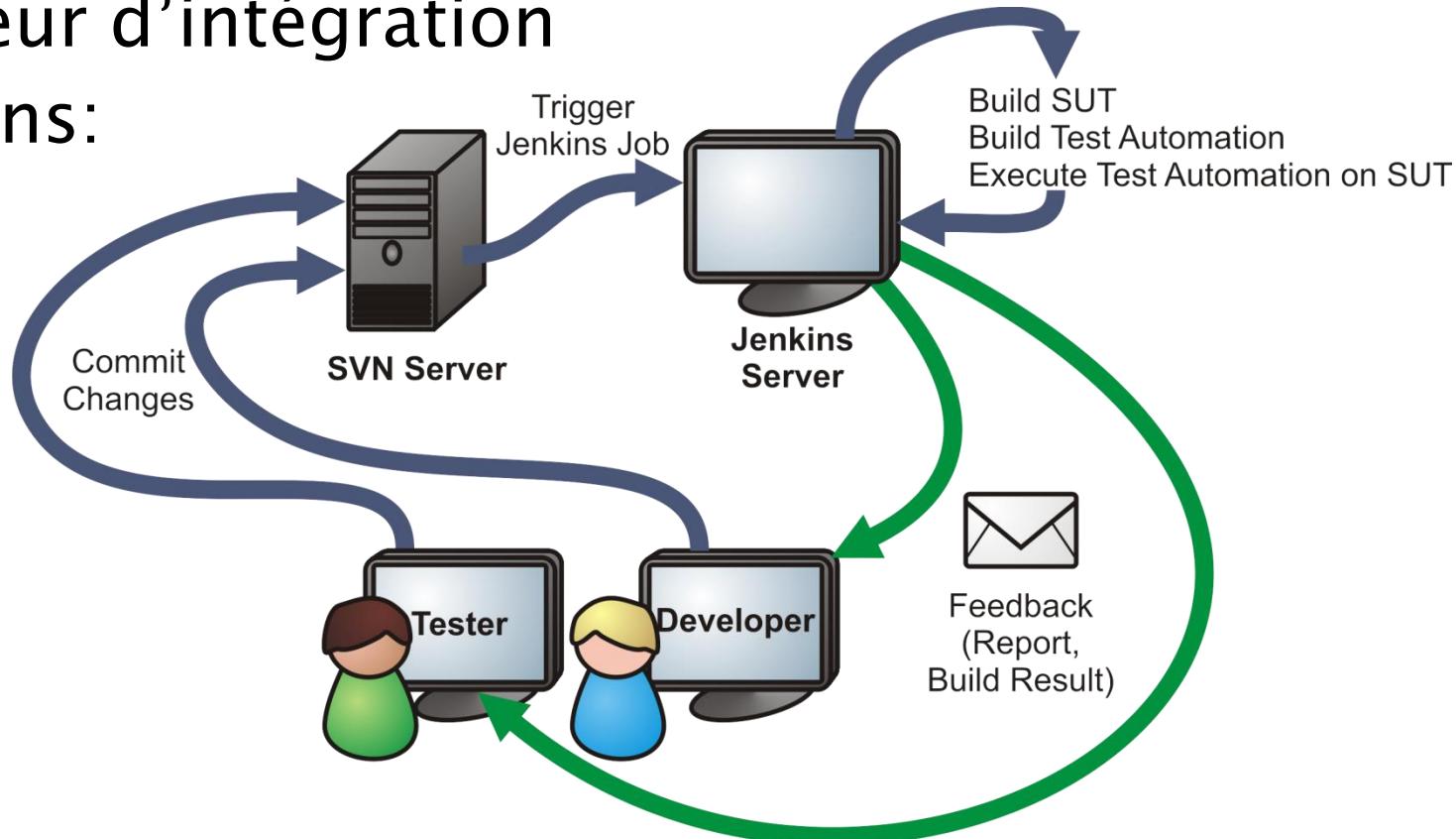
Name	Last Update	Last Commit
app	about 5 hours ago	Dmitriy Zaporozhets Override Issue opened scope. Now reopened issues also included
config	2 days ago	Dmitriy Zaporozhets refactor Issues.js. Remove unused actions. Respect filters while searc...
db	4 days ago	Dmitriy Zaporozhets Prevent same branch in MR seeds
doc	2 days ago	Dmitriy Zaporozhets Merge pull request #3522 from dplarson/correct-install-docs
features	2 days ago	Dmitriy Zaporozhets Fixed mr filter tests
lib	4 days ago	Dmitriy Zaporozhets Backup/restore wiki repos too
log	over 1 year ago	gitlabhq init commit
public	about 1 month ago	Dmitriy Zaporozhets Add logo to deploy.html
script	3 months ago	Dmitriy Zaporozhets Fix sidekiq check and added script/check
spec	2 days ago	Dmitriy Zaporozhets remove outdated routing specs
tmp	7 months ago	Robert Speicher Add tmp/.gitkeep file to ensure tmp folder exists on clone
vendor	23 days ago	Kevin Lyda Result of misspellings run.
.foreman	about 1 year ago	Dmitriy Zaporozhets complete hooks for post receive
.gitignore	5 days ago	Dmitriy Zaporozhets Replace unicorn with Puma
.rspec	14 days ago	AndrewBxx8 Spork support added
.simplecov	3 months ago	Dmitriy Zaporozhets organize simplecov
.travis.yml	40 minutes ago	Dmitriy Zaporozhets Include ruby 2.0 in travis
CHANGELOG	3 days ago	Dmitriy Zaporozhets New entries in CHANGELOG



Environnement de test

3. Serveur d'intégration

▶ Jenkins:



+ plugin de test qualité sur le code
(redondance, etc.) mais moins puissant que Sonar

Environnement de test

4. IDE de test (intégré à l'ide de dev ou pas)

- Infinitest,
- Eclipse,
- Visual studio,
- Selenium ide,
- Archilian,

Mais aussi...

- ▶ Qualité du code écrit, sécurité, etc.
 - Ex:Format, boucles, commentaires,...
 - **Jdepend** (automatically measure the quality of a design in terms of its extensibility, reusability, and maintainability to manage package dependencies effectively),
 - **Jtest(parasoft)**
 - **Jcrasher** (test robustesse automatique en Java)
 - **SOAPIU** (test de services web rest et soap)
 - **HPQuality Center** (test d'intégration, de regression),
 - **Profiler** (Netbeans fuites mémoire)
 - **Testlink** (test d'integration)

Outils

- ▶ Outils de contrôle qualité: Sonar,
 - 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

Sonar screenshot showing a dashboard with a list of projects, a radar chart for rules compliance, and a violations summary.

Projects Overview:

Project	Lines of code	Technical Debt ratio	Coverage	Duplicated lines (%)	Build time
Neo4j - Neo	24 707	10,5%	67,8%	2,4%	12/08/2009
Spring Security	24 995	16,4%	0,0%	0,0%	12/08/2009
XStream Parent	16 035	5,4%	77,3%	0,6%	12/08/2009
Struts 2	42 678		50,3%	2,0%	03/08/2009
Swizzle	6 507	15,3%	34,6%	0,9%	12/08/2009
Spring Web Services	26 530	8,2%	67,3%	4,6%	12/08/2009
XWiki Platform External POM	144 619		17,1%	1,3%	27/07/2009
Commons SCXML	7 331	7,3%	71,7%	7,2%	12/08/2009
swingjavabuilderext	1 167		7,5%	0,0%	13/07/2009
Jetspeed-2 Enterprise Portal	109 605	19,4%	0,0%	4,0%	12/08/2009
utPLSQL	13 769			2,5%	18/06/2009
Commons Configuration	19 703	8,1%	83,7%	1,2%	12/08/2009
EasyBeans	58 887		7,9%	6,1%	02/08/2009
Commons Collections	26 558	21,7%	0,0%	8,1%	12/08/2009
Commons Logging	2 695	28,7%	2,0%	5,7%	11/08/2009
Commons VFS	23 880	11,4%	39,7%	1,6%	12/08/2009
Apache CXF	175 295			2,1%	02/08/2009
Archiva	27 938	8,1%	62,7%	2,3%	09/08/2009
Apache Felix					04/25
Apache Maven 3.x	36 675	13,1%	29,3%	1,0%	04.27
Apache Axis2 - Root	173 281	20,2%	27,7%	5,5%	12/08/2009
Wicket Parent	110 997	7,5%	45,6%	1,2%	12/08/2009
XWork: Parent	21 421	10,4%	62,2%	2,5%	12/08/2009
Apache Akdera	52 643	8,4%		2,6%	11/08/2009

Rules compliance: 88,5% (Violations: 1 100)

Violations Summary:

Category	Count
Blocker	0
Critical	0
Major	981
Minor	119
Info	0