Gestion de configuration

Cours LPSIL IDSE

2013

Questions ?

Agenda

- **Gestion du code source**
- Gestion du build

Gestion du code source

Problèmes rencontrés ?

Pourquoi gérer le code source ?

Collaboration

- Pour permettre aux membres d'une équipe de **travailler ensemble sur un projet commun**

Pourquoi gérer le code source ?

Versioning

- Pour être capable de **gérer les différentes versions du logiciel**

Pourquoi gérer le code source ?

Rollback

- Pour être capable de **revenir à une version précédente à tout moment**

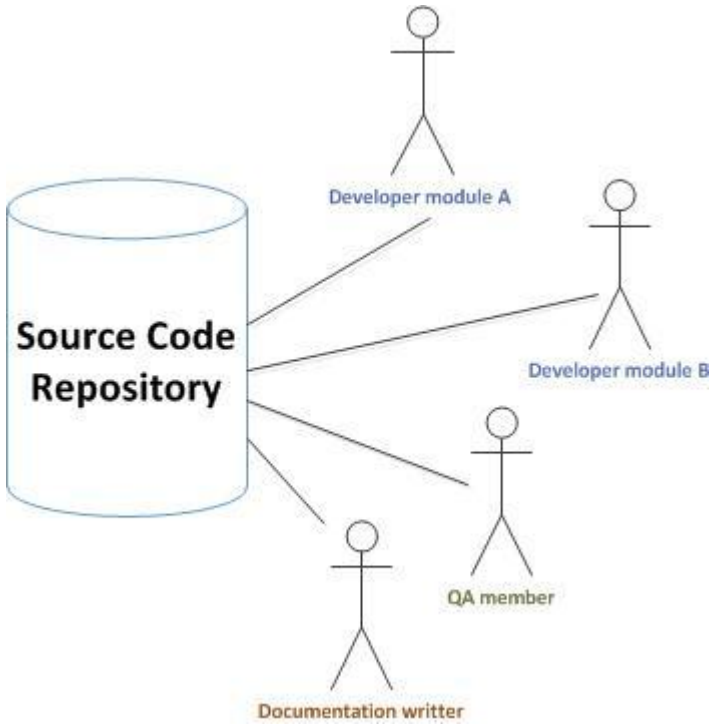
Collaboration

- Utilisation d'un dépôt (repository)
  - **Centralisé / Distribué**
  - **Partagé**

- Qui permet…
  - un **travail collaboratif** sur le code (gestion des modifications concurrentes)
  - **l'accès** à chacun **à la dernière version** du code
  - de connaître et tracer les **changements** effectués

Collaboration

- Lors d'un projet de développement logiciel, plusieurs intervenants contribuent sur la même base de code.
  - Développeurs,…
  - …mais aussi, ingénieurs qualité, ingénieurs packaging, ingénieurs tests, rédacteurs de documentation techniques, gestionnaire de releases, architectes, etc …
- Le respect des bonnes pratiques de gestion du code source est incontournable…

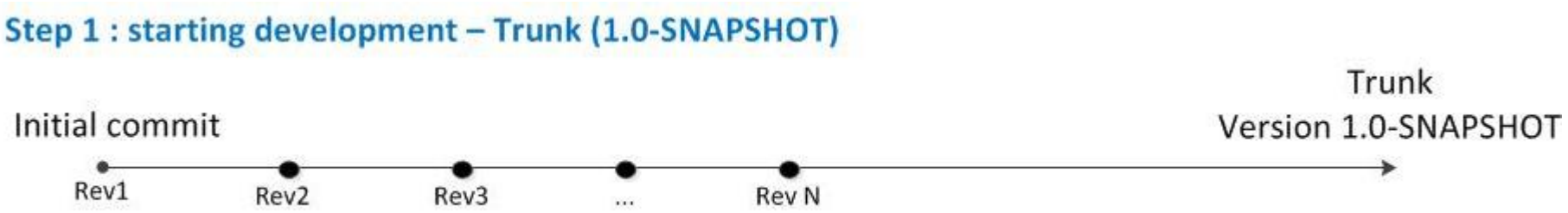


Versioning

- Etre capable de gérer les différentes versions du logiciel
  - Développement en cours (trunk ; version n+1)
  - Maintenance des versions déjà livrées
  - → Utilisation des notions de **tags** et de **branches**

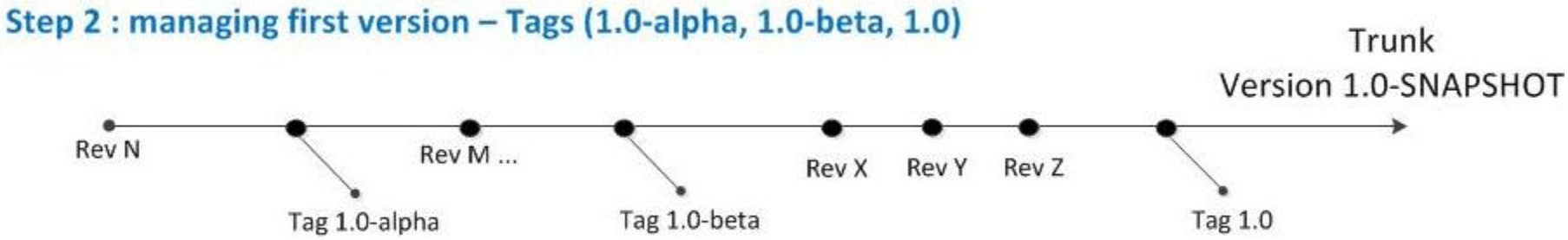
Démarrer un nouveau projet

→ **Trunk** *(le tronc)*



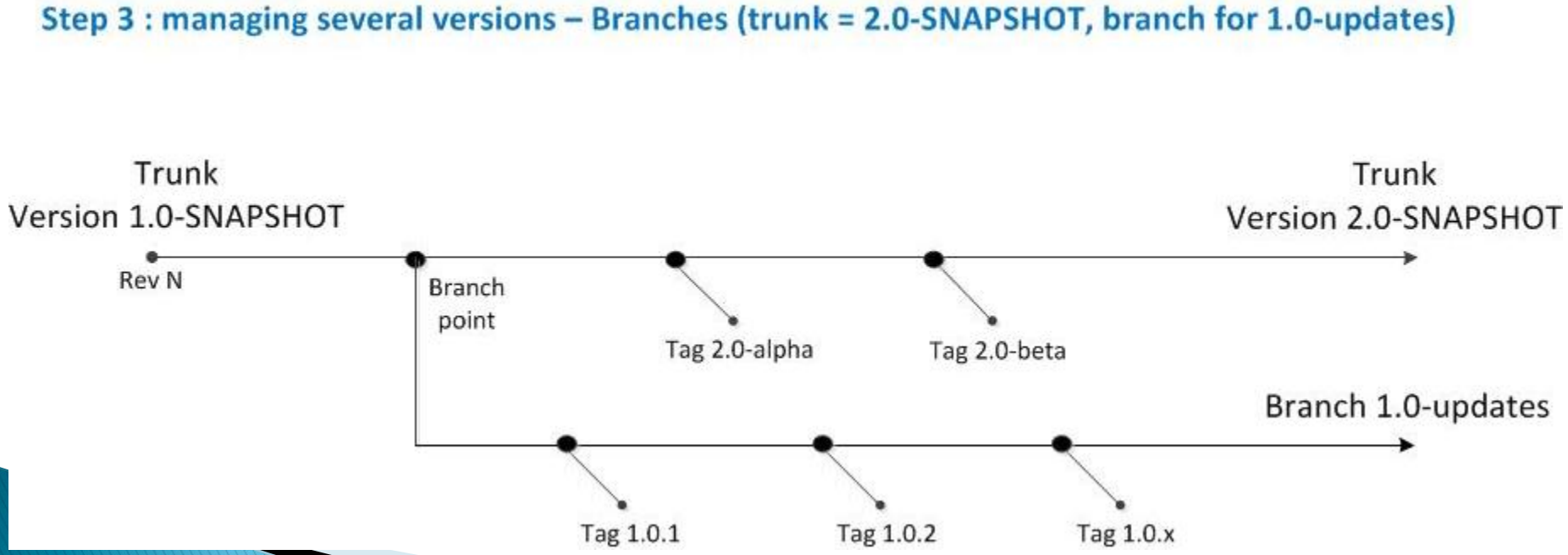
Gérer la première version

→ **Tags** *(les versions buildées, livrées, archivées doivent être « tagguées » = « labélisées »)*



Gérer plusieurs versions

- Nécessité de maintenir les versions livrées (1.0.x)
- Besoin de développer une nouvelle version en parallèle (la 2.0)



Rollback

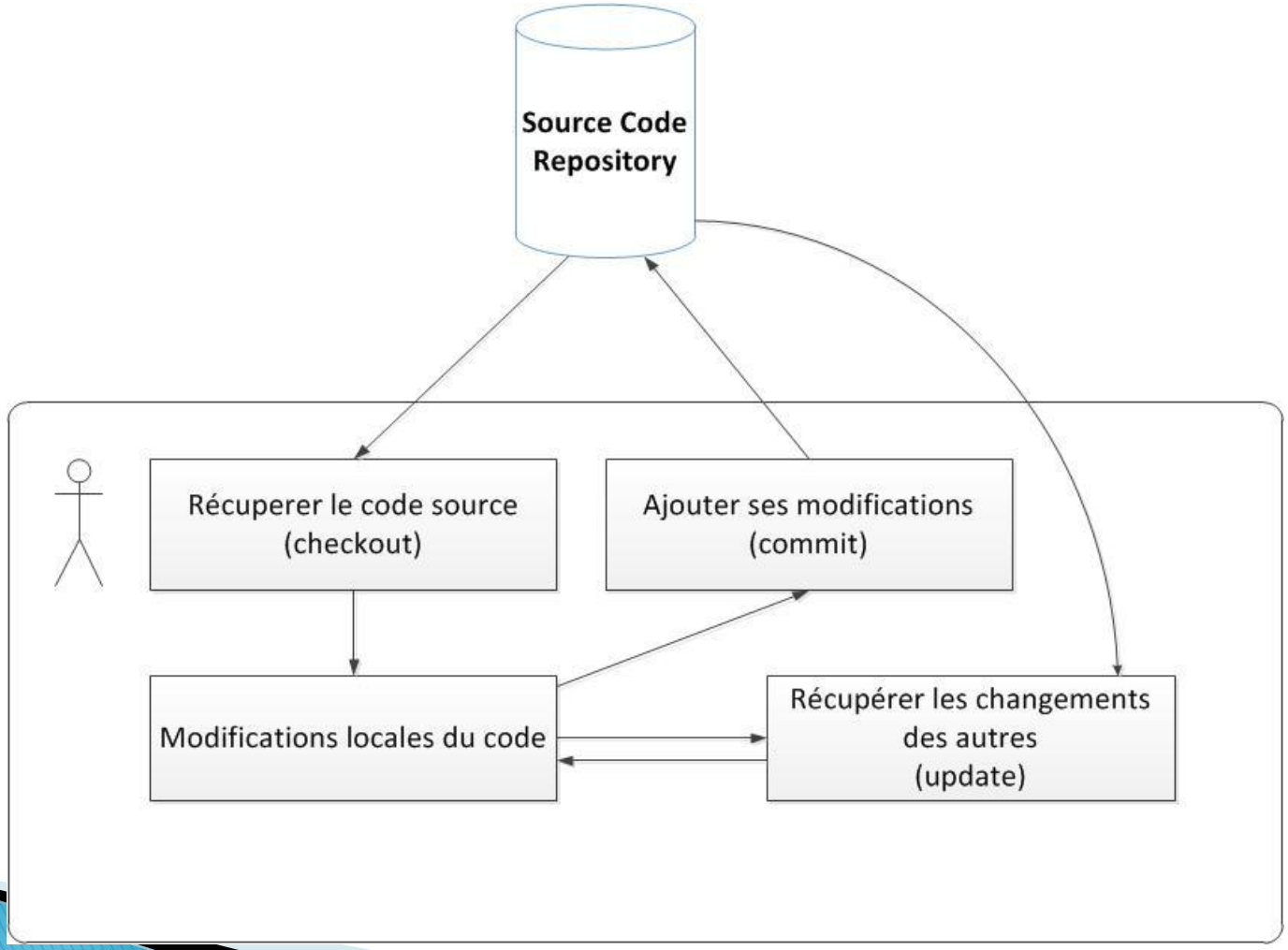
- Etre capable de revenir à une version précédente à tout moment
  - Garder tout **l'historique des changements (traçabilité)**
  - Connaitre l'auteur, la date, la nature et le contenu de chaque changement
  - Connaitre les raisons du changement (bug fix ?)
  - Identifier chaque changement par un numéro de révision
  - Etre capable de revenir à l'état du logiciel à n'importe quel niveau de révision

Cycle de vie

Cycle de vie

- **Checkout**
  Récupérer le code source en local sur sa machine
- **Local changes**
  Modifier le code en local
- **Update**
  Récupérer les modifications des autres
- **Commit**
  Ajouter ses modifications dans le repository

Cycle de vie


Source Code Repository
Récuperer le code source (checkout)
Ajouter ses modifications (commit)
Modifications locales du code
Récupérer les changements des autres (update)

Modifications concurrentes

Modifications concurrentes ?

- Les contributeurs peuvent travailler à plusieurs en même temps sur un même fichier

Conflits

Qu'est-ce qu'un conflit ?

- Les contributeurs veulent apporter une modification sur le même fichier voire sur la même partie d'un même fichier …

Comment éviter les conflits ?

- Approche pessimiste
- Empêcher toute modification sur un fichier en cours d'édition → Verrou
- Conflits impossibles, car gestion concurrente impossible
- Trop rigide…

Comment éviter les conflits ?

- Approche optimiste !
- Utiliser une synchronisation pour gérer les conflits
- Beaucoup plus souple, considère que les conflits ne sont pas fréquents et qu'ils seront gérés au cas par cas, par le développeur lui-même
- → Utilisation des fonctionnalités des systèmes de Gestion de Code Source pour résoudre le conflit (update / fusion / merge)

Exemple de conflit

- John et Tim font chacun un checkout du dépôt et obtiennent le fichier README à la révision #1
- Ils l'éditent chacun de leur coté
- John, une fois ses changements terminés, fait un commit de ses changements
  - Il soumet une mise à jour de son fichier sur le dépôt
  - Le **système créé la révision #2** du fichier
- Plus tard Tim va, à son tour, vouloir commiter ses changements
- **Il y a conflit !** Le système renvoie une erreur et n'autorise pas ce commit…

Pourquoi y'a-t-il conflit ?

La copie locale de Tim, au moment du commit, n'est pas à jour: Rev#1 au lieu de Rev#2 dans le repository.

Si le système laisse Tim commiter ses changements, cela écraserait purement et simplement les changements réalisés par son collègue John.

Comment le résoudre (1) ?

Si les modifications ne se recoupent pas

- Merge automatique (fusion)
- update: on récupère la dernière version du repository, le SCM fusionera automatiquement les changements

Merge automatique

Ex: README Rev#1

```
Voici le contenu du fichier initial
```

Ex: Changement commité par John

```
Je propose d'ajouter du texte avant
Voici le contenu du fichier initial
```

Merge automatique

Changement proposé par Tim

```
Voici le contenu du fichier initial
Je pense qu'il faut développer la partie en dessous
```

Proposition de merge + création de la Rev#3

```
Je propose d'ajouter du texte avant
Voici le contenu du fichier initial
Je pense qu'il faut développer la partie en dessous
```

Comment le résoudre (2) ?

Et si les modifications se recoupent ?

- Lorsque des modifications portent sur la même partie du code, le système ne sait pas résoudre le conflit tout seul
- Le système indiquera un conflit et proposera plusieurs possibilités pour résoudre le conflit (choix de la version à garder)
- Le développeur doit prendre une décision
- Merge manuel (fusion)

Strategies

- Release branch
- Feature branch
- Team branch

Voir : http://svnbook.red-bean.com/en/1.7/svn.branchmerge.commonpatterns.html

Quelques outils du marché

- Gestion centralisée
  - CVS, SVN, Clearcase, Perforce, etc…
- Gestion décentralisée
  - GIT, Mercurial, BitKeeper, RTC, etc…

Un peu d'historique

- CVS → SVN
  - Historique
  - SVN aka Subversion est le successeur de CVS
  - SVN est l'un des gestionnaires de code source les plus répandus actuellement
  - SVN se base sur un repository central

- GIT
  - http://git-scm.com/documentation
  - En plein essor, de plus en plus répandu
  - Approche **distribuée** de la gestion du code source

Présentation de SVN

Documentation en ligne:

- http://subversion.apache.org/docs/
- http://svnbook.red-bean.com/

- SVN est un système centralisé (dépôt)
- Dépôt = server qui stocke tous les fichiers et toutes leurs modifications
- Chaque développeur se connecte en tant que « client » de ce « server »

Présentation de SVN

- L'accès à SVN se fait par HTTP généralement par l'intermédiaire d'un client SVN

- Quelques clients
  - ligne de commande (il sait tout faire !)
  - TortoiseSVN qui s'intègre dans l'explorateur de Windows
  - Subversive pour l'intégration dans Eclipse
  - …

- D'autres exemples ?
  - http://fr.wikipedia.org/wiki/Comparaison_des_clients_pour_Subversion

Présentation de SVN

- L'organisation du code source dans SVN
  - **trunk**
    - moduleA
    - moduleB
    - moduleC
  - **branches**
    - v1updates
      - moduleA
      - moduleB
      - moduleC
    - v2updates
      - …
  - **tags**
    - v1.0
      - moduleA
      - moduleB
      - moduleC
    - v2.0
      - …

Présentation de SVN

- Les commandes courantes
  - svn checkout <URL> <dossier>
  - svn update
  - svn commit <dossier> -m « commentaire »

  - Passer des options (voir le manuel SVN)
  - Ex:
    - -m pour un message
    - --username pour s'authentifier
    - etc …

Présentation de SVN

- Les commandes courantes
  - svn status
  - svn log
  - svn diff
  - svn merge
  - …

Présentation de GIT

- Architecture distribuée
- Avec server central
- Commit en deux temps, concept de repository local (clone)
  - Commit
  - Push
- Reconnu pour ses performances
- Gestion des branches, repositories partageables

Présentation de GIT

- Les commandes courantes
  - git init : créer un repository
  - git clone : cloner un projet existant pour travailler dessus
  - git pull : recuperer les changements des autres
  - git log : voir l'historique des changements
  - git status : voir l'etat des changements dans la copie locale
  - git diff : differences entre 2 versions
  - git add : ajouter des fichiers dans le repo local
  - git commit : ajouter le changement en local
  - git push : partager ses changements
  - …

Agenda

- Gestion du code source
- **Gestion du build**

Gestion du Build

Qu'est-ce que le « build » ?

- Compilation du code source
- Exécution de tests
- Définition du packaging
- Création de livrables, consommables
- Gestion des dépendances
- Production de rapports (errors, tests results, coverage…)
- …

Gestion du Build

Qu'est-ce que le « build » ?

- Intimement lié à la gestion du développement et du code source
- Définit les étapes nécessaires pour construire le produit
- Intégration des modules

Gestion du Build

Qu'est-ce que le « build » ?

- Différents niveaux de granularité
  - Le build continu
    -> snapshot des modules en cours de développement
  - Le build de nuit
    -> snapshot du produit complet
  - Le build de release (ou de milestone)
    -> build d'itération
    -> build de version alpha ou beta
    -> build de release candidate ou finale
    = Version « tagguée » dans le SCM

Gestion du Build

Quel est le but ?

- Valider le code source notamment par l'exécution de la compilation et de tests unitaires
- Être capable de fournir un « installer » ou plus généralement un état courant du développement du produit, à tout moment
  - « Snapshot »
- Un build doit être **reproductible** à tout moment
  - Etre toujours capable de reconstruire un produit livré à un client
- Permettre de détecter très rapidement les « régressions » qui peuvent être introduites pour les corriger le plus rapidement possible
- -> Agilité

Gestion du Build

Quel est le lien avec la gestion de code source ?

- Le build est géré par un ensemble de scripts
- Il est géré comme le code source (il en fait partie !)
- Selon la technologie, il va même définir la structure du code !

- Il définit la version
- Il est associé à la branche et géré dans la branche
- Il évolue avec le produit

- Ex :
  - Sur le trunk = version 2.0-SNAPSHOT
  - Sur une branche v10updates = version 1.0-SNAPHOT

- Permet de construire les différentes versions en parallèle !

Outils de Build

Quelques exemples

- Make
- Ant, Maven, Gradle
- PDEbuild, Tycho
- MSBuild, NAnt
- …

Outils de Build

GNU Make

http://www.gnu.org/software/make/

- Commande unix
- Permettant d'automatiser la compilation de code
- Ex
  - make install
  - make doc
  - make clean
  - make all

Outils de Build

Apache Ant (1)

http://ant.apache.org/

- Remplaçant de make
- Permettant d'automatiser la compilation et d'autres taches de packaging
  - Compilation
  - Génération de javadoc
  - Manipulation de fichiers (copy, move, etc…)
  - Création d'archives JAR
  - …

Outils de Build

Apache Ant (2)

- project
- targets
  - compilation
  - installation
  - execution
- tasks
  - Correspondent a des commandes usuelles
    - javac
    - jar
    - copy
- **Décrits dans un fichier build.xml**

Outils de Build

Apache Ant (3)

- (+) Encore très utilisé, puissant
- (+) Pratique et flexible pour beaucoup de taches

- (-) Difficile à maintenir
- (-) Debugging difficile
- (-) Pas de gestion des dépendances

Outils de Build

Maven (1)

http://maven.apache.org/

- “Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.”

Outils de Build

Maven (2)

- Gestion des dépendances
  - Maven Central Repository
  - Maven Internal Repository (Nexus, Artifactory …)
  - Maven Local Repository (sur la machine du développeur)
- Dépendances transitives
  - J'ai besoin du module M, qui dépend du module N, qui lui-même dépend de…
  - Je n'appelle que le module M ! Maven se charge du reste…

Outils de Build

Maven (3)

- Gestion des dépendances
  - Notion d'artifact Maven
  - Identifié par des coordonnées maven
    - groupId
    - artifactId
    - version
  - Ex JUnit

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.10</version>
</dependency>
```

Outils de Build

Maven (4)

- Gestion du code source
- Maven définit un Layout pour l'organisation du code
  - src/main/java
  - src/main/test
  - pom.xml (Maven project object model)
  - Un pom = un artifact maven
- Le rootpom definit la configuration générale du projet
- Un pom.xml a toujours un pom parent et hérite de ses propriétés (récursivité)

Outils de Build

Maven (5)

- Gestion du build par phase
- Les plus utilisées
  - validate : vérifie les pre-requis d'un projet maven
  - compile : compilation du code source
  - test : lancement des tests unitaires *(cf. cours sur les tests … à suivre)*
  - package : assemble le code compilé en un livrable
  - install : partage le livrable dans le repository maven local
  - deploy : publie le livrable pour d'autres projets dans le repo distant

Outils de Build

Maven (6)

- Liens utiles
  - Apache Maven : http://maven.apache.org/
  - Maven: The Complete Reference : http://www.sonatype.com/books/mvnref-book/reference/
  - **Traduction française:** http://maven-guide-fr.erwan-alliaume.com/ **(A lire !)**

Bonnes pratiques

- Utiliser un gestionnaire de source code
- Expliquer les changements (commentaires de commits)
- Lier les changements a des tickets
- Commiter le code source (pas les binaires !)
- Commiter le code régulièrement pour éviter les conflits
- Mettre en place une politique de gestion des tags/branches, versioner le code
- Reporter les fixes sur les branches actives…
- Automatiser le build et les tests pour le rendre reproductible, éviter les régressions et augmenter le niveau de confiance

Tous ces concepts seront repris dans le cours sur l'intégration continue… (à suivre !)

Questions ?