Les outils de développement logiciel dans l'enseignement de la programmation

Intégration dans l'IDE Eclipse

Stéphane Lopes
stephane.lopes@prism.uvsq.fr

23 mai 2013

Courte bio

Stéphane Lopes (stephane.lopes@prism.uvsq.fr)

- Maître de conférences à l'université de Versailles St-Quentin
  - Chercheur au laboratoire PRiSM
  - Enseignant au dpt. info. de l'UFR des sciences et de l'ISTY

Thèmes de recherche

Bases de données, systèmes d'information, intégration de données, personnalisation, recherche d'information.

Thèmes d'enseignement

Bases de données Administration et tuning de BD

Programmation Programmation objet en Java

Outils de développement Outils de développement, UML

Objectifs de la présentation

- Montrer l'importance des outils dans l'enseignement de la programmation
- Présenter les principales familles d'outils de développement
- Apporter un retour d'expérience sur ce sujet

Plan

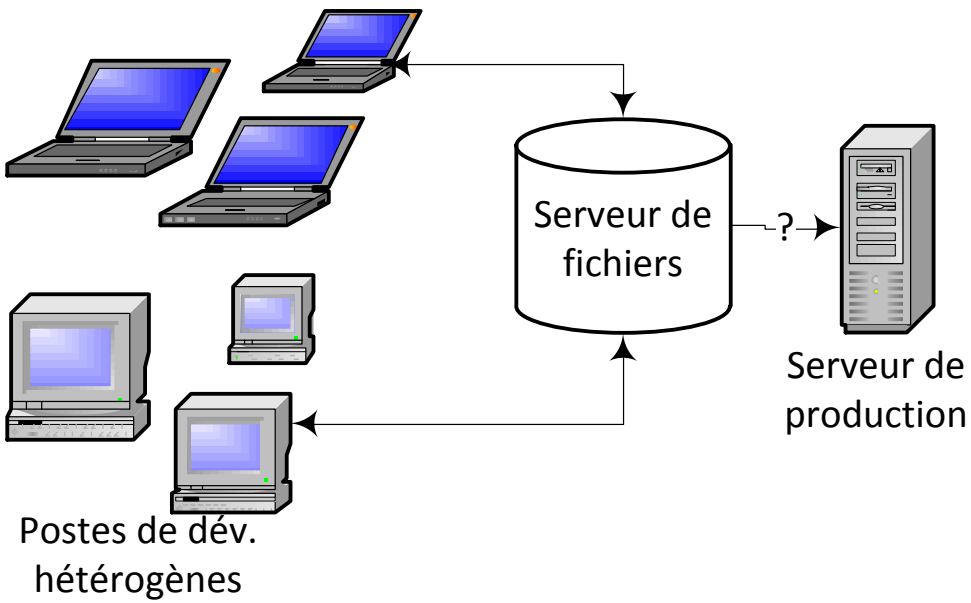
Enseigner la programmation

- L'approche choisie dépend du public
  - informaticien
  - autres sciences
- Principes généraux
  - insister sur les concepts plus que sur le langage
  - aborder les bibliothèques
  - présenter la « boite à outils » du développeur
  - tenir compte de l'état de l'art

Le développement logiciel à l'heure actuelle

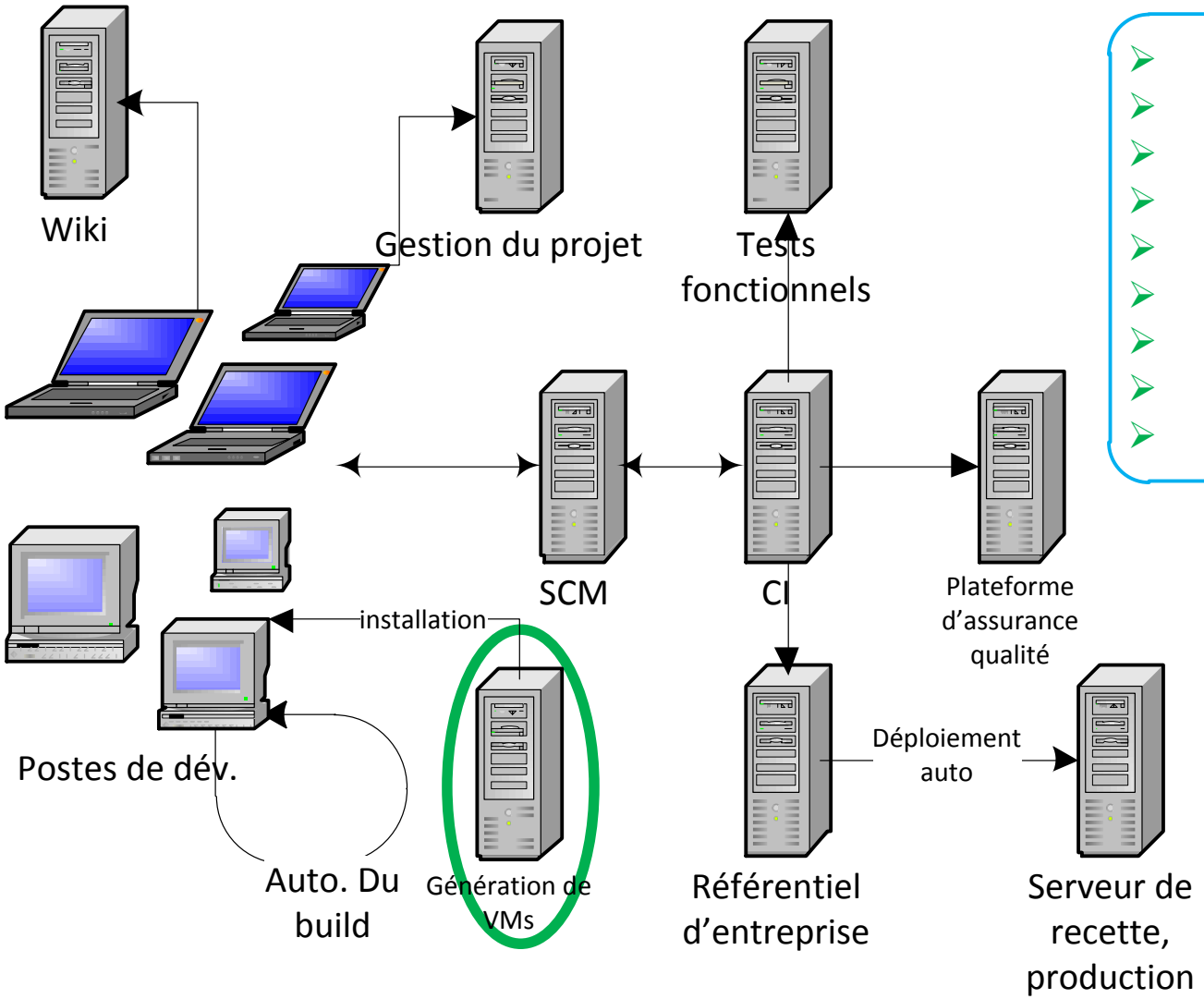
- Industrialisation du logiciel
  - le logiciel est vu comme un produit manufacturé
  - forge/usine logicielle
  - automatisation des traitements répétitifs
- *Software Craftsmanship*
  - met l'accent sur les compétences des développeurs
  - s'inspire du *compagnonage*
- DevOps
  - mouvement visant à réduire le « gap » entre développement (*DEVelopment*) et exploitation (*OPerationS*)
  - l'objectif est de permettre des livraisons fréquentes du logiciel
  - s'appuie sur l'automatisation des déploiements

Situation initiale



- Communication/tâches
- Fusion/versions
- Build
- Distribution des bin.
- Tests/intégration
- Ass. Qualité
- Livraison
- Documentation
- Postes de dév. standards

Processus outillé



- Communication/tâches
- Fusion/versions
- Build
- Distribution des bin.
- Tests/intégration
- Ass. Qualité
- Livraison
- Documentation
- Postes de dév. standards

Exemple de cursus

Enseignement de la programmation objet (POO)

Public

- 3ème année de licence d'informatique (Bac+3)
- 1ère année d'école d'ingénieur en informatique

Cours

- POO (cours et TP)
  - Concepts objets et leur mise en œuvre en Java
  - Bibliothèque standard (I/O, Collection, GUI)
  - Introduction aux *design patterns*
- Techniques et outils de développement (cours, TD et TP)
  - techniques de développement (conventions de codage, débogage, ...)
  - outils (IDE, framework de tests unitaires, ...)
  - notation UML
- Projet

Techniques et outils de développement

- Gestion des sources
  - IDE
  - Conventions de codage et documentation
  - Gestion de versions/travail collaboratif
- Construction (Build) et gestion des binaires
  - Build
  - Gestion des binaires et déploiement
- Mise au point de programmes
  - Analyse statique
  - Assertions et tests
  - Débogage
  - Profiling

Plan

Environnement de développement intégré

- Un environnement de développement intégré (*Integrated Development Environment* ou *IDE*) regroupe un ensemble d'outils de développement
- Principales fonctionnalités
  - Éditeur de code source
  - Gestion de projets (fichiers, dépendances, . . .)
  - Auto-complétion de code
  - Navigation dans les classes
  - *Refactoring*
  - Compilation
  - Débogage
  - Profiling
  - Gestion de versions

Quelques outils

Eclipse, NetBeans, IntelliJ IDEA, Microsoft Visual Studio, BlueJ.

Eclipse

plate-forme Eclipse

- écrite en Java
- extensible par un système de *plugins*
- code initial issu de *Visual Age* (IBM)

Fondation Eclipse

soutenue par de nombreuses organisations (Borland, IBM, Intel, OMG, Oracle, . . .)

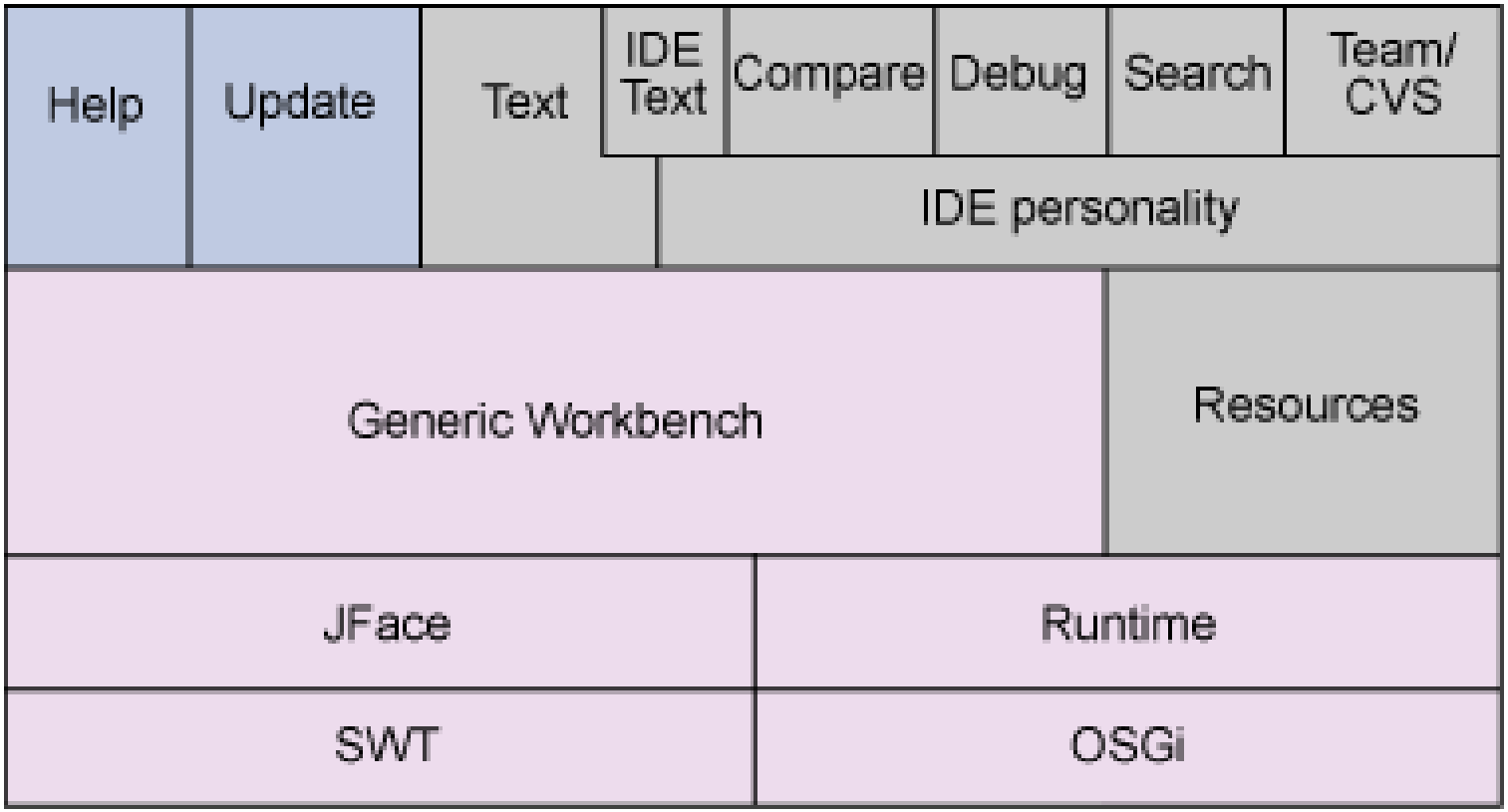
Projets

nombreux développements organisés en *projets*

Les multiples facettes d'Eclipse

- IDE pour Java (plate-forme + JDT)
- *Framework* pour un IDE (CDT pour C/C++, PDT pour PHP, ...)
- Plate-forme pour intégrer des outils (BI, modélisation, ...)
- *Framework* pour des applications (application de type client riche, ...)
- Plate-forme d'exécution (Equinox/OSGi)

Architecture



source : Get started with the Eclipse Platform

Quelques concepts d'Eclipse

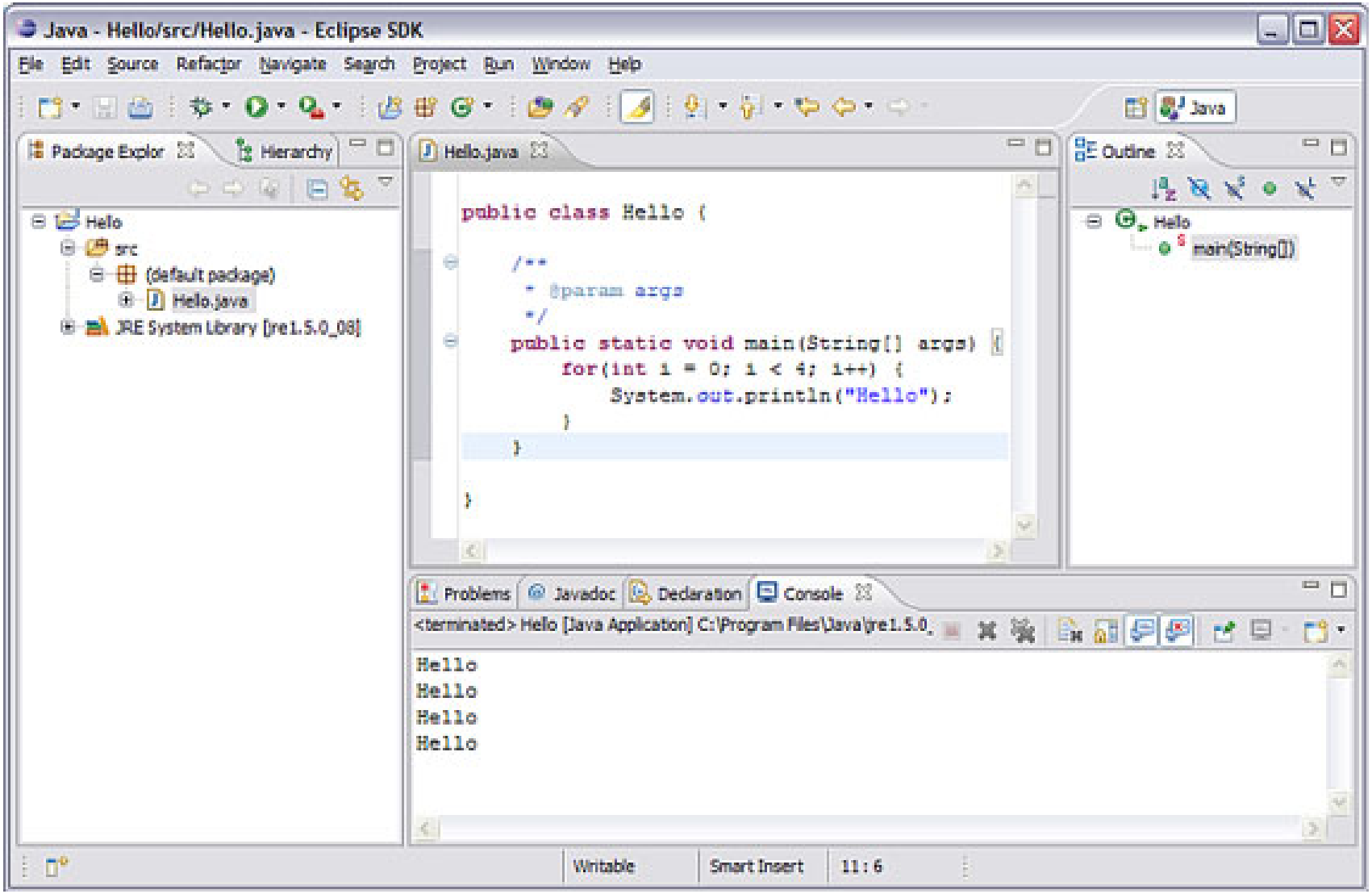
Interface graphique

Workbench	regroupe un ensemble de fenêtres
Éditeur	permet d'ouvrir, d'éditer et de sauver des ressources
Vue	fournit des informations sur les objets manipulés
Perspective	organisation des fenêtres à l'écran

Ressources

Ressource	projet, répertoire et fichier
Workspace	emplacement contenant les projets

Perspective Java



source : Get started with the Eclipse Platform

Eclipse et enseignement

Les +

- IDE à l'état de l'art
- Très utilisé par les professionnels
- Intègre tous les outils (plugins)

Les -

- Environnement complexe
- Nécessite d'introduire tôt certaines notions avancées (*main*, I/O)
- Alternative pour débuter : BlueJ

Style de codage

- Ensemble de règles pour la rédaction du code source (nommage, indentation, commentaires, . . .)
- Améliore la lisibilité du code source
  $\Rightarrow$ plus facile à comprendre
  $\Rightarrow$ plus facile à maintenir
- Doit être supporté par l'IDE et/ou un outil de vérification
  - mise en évidence des violations
  - à configurer pour un projet
- L'important est de choisir un style et de s'y tenir.

Quelques outils

l'IDE, CheckStyle.

Documentation technique du code source

- Commentaires spécifiques dans le code source
- Génération automatique de la documentation
- Aide à garder la documentation en phase avec le code

Quelques outils

JavaDoc, Doxygen.

Exemple de commentaire JavaDoc

Méthode *Applet :: getImage*

```java
/**
 * Returns an Image object that can then be painted on the screen.
 * The url argument must specify an absolute {@link URL}. The name
 * argument is a specifier that is relative to the url argument.
 * <p>
 * This method always returns immediately, whether or not the
 * image exists.
 *
 * @param  url  an absolute URL giving the base location of the image
 *         name the location of the image, relative to the url argument
 * @return      the image at the specified URL
 * @see         Image
 */
public Image getImage(URL url, String name) {
```

getImage

```
public Image getImage(URL url,
                      String name)
```

Returns an `Image` object that can then be painted on the screen. The `url` argument must specify an absolute URL. The `name` argument is a specifier that is relative to the `url` argument.

This method always returns immediately, whether or not the image exists. When this applet attempts to draw the image on the screen, the data will be loaded. The graphics primitives that draw the image will incrementally paint on the screen.

Parameters:

`url` - an absolute URL giving the base location of the image.

`name` - the location of the image, relative to the `url` argument.

Returns:

the image at the specified URL.

See Also:

`Image`

Style de codage et enseignement

Intégration à Eclipse

- Mise en forme du code source supportée
- Génération de la JavaDoc
- Utilisation de la documentation dans l'éditeur
- Intégration de Checkstyle avec le plugin eclipse-cs

En TP (petits exercices et travail individuel)

- Conventions même simples pas toujours respectées
- Peu de documentation
- Configuration délicate de Checkstyle (*faux positifs*)

Gestion de versions

- Consiste à maintenir l'ensemble des versions d'un ensemble de documents
- Activité fastidieuse et complexe
  - $\Rightarrow$ bénéficie d'une façon importante d'un support logiciel
- Intégrée dans certaines applications (*Wiki*, *OpenOffice*, *Microsoft Word*, ...)

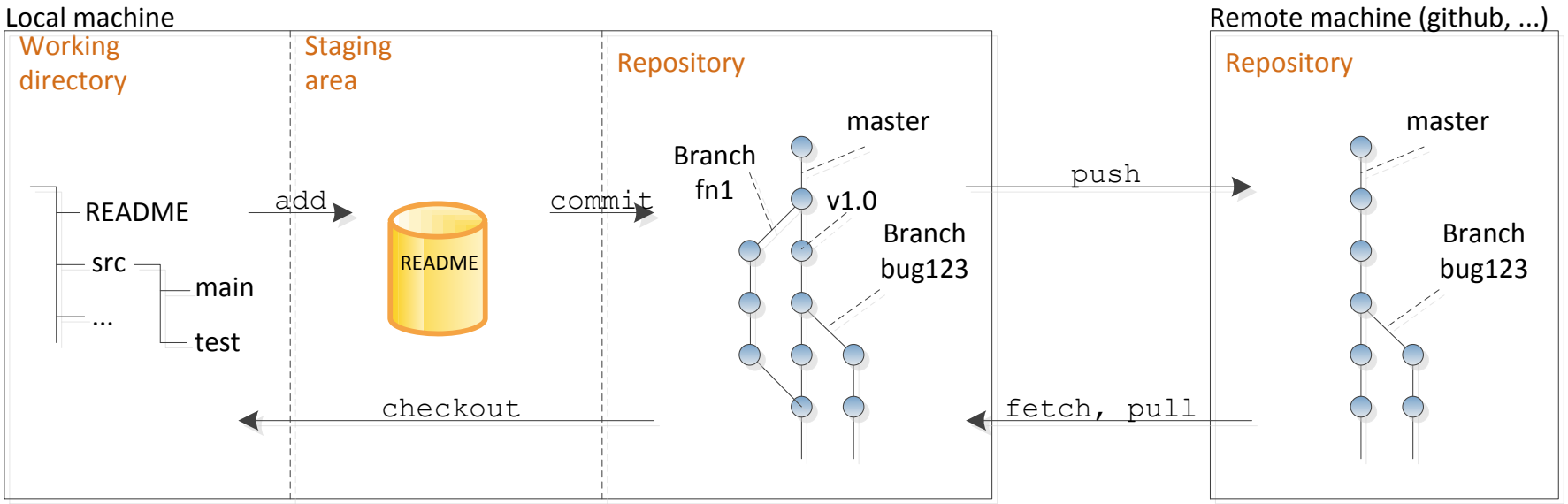
Outils de gestion de versions (VCS)

- Gestion des versions successives d'un document
  - conserve toutes les versions successives dans un *référentiel* ou *dépôt* (*repository*)
  - ne stocke en général que les différences
  - permet de revenir aisément à une version antérieure
- Travail collaboratif
  - chaque utilisateur travaille sur une copie locale (pas de verrouillage)
  - le système signale les conflits
- Architecture centralisée ou distribuée
- Souvent utilisé par l'intermédiaire d'une *forge logicielle*

Quelques outils

Subversion, git.

Git


Local machine
Working directory
Staging area
Repository
Remote machine (github, ...)
Repository
README
src
main
...
test
add
README
commit
master
Branch fn1
v1.0
Branch bug123
push
master
Branch bug123
checkout
fetch, pull

VCS et enseignement

Intégration à Eclipse

- Plugins très complets (subclipse, subversive, EGit)
- Plugins subversion non autonomes

En TP

- Intérêt de ces outils assez évident
- Concepts manipulés nombreux (branches, ...) donc longs à assimiler
- Interface assez complète et complexe (usage de la ligne de commande en parallèle)

Plan

Build automation

- Consiste à automatiser les tâches répétitives (compilation, lancement des tests, ...
- Plus rapide et moins sujette aux erreurs
- Evite les fastidieuses lignes de commande
- Permet une compilation « intelligente » (n'est traité que ce qui est nécessaire)
- Peut être déclenché

  à la demande l'utilisateur exécute un script
  par un ordonnanceur à un instant donné
  par un événement par une action sur le projet (*commit* par exemple)

Quelques outils

Make, Apache Ant, Apache Maven, Gradle.

Maven

- Projet open source de la fondation Apache
- Outil basé sur un ensemble de conventions et de bonnes pratiques
- Approche déclarative

Concepts de Maven

POM Le *modèle objet projet* (*project object model* ou *POM*) est une description du projet

Cycle de vie Le *cycle de vie* (*build lifecycle*) définit précisément les processus de compilation

Hiérarchie standard L'arborescence de répertoires d'un projet respecte un ensemble de conventions

Plugin Un *plugin* est un programme exécuté par maven pour réaliser une tâche

But Un *but* (*goal*) est une des tâches proposées par un plugin

Référentiel Un *référentiel* (*repository*) contient les objets générés et les dépendances

Intégration continue

- Pratique de développement où les membres d'une équipe intègrent fréquement leur travail
- A pour but de détecter les problèmes d'intégration au plus tôt
- Est issue d'*eXtreme Programming*
- S'appuie généralement sur un *serveur d'intégration continue*

Quelques outils

Jenkins, Apache Continuum, CruiseControl.

Build et enseignement

Intégration à Eclipse

- Plugin m2e pour Maven (standard sous Juno)

En TP

- Pourquoi utiliser ces outils par rapport à un IDE ?
- Intégration continue non expérimentée
  - complexité de l'environnement
  - alternative en PaaS (CloudBees, . . .)

Release et déploiement

- Le processus de *release* génère une version de production
  - changements validés, dépendances avec des modules non en version release, tests, ...
- Les modules générés sont placés dans un dépôt binaire
  - mise à disposition des binaires, contrôle d'accès
- L'automatisation du déploiement permet l'installation de l'application
  - *Continuous Delivery*

Quelques outils

Maven Release Plugin, Nexus, artifactory.

Plan

Analyse statique du code

- Permet d'obtenir des informations sur un programme sans l'exécuter
  - recherche de motifs généraux (`;` après un `for`, ...)
- Est un bon complément aux tests
- Outils d'analyse statique
  - moteur
  - ensemble de règles configurables
- Utilisé dans le contexte de la *revue* (*audit*) de code

Quelques outils

FindBugs, Pmd.

Quelques bogues courants en Java

- Boucle récursive infinie

```
public MaClasse() {
  MaClasse m = new MaClasse();
}
```

- Déréférencement d'une référence null

```
if (c == null && c.uneMethode()) //...
```

- Auto affectation d'attribut

```
public MaClasse(String uneChaine) {
  this.chaine = chaine;
}
```

- Valeur de retour ignorée

```
String nom = //...
nom.replace('/', '.');
```

Analyse statique et enseignement

Intégration à Eclipse

- Plugins pour l'intégration dans l'éditeur (FindBugs, PMD)

En TP

- Simple à mettre en œuvre (intégration à l'IDE)
- Réglage difficile de l'outil (éviter les faux positifs, paramétrer le niveau de détail)
- Que faire devant les rapports générés ?

Assertions

- Expression booléenne exprimant une propriété sémantique (invariant, pré-condition, . . .)
- Permet de vérifier que ce que l'on croit « vrai » l'est véritablement à l'exécution
- Technique simple pour assister la réalisation de programmes corrects
- Normalement pas de surcoût dans la version de production
  - destinées aux versions de débogage
  - éliminées lors de la compilation pour les versions de production
- Supportées dans les langages de programmation
- Utilisées également dans
  - les *frameworks* de tests
  - l'approche *Design By Contract*

Tests

- Visent à mettre en évidence des défauts de l'élément testé
- Objectifs
  - réduire le nombre de bogues
  - améliorer la qualité du logiciel
- *Cas de test* : conditions initiales, entrées, observations attendues
- Ne permet pas de prouver l'absence de bogue ($\neq$ méthodes formelles)
- Processus
  - tester après l'implémentation (approche classique)
  - écrire le test d'abord (approche *Test Driven Development*)

Quelques types de tests

Fonctionnel s'assure que les résultats attendus sont bien obtenus

Non fonctionnel analyse les propriétés non fonctionnelles d'un système (performances, sécurité, . . .)

Unitaire vérifie la conformité des éléments de base d'un programme

Intégration vérifie la cohérence des différents modules entre eux

Recette confirme la conformité du système avec les besoins

Quelques outils

JUnit, TestNG, FitNesse, Apache JMeter.

Couverture de code

- Mesure la qualité des tests effectués
- Le degré de couverture est représenté par des indices statistiques

  Statement Coverage mesure le nombre de lignes exécutées
  Condition Coverage vérifie l'évaluation des conditions
- Les portions de codes non testées sont mises en évidence

Quelques outils

Cobertura, EMMA.

Tests et enseignement

Intégration à Eclipse

- JUnit est directement supporté
- Plugin MoreUnit améliore la prise en charge
- Plugin infinitest pour réexécuter les tests automatiquement

En TP

- Tests unitaires assez simples à mettre en œuvre
- Difficultés pour identifier les cas de test
- Concepts avancés (doublure de test, couverture de code, . . .) et autres types de tests non abordés

Débogage

bogue défaut dans un programme qui l'empêche de fonctionner correctement

débogage localiser les bogues dans un programme

- basé sur la *confirmation*
- confirmer les choses que l'on croit vraies jusqu'à en trouver une qui ne l'est pas

débogueur outil fournissant une aide pour le débogage

- Exécution contrôlée du programme (pas à pas)
- Pose de *points d'arrêt* (*breakpoints*, *watchpoint*)
- Visualisation de l'état du programme (variables, pile d'appel)

Quelques outils

L'IDE, jdb.

Profiling

- Collecter des informations sur le comportement d'une application pendant son exécution
  - CPU, mémoire, *threads*, ...
- A un impact sur le comportement de l'application

Quelques outils

Valgrind, HProf, Intel VTune.

Débogage et enseignement

Intégration à Eclipse

- Débogueur intégré performant

En TP

- Scénario simple de débogage
- Débogueur peu utilisé par les élèves (affichages)

Plan

Outils et développement logiciel

- Nombreux outils de développement disponibles
- Apportent une réelle valeur ajoutée
- Difficultés
  - choisir les outils adaptés pour chaque projet
  - paramétrer les outils
  - intégrer ces outils dans le processus

Pour l'enseignement

- Indispensable à l'heure actuelle
- Nombreux concepts à assimiler
  - $\Rightarrow$ nécessite du temps (stages)
- IDE aide à aborder ces outils (interface unifiée)

Merci de votre attention

Questions ?