Ingénierie Méthode : “Source code analyser”

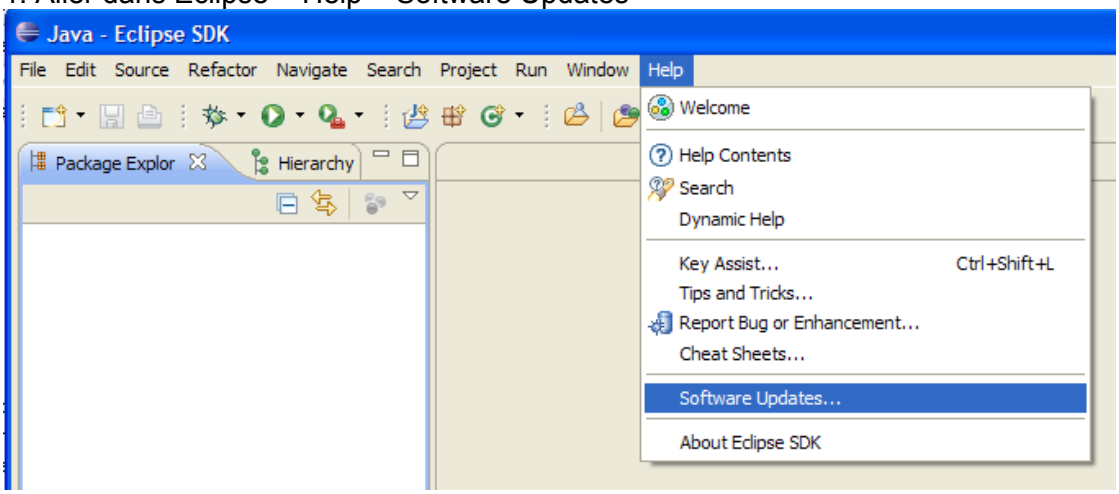
Cette documentation a pour but de présenter et d’analyser l’outils qui sera utilisé lors de notre projet afin de consolider au mieux les soucis de performance, d’optimisation et de sécurité du code. Les technologies qui seront analysées seront Checkstyle, PMD et Findbugs pour lesquelles nous allons détailler leurs installations (par plugin eclipse et par Maven plugin), les principes de fonctionnement de chacun ainsi qu’un comparatif des outils sur lesquels nous allons nous pencher plus en détails.

Enfin, nous appuierons l’un de ces outils qui sera utilisé pour le projet UML Architect.

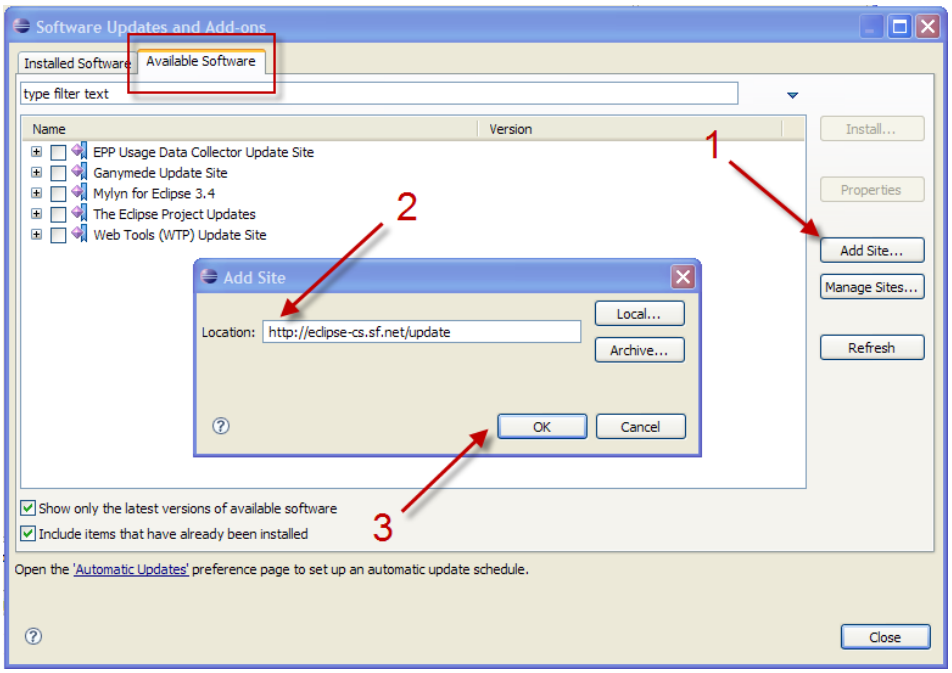
CheckStyle

Procédure d’installation

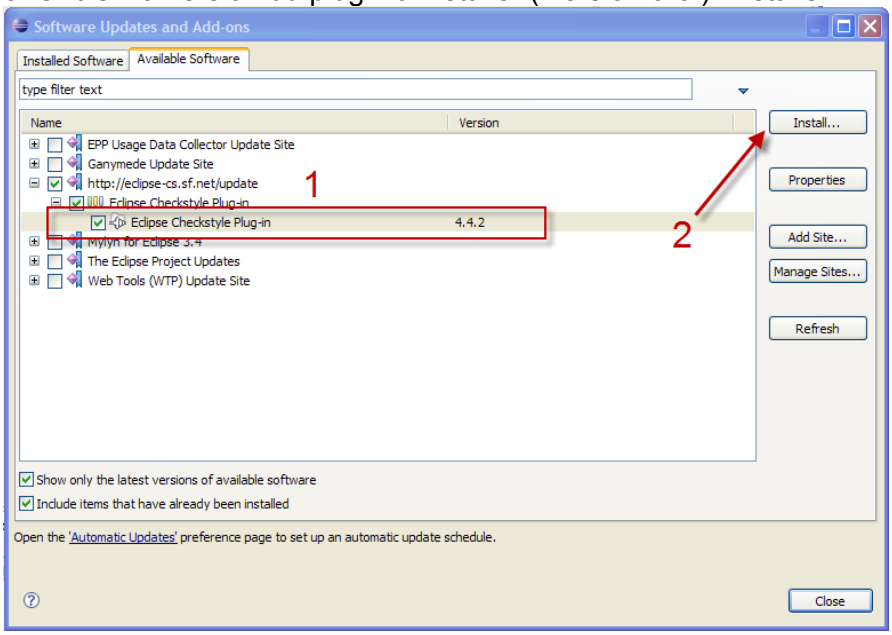
1. Aller dans Eclipse > Help > Software Updates



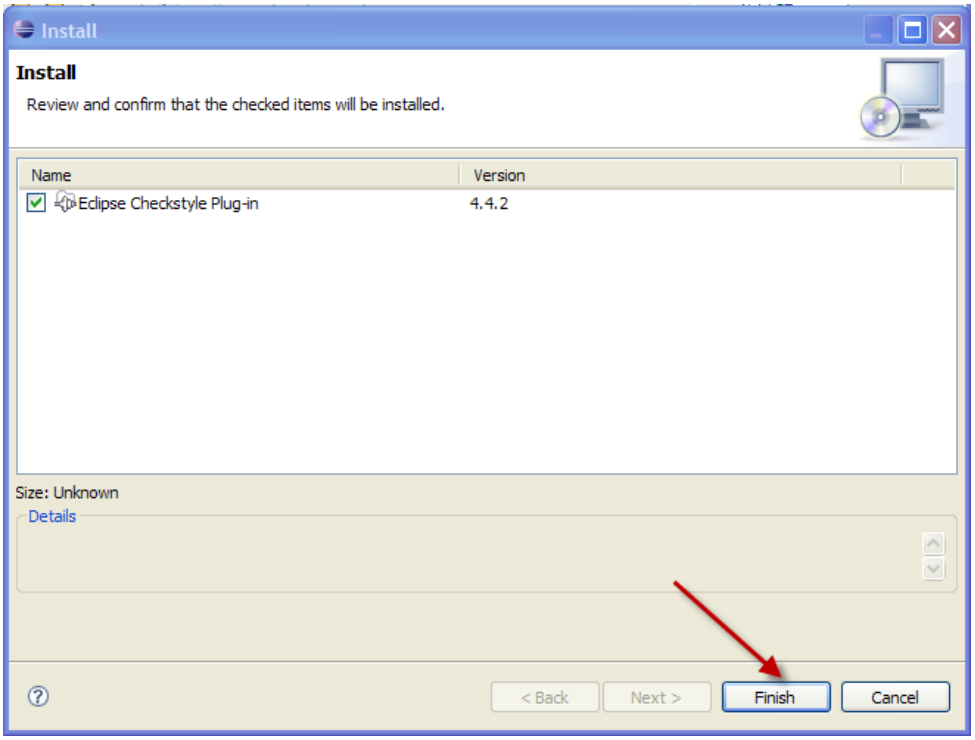
2. Ajouter un nouveau site d’installation avec l’URL : ***http://eclipse-cs.sf.net/update/***



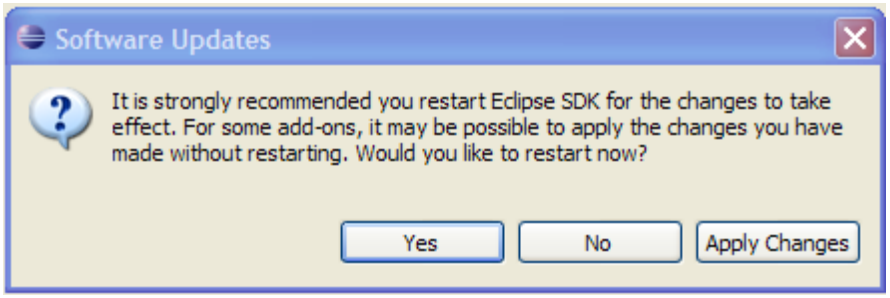
3. Choisir la version du plugin à installer (version 5-5). Installer.



4. Confirmer l'installation des plugins à instal



5. Relancer Eclipse.



Installation par Maven (exemple avec version 2.2)

```
<plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-checkstyle-plugin</artifactId>
        <version>2.2</version>
        <configuration>
                <configLocation>config/sun_checks.xml</configLocation>
                </configuration>
</plugin>
```

Configuration

Aller dans : Windows > Preferences > Checkstyle
Importer les règles de vérification en cliquant sur “New …”. Le fichier de “qualité” doit se trouver dans src/main/resources. Le fichier à importer est “checkstyle.xml”
Choisir ensuite “Set as default” pour utiliser les règles de vérification par défaut.

Utilisation

Par défaut, tous les projets sont automatiquement vérifiés. Vous pouvez désactiver checkstyle sur un projet en selectionnant le projet avec click droit > CheckStyle > Desactivate Checkstyle”

Sur un fichier du projet, vous pouvez faire : Click Droit > Checkstyle > Check code with checkstyle. Un rapport est généré avec toutes les violations de règles de vérification.

Avant chaque commit, vérifier chaque classe en attente de commit avec Checkstyle et résoudre chaque partie/portion de code qui est lève une violation.

Principe

CheckStyle permet de contrôler le respect des conventions de codage et d'avoir quelques métriques.
Ce contrôle est fait avec 126 règles et cela va du plus simple comme la longueur d'une ligne à des choses plus compliquées comme la complexité cyclomatique d'une classe.
Le plugin Maven met à disposition 4 fichiers de règles :

- config/sun_checks.xml
  Convention Sun Microsystems (par défaut),
  http://java.sun.com/docs/codeconv/

- config/maven_checks.xml
  Convention de l'équipe Maven,
  http://maven.apache.org/guides/development/guide-m2-development.html#maven%20code%20style

- config/turbine_checks.xml
  Convention du projet Apache Turbine,
  http://turbine.apache.org/common/code-standards.html

- config/avalon_checks.xml
  Convention du projet Apache Avalon.
  Ce projet n'existe plus.

PMD

Procédure d’installation

Similaire à celle de CheckStyle mise à part les points suivants :
2. Ajouter un nouveau site d’installation avec l’URL : http://pmd.sourceforge.net/eclipse

Installation par Maven (exemple avec version 2.4 et quelques règles de vérifications “ruleset”)

```
<plugin>
	<groupId>org.apache.maven.plugins</groupId>
	<artifactId>maven-pmd-plugin</artifactId>
	<version>2.4<version>
	<configuration>
		<rulesets>
			<ruleset>/rulesets/basic.xml</ruleset>
			<ruleset>/rulesets/imports.xml</ruleset>
			<ruleset>/rulesets/unusedcode.xml</ruleset>
			<exclude name="UnusedPrivateField" />
			<ruleset>/rulesets/finalizers.xml</ruleset>
		</rulesets>
	</configuration>
</plugin>
```

La liste des “ruleset” de pmd se trouve à l’adresse suivante : http://pmd.sourceforge.net/rules/index.html

Configuration

Aller dans Windows > Preferences > PMD > Configuration des règles
Le fichier à importer est “pmd.xml”

Utilisation

Sur un fichier du projet, vous pouvez faire : Click Droit > PMD > Check code with PMD.
Un rapport est généré avec toutes les violations de règles de vérification.

Avant chaque commit, vérifier chaque classe en attente de commit avec PMD et résoudre chaque partie/portion de code qui est lève une violation.

Principe

PMD va analyser le code afin de trouver des problèmes potentiels tel que :

- Bugs possibles,
- Code mort,
- Code non optimal,
- Expressions trop compliquées,
- Problèmes de sécurité,
- Problèmes de couplage entre objet/package

Findbugs

Procédure d’installation

Similaire à celle de CheckStyle mise à part les points suivants :
2. Ajouter un nouveau site d’installation avec URL : http://findbugs.cs.umd.edu/eclipse/

Installation par Maven (exemple avec la version 1.0.0)

```
<plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>findbugs-maven-plugin</artifactId>
        <version>1.0.0<version>
</plugin>
```

Configuration

Aller dans Windows > Preferences > Java > Findbugs > Filter Files.
Le fichier à importer est “findbug.xml”

Utilisation

Sur un fichier du projet, vous pouvez faire : Click Droit > Findbugs > Find bugs. Un rapport est généré avec toutes les violations de règles de vérification.

Avant chaque commit, vérifier chaque classe en attente de commit avec Findbugs et résoudre chaque partie/portion de code qui est lève une violation.

Principe
Findbugs va trouver les bugs potentiels en analysant le bytecode Java. Pour cela il s'appuie sur une notion de 'bug patterns'.

Ces bugs sont classés en plusieurs catégories :

- Correctness
  Regroupe les bugs généraux. Par exemple les boucles infinies, mauvaises utilisations de equals(), ...
- Bad practice
  Regroupe les mauvaises pratiques. Par exemple les problèmes d'Exception, de ressources non fermées, mauvaises utilisations de comparaison de chaîne de caractères, ...
- Performance
  Regroupe les problèmes de performance. Par exemple la création d'objets inutiles.
- Multithreaded correctness
  Regroupe les problèmes liés au code multithread.
- Internationalization

Regroupe les problèmes liés à l'internationalisation d'une application.

- Malicious code vulnerability
  Regroupe les problèmes de vulnérabilité. Par exemple du code qui pourrait être détourné de son utilisation, ...

- Security
  Regroupe les problèmes de sécurité. Par exemple les problèmes liés au protocole http, les SQL injections, ...

- Dodgy
  Regroupe le "smell code". Par exemple les comparaisons redondantes avec null, variables non utilisées, ...

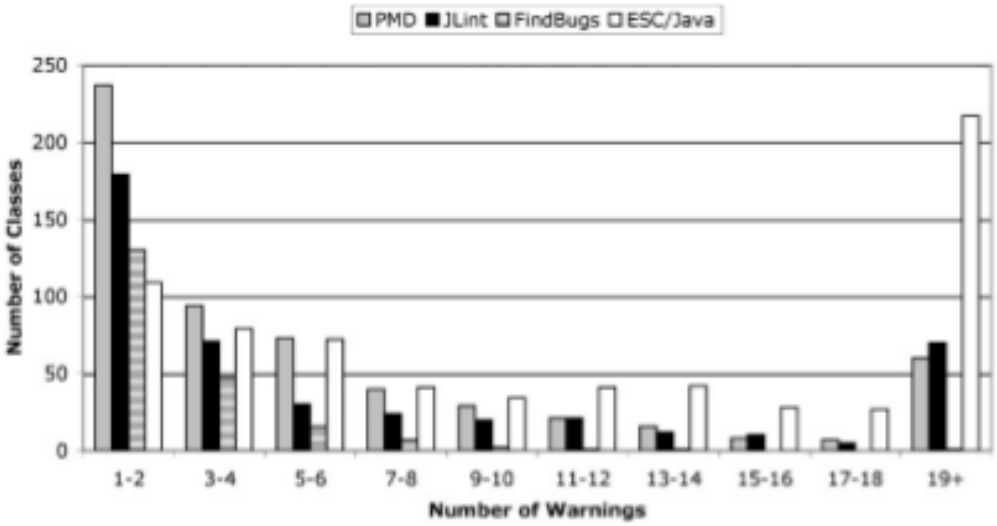
Il existe un autre paquet de règles.
On peut le trouver au format jar sur http://fb-contrib.sourceforge.net/

Comparatifs entre FindBugs et PMD (ainsi que d'autres outils non analysés)

Bug Category	Example	ESC/Java	FindBugs	JLint	PMD
General	Null dereference	√*	√*	√*	√
Concurrency	Possible deadlock	√*	√	√*	√
Exceptions	Possible unexpected exception	√*			
Array	Length may be less than zero	√		√*	
Mathematics	Division by zero	√*		√	
Conditional, loop	Unreachable code due to constant guard		√		√*
String	Checking equality using == or !=		√	√*	√
Object overriding	Equal objects must have equal hashcodes		√*	√*	√*
I/O stream	Stream not closed on all paths		√*		
Unused or duplicate statement	Unused local variable		√		√*
Design	Should be a static inner class		√*		
Unnecessary statement	Unnecessary return statement				√*

√ - tool checks for bugs in this category * - tool checks for this specific example

Name	NCSS (Lines)	Class Files	Time (min:sec.csec)				Warning Count			
			ESC/Java	FindBugs	JLint	PMD	ESC/Java	FindBugs	JLint	PMD
Azureus 2.0.7	35,549	1053	211:09.00	01:26.14	00:06.87	19:39.00	5474	360	1584	1371
Art of Illusion 1.7	55,249	676	361:56.00	02:55.02	00:06.32	20:03.00	12813	481	1637	1992
Tomcat 5.019	34,425	290	90:25.00	01:03.62	00:08.71	14:28.00	1241	245	3247	1236
JBoss 3.2.3	8,354	274	84:01.00	00:17.56	00:03.12	09:11.00	1539	79	317	153
Megamek 0.29	37,255	270	23:39.00	02:27.21	00:06.25	11:12.00	6402	223	4353	536



Estimation de code performant

La catégorie "Optimization Rules" de règles de PMD, ainsi que les catégories "Multithreaded correctness" et "Performance" de FindBugs pourront aider dans une moindre mesure.

Choix technologique

- Concernant checkstyle, cet outils va en effet révéler des soucis de “design” de code, en revanche, il reste orienté sur cet aspect là et est très verbeux.
- PMD va aider à vérifier des contraintes de vérifications plus complexes ou pour des problèmes plus spécifiques comme l’implémentation correcte de la fonction de clonnage d’une instance.
- Findbugs va aider à révéler les bugs applicatifs ou de design de l’application, mais est très largement moins complet que PMD comme nous pouvons le voir dans le schéma de nombre d’alertes ressorties entre PMD et Findbugs

La différence entre le grand nombre d’alertes trouvées entre PMD et Findbugs est notamment due à la richesse de règles de vérification ajoutée à **PMD** qui fait de cet outils une sécurité d’un point de vue qualité, performance et optimisation du code lors d’un projet.