

Contrôler la qualité de ses projets avec Sonar

par [Romain Linsolas](#)

Date de publication : 13/09/2008

Dernière mise à jour : 11/08/2011

Cet article a pour mission de vous faire découvrir ce qu'est l'outil Sonar, comment l'installer et l'utiliser.

I - Introduction.....	4
I-A - De l'intérêt de la qualité d'un logiciel, et de son contrôle.....	4
I-B - Glossaire.....	5
I-C - Description.....	6
I-D - Principales fonctionnalités.....	7
I-E - Version utilisée.....	8
I-F - Licence.....	8
I-G - Historique.....	8
I-H - Démonstration.....	10
II - Installation et configuration.....	10
II-A - Prérequis.....	10
II-B - Installation et configuration.....	11
II-B-1 - Téléchargement.....	11
II-B-2 - Paramétrage de la base de données.....	11
II-B-3 - Installation et configuration.....	11
II-B-2-a - Installation comme service Windows.....	12
II-B-2-b - Intégration à un serveur Tomcat.....	12
II-C - Mise à jour d'un serveur existant.....	12
II-D - Démarrage.....	13
III - Analyse d'un projet.....	13
III-A - Prérequis.....	13
III-B - Utilisation du plugin Java.....	14
III-C - Utilisation d'une tâche Ant.....	15
III-D - Utilisation de l'exécuteur Java.....	16
III-E - Automatisation.....	16
IV - Les outils externes.....	17
IV-A - Les règles.....	18
IV-B - Checkstyle.....	18
IV-C - PMD.....	18
IV-D - Findbugs.....	18
IV-E - Sonar Squid.....	19
IV-F - Cobertura, Clover, JaCoCo.....	19
V - Visualisation des données.....	19
V-A - Liste des projets.....	19
V-B - Vue d'un projet.....	21
V-B-1 - Vue "Dashboard".....	22
V-B-2 - Vue "Components".....	24
V-B-3 - Vue "Violations drilldown".....	24
V-B-4 - Vue "Time machine".....	26
V-B-4-a - Time machine.....	26
V-B-4-b - Vue différentielle.....	27
V-B-5 - Vue "Clouds".....	28
V-B-6 - Vue "Design".....	29
V-B-7 - Vue "Hotspots".....	30
V-B-8 - Vue "Libraries".....	31
V-B-9 - Vue "Settings".....	32
V-B-10 - Vue "Project Roles".....	33
V-C - Les métriques.....	34
V-C-1 - Les différentes métriques et leur mode de calcul.....	34
V-C-2 - Interprétation des résultats.....	34
V-D - Visualisation du code source.....	35
V-D-1 - Vues.....	35
V-D-2 - Revue de code.....	36
VI - Fonctionnalités d'administration.....	37
VI-A - Administration globale.....	37
VI-A-1 - Gestion des profils de qualité.....	38
VI-A-2 - Métriques et événements manuels.....	41
VI-A-3 - Gestion des droits et de la sécurité.....	43

VI-A-4 - Configuration de Sonar.....	44
VI-A-5 - Centre de mise à jour.....	45
VI-B - Administration d'un projet.....	45
VI-C - Modification de la page d'accueil.....	46
VII - Méthodologie d'activation des règles.....	48
VII-A - Étape 1 : définir les règles de codage.....	48
VII-B - Étape 2 : définir les règles les plus graves.....	48
VII-C - Étape 3 : faire évoluer le niveau des règles.....	49
VII-D - Variante.....	49
VIII - Plugins.....	50
VIII-A - Intérêt des plugins.....	50
VIII-B - Installation.....	53
VIII-C - Développement de nouveaux plugins.....	53
VIII-D - API web services de Sonar.....	53
IX - Conclusion.....	54
X - Références.....	54

I - Introduction

I-A - De l'intérêt de la qualité d'un logiciel, et de son contrôle

Sonar offre une solution performante du contrôle de la qualité du logiciel. Mais qu'est-ce que la qualité d'un logiciel, et en quoi est-il important de la contrôler ?

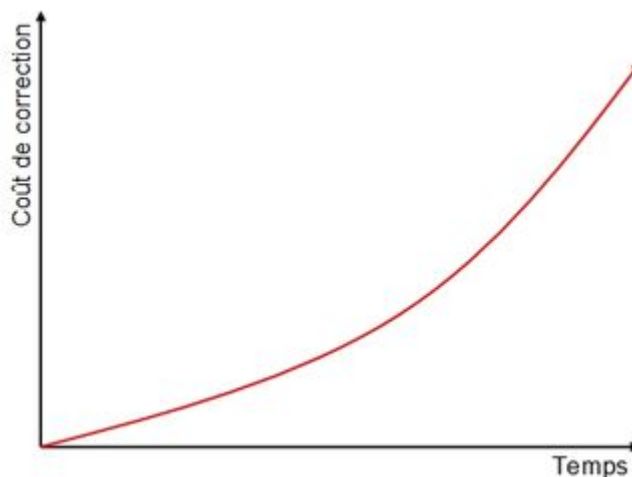
En paraphrasant **Wikipédia**, *la gestion de la qualité est l'ensemble des activités qui concourent à l'obtention de la qualité dans un cadre de production de biens ou de services* (dans notre cas, d'un logiciel). *Plus largement, c'est aussi un moyen que se donnent certaines organisations, dans des buts tels que la mise en conformité par rapport aux standards du marché.*

Dans le monde informatique en général, et en Java en particulier, la qualité d'une application va être directement liée à la qualité du code. De nombreux outils s'affairent à contrôler certains aspects de cette qualité du code : exécution de tests unitaires, analyse de la couverture du code par ces tests, vérifications du respect des règles de codage, etc. Il est donc possible de contrôler la qualité de son code grâce à ces outils, et d'avoir une confiance accrue en son application !

Le contrôle de la qualité va donc pousser l'équipe de développement à adopter et à respecter certains standards de développement. Le but de tout cela étant bien entendu de rendre le code plus sûr, mais de permettre d'y déceler les erreurs le plus rapidement possible... et donc de les corriger !

Le "**Toyota Way**" correspond à une méthodologie extrêmement appréciée aujourd'hui, aussi appelée le "**Lean**". Celle-ci est basée sur 14 principes dont l'un d'eux est le "*Build a culture of stopping to fix problems, to get quality right the first time*". Ce principe est là pour nous rappeler qu'il est impossible d'accélérer et de soutenir une fréquence de production sans que la qualité soit au coeur de toutes les actions. Autrement dit, il n'est pas possible d'aller vite sans qualité, mais qu'il n'est pas possible non plus de faire de la qualité sans vitesse. C'est aussi pour cela qu'il est aujourd'hui primordial de disposer d'une intégration continue et d'un processus itératif et incrémental.

L'un des intérêts de la surveillance de la qualité est la détection précoce des éventuels problèmes. Or lorsque l'on sait que le coût de la correction d'une erreur augmente considérablement avec le temps (voir image ci-dessous), on comprend très vite l'importance de la détection rapide des erreurs...



Le coût de correction d'une erreur croît exponentiellement avec le temps...

Comme le rappelle **Tom DeMarco**, "*You can't control what you can't measure*", "On ne peut contrôler ce que l'on ne mesure pas"... D'où l'importance d'utiliser un outil tel que Sonar !

I-B - Glossaire

Dans cet article, certains termes seront utilisés, et pour aider à leur compréhension, voici un petit glossaire.

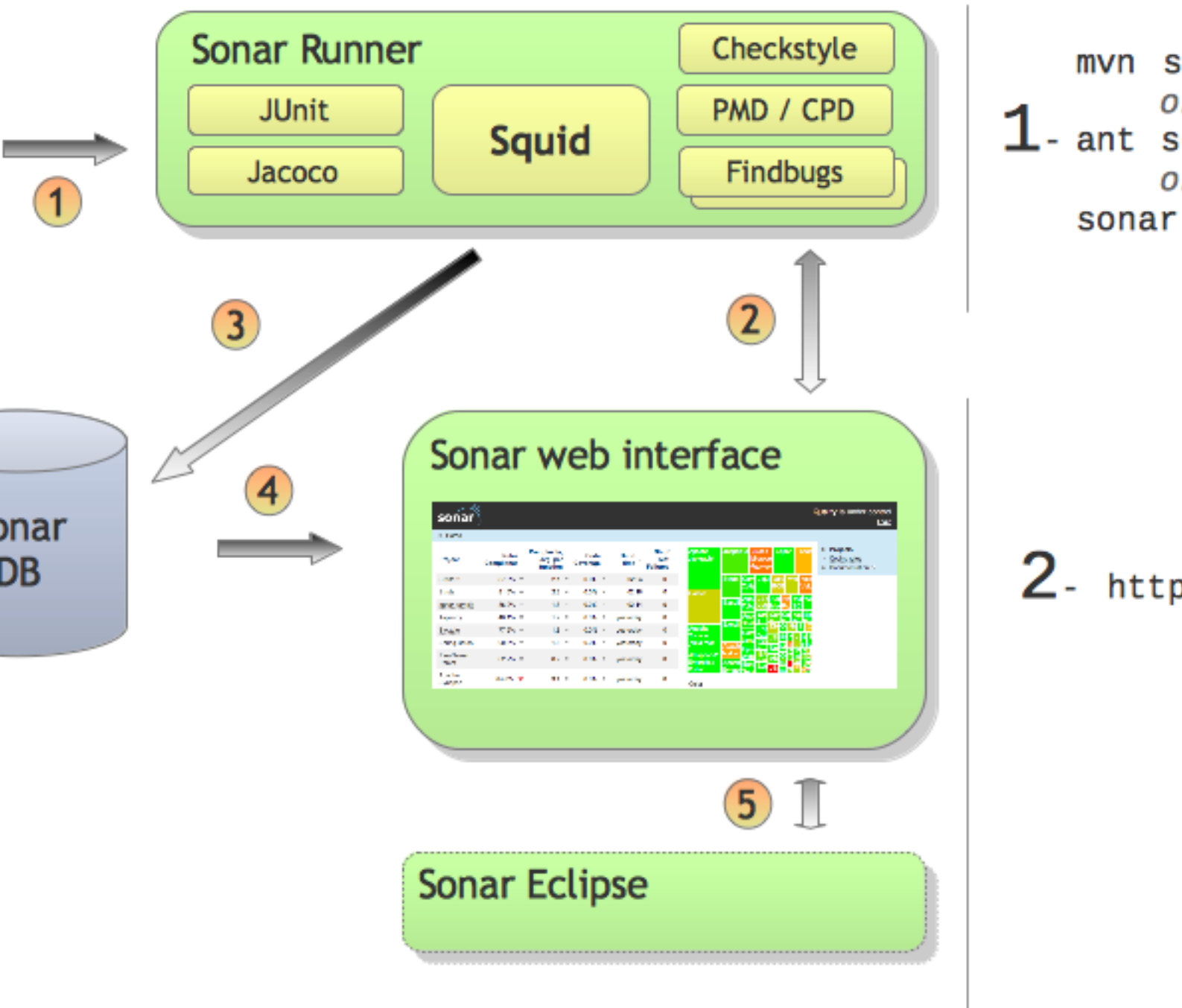
Mot	Définition	Exemple
Complexité cyclomatique	Métrique destinée à mesurer la complexité d'un code informatique, en comptabilisant le nombre de "chemins" possibles au sein de ce code. Ainsi, l'utilisation de boucles (<i>while</i> , <i>for</i> ...) ou encore d'instructions de branchements conditionnels (<i>if</i> , <i>else</i>) augmentent cette complexité. (Définition de Wikipédia)	
Dette technique	La dette technique est une dette que vous encourez à chaque fois que vous faites quelque chose de la mauvaise façon en laissant la qualité du code se détériorer. Tout comme les dettes financières, agir ainsi est plus facile sur le court terme. Mais au fil du temps, les "intérêts" que vous payez sur cette dette deviennent énormes, jusqu'à atteindre un point où une réécriture complète de l'application devient plus facile que de maintenir ou de modifier le code existant (Explications de Martin Fowler en anglais).	
Métrique	Une métrique est une caractéristique (ou une propriété) d'une application.	Nombre de lignes de code, pourcentage de couverture de code par les tests unitaires, nombre de violations de règles de codage, etc.
Mesure	Valeur d'une métrique à un moment précis de la vie de l'application.	Au 5 juin, l'application comptait 42.000 lignes de code, 317 violations et avait une couverture de code de 78 %.
Règle de codage	Définition d'une bonne pratique à respecter dans son code. Cela peut également concerner des mauvaises pratiques à éviter, car souvent source d'erreur.	Ne pas dépasser une complexité cyclomatique de 10 pour une méthode, redéfinir la méthode <i>hashCode</i> lorsque l'on redéfinit <i>equals</i> , etc.
Violation (d'une règle de codage)	Non-respect, dans une partie du code précise, d'une règle de codage particulière.	

I-C - Description

Sonar est un outil open source initialement développé par la société suisse **Hortis**. Depuis novembre 2008, c'est la société suisse **SonarSource** qui se charge du développement et du support de Sonar. Le but principal de cet outil est de fournir une analyse complète de la qualité d'une application en fournissant de nombreuses statistiques (ou **métriques**) sur ses projets. Ces données permettent ainsi d'évaluer la qualité du code, et d'en connaître l'évolution au cours du développement.

D'un point de vue architectural, Sonar est composé de plusieurs couches :

- un exécuteur (basé sur Maven 2/3, Ant ou un exécuteur Java) dont le but sera de lancer un certain nombre d'outils d'analyse, et d'en agréger les résultats ;
- une base de données, qui stocke et historise les informations sur les projets surveillés par Sonar ;
- le serveur web qui permet la navigation et la consultation des analyses réalisées sur les projets ;
- éventuellement un plugin pour Eclipse qui offre une meilleure intégration des données de Sonar dans son outil de développement.



Architecture de Sonar

I-D - Principales fonctionnalités

Nous listons ici les principales fonctionnalités de l'outil Sonar. Elles seront décrites en détail dans les chapitres suivants de cet article.

- 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.
- 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.

I-E - Version utilisée

Dans cet article, nous utiliserons la version de démonstration en ligne, appelée **Nemo**. L'accès à cette instance de démonstration permet de tester Sonar sans pour autant l'installer sur sa machine, mais également de voir directement les nouveautés des dernières versions, puisque Nemo est régulièrement mis à jour.

Nous nous baserons ici sur la version **2.9** de Sonar, sortie mi-juillet 2011. De fait, certaines fonctionnalités, ainsi que les processus d'installation et de configuration sont propres à cette version. La plupart des captures d'écran ont été réalisées à partir de l'instance de démonstration **Nemo**.

I-F - Licence

Sonar est un logiciel open source (licence *Gnu LGPL*) et est donc librement utilisable. Cette licence est visible [ici](#). Le code source est disponible et consultable [ici](#).

I-G - Historique

Sonar est un produit relativement jeune, bien que déjà très apprécié. En effet, la toute première version ne date que de la fin de l'année 2007. Le tableau ci-dessous montre succinctement l'historique de l'outil, en mettant en avant les principales nouveautés apportées par chaque version.

Version	Date de sortie	Commentaires
2.9	18 juillet 2011	Améliorations des revues manuelles, historisation des profils qualité.
2.8	19 mai 2011	Revue de code manuelles, comparateur de profils qualité.
2.7	1er avril 2011	Couverture du nouveau code, meilleure intégration avec les gestionnaires de sources.
2.6	18 février 2011	Support de Ant.
2.5	14 janvier 2011	Vues différentielles, suivi temporel des violations, nouvelles règles.
2.4	16 novembre 2010	Tableaux de bord paramétrables, centre de

		mise à jour, ajout de règles d'architecture.
2.3	13 octobre 2010	Exportation / importation des profils qualité.
2.2	15 juillet 2010	Ajout des favoris et des filtres.
2.1	5 mai 2010	Analyse des dépendances des projets, nouvelles règles.
2.0	10 mars 2010	Ajout de métriques de Chidamber et Kemerer, matrice de structure.
1.12	7 décembre 2009	Gestion des utilisateurs et des groupes, coloration syntaxique du code source
1.11	5 octobre 2009	Nouvelle page de composants, nouvelles métriques.
1.10	mi-août 2009	De nombreux problèmes ont été corrigés.
1.9	25 mai 2009	Intégration du moteur Sonar-Squid (remplaçant JavaNCSS).
1.8	17 avril 2009	Introduction des "Hotspots", support de Maven 2.1.
1.7	18 mars 2009	Possibilité d'ignorer certaines classes ou packages d'un projet, affichage du détail des erreurs des tests unitaires, "nuage de couverture" pour les packages et modules.
1.6	9 février 2009	Intégration des seuils d'alerte, amélioration de la gestion des profils de qualité.
1.5	16 décembre 2008	Début du support des plugins, gestion de Findbugs.
1.4	8 août 2008	Version de transition, avant la version 1.5.
1.3	17 juin 2008	Amélioration des performances et de la stabilité de Sonar, déploiement sur Tomcat.
1.2	27 mars 2008	Refonte importante de l'aspect graphique et de l'affichage des rapports.
1.1	29 février 2008	Gestion de la "time machine", fonctionnalités d'administration. Gestion des extensions PMD et Checkstyle.
1.0	21 novembre 2007	Première version finale de Sonar. Essentiellement des corrections de bogues de la version bêta.

La principale nouveauté de la version **2.10** qui devrait sortir pour la rentrée 2011 sera la support de l'internationalisation ("i18n") et pourra ainsi être utilisé en français. Le site Nemo (voir le chapitre suivant) permet déjà d'avoir un aperçu de cette traduction.

I-H - Démonstration

SonarSource met à disposition le site <http://nemo.sonar.codehaus.org>, qui regroupe un certain nombre de projets open source. Cela permet de voir les différentes fonctionnalités de l'outil, sans nécessairement l'installer soi-même. De plus, cette version publique est toujours mise à jour, ce qui permet également de tester les nouvelles fonctionnalités.


Configuration Log in

Projects Activity over 90 days Open Source Forges Sonar Platform Activity Sonar Plugins

Time changes

Alert	Name ^	Lines of code	Coverage	SQALE Remediation Cost	Duplicated lines (%)
	MasterProject	7,344,537 ▲	24.5%	62,212.5 ▲	4.1%
	AS3 Core Lib	2,814		0.0	4.7%
⚠	ActiveMQ	162,103 ▲		1,096.3 ▲	20.4%
⚠	Adobe Flex PMD Java Parent	16,696	5.2%	101.1 ▼	0.0%
✓	Aisl ib application framework	12,187	38.6%	106.7	1.1%
⚠	All Sonar plugins	40,394 ▼	53.0%	95.9 ▼	0.5%
✓	Apache Abdera	49,519		252.6 ▲	2.5%
⚠	Apache Archiva	26,276	60.3%	148.9 ▲	2.0%
✓	Apache Aries	49,209 ▼	38.6% ▲	496.4 ▼	3.1%
⚠	Apache CXF	212,849 ▼	43.1%	2,170.6 ▼	2.2%
✓	Apache Cocoon 3: Root	15,487 ▲	26.8%	132.1 ▲	1.2%
✓	Apache Cocoon [build root]	100,871	10.2%	1,291.6 ▼	2.3%
✓	Apache Commons Digester	9,306 ▲	70.6%	43.2	0.0%
⚠	Apache Empire-db	31,958 ▲	13.5%	418.9 ▲	8.6%
⚠	Apache Felix	167,657 ▲	10.5%	2,412.4 ▲	2.4%
⚠	Apache Jackrabbit	215,778 ▲	30.9% ▲	2,212.0 ▼	3.2%
✓	Apache James Server	28,496 ▲	28.6% ▲	343.0 ▲	2.4%
✓	Apache Lucene	67,288 ▼		283.3 ▲	2.0%
✓	Apache MINA	36,383	39.6%	352.0	1.7%
✓	Apache Maven	55,095	33.9%	409.2	0.6%
✓	Apache Pluto	23,607	10.5%	225.8 ▼	3.4%
✓	Apache Qpid	44,868		169.4	3.1% ▼
⚠	Apache Shindig Project	43,091 ▲	74.5%	213.6 ▲	0.3%

II - Installation et configuration

Il est à noter que la procédure d'installation et de configuration expliquée ici se base sur la version **2.9** de Sonar. Même si cette procédure n'a que peu évolué depuis plusieurs versions, en cas de besoin,  le site de Sonar propose les procédures d'installation des versions antérieures de l'outil.

II-A - Prérequis

Pour pouvoir fonctionner, Sonar nécessite l'installation d'une version 5 (ou supérieure) de Java.

Sonar historise les données (mesures et métriques) des projets, et demande donc un accès à une base de données. Par défaut, Sonar est livré avec une base de données interne (**Apache Derby**), mais c'est avant tout dans le but

de tester facilement l'outil. Il est toutefois très fortement conseillé d'avoir recours à une base de données de type **MySQL 5.x+**, **Oracle 10g+**, **PostgreSQL 9.x+** ou **MS SQLServer 2005** en local. En termes d'espace de stockage, Sonar indique n'avoir besoin "que" de 2 Go pour historiser deux années d'analyse sur un million de lignes de code !

Il peut être soit démarré en tant que serveur propre (et pouvant également s'installer comme un service Windows), soit être déployé sur un serveur d'application type Tomcat (la création d'un .WAR est requise).

II-B - Installation et configuration

II-B-1 - Téléchargement

En premier lieu, il faut télécharger l'archive Sonar sur le site de [SonarSource](#). À l'heure actuelle, la dernière version disponible est la **2.9** (juillet 2011). Il s'agit d'un fichier ZIP dont on décompressera le contenu dans un répertoire.

II-B-2 - Paramétrage de la base de données

Comme nous l'avons dit, Sonar embarque une base de données Apache Derby. Toutefois, pour des raisons de confort, il peut être préférable de configurer Sonar pour accéder à une autre base de données, MySQL 5.x, Oracle 10g XE et PostgreSQL 8.3 étant supportées. Il faut ensuite éditer le fichier de configuration `conf/sonar.properties`. Par défaut, les trois lignes suivantes correspondent à la configuration Apache Derby :

```
sonar.jdbc.url: jdbc:derby://localhost:1527/sonar;create=true
sonar.jdbc.driver: org.apache.derby.jdbc.ClientDriver
sonar.jdbc.defaultTransactionIsolation: 1
```

Pour utiliser une base de données tierce, il faudra tout d'abord commenter ces trois lignes. Pour cela, il suffit d'ajouter un **#** devant chaque ligne. Il faut ensuite décommenter les lignes correspondant à la base de données choisie. Par exemple, pour MySQL, cela nous donnera :

```
sonar.jdbc.url: jdbc:mysql://localhost:3306/sonar?useUnicode=true&characterEncoding=utf8
sonar.jdbc.driver: com.mysql.jdbc.Driver
sonar.jdbc.validationQuery: select 1
```

Au niveau de la base de données, il faut créer la base **sonar** avec les droits de création de tables. Dans le cas où l'on utilise la base de données MySQL, il est possible d'utiliser le script se trouvant dans le répertoire *extras/database/mysql*.

II-B-3 - Installation et configuration

Une fois le ZIP téléchargé et décompressé et la configuration de la base de données définie, il suffit d'exécuter la commande suivante (pour Windows) :

bin\windows-x86-32\StartSonar.bat

ou, pour les autres environnements :

bin/[OS]/sonar.sh console

II-B-2-a - Installation comme service Windows

Sonar peut être installé comme service Windows. La configuration de ce service est définie dans les fichiers *conf/wrapper.conf* et *conf/sonar.properties*. Il est préférable de ne pas toucher au premier de ces fichiers, celui-ci décrivant les caractéristiques du service Windows.

Il est possible de changer le port du serveur Sonar, qui est par défaut le port 9000. On éditera le fichier *conf/sonar.properties* pour changer la ligne suivante :
`sonar.web.port:9000`

À noter que cette valeur n'est utilisée que si Sonar est exécuté en serveur propre (donc ne concerne pas l'intégration à Tomcat).

Il suffit maintenant d'installer le service Windows en exécutant le fichier *bin/windows-x86-32/InstallNTService.bat*. Le démarrage de Sonar se fait désormais en démarrant le service Windows ainsi créé.

Notez la présence du fichier *bin/windows-x86-32/UninstallNTService.bat* qui permet de supprimer le service Windows, en cas de besoin.

II-B-2-b - Intégration à un serveur Tomcat

Il est possible d'intégrer Sonar à un serveur **Tomcat** 5.x, 6.x, 7.x ou **Jetty** 6.x.

La première chose est de configurer Sonar via le fichier *conf/sonar.properties*. Ensuite, il faut exécuter le script *war/build-war.bat* (ou *war/build-war.sh*). Une fois la commande terminée, le fichier **sonar.war** se trouvera dans le répertoire *war/*. Il suffira donc de le déployer sur le serveur Tomcat (on peut par exemple déposer ce nouveau fichier dans le répertoire *webapps/* de Tomcat).

Sonar étant assez gourmand en mémoire, il est nécessaire de modifier le paramètre *CATALINA_OPTS* (se trouvant dans *bin/catalina.bat* par exemple) pour affecter un minimum de 512 Mo de mémoire :

```
CATALINA_OPTS="-Xmx=512m"
```

Par défaut, le port de Tomcat est le 8080. L'accès au serveur de Sonar se fera donc à l'adresse *http://url-du-serveur-tomcat:8080/sonar*.

II-C - Mise à jour d'un serveur existant

Dans le cas où vous désirez mettre à jour un serveur Sonar déjà existant, il vous faudra suivre la procédure décrite sur [cette page](#). À noter que les versions récentes de Sonar proposent un centre de mise à jour (voir le chapitre d'administration de Sonar) qui indiquera la disponibilité éventuelle d'une nouvelle version de l'outil.

[Installed Plugins](#)
[Available Plugins](#)
[Plugin Updates](#)
[System Updates](#)

Sonar 2.6

Date: Feb 18, 2011

Release Notes ⓘ : Support native Ant projects and +40 improvements/bug-fixes

How to upgrade: Follow those steps to upgrade Sonar from version 2.5 to version 2.6 :

1. ⚠ **Uninstall** the plugin Clover which is not compatible with Sonar 2.6.
2. **Upgrade** the plugin Views to version 1.5.4
3. Stop Sonar
4. **Download** ⓘ and install Sonar 2.6 after having carefully read the upgrade guide.
5. Start Sonar

Updated on Fri Jul 15 15:51:45 CEST 2011. [Refresh](#)

Il conviendra de réaliser un backup de la base de données avant toute mise à jour !


II-D - Démarrage

Si Sonar n'a pas été installé comme service Windows, la commande **bin/windows-x86-32/StartSonar.bat** (Windows) ou **bin/[plateforme]/sonar.sh start** (pour les autres plateformes) permet de démarrer le serveur Sonar. Par défaut, il suffit de se rendre à l'adresse **http://url-du-serveur-sonar:9000** pour accéder à Sonar. Si Sonar est déployé sur un serveur Tomcat, l'adresse sera plutôt **http://url-du-serveur-tomcat:8080/sonar** (8080 étant le port par défaut de Tomcat). Lors du premier démarrage, une page vide nous accueille :



Lorsqu'il s'agit d'une migration de version de Sonar, le premier accès au serveur va afficher une page indiquant qu'une mise à jour de la base de données doit être réalisée.

Il ne nous reste maintenant plus qu'à lancer l'analyse d'un projet (voir chapitres suivants).

 *En cas de problème, il est possible de consulter les logs de l'application dans le répertoire logs/ (en particulier le fichier sonar.log).*

III - Analyse d'un projet

III-A - Prérequis

Sonar est capable d'analyser des projets de différents langages.

- Java
- **C**
- **C#**
- **PHP**
- **Cobol** (plugin commercial)
- **Natural** (plugin commercial)
- **PL/SQL** (plugin commercial)
- **Flex**
- **Visual Basic 6** (plugin commercial)
- **JavaScript**
- **XML**
- **Groovy**
- **JSP et JSF**

Dans la suite de cet article, nous nous concentrerons plus particulièrement sur l'analyse de projets Java. Pour connaître la procédure nécessaire à l'analyse d'un projet autre que Java, il suffit de se référer à la documentation du plugin gérant ce langage. À noter que les résultats et fonctionnalités proposées par Sonar sont généralement indépendants du langage, et donc ce qui sera par la suite expliqué sur l'analyse des projets Java est généralement vrai pour les analyses des autres projets.

Concernant l'analyse d'un projet Java en elle-même, elle peut être réalisée de trois façons :

- **en utilisant Maven ;**
- **grâce à une tâche Ant ;**
- **à l'aide d'un exécuteur Java.**

III-B - Utilisation du plugin Java

Si votre projet utilise déjà **Maven 2 ou 3**, le plus simple est de faire appel au plugin Maven de Sonar. La première chose à faire est de définir quelques paramètres Maven à placer de préférence dans son fichier *settings.xml*. Nous redéfinissons là les paramètres de connexion à la base de données, ainsi que l'URL du serveur Sonar.

```
...
  <profile>
    <id>sonar</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <properties>
      <sonar.jdbc.url>jdbc:mysql://localhost:3306/sonar?
useUnicode=true&characterEncoding=utf8</sonar.jdbc.url>
      <sonar.jdbc.driver>com.mysql.jdbc.Driver</sonar.jdbc.driver>
      <sonar.jdbc.username>sonar</sonar.jdbc.username>
      <sonar.jdbc.password>sonar</sonar.jdbc.password>
      <!-- La ligne ci-
dessous n'est utile que si le port par défaut (9000) est modifié dans le fichier de configuration. -->
      <sonar.host.url>http://url-serveur:1234</sonar.host.url>
    </properties>
  </profile>
...
```

Il suffit ensuite de lancer la commande suivante :

mvn sonar:sonar

Il est recommandé de réaliser une installation des artefacts Maven de son projet avant (surtout dans le cas d'un projet multimodules). Nous procéderons ainsi (on notera que les tests sont ignorés lors de la première commande, car ils seront exécutés par le plugin Sonar) :


```
mvn clean install -Dtest=false -DfailIfNoTests=false
mvn sonar:sonar
```

Si la commande précédente ne peut être utilisée, ou que l'on souhaite réaliser l'analyse en une seule commande Maven, on privilégiera la commande suivante, ce qui aura toutefois l'inconvénient d'exécuter les tests *deux* fois :

```
mvn clean install sonar:sonar -Dmaven.test.failure.ignore=true
```

Une fois cette commande exécutée, il sera possible de consulter l'analyse en se rendant sur le serveur de Sonar.

III-C - Utilisation d'une tâche Ant

Pour lancer une analyse Sonar en utilisant Ant, il est nécessaire de disposer des versions au moins égales à 1.7.1 d'Ant, 1.5 de Java et 2.8 de Sonar.

Il faut ensuite télécharger la tâche Ant **ici**, puis déclarer cette tâche dans son script Ant :

```
...
<!-- Ajout de la tâche Sonar pour Ant. -->
<taskdef uri="antlib:org.sonar.ant" resource="org/sonar/ant/antlib.xml">
  <!-- Cette librairie peut aussi être ajoutée dans le répertoire ${ANT_HOME}\lib -->
  <!-- Dans ce cas, le node classpath ci-dessous est inutile. -->
  <classpath path="path/to/sonar/ant/task/lib" />
</taskdef>

<!-- Propriétés propres à Sonar, par exemple pour MySQL : -->
<property name="sonar.jdbc.url"
  value="jdbc:mysql://localhost:3306/sonar?useUnicode=true&characterEncoding=utf8" />
<property name="sonar.jdbc.driverClassName" value="com.mysql.jdbc.Driver" />
<property name="sonar.jdbc.username" value="sonar" />
<property name="sonar.jdbc.password" value="sonar" />
<!-- Et dans le cas où le serveur se trouve sur une autre machine -->
<property name="sonar.host.url" value="http://url-serveur:1234" />
...
```

On définira ensuite une *target* Ant dans son script :

```
<project name="Mon projet">
  <!-- Définition de la tâche Sonar pour Ant. -->
  <taskdef uri="antlib:org.sonar.ant" resource="org/sonar/ant/antlib.xml">
    <classpath path="path/to/sonar/ant/task/lib" />
  </taskdef>

  <!-- Ajout de la cible Ant -->
  <target name="sonar">
    <!-- On définit les propriétés obligatoires pour Sonar -->
    <property name="sonar.sources" value="list of source directories separated by a comma" />

    <!-- On définit les propriétés optionnelles pour Sonar -->

    <property name="sonar.projectName" value="this value overrides the name defined in Ant root node" />

    <property name="sonar.binaries" value="list of directories which contain for example the Java bytecode" />
    <property name="sonar.tests" value="list of test source directories separated by a comma" />

    <property name="sonar.libraries" value="list of paths to libraries separated by a comma (These libraries are fo
    ...

    <sonar:sonar key="org.example:example" version="0.1-SNAPSHOT" xmlns:sonar="antlib:org.sonar.ant"/>
  </target>
</project>
```

Une fois ceci fait, il suffit de lancer la commande suivante pour lancer l'analyse :

ant sonar

Il est possible d'analyser de cette façon des projets multimodules. Pour ce faire, on se rendra sur [la page wiki dédiée de Sonar](#).

III-D - Utilisation de l'exécuteur Java

La dernière possibilité pour analyser un projet Java est de passer par un exécuteur Java. Cela nécessite Java 1.5 et Sonar 2.6. Il faut ensuite télécharger l'exécuteur Java, disponible [ici](#).

Pour analyser un projet, il faut créer un fichier appelé *sonar-project.properties*, dont le contenu sera le suivant :

```
# Métadonnées du projet.
sonar.projectKey=my:project
sonar.projectName=My project
sonar.projectVersion=1.0

# Chemin vers les sources (obligatoire).
sources=srcDir1,srcDir2

# Chemin vers les sources de test (optionnel).
tests=testDir1,testDir2

# Chemin vers les binaires du projet (optionnel).
binaries=binDir

# Chemin vers les bibliothèques du projet (optionnel).
libraries=junit.jar

# Décommenter ces lignes si le projet utilise des spécificités de Java 1.5 ou 1.6
#sonar.java.source=1.5
#sonar.java.target=1.5

# Décommenter cette ligne dans le cas où le projet n'est pas un projet Java.
# Dans ce cas, il faut indiquer de quelle nature est le projet (ici Cobol).
#sonar.language=cobol

# Paramètres avancés, spécifiques au projet.
my.property=value
```

Une fois tout ceci configuré, il suffira de lancer la commande suivante pour analyser le projet :

sonar-runner

III-E - Automatisation

Nous venons de voir comment lancer l'analyse Sonar manuellement. Mais il est bien plus profitable de faire réaliser cette tâche automatiquement, grâce à un outil d'intégration continue. Parmi les serveurs d'intégration continue les plus connus, on citera :

- **Jenkins** ou **Hudson** (à lire mon article sur cet outil [ici](#)) ;
- **Bamboo** ;
- **Cruise Control** ;
- **Et bien d'autres encore...**

Sonar propose des plugins pour **Jenkins / Hudson**, **Bamboo** ou **AnthillPro**, offrant ainsi une meilleure interaction des outils.

Nous voyons ici la configuration de notre serveur Sonar depuis l'interface Jenkins / Hudson :

Sonar

Sonar installations

Name: Sonar 2.9

Disable: ☐

Check to quickly disable Sonar on all jobs.

Server URL: http://localhost:9000

Default is http://localhost:9000

Server Public URL: http://localhost:9000

If not specified, then Server URL will be used

Database URL: jdbc:mysql://localhost:3306/sonar?useUnicode=true&characterEncoding=utf8

Do not set if default embedded database.

Database login: sonar

Default is sonar

Database password: *****

Default is sonar

Database driver: com.mysql.jdbc.Driver

Do not set if you use the default embedded database on localhost.

Version of sonar-maven-plugin:

If not specified, then sonar:sonar will be used.

Additional properties: -Dsonar.host.url=http://localhost:9000

Additional properties to be passed to the mvn executable (example : -Dsome.property=some.value)

Triggers

☐ Poll SCM

☒ Build periodically

☒ Manually started by user

☐ Build whenever a SNAPSHOT dependency is built

☐ Skip analysis on build failure

Add Sonar

Delete Sonar

List of Sonar installations

Configuration du plugin Sonar sur un serveur Jenkins / Hudson

IV - Les outils externes

Afin d'analyser un projet Java, Sonar va se baser en partie sur des outils externes, dont :

Outil	Site
Sonar Squid *	http://www.sonarsource.org/
Checkstyle	http://checkstyle.sourceforge.net
PMD	http://pmd.sourceforge.net
Findbugs	http://findbugs.sourceforge.net/
Cobertura **	http://cobertura.sourceforge.net
JaCoCo **	http://www.eclemma.org/jacoco/index.html
Clover **	http://www.atlassian.com/software/clover
Surefire	http://maven.apache.org/plugins/maven-surefire-plugin

* Depuis la version 1.9 de Sonar, l'obsolète projet **JavaNCSS** a été remplacé par Sonar Squid.

** Il est possible d'utiliser l'outil Clover grâce à **un plugin dédié**, mais il nécessitera toutefois un fichier de licence (il s'agit d'un outil commercial). Il est également possible d'utiliser **JaCoCo** pour calculer la couverture de code, en utilisant **le plugin adéquat**.

Une **page wiki** permet de lister les versions de ces outils utilisées nativement par Sonar.

IV-A - Les règles


Sonar propose trois profils qualité par défaut : *Sonar way*, *Sonar way with Findbugs* et *Sun checks*. Il est possible de définir d'autres profils de qualité en fournissant ses propres fichiers de configuration pour Checkstyle, PMD et Findbugs. Nous aborderons cela dans la partie administration de Sonar.

En cumulant ces trois outils (plus les règles spécifiques de Sonar), nous disposons de près de 800 règles possibles ! Chacune de ces règles peut être activée ou non, et est ensuite affectée à un niveau de sévérité :

- Blocker ;
- Critical ;
- Major ;
- Minor ;
- Info.

IV-B - Checkstyle

Checkstyle est une aide précieuse pour les développeurs afin de leur faire respecter des standards de codage précis. On y retrouve une liste assez complète de règles paramétrables (environ 130 règles) permettant de valider à peu près n'importe quel type de standard. On pourra ainsi vérifier que les lignes n'excèdent pas une certaine longueur, que la Javadoc est bien présente, que les standards de nommage sont bien respectés, etc. Checkstyle permet aussi d'améliorer l'écriture et la qualité de son propre code, en indiquant par exemple quelles expressions peuvent être simplifiées, quels blocs peuvent être supprimés, quelle classe doit être déclarée finale, etc.

La liste complète des quelque 130 règles disponibles sur Checkstyle se trouve  [ici](#).

IV-C - PMD

PMD va scanner le code Java à la recherche de problèmes potentiels, tels que :

- bogues possibles, blocs de code vides (*try ... catch*, *switch*) ;
- code "mort", non utilisé (essentiellement des méthodes privées, ou des paramètres) ;
- expressions trop lourdes, mauvaises utilisations des opérateurs de boucles ;
- code dupliqué, "erreurs de copier-coller"...

IV-D - Findbugs

Findbugs est un outil apprécié par les développeurs car il apporte une analyse assez fouillée de programmes Java, et permet de détecter des problèmes assez complexes. Il pourra ainsi nous informer sur différents aspects, tels que :

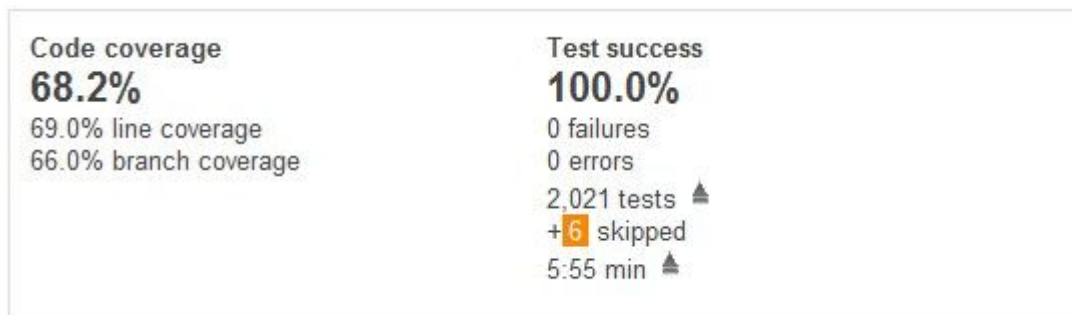
- mauvaises pratiques ;
- utilisation de vulnérabilités de certains bouts de code ;
- lister des problèmes épineux ;
- possibles pertes de performances ;
- gestion du *multithreading* ;
- problèmes liés à la sécurité ;
- etc.

IV-E - Sonar Squid

L'outil **JavaNCSS** était utilisé dans les premières versions de Sonar (jusqu'à la version 1.8). Hélas, ce projet n'a plus beaucoup évolué, et de nombreuses spécificités des dernières versions du langage Java n'étant pas supportées, l'équipe de Sonar avait alors décidé de développer son propre outil, Sonar Squid. Sonar Squid est partie intégrante de Sonar.

IV-F - Cobertura, Clover, JaCoCo

Cobertura, **Clover** et **JaCoCo** sont des outils permettant la mesure de la couverture de code par l'exécution des tests unitaires. Il s'agit là d'un indicateur extrêmement utile pour déterminer avec précision quelles parties de l'application ne sont pas suffisamment testées. Sonar propose d'obtenir une vision au niveau d'un projet, d'un package, d'une classe ou d'une méthode de cette couverture de test.



V - Visualisation des données

V-A - Liste des projets

Lorsque l'on se connecte sur la page d'accueil de son serveur Sonar, on visualise l'ensemble des projets gérés par Sonar. Cette liste est un tableau où sont affichées les principales informations pour chaque projet, comme par exemple :

- le nom du projet ;
- sa version ;
- la taille du projet ;
- le taux de respect des règles ;
- quand a été réalisé le dernier build ;
- les principaux liens (vers l'application, le site du projet, le serveur d'intégration continue, le serveur de gestion des sources, etc.) ;
- les éventuelles alertes sur le projet ;
- etc.

Projects		Activity over 90 days		Open Source Forges		Sonar Platform Activity		Sonar Plugins		Time changes	
Alert	Name ^	Lines of code	Coverage	SQALE	Remediation Cost	Duplicated lines (%)					
	MasterProject	7,344,537 ▲	24.5%		62,212.5 ▲	4.1%					
	AS3 Core Lib	2,814			0.0	4.7%					
	ActiveMQ	162,103 ▲			1,096.3 ▲	20.4%					
	Adobe Flex PMD Java Parent	16,696	5.2%		101.1 ▼	0.0%					
	AislLib application framework	12,187	38.6%		106.7	1.1%					
	All Sonar plugins	40,394 ▼	53.0%		95.9 ▼	0.5%					
	Apache Abdera	49,519			252.6 ▲	2.5%					
	Apache Archiva	26,276	60.3%		148.9 ▲	2.0%					
	Apache Aries	49,209 ▼	38.6% ▲		496.4 ▼	3.1%					
	Apache CXF	212,849 ▼	43.1%		2,170.6 ▼	2.2%					
	Apache Cocoon 3- Root	15,487 ▲	26.8%		132.1 ▲	1.2%					
	Apache Cocoon [build root]	100,871	10.2%		1,291.6 ▼	2.3%					
	Apache Commons Digester	9,306 ▲	70.6%		43.2	0.0%					
	Apache Empire-db	31,958 ▲	13.5%		418.9 ▲	8.6%					
	Apache Felix	167,657 ▲	10.5%		2,412.4 ▲	2.4%					
	Apache Jackrabbit	215,778 ▲	30.9% ▲		2,212.0 ▼	3.2%					
	Apache James Server	28,496 ▲	28.6% ▲		343.0 ▲	2.4%					
	Apache Lucene	67,288 ▼			283.3 ▲	2.0%					
	Apache MINA	36,383	39.6%		352.0	1.7%					
	Apache Maven	55,095	33.9%		409.2	0.6%					
	Apache Pluto	23,607	10.5%		225.8 ▼	3.4%					
	Apache Qpid	44,868			169.4	3.1% ▼					
	Apache Shindig Project	43,091 ▲	74.5%		213.6 ▲	0.3%					

L'administrateur de Sonar peut toutefois éditer les informations affichées dans ce tableau afin de présenter d'autres informations sur cette page de garde (voir le chapitre sur l'administration).

Le second onglet de cette page, appelé "Treemap" offre une vue où chaque projet est symbolisé par un carré, dont la taille et la couleur indiquent le niveau de certaines métriques (par exemple le nombre de lignes de code, des informations sur les tests unitaires, etc.). En survolant le carré correspondant à un projet, une info-bulle donnera de plus amples informations.



Pour obtenir les détails d'un projet en particulier, il suffit de cliquer sur son nom, dans la liste. La vue d'un projet s'affiche alors...

V-B - Vue d'un projet

La vue d'un projet est composée du dashboard (l'écran de contrôle), ainsi que d'un menu à sa gauche. Ce menu permet d'accéder aux principales informations :

- le "*Dashboard*" qui propose en une page une visualisation complète de la qualité du projet ;
- le "*Component*" qui permet de voir la vue de chaque module composant le projet ;
- le "*Violations drilldown*" est la page principale regroupant l'ensemble des violations des règles de qualité ;
- la "*Time machine*" montre l'évolution du projet au cours du temps ;
- les "*Clouds*", ou nuages, montrent les "*quick wins*" ou les points critiques du projet ;
- le "*Design*", donnant des informations sur l'aspect design du projet, telles que les dépendances entre les modules ;
- les "*Hotspots*" regroupent en une page les classes les plus influentes sur la qualité ;
- les "*libraries*" listant les dépendances tierces de chaque module ;
- les "*Settings*" permettent de configurer le projet (option visible seulement pour l'administrateur) ;
- les "*Project Roles*" pour définir les droits des différents groupes sur ce projet (option visible seulement pour l'administrateur).



D'autres menus peuvent apparaître avec l'utilisation de plugins (plugins *SQALE*, *Motion chart*, *Radiotor* ou *Timeline* par exemple).

V-B-1 - Vue "Dashboard"

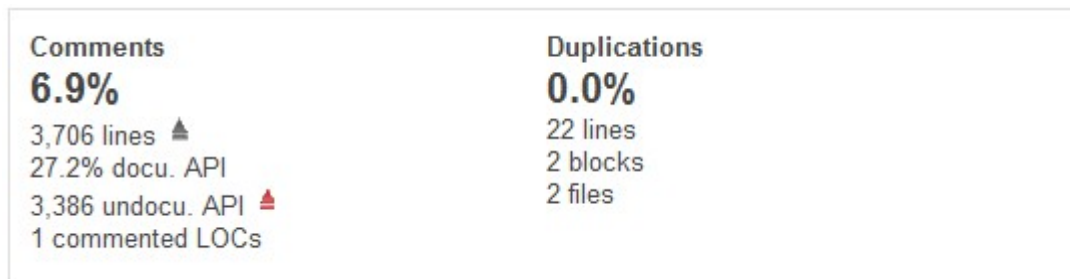
La page d'accueil d'un projet est son dashboard, qui se divise en plusieurs blocs d'informations, catégorisant ainsi les métriques (voir chapitre suivant traitant de ces métriques). Ces blocs sont appelés "*widgets*" et sont paramétrables par l'administrateur (possibilité d'ajouter, supprimer et déplacer ces widgets).

Sans être exhaustif, voici quelques-uns de ces blocs.

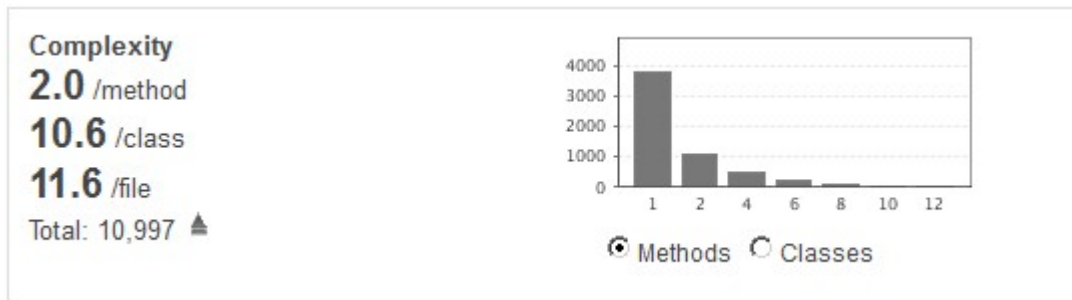
Le premier bloc montre les statistiques principales, comme le nombre de lignes de code, de packages, de classes Java, etc.



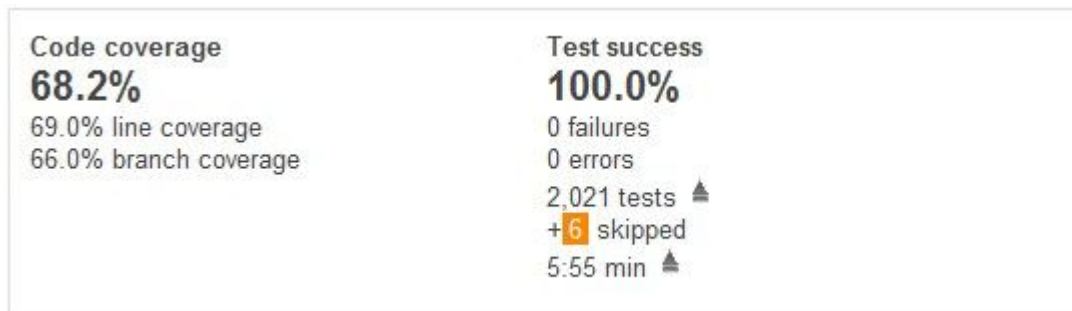
Le bloc suivant montre le pourcentage de commentaires présents dans le code, ainsi que la part de code dupliqué.



Le troisième bloc d'informations se focalise sur la répartition de la complexité cyclomatique par méthode ou par classe.



Les blocs suivants sont destinés à afficher les résultats de l'exécution des tests unitaires : couverture du code, pourcentage de réussite, etc.



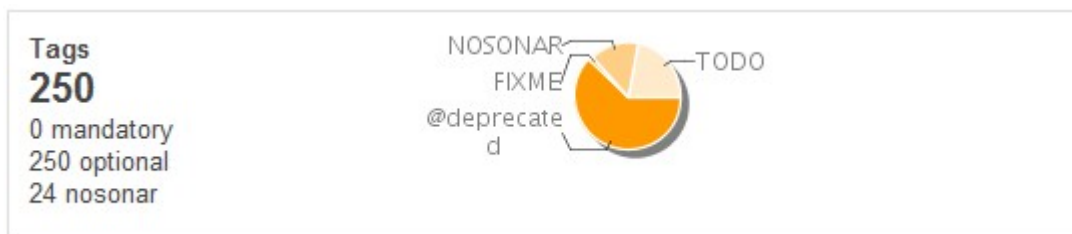
Un autre bloc nous montre le taux de respect des règles de codage, ainsi que la répartition, en fonction de leur sévérité, des violations de ces règles présentes dans le projet.

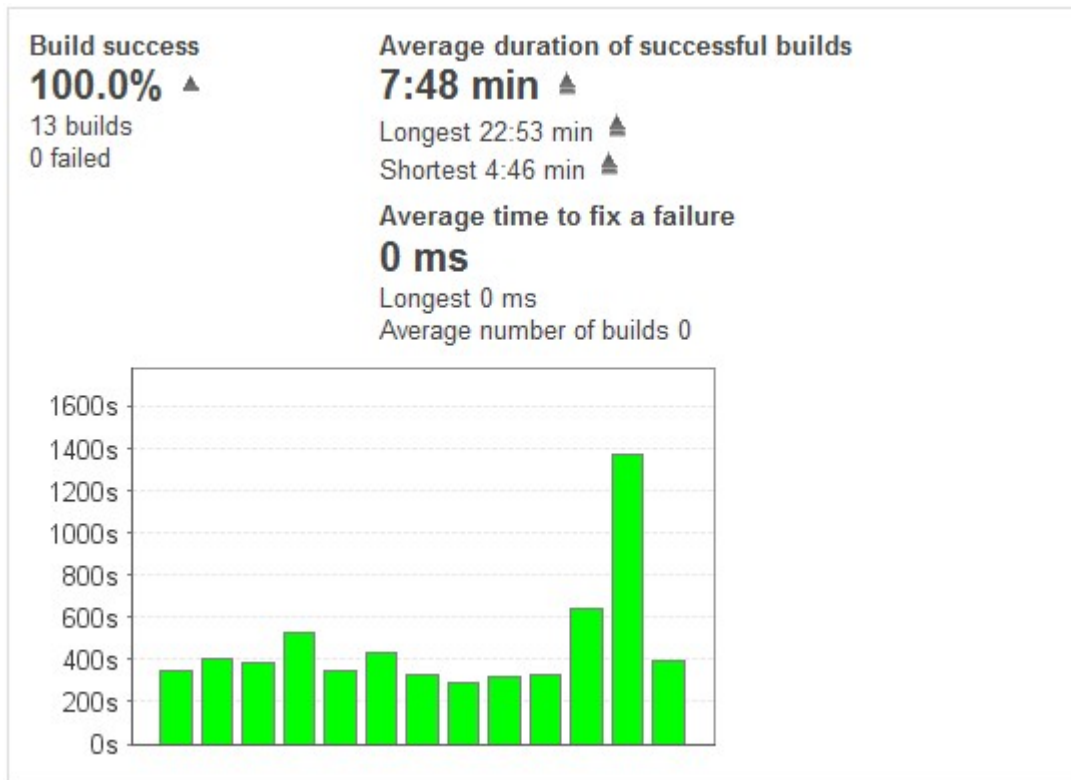


Un autre *widget* est dédié à l'affichage des éventuelles alertes, comme nous pouvons le voir ci-dessous :



Enfin, il est possible que certains plugins additionnels proposent leur propre *widget*, qui pourra alors afficher les informations relatives à leur fonction. On peut voir ainsi les *widgets* des plugins *Tags*, *Build Stability*, *JaCoCo* ou encore *Technical Debt*:





V-B-2 - Vue "Components"

La vue "Components" permet d'afficher la même vue que la page d'accueil de Sonar, mais où chaque élément affiché correspond à un module de l'application. Un module peut être soit un module au sens Maven (dans le cas de projet multimodules), soit la liste des packages Java. Cette vue permet ainsi d'obtenir une vision plus fine de la qualité.

V-B-3 - Vue "Violations drilldown"

La vue "Violations drilldown" montre l'ensemble des violations des règles de qualité présentes dans le projet.

Version 2.10-SNAPSHOT - Mon, 25 Jul 2011 18:32:49 +0000 - Profile Sonar for Sonar - Time changes...

Severity

- Blocker 0
- Critical 9
- Major 597
- Minor 238
- Info 270

Rule

- Bad practice - Class defines compareTo(...) and uses Object.equals()
- Bad practice - equals() method does not check for null argument
- Empty While Stmt
- Bad practice - equals method fails for subtypes
- Bad practice - Equals method should not assume anything about the type of its argument
- Insufficient branch coverage by unit tests

Sonar :: Plugin API	422	org.sonar.wsclient.services	99	Snapshot
Sonar :: Server	135	org.sonar.api.database.model	84	Rule
Sonar :: Batch	113	org.sonar.api.rules	79	CoreMetrics
Sonar :: Web Service Client	108	org.sonar.api.resources	74	DefaultProjectFileSystem
Sonar :: Core	54	org.sonar.api.measures	60	Project
Sonar :: Plugins :: Core	44	org.sonar.batch	39	ActiveRule

Path: Any severity » Any rule »

Le premier bloc affiche le nombre total de violations sur le projet. Celles-ci sont ventilées soit selon leur sévérité (*bloquante, critique, majeure, mineure ou information*).

Severity		
	Blocker	0
	Critical	9
	Major	597 <div></div>
	Minor	238 <div></div>
	Info	270 <div></div>

Le deuxième bloc affiche l'ensemble des règles non respectées sur le projet, ainsi que l'occurrence de ces violations.

Rule		
	Bad practice - Class defines compareTo(...) and uses Object equals()	3
	Bad practice - equals() method does not check for null argument	2
	Empty While Stmt	2
	Bad practice - equals method fails for subtypes	1
	Bad practice - Equals method should not assume anything about the type of its argument	1
	Insufficient branch coverage by unit tests	198 <div></div>

Les autres blocs montrent, pour chaque module, package ou classe, le nombre total de violations.

	Sonar :: Plugin API	422		org.sonar.wsclient.services	99		Snapshot	60
	Sonar :: Server	135		org.sonar.api.database.model	84		Rule	23
	Sonar :: Batch	113		org.sonar.api.rules	79		CoreMetrics	21
	Sonar :: Web Service Client	108		org.sonar.api.resources	74		DefaultProjectFileSystem	19
	Sonar :: Core	54		org.sonar.api.measures	60		Project	18
	Sonar :: Plugins :: Core	44		org.sonar.batch	39		ActiveRule	17

En cliquant sur l'une de ces règles (affichée dans le deuxième bloc), on fait afficher, dans les blocs du bas, les modules, packages et classes Java violant cette règle-là. Par exemple, en cliquant sur la règle "*Bad practice - equals() method does not check for null argument*", on visualise où se situent ces violations :

Severity		
Blocker	0	
Critical	9	
Major	597	<div></div>
Minor	238	<div></div>
Info	270	<div></div>

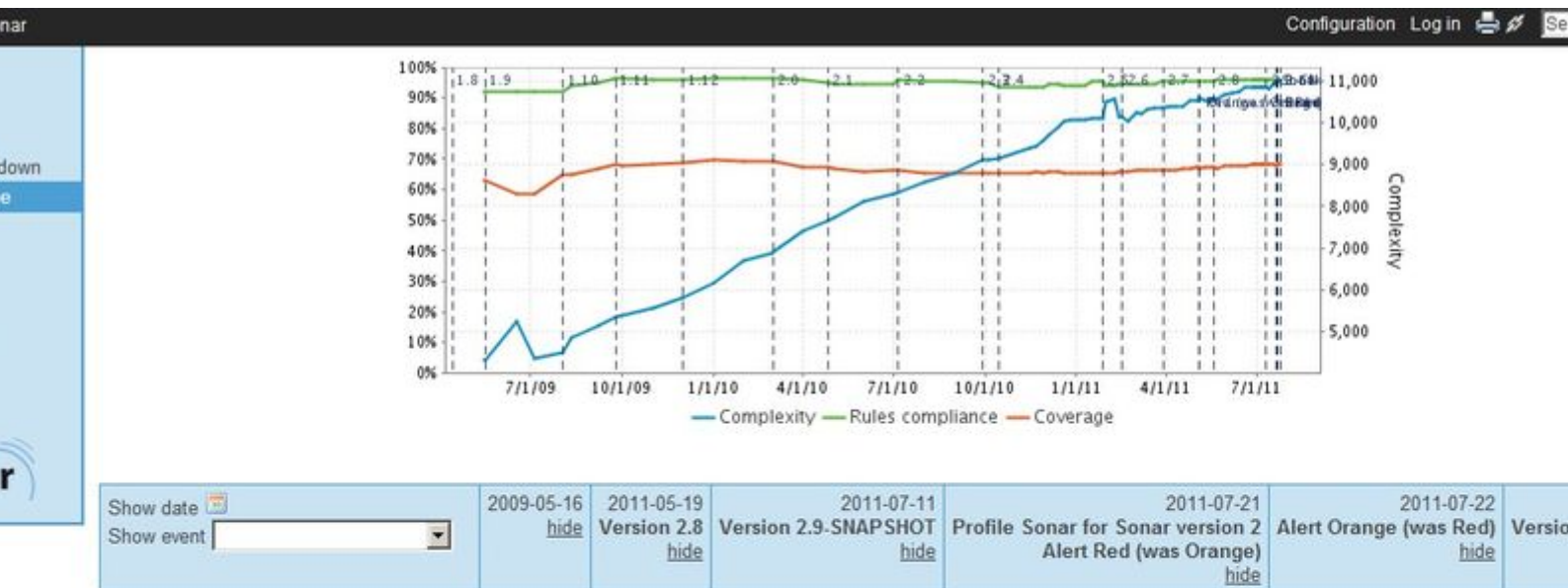
Rule		
	Bad practice - equals() method does not check for null argument	2
	Empty While Stmt	2
	Bad practice - equals method fails for subtypes	1
	Bad practice - Equals method should not assume anything about the type of its argument	1
	Insufficient branch coverage by unit tests	198 <div></div>
	Malicious code vulnerability - May expose internal representation by returning reference to mutable object	94 <div></div>

	Sonar :: Plugin API	1		org.sonar.api.measures	1		RuleMeasure
	Sonar :: Duplications	1		org.sonar.duplications.cpd	1		Match

V-B-4 - Vue "Time machine"

V-B-4-a - Time machine

Depuis les toutes premières versions de Sonar, cet outil propose une fonctionnalité très intéressante, à savoir la "Time Machine". Elle permet de visualiser simplement l'évolution des différentes mesures au cours du temps sur un projet donné, et donc également de comparer l'état de l'application entre deux dates. L'accès à cette fonctionnalité se fait via le bouton "Time machine" présent dans le menu à gauche.



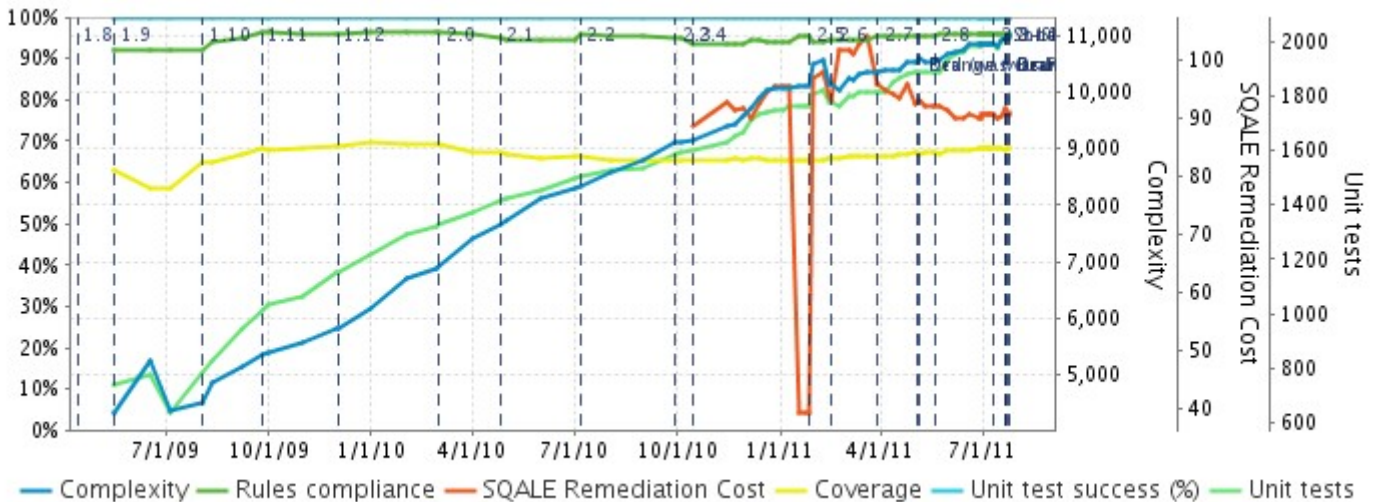
Complexity

<input checked="" type="checkbox"/> Complexity	4,351	10,551	10,863	10,954	10,980
<input type="checkbox"/> Complexity /class	11.8	10.6	10.7	10.6	10.6
<input type="checkbox"/> Complexity /file		11.6	11.7	11.5	11.6
<input type="checkbox"/> Complexity /method	2.0	2.0	1.9	1.9	2.0

Continuous integration

<input type="checkbox"/> Average Duration	39:42 min	39:42 min	39:42 min
<input type="checkbox"/> Average number of builds between fixes	0	0	0
<input type="checkbox"/> Average time to fix a failure	0 ms	0 ms	0 ms
<input type="checkbox"/> Builds	1	1	1
<input type="checkbox"/> Failed Builds	0	0	0
<input type="checkbox"/> Longest duration	39:42 min	39:42 min	39:42 min
<input type="checkbox"/> Longest time to fix a failure	0 ms	0 ms	0 ms

Au-dessous du graphique, de nombreux indicateurs sont présents. Il suffit de sélectionner ceux pour lesquels nous souhaitons voir l'évolution, puis de cliquer sur le bouton "Compare on chart". Le graphique se met alors à jour pour afficher les nouvelles métriques.



On voit que Sonar intègre sur le graphique certaines dates "importantes" (marquées par une ligne verticale) :

- le changement de version (dans le *pom.xml*) ;
- un seuil d'alerte a été franchi (appelé "Alert Red (was Orange)" par exemple) ;
- une alerte a été stoppée (appelée "Green (was Orange)" par exemple).

Chacun de ces événements affichés résulte en une colonne dans le tableau présenté sous le graphique. Grâce au bouton "Show date" situé dans l'entête de tableau, on peut ajouter une nouvelle date pour une comparaison affinée. De même, chaque colonne dispose d'un bouton "hide" pour la faire disparaître du tableau.

V-B-4-b - Vue différentielle

Sur les versions récentes de Sonar a été intégrée la fonctionnalité de vue différentielle. Elle permet de connaître l'évolution des métriques sur un laps de temps donné. Par défaut, nous pouvons comparer l'état courant avec l'état :

- 5 jours auparavant ;
- 30 jours auparavant ;
- l'une des deux dates ou périodes fixées dans la configuration du projet (options *Period 4* et *Period 5*).

Ces dates sont paramétrables.



Nous pouvons ainsi demander à Sonar d'afficher dans la vue "Dashboard" l'évolution des métriques sur une période d'un mois :

Version 2.10-SNAPSHOT - Mon, 25 Jul 2011 18:32:49 +0000 - Δ over 30 days

Lines of code
49,974 (+633)
37,617 lines (+1,357)
19,141 statements (+249)
950 files (+21)

Classes
1,033 (+21)
128 packages (+8)
5,639 methods (+66)
1,100 accessors (+17)

Comments
5.9% (+0.0)
3,706 lines (+43)
27.2% docu. API (+0.0)
3,386 undocu. API (+44)
1 commented LOCs (+0)

Duplications
0.0% (+0.0)
22 lines (+0)
2 blocks (+0)
2 files (+0)

Code coverage
68.2% (+0.1)
69.0% line coverage (+0.2)
66.0% branch coverage (+0.0)

Test success
100.0% (+0.0)
0 failures (+0)
0 errors (+0)
2,021 tests (+36)
+6 skipped (-3)
5:55 min (+49.1 sec)

On new code:
69.8%
688 lines to cover
71.1% line coverage
66.4% branch coverage

Events All

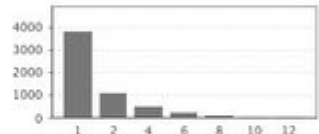
Violations
1,114 (+38)

Rules compliance
95.8% (-0.1)

Blocker 0 (+0)
Critical 9 (+1)
Major 597 (+13)
Minor 238 (+17)
Info 270 (+7)

Alerts : Critical violations > 0, Skipped unit tests > 0.

Complexity
2.0 /method (+0.1)
10.6 /class (-0.1)
11.6 /file (-0.1)
Total: 10,997 (+145)



Methods Classes

Package tangle index
6.2% (+0.1)
> 32 cycles (+1)

Dependencies to cut
17 between packages (+1)
25 between files (+1)

LCOM4
1.5 /class (+0.0)
24.5% files having LCOM4>1 (-0.2)

RFC
18 /class (0)

On peut noter que cette vue différentielle se retrouve à d'autres endroits, comme nous le voyons ci-dessous avec la vue d'une classe Java :

Version 2.10-SNAPSHOT - Mon, 25 Jul 2011 18:32:49 +0000 - Profile Sonar for Sonar - Added over 30 days

Severity	Rule	
Blocker +0	Empty While Stmt	+1
Critical +1	Insufficient branch coverage by unit tests	+8
Major +33	Unused Private Field	+7
Minor +26	Malicious code vulnerability - May expose internal representation by returning reference to mutable object	+4
Info +10	Malicious code vulnerability - May expose internal representation by incorporating reference to mutable object	+4
	Final Class	+1

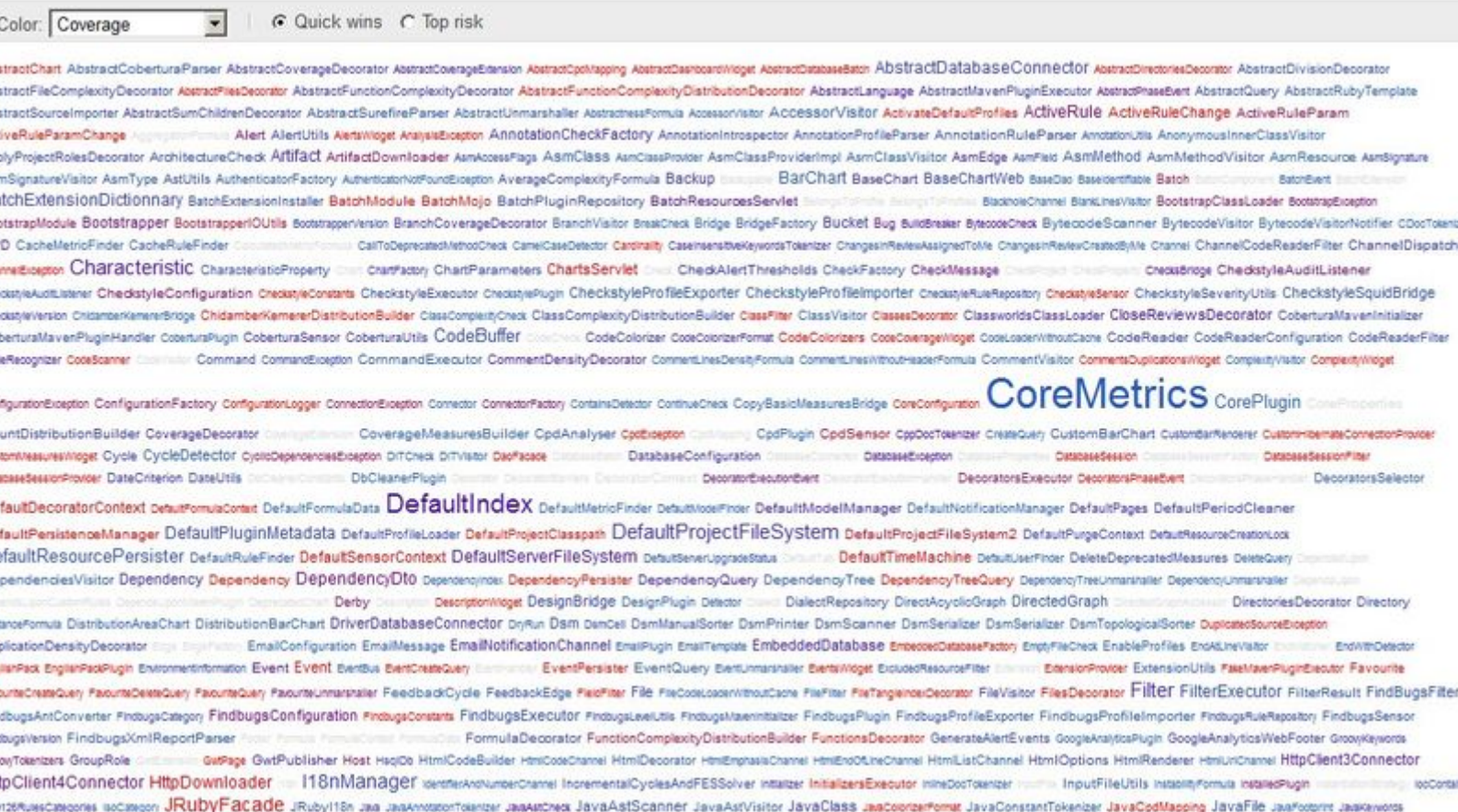
Sonar :: Core	+12	org.sonar.jp.a.entity	+12	Review	+7
Sonar :: Plugins :: Squid Java	+9	org.sonar.java.ast.visitor	+7	Match	+4
Sonar :: Plugins :: Core	+7	org.sonar.wsclient.services	+5	EmailNotificationChannel	+4
Sonar :: Plugin API	+6	org.sonar.plugins_email	+4	NotificationService	+4
Sonar :: Web Service Client	+6	org.sonar.duplications.cpd	+4	NotificationQueueElement	+3
Sonar :: Server	+5	org.sonar.server.notifications	+4	118nManager	+3

V-B-5 - Vue "Clouds"

La vue "Clouds", ou "Nuages" est une double page mettant en lumière les classes ayant un intérêt particulier. Dans ces vues, le nom des classes Java est affiché plus ou moins grand, en fonction de l'importance de la classe. Il y a deux catégories de nuages :

- les "Quick wins" ;
- les "Top risk".

La page "*Quick wins*" montre ainsi les classes qui, pour un effort relativement faible, augmenteront significativement la qualité générale de l'application. Plus un nom de classe est écrit grand, plus la classe disposera d'une complexité cyclomatique forte. Quant à la couleur, elle correspond à la couverture du code, ou au respect des règles de codage. Ainsi, dans la capture suivante, la classe "*CoreMetrics*" est écrite en grand (elle a une forte complexité) mais elle est en bleu, car sa couverture est bonne.

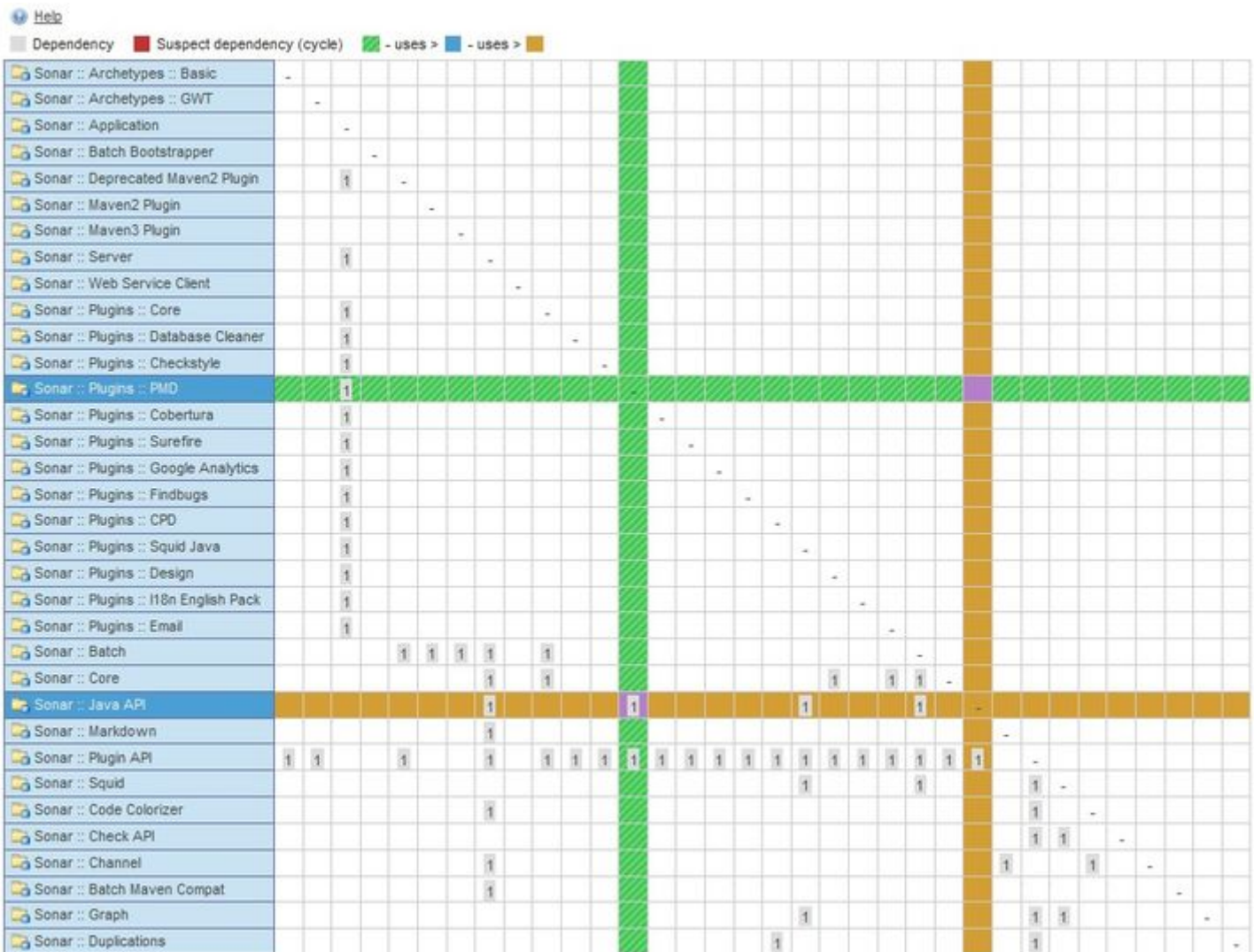


La page "*Top risk*" reprend le même principe, mais se base sur d'autres critères. Ici, la taille du texte indique la complexité par méthode (et non plus la complexité générale de la classe) et la met en parallèle avec le respect des règles de codage ou la couverture de code. Ainsi, une classe écrite en grand et en rouge montre une classe ayant un risque d'erreur important.

Lorsque l'on clique sur le nom d'une classe (quel que soit le type de nuage), on est automatiquement redirigé vers le code source de la classe.

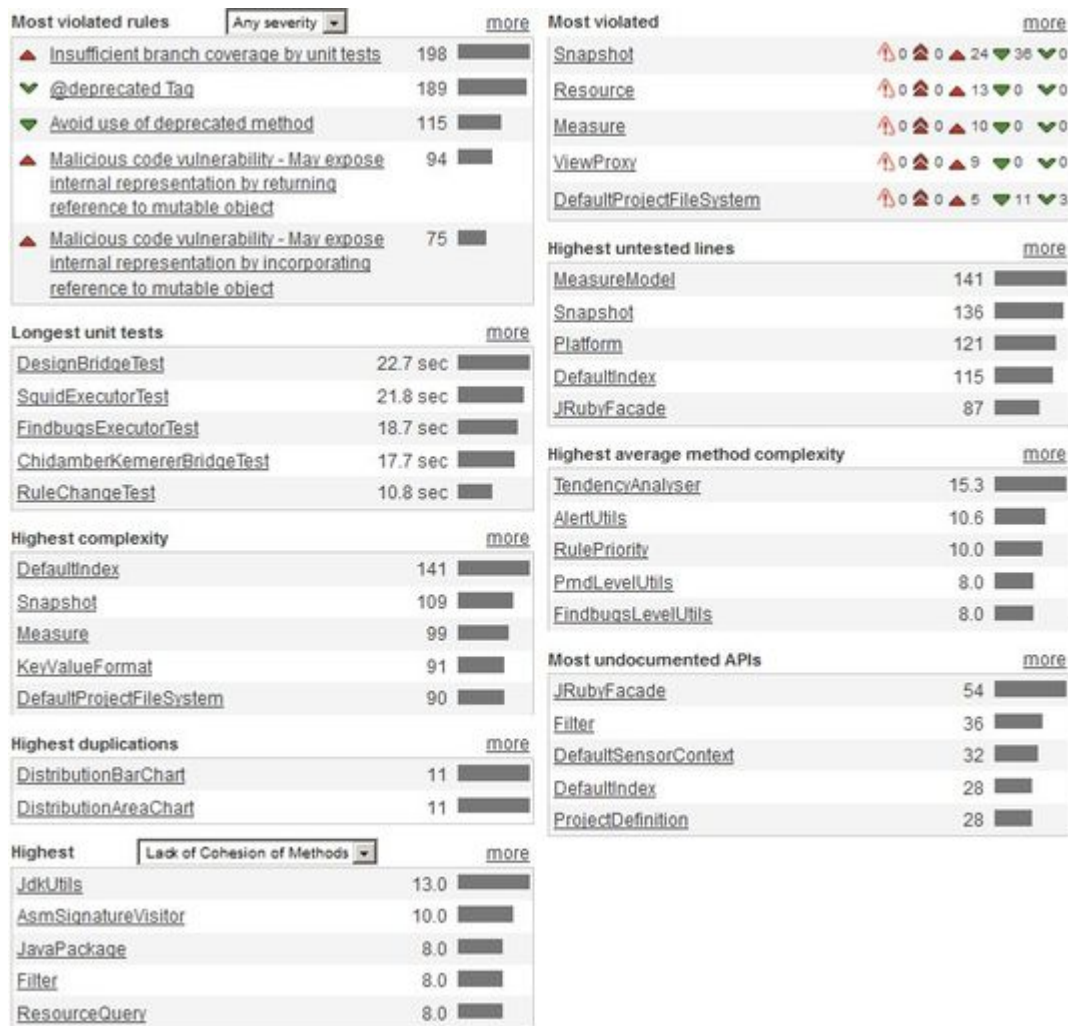
V-B-6 - Vue "Design"

La vue "*Design*" montre sous forme de tableau, les dépendances entre les modules (ou les packages ou les classes, selon le niveau où l'on se trouve).



V-B-7 - Vue "Hotspots"

La vue "*Hotspots*" montre, au sein d'une même page, les points d'intérêt sur le projet.

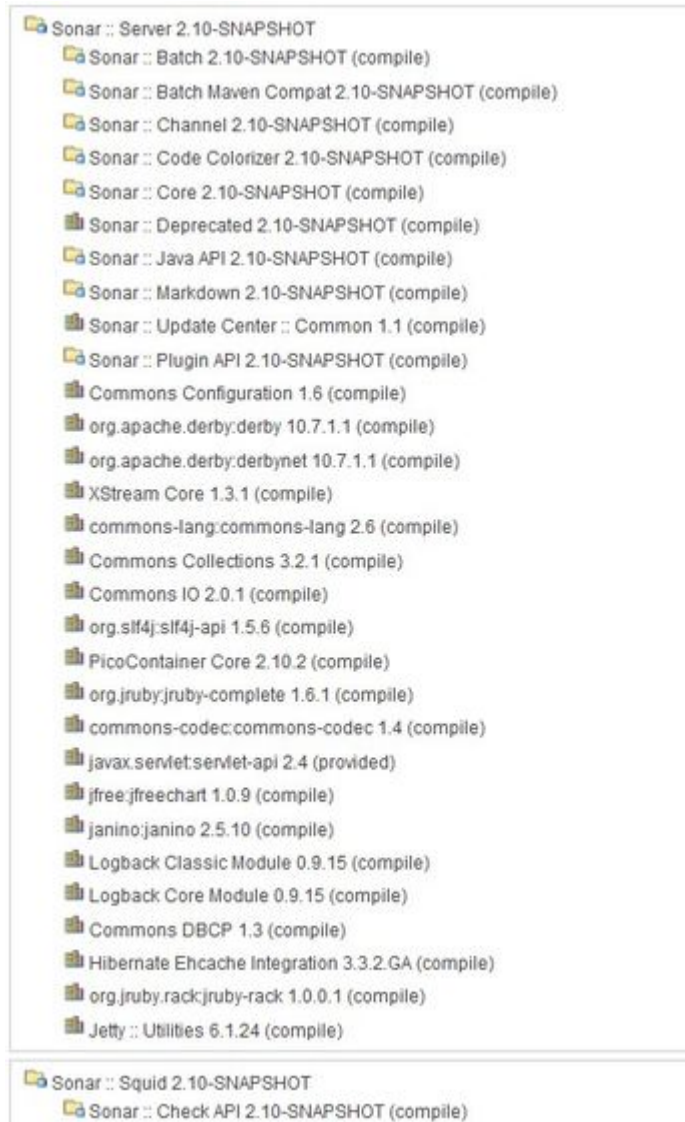


Ces informations sont séparées en blocs distincts :

- "Most violated rules" liste les règles ayant le plus de violations.
- "Most violated" montre les classes Java ayant le plus de violations de règles.
- "Longest unit tests" indique quelles classes prennent le plus de temps pour exécuter les tests unitaires.
- "Highest untested lines" indique les classes ayant le plus de lignes non couvertes par les tests unitaires.
- "Highest complexity" montre les classes ayant la plus forte complexité cyclomatique.
- "Highest average method complexity" liste les classes ayant les plus fortes complexités moyennes par méthode.
- "Highest duplications" montre les classes ayant le plus de code dupliqué.
- "Most undocumented file APIs" met en lumière les classes ayant peu de documentation.
- "Highest (Lack of cohesion of methods | Response for class)" montrent les classes ayant un fort **LCOM4** ou **RFC**.

V-B-8 - Vue "Libraries"

Cette vue affiche pour chaque composant les dépendances avec des librairies tierces.



V-B-9 - Vue "Settings"

Cette vue n'est accessible qu'aux administrateurs. Elle permet de paramétrer le projet (voir le sous-chapitre "Gestion des projets" du chapitre "Administration" pour plus de détails).

Dashboard

Components

Violations drilldown

Time machine

Clouds

Design


Hotspots

Libraries

SYSTEM

Settings

Project roles



Plugin settings

Select plugin

- [Artifact Size](#)
- [Cobertura](#)
- [Core](#)**
- [Database Cleaner](#)
- [Design](#)
- [Duplications](#)
- [Findbugs](#)
- [JaCoCo](#)
- [SCM Activity](#)
- [Sonar Build Stability Plugin](#)
- [Sonar SCM Commit Plugin](#)
- [Squid for Java](#)
- [Surefire](#)

[save parameters](#)

V-B-10 - Vue "Project Roles"

Cette vue permet de définir les utilisateurs ou groupes (voir chapitre sur l'administration de Sonar) pour chaque rôle, qui sont :

- **Administrateurs** : peuvent tout faire sur ce projet ;
- **Utilisateurs** : peuvent naviguer dans le projet, mais n'ont pas accès au code source ou à la configuration ;
- **Accès au code** : peuvent voir le code source du projet.

Dashboard

Components

Violations drilldown

Time machine

Clouds

Design


Hotspots

Libraries

SYSTEM

Settings

Project roles




Project roles

Role	Users	Groups
Administrators Ability to perform administration functions for a project by accessing its settings.	(select)	sonar-administrators (select)
Users Ability to navigate through every service of a project, except viewing source code and settings.	(select)	Anyone, sonar-users (select)
Code viewers Ability to view source code of a project.	(select)	Anyone, sonar-users (select)

V-C - Les métriques

V-C-1 - Les différentes métriques et leur mode de calcul

Nous n'allons pas ici parler de toutes les métriques, cela serait fastidieux et pas forcément utile. Comme nous l'avons déjà vu, Sonar remonte pour chaque projet un nombre assez important de métriques, ces métriques étant de plus disponibles sur plusieurs niveaux (projet, module, package, classe, etc.).

Il peut être intéressant de connaître la façon dont telle ou telle mesure a été réalisée sur une métrique donnée. C'est dans cette optique qu'une  [page du wiki de Sonar](#) a été écrite, regroupant les principales métriques, leur définition et leur méthode de calcul. On pourra ainsi découvrir la différence entre la couverture par ligne, par branche ou globale (voir chapitre suivant), ou encore la façon dont Sonar va calculer, pour un fichier donné, le nombre de lignes de code, d'instructions, de commentaires, etc.

V-C-2 - Interprétation des résultats

Obtenir des métriques est une chose, savoir les interpréter en est une autre, bien plus compliquée qu'il n'y paraît ! Prenons un exemple : la couverture de code. Il est évident que nous chercherons à obtenir le meilleur score sur cette métrique, mais signifie-t-elle pour autant que notre code est parfait ? Considérons le code Java suivant :

```
public String maMethode(boolean condition) {  
    String str = null;  
    if (condition) {  
        str = "ma chaine";  
    }  
    return str.trim();  
}
```

Nous pourrions écrire un test JUnit suivant pour tester notre méthode :

```
@Test  
public void testMaMethode() {  
    assertEquals("ma chaine", maClasse.maMethode(true));  
}
```

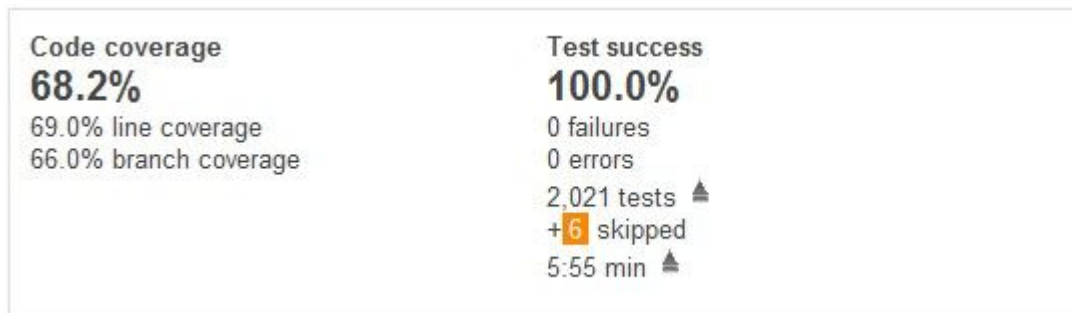
Ce test JUnit nous permet d'obtenir 100 % de couverture de code, mais pourtant, mon code n'est pas exempt de bogue, puisqu'il retournera un *NullPointerException* si je donne la valeur *false* au paramètre *condition*. En réalité, la couverture de code n'est pas ici de 100 %, c'est la couverture de ligne qui est à 100 %. Autrement dit, chaque ligne du code a été exécutée par le test JUnit. Ici, la couverture **par branche** n'est pas de 100 %, car la méthode peut en réalité être lue comme ceci :

```
public String maMethode(boolean condition) {  
    String str = null;  
    if (condition) {  
        str = "ma chaine";  
    } else {  
        // Ne rien faire.  
    }  
    return str.trim();  
}
```

Or le bloc *else* n'a pas été exécuté par notre test JUnit. C'est d'ailleurs pour ceci que Sonar propose trois valeurs de couverture de test :

- couverture de lignes : pourcentage de lignes de code exécutées par les tests unitaires ;

- couverture de branches : pourcentage de branches effectivement parcourues par les tests unitaires ;
- couverture globale : sorte de pondération entre les deux précédentes valeurs.




Et même si nous atteignons une couverture totale de 100 %, cela ne signifie absolument pas que le code soit exempt de bogue ! Toutefois, cela nous permet d'avoir une **plus grande confiance** dans le livrable final.

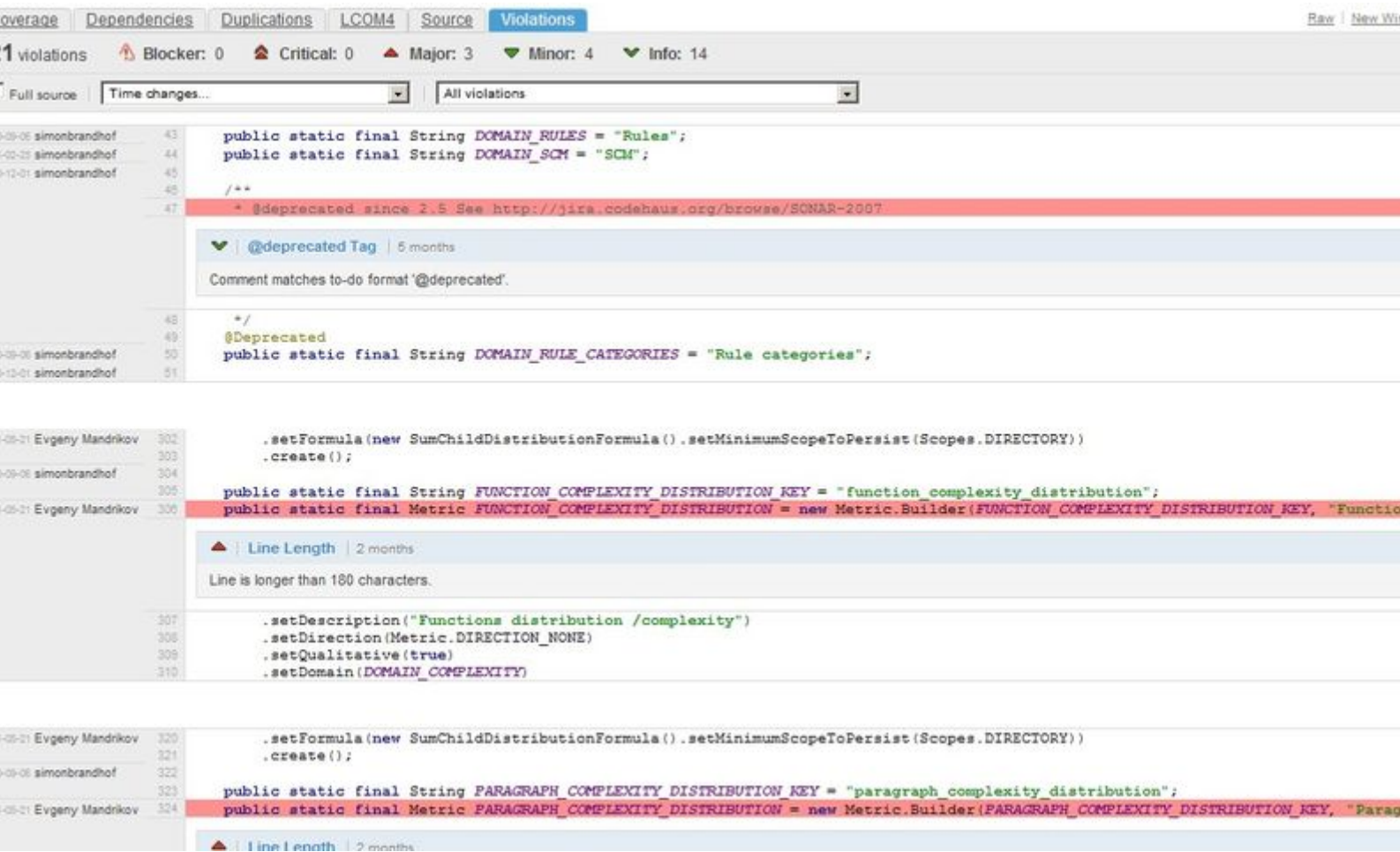
V-D - Visualisation du code source

V-D-1 - Vues

Il est possible d'arriver de différentes façons jusqu'au niveau d'une classe du projet. À partir de là, il est possible de voir le code source selon plusieurs critères :

- "Coverage" pour voir la couverture du code de cette classe par les tests unitaires ;
- "Dependencies" pour connaître les dépendances de cette classe ;
- "Duplications" qui affiche les lignes de code dont il a été trouvé des duplications ;
- "LCOM4" ("Lack of cohesion of methods") qui mesure le nombre de composants connectés à cette classe (voir  [ici](#) pour plus d'informations) ;
- "Source" qui montre le code source brut ;
- "Violations" qui montre le code source en y indiquant les violations rencontrées.

Ainsi, en visualisant le code source d'une classe avec les violations, nous voyons apparaître ce type d'écran :



Overview Dependencies Duplications LCOM4 Source **Violations**

1 violations Blocker: 0 Critical: 0 Major: 3 Minor: 4 Info: 14

Full source: Time changes... All violations

43 public static final String DOMAIN_RULES = "Rules";
 44 public static final String DOMAIN_SCM = "SCM";
 45 /**
 46 * @deprecated since 2.5 See <http://jira.codehaus.org/browse/SONAR-2007>.
 47 *
 48 *
 49 *
 50 public static final String DOMAIN_RULE_CATEGORIES = "Rule categories";
 51

@Deprecated Tag 5 months
 Comment matches to-do format '@deprecated'.

302 .setFormula(new SumChildDistributionFormula().setMinimumScopeToPersist(Scopes.DIRECTORY))
 303 .create();
 304
 305 public static final String FUNCTION_COMPLEXITY_DISTRIBUTION_KEY = "function_complexity_distribution";
 306 public static final Metric FUNCTION_COMPLEXITY_DISTRIBUTION = new Metric.Builder(FUNCTION_COMPLEXITY_DISTRIBUTION_KEY, "Function
 307
 308
 309
 310

Line Length 2 months
 Line is longer than 180 characters.

320 .setFormula(new SumChildDistributionFormula().setMinimumScopeToPersist(Scopes.DIRECTORY))
 321 .create();
 322
 323 public static final String PARAGRAPH_COMPLEXITY_DISTRIBUTION_KEY = "paragraph_complexity_distribution";
 324 public static final Metric PARAGRAPH_COMPLEXITY_DISTRIBUTION = new Metric.Builder(PARAGRAPH_COMPLEXITY_DISTRIBUTION_KEY, "Parag

Line Length 2 months

Pour plus de lisibilité, des options permettent de n'afficher que certaines violations (via le "filter") et n'affichent pas le code propre (on peut voir l'intégralité du code source en cliquant sur la case "Expand").

Pour des raisons de sécurité, il est possible de demander à Sonar de ne pas importer le code source dans Sonar. Ainsi, cette visualisation ne sera plus possible si cette option est désactivée (elle est active par défaut). Pour faire cela, il suffit de se rendre (en tant qu'administrateur) dans la configuration de Sonar, puis dans "General", et enfin de changer la valeur de l'option "Import sources". Il est également possible de le faire en ajoutant à sa commande Maven l'option "-Dsonar.importSources=false".

V-D-2 - Revue de code

Depuis la version 2.8 de Sonar, il est possible de réaliser des revues de code sur Sonar. Dans la vue "Violations", pour chaque violation, il est possible de réaliser les actions suivantes :

- marquer comme faux positif : cette violation n'est pas considérée comme une erreur dans ce cas précis, et Sonar doit désormais l'ignorer ;
- supprimer un faux positif : dans le cas où une violation a été considérée comme un faux positif, alors qu'il s'agissait bien d'une erreur ;
- faire une revue : laisser un commentaire sur cette violation, ce qui permettra par exemple de créer une demande de correction par un développeur ;
- réassigner (dans le cas d'une revue existante) : permet de réassigner à une autre personne une revue existante.



Au sein de la page d'accueil, on peut lister l'ensemble des revues et les filtrer en fonction de leur sévérité (ou plus exactement de la sévérité de la violation qui y est liée), du statut ou encore du projet :

Reviews

Status	Severity	Project	Created by	Id	Assigned to	Without false positives	Search
Any	Any	Any					
Open	Blocker	AS3 Core Lib					
Reopened	Critical	ActiveMQ					
Resolved	Major	Adobe Flex PMD Ja...					
Closed	Minor	AlisLib applicatio...					
	Info	All Sonar plugins					

St.	Id	Se.	Title	Project	Assignee
44			switch without "default" clause. Evgeny Mandrikov : default/Value can be set to null in default clause	Sonar org.sonar.batch.components.PastSnapshotFinder	Evgeny Mandrikov
43			Unused import - org.sonar.api.resources.ProjectFileSystem. Evgeny Mandrikov : Indeed - import can be removed.	Sonar org.sonar.api.CoreProperties	Evgeny Mandrikov
41			Unused import - org.sonar.api.resources.Project. Evgeny Mandrikov : Indeed - import can be removed.	Sonar org.sonar.batch.index.EventPersister	Evgeny Mandrikov
21			switch without "default" clause. Evgeny Mandrikov : I'll take care of this.	Sonar org.sonar.squid.text.StringArrayReader	Evgeny Mandrikov
9			org.sonar.graph.FeedbackCycle defines compareTo(FeedbackCycle) and uses Object.equals() Freddy Mallet : The description of the rule is pretty clear : "This class defines a compareTo(...) method but inher ...	Sonar org.sonar.graph.FeedbackCycle	Simon Brandhof
11			org.sonar.plugins.pmd.xml.PmdRule defines compareTo(Object) and uses Object.equals() Evgeny Mandrikov : See comment from Freddy in http://nemo.sonarsource.org/reviews/view/9	Sonar org.sonar.plugins.pmd.xml.PmdRule	Fab
8			Avoid empty while statements Evgeny Mandrikov : Another test with several lines.	Sonar org.sonar.plugins.surefire.data.SurefireStaxHandler	-

7 results

VI - Fonctionnalités d'administration

Ce chapitre va s'intéresser aux possibilités offertes à l'administrateur de Sonar et / ou d'un projet en particulier. Ces options ne sont donc possibles qu'à partir du moment où l'on est connecté en tant qu'administrateur.

VI-A - Administration globale

Pour administrer Sonar, il faut cliquer sur le lien "Configuration" en haut de l'écran. L'écran qui s'affiche nous propose d'accéder aux menus suivants :

- *Quality profile* : gestion des profils qualité ;
- *Event categories* : gestion de la catégorie des événements pouvant avoir lieu sur un projet ;
- *Manual metrics* : gestion des métriques manuelles ;
- *Default filters* : filtres activés sur la page d'accueil ;

- *Default dashboards* : gestion des dashboards existants ;
- *My profile* : gestion du profil de l'administrateur ;
- *Users* : liste des utilisateurs ;
- *Groups* : liste des groupes d'utilisateurs ;
- *Global roles* : liste des rôles globaux ;
- *Project roles* : liste des rôles spécifiques aux projets ;
- *Settings* : configuration générale de Sonar et des plugins installés ;
- *Backup* : sauvegarde des paramètres de profils ;
- *System info* : informations sur le système ;
- *Update center* : centre de mises à jour des plugins.

VI-A-1 - Gestion des profils de qualité

Le rôle d'administrateur permet de gérer les différents profils de règles de codage. Chaque profil existant peut être choisi comme le profil par défaut, supprimé ou dupliqué. Il est également possible de créer un nouveau profil. Pour ce faire, il suffit d'en saisir le nom et de donner les fichiers XML de configuration de PMD, Checkstyle et FindBugs.

En cliquant sur un profil, on peut en voir les détails, disposés dans différents onglets :

- *Coding rules* : gestion détaillée des règles ;
- *Alerts* : définition des alertes liées à ce profil ;
- *Projects* : liste des projets qui seront analysés avec ce profil ;
- *Permalinks* : liens permanents vers les configurations Checkstyle, PMD et Findbugs ;
- *Profile inheritance* : permet d'indiquer le profil parent, et donc d'en hériter les propriétés.

Quality profiles / java / Sonar way with Findbugs

Coding rules Alerts Projects Permalinks Profile inheritance

This profile can not be updated but it can be used as a template for your own configuration. Just copy it from the profiles page.

Name/Key

Plugin
 Any
 Checkstyle
 Findbugs
 PMD
 Sonar

Severity
 Any
 Blocker
 Critical
 Major
 Minor
 Info

Status
 Any
 Active
 Inactive

Search

490 results

Download

Active/Severity	Name [expand/collapse]	Plugin
<input checked="" type="checkbox"/> Major	Anon Inner Length	Checkstyle
<input checked="" type="checkbox"/> Major	Avoid Array Loops	Pmd
<input checked="" type="checkbox"/> Major	Avoid Assert As Identifier	Pmd
<input checked="" type="checkbox"/> Major	Avoid Calling Finalize	Pmd
<input checked="" type="checkbox"/> Major	Avoid Catching NPE	Pmd

Dans le premier onglet, *Coding rules*, on peut modifier la configuration actuelle des règles, en choisissant d'activer ou de désactiver telle règle, ou en changeant sa sévérité.

Dans l'onglet *Alerts*, il est possible de définir des alertes pour ce profil. La définition d'une alerte passe par le choix d'une métrique, d'un seuil pour le niveau "alerte" et un autre seuil pour le niveau "erreur". On pourra par exemple définir les alertes suivantes :

- si la couverture de code est inférieure à 60 %, on est en erreur, si elle est inférieure à 80 %, alors on est en avertissement ;
- si l'on a plus de 100 violations bloquantes, on passe en avertissement, si le nombre est supérieur à 200, on passe en erreur ;
- etc.

Coding rules Alerts Projects Permalinks Profile inheritance

Create alert

Select a metric

Warning threshold Error threshold

Create

Blocker violations is greater than 100 200 Update Delete

Coverage is less than 80 % 60 % Update Delete

Notes

Only project measures are checked against thresholds. Modules, packages and classes are ignored.

Project health icons represent :

- ✓ at least one threshold is defined, no threshold is reached.
- ⚠ at least one warning threshold is reached, no error threshold is reached.
- ! at least one error threshold is reached.

Lorsqu'une alerte est active, cela va entraîner les conséquences suivantes :

- un affichage adéquat sera fait sur le dashboard du projet ;
- une icône sera affichée dans la liste des projets, dans la page d'accueil ;
- les flux RSS seront alors alimentés avec ces alertes (le lien vers ce flux est disponible sur la page d'accueil) ;
- un événement sera créé au niveau du projet, et sera donc sélectionnable dans la Time machine.

⚠ Alerts : Critical violations > 0, Skipped unit tests > 0.

Il est à noter que nous pouvons définir un lien d'héritage entre profils. Ainsi, on pourra dire que le profil X hérite du profil Y, ce qui permet de disposer de la configuration du profil Y, mais en ajoutant d'autres règles. Cela pourra être utile dans le cas où nous disposons d'un jeu de règles pour une entreprise, mais que pour certains projets (des projets *legacy* par exemple), nous souhaitons surcharger ces règles standard.

Quality profiles / java / DVP

Coding rules Alerts Projects Permalinks Profile inheritance

Sonar way with Findbugs (490 rules)

⬆

DVP (490 rules, incl. 115 overriding!)

Set parent:
Inherit rules configuration from the profile:
Sonar way with Findbugs Change

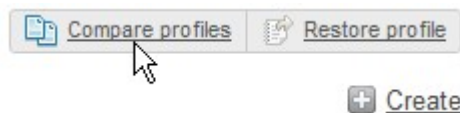
Le profil 'DVP' hérite du profil 'Sonar way with Findbugs'

À partir de la version 2.9 de Sonar, il est également possible de suivre les modifications apportées sur un profil, une sorte de gestionnaire de sources en version allégée :

Quality profiles / java / Legacy projects

Coding rules	Alerts	Projects	Permalinks	Profile inheritance	Changelog
Changelog from <input type="text" value="no version"/> to <input type="text" value="last version 2 (21 Jul 2011 10:47)"/> <input type="button" value="Load"/>					
Profile version	Date	User	Action	Rule	Parameters
1 -> 2	21 Jul 2011 10:47	Administrator	modified	Close Resource	Severity changed from MAJOR to BLOCKER
0 -> 1	21 Jul 2011 10:46	Administrator	off	Integer Instantiation	Severity was MAJOR
0 -> 1	21 Jul 2011 10:46	Administrator	off	If Stmts Must Use Braces	Severity was MAJOR
0 -> 1	21 Jul 2011 10:46	Administrator	off	If Else Stmts Must Use Braces	Severity was MAJOR

Une dernière fonctionnalité intéressante proposée par Sonar sur les profils est de pouvoir comparer deux profils. Il faut pour cela se rendre sur la page listant les profils, de cliquer sur le bouton "Compare profiles", et de choisir les deux profils à comparer.



Quality profiles / Compare

Une page s'affiche alors montrant les différences entre les deux profils :

- un récapitulatif de la comparaison des deux profils ;
- la liste des règles présentes seulement dans le premier profil ;
- la liste des règles présentes seulement dans le second profil ;
- les règles activées dans les deux profils, mais ayant des configurations différentes.

Only in Sonar way 91 rules	Only in Sun checks 35 rules	With different configuration 6 rules	With same configuration 18 rules
--------------------------------------	---------------------------------------	--	--

6 rules have a different configuration Sonar way

▲ **Hidden Field** checkstyle
tokens: VARIABLE_DEF
ignoreConstructorParameter: true
ignoreSetter: true
ignoreAbstractMethods: true

▲ **Inner Assignment** checkstyle

▼ **Magic Number** checkstyle

▼ **Redundant Modifier** checkstyle

▼ **Redundant Throws** checkstyle

▲ **Visibility Modifier** checkstyle

Sun checks

▲ **Hidden Field** checkstyle
tokens: PARAMETER_DEF, VARIABLE_DEF
ignoreConstructorParameter: false
ignoreSetter: false
ignoreAbstractMethods: false

▲ **Inner Assignment** checkstyle
tokens: ASSIGN, BAND_ASSIGN, BOR_ASSIGN, BSR_ASSIGN, BXOR_ASSIGN, DIV_ASSIGN, MINUS_ASSIGN, MOD_ASSIGN, PLUS_ASSIGN, SL_ASSIGN, SR_ASSIGN, STAR_ASSIGN

▼ **Magic Number** checkstyle
tokens: NUM_DOUBLE, NUM_FLOAT, NUM_INT, NUM_LONG
ignoreNumbers: -1, 0, 1, 2

▼ **Redundant Modifier** checkstyle
tokens: METHOD_DEF, VARIABLE_DEF, ANNOTATION_FIELD_DEF

▼ **Redundant Throws** checkstyle
allowUnchecked: false
allowSubclasses: false

▲ **Visibility Modifier** checkstyle
packageAllowed: false
protectedAllowed: false
publicMemberPattern: ^serialVersionUID\$

VI-A-2 - Métriques et événements manuels

Certaines métriques ne peuvent pas être alimentées automatiquement, comme par exemple la taille de l'équipe de développement. Il est donc possible de définir de nouvelles métriques, dites manuelles, par le biais de l'écran "Manual metrics". Pour cela, on utilisera le petit formulaire à la droite de l'écran où l'on spécifiera le nom de la métrique, sa description, son domaine ou encore son type de valeur.

Quality profiles

Event categories

Manual metrics

Default filters

Default dashboards

My profile

SECURITY

Users

Groups

Global roles

Project roles

SYSTEM

Settings

Backup

System Info

Name	Description	Domain	Type	Operations
Burned budget	The budget already used in the project	Management	Float	
Team size	Size of the project team	Management	Integer	
Business value	An indication on the value of the project for the business	Management	Float	

Name:

Ex. : Configuration management

Description:

Ex. : Respect level of the configuration management procedure (branch, label, ...), from 0% (worst) to 100% (best).

Domain:

 or

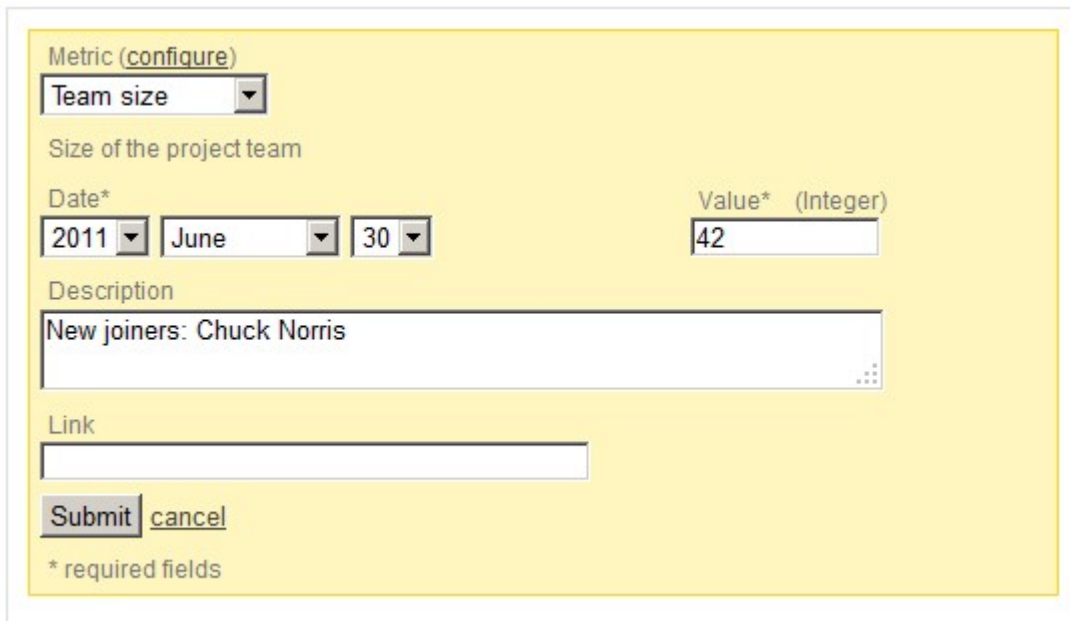
Type:

Create

Sur le dashboard d'un projet, il devient alors possible d'ajouter une nouvelle métrique, comme le montre la capture suivante :

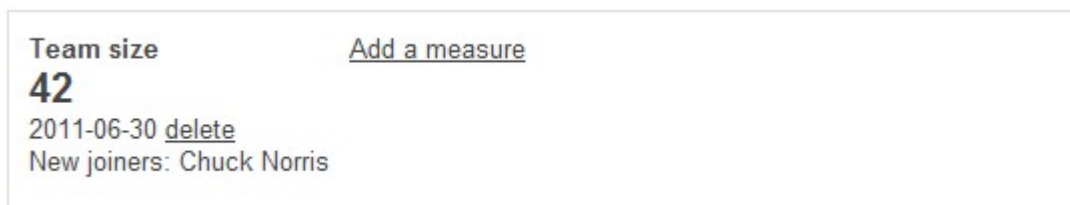
Add a measure

Un widget permet d'ajouter manuellement des métriques.



Metric (configure)
 Team size
 Size of the project team
 Date* 2011 June 30 Value* (Integer) 42
 Description
 New joiners: Chuck Norris
 Link
 Submit cancel
 * required fields

Le formulaire permet de saisir la nouvelle valeur pour cette métrique.



Team size [Add a measure](#)
42
 2011-06-30 [delete](#)
 New joiners: Chuck Norris

La mesure a été saisie et est désormais affichée.

Le principe est identique pour les événements. Ces derniers sont configurables par l'écran "Event categories". À noter que concernant les événements, Sonar ajoute automatiquement certains types d'événements :

- lorsque la version de l'application change ;
- lorsqu'une alerte est levée ;
- lorsque le niveau d'une alerte change (un avertissement devenant une erreur, ou inversement) ;
- lorsqu'une alerte se termine.

Events All		
2011-07-25	Version	2.10-SNAPSHOT
2011-07-22	Alert	Orange (was Red) i
2011-07-21	Profile	Sonar for Sonar version 2 i
2011-07-21	Alert	Red (was Orange) i
2011-07-11	Version	2.9-SNAPSHOT
2011-05-19	Version	2.8
2011-05-05	Alert	Orange (was Red) i
2011-05-04	Alert	Red (was Orange) i
2011-03-29	Version	2.7
2011-02-16	Version	2.6
Show more		

VI-A-3 - Gestion des droits et de la sécurité

Par défaut, Sonar est livré avec deux types d'utilisateurs : l'administrateur qui a tout pouvoir, et l'utilisateur, qui peut accéder à la visualisation des projets. Il est toutefois possible de gérer plus finement ces droits.

Tout d'abord, il faut créer des utilisateurs, via l'écran dédié.

- Quality profiles
- Event categories
- Manual metrics
- Default filters
- Default dashboards
- My profile
- SECURITY**
- Users**
- Groups
- Global roles
- Project roles

Users

Login	Name	Groups	Operations
admin	Administrator	sonar-administrators, sonar-users (select)	edit delete
romain	Romain	group_test, sonar-users (select)	edit delete

Add new user

Login:

Name:

Password:

Confirm password:

[Create](#) [cancel](#)

On pourra ensuite créer des groupes, de la même façon que pour les utilisateurs. Cet écran permet également d'affecter les utilisateurs aux différents groupes.

- Quality profiles
- Event categories
- Manual metrics
- Default filters
- Default dashboards
- My profile
- SECURITY**
- Users
- Groups**
- Global roles
- Project roles

Groups

Name	Description	Members	Operations
group_test		1 (select)	edit delete
sonar-administrators	System administrators	1 (select)	edit delete
sonar-users	Any new users created will automatically join this group	3 (select)	edit delete

Add new group

Name:

Ex: my-group

Description:

[Create](#) [cancel](#)

L'écran "Global roles" permet de configurer les rôles globaux à Sonar, en l'état le rôle d'administrateur. On peut ainsi affecter le rôle d'administrateur de Sonar à d'autres utilisateurs.

Ce même type de paramétrage est ensuite possible au niveau des projets, dans la page "*Project roles*". On pourra ainsi offrir les options d'administration **des projets** à certains utilisateurs, et on pourra également donner à chaque utilisateur un rôle spécifique pour chaque projet, parmi lesquels :

- **Administrateur** : permet l'administration de ce projet ;
- **Utilisateur** : permet l'accès à ce projet ;
- **Accès au code** : permet de visualiser le code source de ce projet.

Default roles for new projects

Role	Users	Groups
Administrators Ability to perform administration functions for a project by accessing its settings.	(select)	sonar-administrators (select)
Users Ability to navigate through every service of a project, except viewing source code and settings.	(select)	Anyone, sonar-users (select)
Code viewers Ability to view source code of a project.	(select)	Anyone, sonar-users (select)

Projects

Project	Role: Administrators	Role: Users	Role: Code viewers
My wonderful Project org.linsolas:save-the-world	(select users) sonar-administrators (select groups)	(select users) sonar-users, Anyone (select groups)	(select users) sonar-users, Anyone (select groups)

VI-A-4 - Configuration de Sonar

C'est ici qu'il faut se rendre pour configurer Sonar, ainsi que les plugins qui y sont installés. On retrouvera ainsi différentes options, propres au fonctionnement de Sonar. Ces options sont divisées ainsi :

- *Checkstyle* : pour la configuration de Checkstyle ;
- *Core* : pour les fonctionnalités du cœur de Sonar ;
- *Cobertura* : pour la configuration de Cobertura ;
- *Database Cleaner* : pour nettoyer les anciennes données de la base de données de Sonar ;
- *Design* : pour activer (ou désactiver) l'analyse de design par Sonar ;
- *Duplications* : paramètres pour la détection de code dupliqué (faite par CPD) ;
- *Findbugs* : configuration de l'outil Findbugs ;
- *Google analytics* : si vous souhaitez brancher une analyse Google Analytics sur le site de Sonar ;
- *Squid for Java* : quelques paramètres pour l'analyse par le plugin Squid de Sonar ;
- D'autres menus de configuration peuvent apparaître en fonction des plugins installés sur l'instance de Sonar.

Quality profiles
Event categories
Manual metrics
Default filters
Default dashboards
My profile

SECURITY
Users
Groups
Global roles
Project roles

SYSTEM
Settings
Backup
System Info
Update Center

sonar

Select plugin

[Checkstyle](#)
[Cobertura](#)
[Core](#)
[Database Cleaner](#)
[Design](#)
[Duplications](#)
[Findbugs](#)
[Google analytics](#)
[JaCoCo](#)
[Squid for Java](#)

save parameters

Provides all common components required to cover all languages.

Code coverage plugin [sonar.core.codeCoveragePlugin]
Key of the code coverage plugin to use.
 Default : cobertura

Import sources [sonar.importSources]
Set to false if sources should not be displayed, e.g. for security reasons.
 Default : true

Tendency period
Number of days the tendency should be calculated on.
 Default : 30

Skip tendencies [sonar.skipTendencies]
Skip calculation of measure tendencies
 Default : false

Rules weight

À noter que cette page est aussi présente (bien qu'un peu différente) au niveau des projets.

VI-A-5 - Centre de mise à jour

Cette page répertorie les plugins installés, ainsi que leur version, et va vérifier si une mise à jour de ceux-ci n'est pas présente. On peut également y trouver la liste des plugins disponibles au téléchargement et les installer.

Quality profiles
Event categories
Manual metrics
Default filters
Default dashboards
My profile

SECURITY
Users
Groups
Global roles
Project roles

SYSTEM
Settings
Backup
System Info
Update Center

sonar

Installed Plugins Available Plugins Plugin Updates System Updates

Plugins

Plugin	Version	Description
JaCoCo	0.6	JaCoCo is an alternative to Clover and Cobertura to measure coverage by unit tests.

System plugins

Checkstyle	Analyze Java code with Checkstyle 5.1 .
Cobertura	Get code coverage with Cobertura .
Core	Provides all common components required to cover all languages.
Database Cleaner	Optimizes database performances by removing old and useless data.
Design	Analyze Java bytecode to compute O.O. metrics and extract dependencies between resources.
Duplications	Find duplicated source code within project.
Findbugs	Analyze Java code with Findbugs 1.3.9 .
Google analytics	Add the Google Analytics tracking script to the Sonar web application
PMD	Analyze Java code with PMD 4.2.5 .
Squid for Java	Squid analyzer for Java.
Surefire	Get results of unit tests with Surefire .

VI-B - Administration d'un projet

Nous venons de voir l'administration globale du serveur Sonar. Au sein de chaque projet analysé par Sonar, il est possible de configurer certains paramètres. Pour ce faire, il faut se rendre sur la page d'un projet en étant connecté comme administrateur (ou avec le compte d'un utilisateur ayant les droits d'administration sur ce projet). Le menu à gauche proposera les options "Settings" et "Project roles".

L'option "*Project roles*" est identique à l'option de même nom présente dans l'administration globale, mais cette fois-ci limitée au seul projet courant. Elle permet de donner des droits particuliers à certaines personnes ou certains groupes d'utilisateurs sur ce projet.

Dans le menu "*Settings*", on retrouvera à peu près la même liste d'options que celles proposées par ce menu dans l'administration globale. Bien entendu, les paramètres définis pour un projet donné seront prioritaires sur les paramètres globaux, et ne seront valables que pour ce projet.

Lorsque l'on navigue sur ce menu "*Settings*", la partie basse de l'écran est dédiée aux informations générales sur ce projet, en particulier :

- la liste des répertoires ou fichiers à exclure de l'analyse Sonar (ce qui peut s'avérer pratique dans le cas où certains packages sont générés aléatoirement par exemple) ;
- le lien vers le site web du projet ;
- le lien vers le serveur d'intégration continue gérant ce projet ;
- le lien vers le gestionnaire de défauts ;
- le lien vers l'hébergeur des sources du projet ;
- d'autres liens possibles ;
- le bouton pour supprimer le projet de Sonar (**attention, il s'agit d'une action irréversible**).

Exclude sources from code analysis
Changes will be applied during next code analysis.

om/mycompany/"*.java
/*Dummy.java

Save filters Delete all filters

Project links

Title	URL
Home	<input type="text"/>
Continuous integration	<input type="text"/>
Issue tracker	<input type="text"/>
Sources	<input type="text"/>
Developer connection	<input type="text"/>
Save links	

Wildcards

*	Match zero or more characters
**	Match zero or more directories
?	Match a single character

Title	URL
 <input type="text"/>	<input type="text"/>
 <input type="text"/>	<input type="text"/>
 <input type="text"/>	<input type="text"/>
 <input type="text"/>	<input type="text"/>
 <input type="text"/>	<input type="text"/>

Delete project



This operation can not be undone.
Delete project

VI-C - Modification de la page d'accueil

Lorsque l'on se connecte sur la page d'accueil, celle-ci est divisée en plusieurs onglets, parmi lesquels :

- l'onglet "*Projects*" qui affiche les projets gérés par Sonar ;
- l'onglet "*Treemap*" qui affiche le "*tree map*" des projets ;
- l'onglet "*My favorites*" qui liste les projets favoris (à condition toutefois de s'être connecté).

L'administrateur de Sonar peut modifier cette page d'accueil soit en ajoutant / supprimant des onglets, soit en modifiant le contenu de chaque onglet. Ces opérations se réalisent de la même façon, en utilisant au choix l'option "*Add filter*" (pour l'ajout d'un nouvel onglet), "*Edit filter*" (pour modifier l'onglet courant) ou "*Manage filters*" (pour réorganiser les onglets, ou en supprimer un).

Configuration  Administrator » Log out 

La page d'édition (ou de création) se divise en deux ou trois blocs.

Le premier bloc va permettre de nommer l'onglet et de sélectionner les projets qui seront affichés dedans. Cette sélection peut se faire selon différents critères. Tout d'abord, on choisit si les critères concernent les projets, les modules, les répertoires (ou packages), les fichiers (ou classes) ou les tests unitaires. Ensuite, on définit la liste des critères que doivent respecter les projets pour être inclus dans cette vue. On pourra ainsi choisir de n'afficher que les projets dont la couverture de code par les tests est inférieure à 60 %, les projets dont l'analyse date de moins de 7 jours, etc.

Name: Shared: ☐

Path:

Search for: ☒ Projects ☐ Sub-projects ☐ Directories/Packages ☐ Files/Classes ☐ Unit tests

Criteria:

and:

and:

Default period:

Language: ☐ Java When no language is selected, no filter will apply

★ Favourites only: ☐

Resource key like: Use the character * to match zero or more characters.

Resource name like: Use the character * to match zero or more characters.

Build date: days

Le deuxième bloc, appelé "Display" va permettre de choisir les options d'affichage. Tout d'abord, on choisira d'afficher soit un tableau, soit une "treemap".

Dans le cas d'un tableau, on pourra choisir d'ajouter une nouvelle colonne parmi la liste de métriques proposées, le nombre maximum de projets affichés par page, etc.

Display

Display as: ☒ Table ☐ Treemap

Add column:

Default sorted column:


Page size: Min 20, max 200

Dans le cas d'une "treemap", on choisira la métrique définissant la taille de chaque carré, ainsi que la métrique définissant la couleur de ces mêmes carrés.

Display

Display as: ☐ Table ☒ Treemap

Size:

Color: 0.0%  100.0%

Enfin, le dernier bloc est le tableau lui-même (seulement si l'affichage se fait sous forme de tableau, et non de "treemap"). Chaque entête de colonne affiche des petites icônes permettant au choix de décaler la colonne à gauche ou à droite, de la supprimer ou encore de choisir si elle sera utilisée pour trier par défaut les éléments du tableau. Ainsi, il sera possible de trier alphabétiquement les projets, de les trier par date d'analyse, ou selon la valeur d'une métrique.

Name ^	Lines of code	Rules compliance	Blocker violations	Build date	Links
My wonderful Project	46,683 ▼	63.9%	0	2011-07-25	

Il est finalement possible de créer autant d'onglets que l'on souhaite.

VII - Méthodologie d'activation des règles

Il est très fréquent que la qualité d'un projet ne soit pas surveillée. Or, lorsque l'on met en place un outil comme Sonar sur un tel projet, il est à peu près assuré que le nombre d'erreurs soit très important (plusieurs milliers d'erreurs). Afin de ne pas se décourager (ou pire, de décourager l'équipe de développement) il est préférable de suivre un processus que l'on définira ici.

VII-A - Étape 1 : définir les règles de codage

Comme nous l'avons vu, Checkstyle propose plus d'une centaine de règles, et il en va de même avec PMD ou Findbugs (sans compter les règles propres à Sonar). Il est donc souhaitable de n'activer que les règles importantes, et de désactiver les autres. Il est également important de paramétrer certaines règles. Par exemple, Checkstyle permet de vérifier que la longueur maximale d'une ligne n'est pas dépassée. Or, la longueur maximale d'une ligne est variable d'une équipe à une autre, d'un projet à un autre. 80, 120, 150 ? Cette valeur reste à définir au sein de l'équipe.

Il va donc falloir définir convenablement les fichiers de configuration de PMD, Checkstyle et Findbugs. Une bonne idée est de partir de la configuration de Sun ou Sonar par exemple, et de l'adapter à ses besoins. Une très bonne pratique consistera également à appliquer certaines de ces règles dans le formateur intégré à son IDE (Eclipse par exemple), et de partager ce fichier de configuration au sein de l'équipe. Certains IDE permettent également d'appliquer le formatage automatiquement lorsque le fichier est sauvegardé (option de base d'Eclipse 3.2 ou supérieur, sinon un plugin existe pour ça). Cette bonne pratique permet ainsi d'obtenir une plus grande rigueur sur le formatage, et donc le respect de certaines règles de codage.

VII-B - Étape 2 : définir les règles les plus graves

Pour chaque règle, on peut définir d'une part son activation ou non, et d'autre part son niveau de sévérité parmi cinq niveaux (*Bloquant*, *Critique*, *Majeur*, *Mineur*, *Information*). Concrètement, dans l'interface d'administration de Sonar, il est possible de définir ces informations pour chaque règle. Il est également possible de jouer directement avec les fichiers XML...

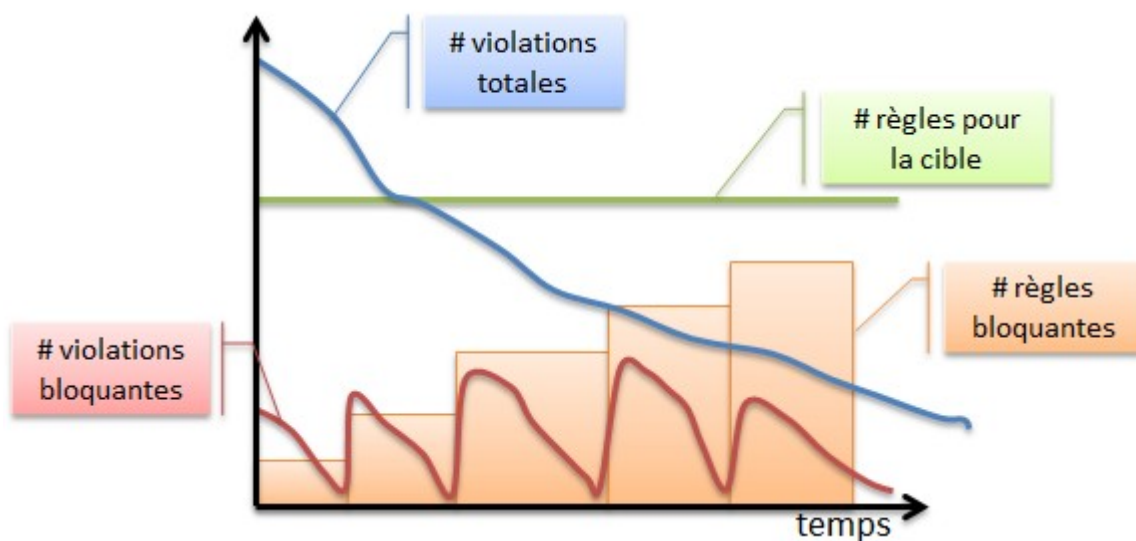
Lorsque l'on intègre Sonar (ou tout autre outil de qualité), le nombre d'erreurs est généralement très important. Après avoir défini les règles à appliquer, il est préférable de les mettre à des niveaux de sévérité assez faibles, puis d'en sélectionner un petit nombre (5 ou 10 par exemple) qui seront considérées comme obligatoires et donc avec un

niveau élevé (*Bloquant* ou *Critique*). L'équipe de développement va donc se charger de corriger les violations de ces dernières règles en priorité.

VII-C - Étape 3 : faire évoluer le niveau des règles

Lors de l'étape 2, un petit nombre de règles de codage ont été définies comme critiques. L'équipe de développement s'est donc attelée à corriger les violations de ces quelques règles. Une fois que ces règles obligatoires ne subissent plus ou presque plus de violations, l'étape suivante va consister à choisir un nouveau petit groupe de règles, pour en augmenter la sévérité. Ces règles seront donc le nouveau point d'intérêt des développeurs, en vue d'améliorer la qualité du code de l'application.

En procédant ainsi, et de façon incrémentale, le nombre de règles critiques va augmenter, et le nombre de violations totales des règles va peu à peu diminuer... En schématisant ce processus, cela nous donnerait quelque chose comme ceci :



Le nombre total de règles (ligne bleu clair) est défini dès le début et reste (à peu près) fixe au cours du temps. Le nombre total de violations (courbe orange) est très élevé au départ, mais diminue progressivement. Régulièrement, l'équipe de développement va sélectionner un petit groupe de règles comme étant prioritaires (courbe bleu foncé), et s'occupera alors de résoudre leurs violations. Ce qui explique les variations subies par la courbe rouge, dénombrant le nombre de violations de ces règles principales.

VII-D - Variante

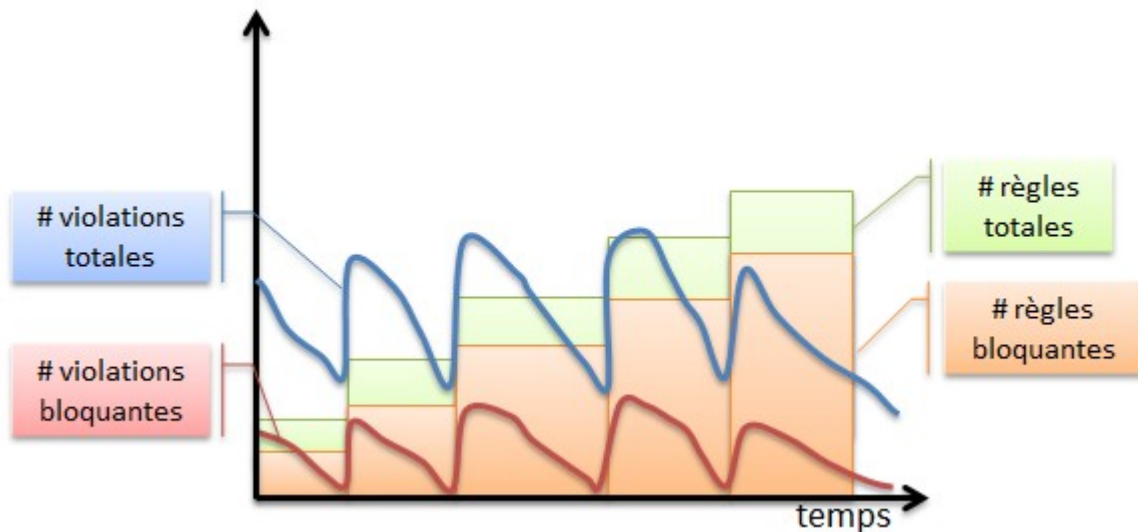
L'inconvénient de la méthode précédemment exposée est que l'instauration d'un contrôle de qualité sur un projet peut aboutir à un très grand nombre d'erreurs (ayant une sévérité faible). Il peut donc être intéressant d'utiliser une variante de cette solution. Cette variante fonctionne de la même façon, mais au lieu de n'augmenter que les règles critiques, elle va également faire varier le nombre total de règles.

Le principe est donc presque le même.

- On commence par choisir un petit nombre de règles critiques (une dizaine ou un peu plus, selon l'état initial du projet), ainsi qu'un autre petit nombre de règles moins importantes (avec une sévérité plus faible donc).
- L'équipe de développement va donner la priorité à la résolution des violations des règles de plus forte sévérité.
- Dès que le respect de ces règles s'approche des 100 %, on se charge de régler les règles moins importantes.
- Dès que ces règles de plus faible sévérité ont un taux de violation inférieur à 20 ou 30 %, on augmente la sévérité de ces règles.

- On définit désormais de nouvelles règles avec une sévérité faible.
- On réitère ainsi, jusqu'à ce que le standard soit atteint.

Cette variante serait plutôt schématisée de cette façon :



VIII - Plugins

VIII-A - Intérêt des plugins

Il existe aujourd'hui près d'une quarantaine de plugins pour Sonar. La liste de ces plugins est disponible [ici](#).

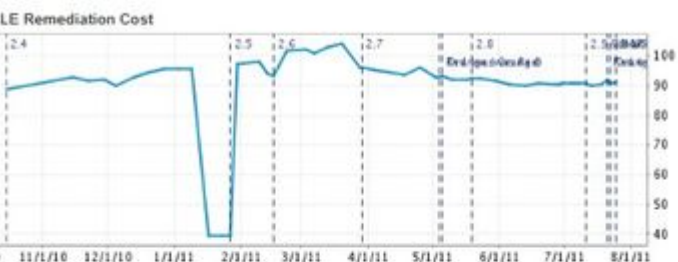
Actuellement, les plugins sont divisés en plusieurs catégories, selon leur utilité première.

- **Métriques additionnelles** : permet de calculer de nouvelles métriques pour Sonar. Il s'agit en général de permettre le support d'outils externes (JaCoCo, JIRA, JMeter, Mantis, SonarJ, etc.).
- **Nouveaux langages** : par défaut, Sonar ne gère que le langage Java, mais grâce à l'ajout de plugins, de nombreux autres langages peuvent être gérés (C, C#, Groovy, JavaScript, PHP, Visual Basic 6, etc.). Attention, certains d'entre eux sont des plugins commerciaux.
- **Visualisation** : plugins permettant un affichage différent des métriques calculées par Sonar.
- **Gouvernance** : offre le support de nouvelles méthodologies d'analyse, comme par exemple SQALE.
- **Intégration** : permet une meilleure intégration de l'outil Sonar dans l'usine logicielle, par exemple en offrant un support avec Hudson ou Bamboo (serveurs d'intégration continue). Certains de ces plugins sont destinés à être installés en dehors de Sonar.
- **IDE** : plugins destinés aux IDE (outil de développement comme Eclipse ou IntelliJ IDEA) offrant une intégration plus poussée avec Sonar.

Ci-dessous, quelques captures d'écran de l'utilisation de ces plugins :

2.10-SNAPSHOT - Mon, 25 Jul 2011 18:32:49 +0000 - Time changes...

LE Rating
Remediation Cost
91 days
Lines of Code
49K ▲



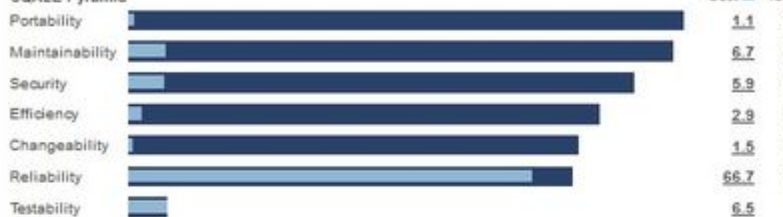
LE Kiviati



File Distribution by SQALE Rating



SQALE Pyramid



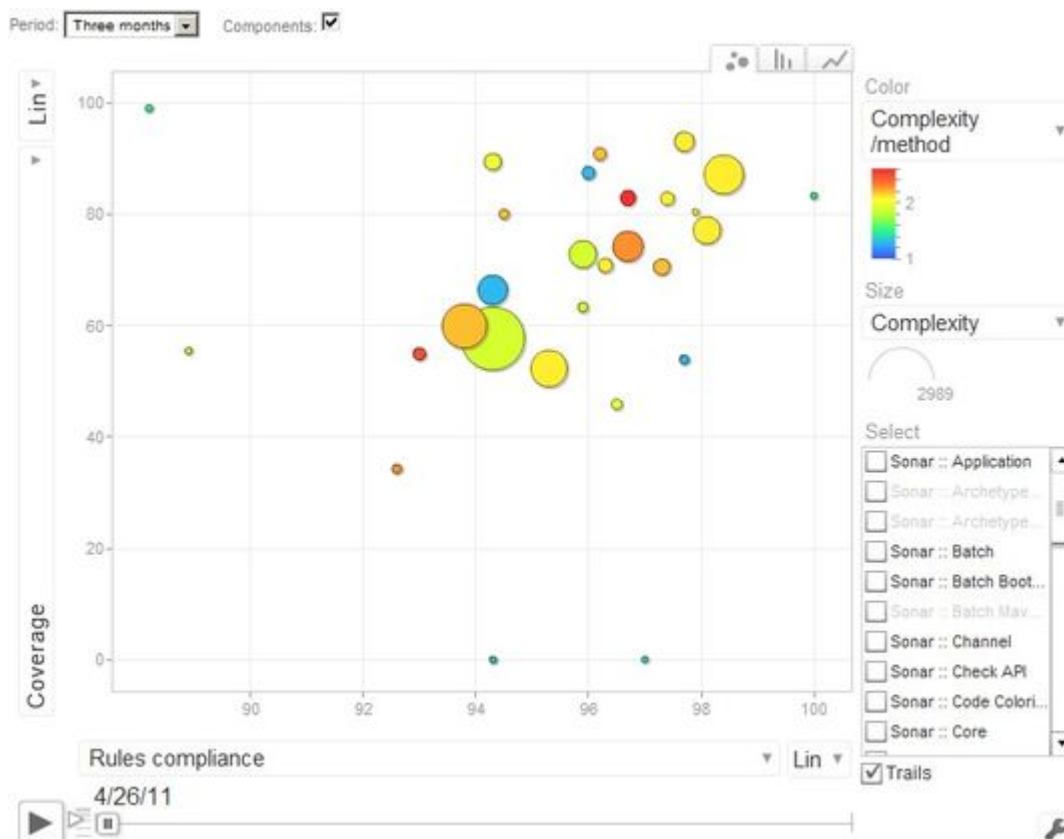
SQALE Sunburst



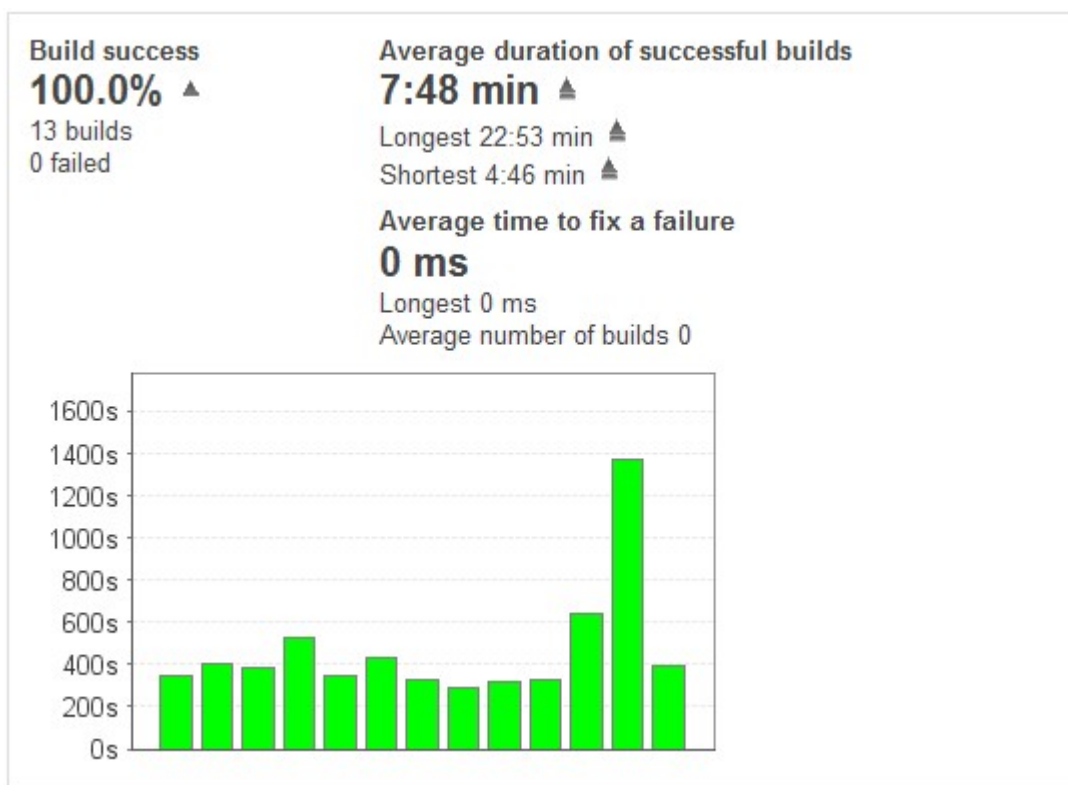
Le plugin 'SQALE' affiche le coût de remédiation de la dette technique.



Le plugin 'Timeline' affiche l'historique des métriques en relation avec les événements arrivés sur le projet, à la façon Google.



Le plugin 'Motion chart' affiche en animation l'évolution des métriques du projet au cours du temps.



Le plugin 'Build Stability' indique l'état de stabilité des builds de l'intégration continue.

On pourra également s'intéresser au plugin **SCM Activity**, qui va récupérer un certain nombre d'informations depuis le gestionnaire de sources (CVS, SVN par exemple) pour s'en servir dans Sonar. Il sera dès lors possible d'améliorer le niveau de qualité de l'inspection continue que l'on réalise sur ses projets, comme cela est expliqué [ici](#).

VIII-B - Installation

En règle générale, le plugin se présente sous la forme d'un fichier JAR qu'il faut télécharger [ici](#). Ensuite, il suffit de déposer ce fichier dans le répertoire `extensions/plugins/` du serveur Sonar (que l'on aura au préalable arrêté), puis de redémarrer le serveur. Toutefois, il se peut que cette règle diffère légèrement d'un plugin à un autre (en particulier les plugins d'intégration).

Avant l'installation d'un plugin, il est préférable de consulter sa compatibilité avec les versions de Sonar. Ceci peut se faire soit en consultant la page du plugin, soit sur [la page de matrice de compatibilité](#).

⚠ Attention, l'installation des plugins se faisant dans le répertoire `extensions/plugins/` du serveur Sonar, lorsque l'on met à jour Sonar, il est important de réaliser une sauvegarde de ce répertoire !

VIII-C - Développement de nouveaux plugins

Nous n'aborderons pas ici le développement d'un nouveau plugin pour Sonar, mais dans le cas où le sujet vous intéresse, vous pouvez vous rendre sur la [page wiki de Sonar](#) traitant de ce sujet.

VIII-D - API web services de Sonar

Sonar propose une série d'API qu'il est possible d'appeler afin de récupérer un certain nombre d'informations. On trouvera le détail de ces API sur [cette page](#).

Pour appeler l'un de ces services, il suffit d'appeler une URL spécifique, en renseignant le nom du service appelé, et des différents paramètres propres à ce service, de cette façon :

```
http://url-sonar/api/[nom-du-service]?param1=value1&param2=value2
```

En général, on pourra spécifier la ressource (qui peut être un projet, un module, un package ou même une classe) ou encore le format (selon le service appelé, on peut disposer des formats texte, CSV, XML, JSON, etc.)

Voici quelques exemples d'appels à ces web services :

URL	Définition
<code>http://url-sonar/api/resources</code>	Récupération de la liste des projets analysés par Sonar.
<code>http://url-sonar/api/resources?resource=foo:bar&metrics=blocker_violations,ncloc</code>	Récupération de la valeur des métriques <i>blocker_violations</i> (nombre de violations bloquantes) et <i>ncloc</i> (nombre de lignes de code) sur le projet <i>foo:bar</i> .
<code>http://url-sonar/api/reviews?resource=foo:bar</code>	Récupération de la liste des revues de code pour le projet <i>foo:bar</i> .
<code>http://url-sonar/api/reviews?resource=foo:bar&format=json</code>	Récupération de la liste des revues de code pour le projet <i>foo:bar</i> mais au format JSON.
<code>http://url-sonar/api/rules</code>	Liste des règles en vigueur.
<code>http://url-sonar/api/sources?resource=foo:bar:my.package.SomeClass&format=text</code>	Récupération (sous format texte) du code source (lequel-ci est accessible) de la classe <i>my.package.SomeClass</i> du projet <i>foo:bar</i> .

D'autres exemples sont disponibles sur [le wiki de Sonar](#).

Notons également que Sonar propose une petite application Java pour simplifier les appels vers ces web services. On trouvera davantage d'informations [ici](#).

IX - Conclusion

Sonar n'est pas le premier outil - et sans doute pas le dernier non plus - dont le but est d'offrir un suivi de la qualité d'un logiciel. Il s'agit toutefois là d'une solution facile à déployer, très agréable à utiliser, et vraiment complète. De plus, elle est aujourd'hui la seule solution (open source en tout cas) à proposer la visualisation et le regroupement de ces données pour un ensemble de projets, et non pour un seul projet.

L'évolution et l'activité de Sonar sont très encourageantes, et de nombreuses améliorations ont déjà vu le jour. La gestion des plugins est un aspect important pour le développement futur de Sonar, ainsi que pour favoriser son adoption par une communauté grandissante.

De même, le fait que Sonar ne se cantonne plus seulement à l'analyse de projets Java (et avec Maven) permet ainsi d'élargir encore davantage son champ d'action !

Enfin, vous pouvez suivre sur la page [Roadmap](#) de Sonar les évolutions futures de cet outil.

X - Références

- Page du projet Sonar : <http://www.sonarsource.org>.
- [Nemo, l'instance de démonstration de Sonar](#).
- [Quelques articles sur l'outil](#).
- Le [wiki de Sonar](#).

- La page des **plugins de Sonar**.
- **Suivez Sonar sur Twitter !**

Je souhaite remercier l'équipe de développement de Sonar, et tout particulièrement **Freddy Mallet** et **Simon Brandhof**, pour toute l'aide qu'ils ont pu m'apporter pour bien maîtriser cet outil. Je remercie également très chaleureusement **Claude Leloup** et **_Mac_** pour leurs relectures attentives !