

EPELFI

Marché de Maintenance applicative du système AMALFI

Résultat de demande de travaux ponctuels

Résumé

Type de document	TMA-TP1-RTP
Référence	TMA-TP1-RTP#TMAC-4048
Version	1.2
Date	31/03/2016
Nom de fichier	TMA-8.2.7.1 Résultats SonarQube.docx
Titre	Résultat de demande de travaux ponctuels
Sujet	Prestation TP1 : Résultat de demande de travaux ponctuels
Statut du document	Pour acceptation

Diffusion

Société	Nom	Fonction
EPELFI	Dominique Doyen	Responsable TMA
ASTEK	Ndongo Fall	Responsable TMA

Historique du document

HISTORIQUE DU DOCUMENT			
VERSION	DATE	AUTEUR	DESCRIPTION
1.0	28/01/2016	Damien WAHL Yves Grill	Version initiale
1.1	12/02/2016	EPELFI	Relecture
1.2	31/03/2016	Yves Grill Ndongo FALL	Version mise à jour en réunion du 31/03/16 de présentation du RTP

HISTORIQUE DES RELECTURES			
VERSION	DATE	AUTEUR	DESCRIPTION

TABLE DES MATIERES

1	Introduction.....	6
1.1	Objectif du document	6
1.2	Documents de référence	6
2	Présentation de la demande.....	7
2.1	Description de la demande	7
2.2	Actions à réaliser dans le cadre de la demande.....	7
2.3	Organisation du document.....	7
3	Actions réalisées dans le cadre de la demande.....	8
3.1	Installation du serveur SonarQube	8
3.2	Définition des profils de qualité	8
3.3	Mise en place du tableau de bord SonarQube.....	10
3.4	Modification de la construction des projets Amalfi	12
4	L'analyse quantitative : les métriques de code	14
4.1	Métriques de volumétrie	15
4.2	Documentation.....	15
4.3	Dette technique & défauts	15
4.4	Pyramide de dette technique	16
4.5	Design des fichiers	17
4.6	Duplications	18
4.7	Complexité	18
4.8	Rgèles les plus enfreintes.....	19
4.9	Toxicity.....	19
4.10	Détail des métriques par « projet »	21
4.10.1	Le projet AmalfiSc : le cœur d'Amalfi.....	22
4.10.2	Le projet AmalfiSsee : une extension d'AmalfiSc.....	25
4.10.3	Le projet AmalfiWar : le projet Web détenant la moins bonne note	28
4.10.4	Le projet AmalfiSseLra : le projet détenant la meilleure note	30
4.10.5	Les projets des Batchs.....	33
4.11	Parangonnage des valeurs obtenues	36
4.12	Analyse des défauts.....	39
5	Synthèse des résultats de l'audit	42
6	Préconisations et plan d'actions	43

7	Glossaire.....	44
8	Annexes.....	44
8.1	Classification des règles	44
8.2	Règles de codage Java.....	45
8.3	Règles de codage Javascript	52
8.4	Règles de codage Web	55
8.5	Règles de codage CSS.....	56
8.6	Règles de codage XML.....	58

TABLE DES ILLUSTRATIONS

Figure 1: Classification des règles de codage CSS	9
Figure 2: Classification des règles de codage Java	9
Figure 3: Classification des règles de codage Javascript	9
Figure 4: Classification des règles de codage Web	9
Figure 5: Classification des règles de codage XML	10
Figure 6: Tableau de bord de l'analyse SonarQube	10
Figure 7: Métriques de volumétrie globales	15
Figure 8: Métriques de documentation globale	15
Figure 9: Dette technique globale et défauts	15
Figure 10: Pyramide de la dette technique	17
Figure 11: Métriques de design globales	17
Figure 12: Métriques de duplication globales	18
Figure 13: Répartition par taille des projets AMALFI	21
Figure 14: Métriques de volumétrie des projets d'AmalfiSc	22
Figure 15: Métriques de documentation des projets d'AmalfiSc	23
Figure 16: Dette technique des projets AmalfiSc	23
Figure 17: Métriques des dépendances des projets d'AmalfiSc	23
Figure 18: Métriques de duplication des projets d'AmalfiSc	24
Figure 19: Métriques de complexité des projets d'AmalfiSc	24
Figure 20: Métriques de volumétrie des projets d'AmalfiSsee	25
Figure 21: Métriques de documentation des projets d'AmalfiSsee	25
Figure 22: Dette technique des projets d'AmalfiSsee	26
Figure 23: Métriques de design des projets d'AmalfiSsee	26
Figure 24: Métriques de duplication des projets d'AmalfiSsee	26
Figure 25: Métriques de complexité des projets d'AmalfiSsee	26
Figure 26: Métriques de volumétrie du projet AmalfiWar	28
Figure 27: Métriques de documentation des projets du projet AmalfiWar	28
Figure 28: Dette technique du projet AmalfiWar	29
Figure 29: Métriques de duplication du projet AmalfiWar	29
Figure 30: Métriques de duplication du projet AmalfiWar	29
Figure 31: Métriques de volumétrie des projets d'AmalfiSseLra	30
Figure 32: Métriques de documentation des projets d'AmalfiSseLra	31
Figure 33: Dette technique des projets d'AmalfiSseLra	31
Figure 34: Métriques de design des projets d'AmalfiSseLra	31
Figure 35: Métriques de duplication des projets d'AmalfiSseLra	31
Figure 36: Métriques de complexité des projets d'AmalfiSseLra	32
Figure 37: Métriques de volumétrie des Batches	33
Figure 38: Métriques de documentation des Batches	33
Figure 39: Dette technique des Batches	33
Figure 40: Métriques de duplication des Batches	34
Figure 41: Métriques de complexité des Batches	34

1 INTRODUCTION

1.1 Objectif du document

Ce document a pour but de répondre à la demande de travaux ponctuels TMAC-4048 (Analyse du code AMALFI avec SonarQube).

1.2 Documents de référence

Référence	Version	Nom du document
	Date	Titre
[TMA-TP1-PTP#TMAC-4048]	V1.0	TMA-TP1-PTP#TMAC-4048.docx
	13/11/2015	

2 PRESENTATION DE LA DEMANDE

Les informations en italique qui suivent, décrivant la demande à étudier, sont extraites de Jira.

2.1 Description de la demande

L'EPELFI veut avoir une idée de la qualité du logiciel AMALFI.

Une analyse de celle-ci doit être faite avec l'outil SonarQube sur l'ensemble des applications et du code d'AMALFI.

2.2 Actions à réaliser dans le cadre de la demande

Identifiant	Libellé
ACT-01	Installation du serveur et des modules SonarQube sur un poste développeur et définition du profil de qualité.
ACT-02	Mise en place du tableau de bord SonarQube et choix des modules de présentation des résultats.
ACT-03	Analyse des résultats

2.3 Organisation du document

Le document comprend les parties suivantes :

- actions réalisées dans le cadre de la demande : cette partie décrit les opérations réalisées pour pouvoir auditer le code
- analyse quantitative : cette partie décrit et commente les résultats de l'analyse fournis par SonarQube
- synthèse des résultats de l'audit de code : cette partie est une synthèse de la précédente
- préconisations et plan d'action : sur la base des résultats de l'audit, cette partie propose des actions d'amélioration de la qualité du code
- glossaire : cette partie explique certains acronymes et notions utilisés dans le document
- annexes : cette partie liste les règles utilisées par SonarQube lors de l'audit de code

3 ACTIONS REALISEES DANS LE CADRE DE LA DEMANDE

3.1 Installation du serveur SonarQube

Choix de la version du serveur

La version la plus récente de SonarQube est la version 5.2 qui date du 2 novembre 2015. Celle-ci n'a pas été retenue car comme stipulé dans les notes de parution, cette version ne propose plus la détection transversale (inter-projets) de la duplication. Donc, au vu du manque de maturité de cette version, la version 5.1.2 (l'avant dernière version qui date du 7 juillet 2015) a été installée sur un poste de développement.

Choix des modules/plugins

L'installation standard de la version 5.1.2, SonarQube intègre le module java 3.0. A celui-ci nous avons ajouté le module css 1.6, le module javascript 2.9, le module web 2.4 (analyse des pages web et des jsp) et le module xml 1.4. Ceci couvre l'ensemble des fichiers source d'AMALFI à analyser.

3.2 Définition des profils de qualité

Nous n'avons pas modifié les règles issues des modules : ni changement de sévérité, ni ajout de règles spécifiques, ni ajout d'étiquettes. L'analyse porte donc sur les conventions standards de programmation issues des modules installés.

Ci-dessous l'inventaire des profils Qualité SonarQube utilisés lors de l'analyse

Profils Qualité	
Profils CSS	
NOM	RÈGLES
Sonar way	46
Profils Java	
NOM	RÈGLES
Sonar way	192
Profils JavaScript	
NOM	RÈGLES
Sonar way	78
Profils Web	
NOM	RÈGLES
Sonar way	14
Profils XML	
NOM	RÈGLES
Sonar way	4

Les figures suivantes fournissent pour chaque langage la typologie des règles utilisées.

Sonar way

← Profils Qualité

Règles de codage Projets Liens permanents Héritage de profil Journal des modifications

Règles

46

! Bloquant	0
⬆ Critique	5
⬇ Majeur	31
✓ Mineur	7
⬆ Info	3

Figure 1: Classification des règles de codage CSS

Sonar way

← Profils Qualité

Règles de codage Projets Liens permanents Héritage de profil Journal des modifications

Règles

192

! Bloquant	15
⬆ Critique	59
⬇ Majeur	83
✓ Mineur	33
⬆ Info	2

Figure 2: Classification des règles de codage Java

Sonar way

← Profils Qualité

Règles de codage Projets Liens permanents Héritage de profil Journal des modifications

Règles

78

! Bloquant	5
⬆ Critique	22
⬇ Majeur	39
✓ Mineur	9
⬆ Info	3

Figure 3: Classification des règles de codage Javascript

Sonar way

← Profils Qualité

Règles de codage Projets Liens permanents Héritage de profil Journal des modifications

Règles

14

! Bloquant	0
⬆ Critique	1
⬇ Majeur	13
✓ Mineur	0
⬆ Info	0

Figure 4: Classification des règles de codage Web

Sonar way

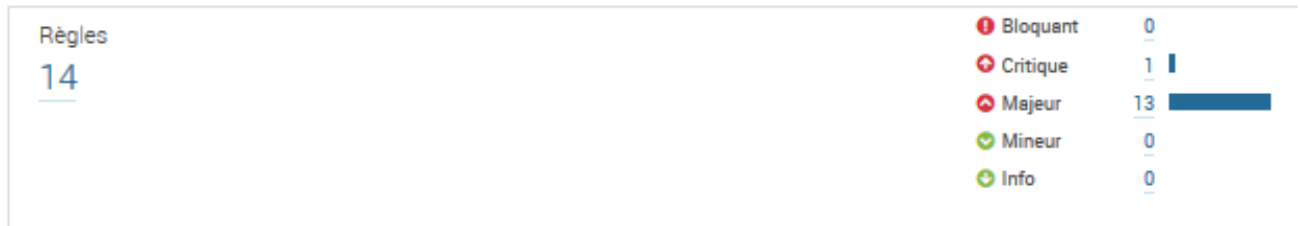
[Profilo Qualité](#)
[Règles de codage](#) [Projet](#) [Lien permanent](#) [Héritage de profil](#) [Journal des modifications](#)


Figure 5: Classification des règles de codage XML

On trouvera en [annexe](#) l'ensemble des règles pour chaque langage.

3.3 Mise en place du tableau de bord SonarQube

Voici le tableau de bord principal de l'analyse effectuée par SonarQube

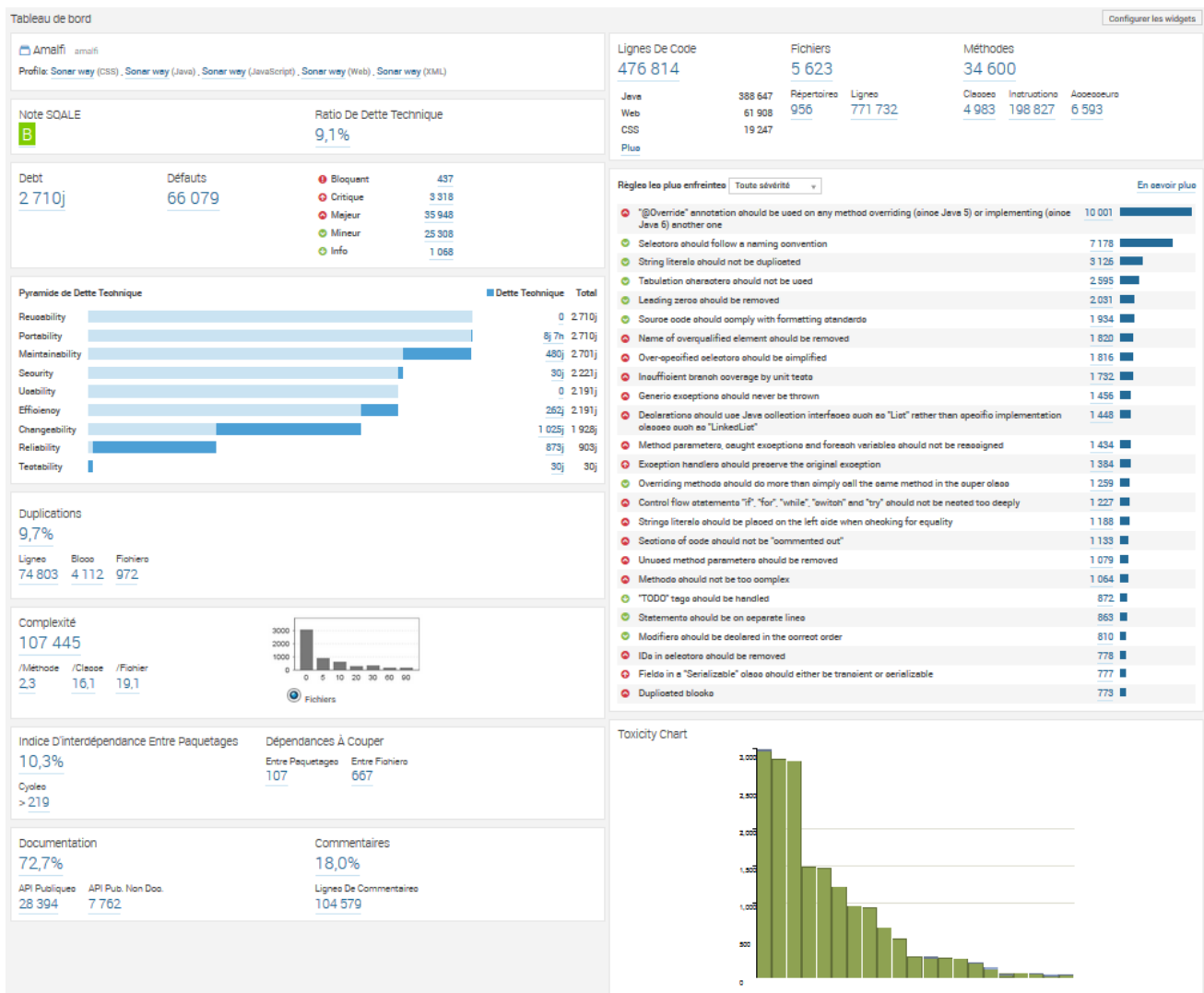


Figure 6: Tableau de bord de l'analyse SonarQube

Les composants visuels utilisés lors de l'analyse et présents sur le tableau de bord sont décrits ci-dessous :

- **Description :**
Affiche la liste des profils utilisés dans le cadre du projet.
- **Synopsis de la dette technique :**
Affiche la note SQUALE du projet et le ratio de la dette technique.
- **Dette technique et défauts :** Affiche un rapport sur les problèmes rencontrés et calcule la dette technique du projet. La dette technique se configure avec l'écran ci-dessous. Nous avons modifié la grille de notation afin de respecter celle utilisée par le site de parangonnage de SonarQube.

Dette Technique

Development cost Défaut: 30
Cost to develop one unit of code. If the unit is a line of code (LOC), and the cost to develop 1 LOC has been estimated at 30 minutes, then the value of this property would be 30.
Clé: sonar:teohnioalDebt.developmentCost

Language specific parameters

LANGUAGE KEY	DEVELOPMENT COST	SIZE METRIC
EX: JAVA, CS, CPP...	IF LEFT BLANK, THE GENERIC VALUE DEFINED IN THIS SECTION WILL BE USED.	IF LEFT BLANK, THE GENERIC VALUE DEFINED IN THIS SECTION WILL BE USED.

The parameters specified here for a given language will override the general parameters defined in this section.
Clé: languageSpecificParameters

Number of working hours in a day Défaut: 8
Clé: sonar:teohnioalDebt.hoursInDay

Rating grid Défaut: 0.1,0.2,0.5,1
SQUALE ratings range from A (very good) to E (very bad). The rating is determined by the value of the Technical Debt Ratio, which compares the technical debt on a project to the cost it would take to rewrite the code from scratch. The default values for A through D are 0.1,0.2,0.5,1. Anything over 1 is an E. Example: assuming the size metric is lines of code (LOC), and the work unit is 30 (minutes to produce 1 LOC), a project with a technical debt of 24,000 minutes for 2,500 LOC will have a technical debt ratio of 24000/(30 * 2,500) = 0.32. That yields a SQUALE rating of C.
Clé: sonar:teohnioalDebt.ratingGrid

Size metric Défaut: nolor
Metric used to estimate artifact's development cost.
Clé: sonar:teohnioalDebt.sizeMetric

- **Pyramide de la dette technique :**
Affiche la dette technique par caractéristique. Elle se lit en commençant par le bas.
- **Duplications :**
Affiche un rapport sur la duplication de code.
- **Complexité :**
Affiche un rapport sur la complexité, la complexité moyenne et la distribution de la complexité.
- **Design des fichiers :**
Affiche un rapport sur les cycles de dépendance et d'enchevêtrement des fichiers.
- **Règles les plus enfreintes :**
Affiche les règles les plus enfreintes.
- **Métriques de volumétrie :**
Affiche un rapport sur la taille du projet.

- **Documentation :**

Affiche les métriques de pourcentage de commentaires, d'API publiques (méthodes publiques) et d'API publiques commentées

- **Toxicité :**

Présente les classes qui accumulent le plus de problèmes selon les critères suivants :

Code Squid:	Nom	Description	Valeur par défaut
S00104	File Length	Nombre maximum de lignes d'un fichier.	1000
S138	Method Length	Nombre maximum de lignes d'une méthode.	100
MethodCyclomaticComplexity	Cyclomatic Complexity	Complexité cyclomatique maximale d'une méthode.	10
S00107	Parameter Number	Nombre maximum de paramètres d'une méthode.	7
S1067	Boolean Expression Complexity	Nombre d'opérateurs logiques imbriqués (&&, , &, and ^)	3
S134	Nested If Depth	Nombre d'if-else imbriqués	3
S1141	Nested Try Depth	Nombre de try imbriqués	N/A
SwitchLastCasesDefaultCheck	Missing Switch Default	Nombre d'instructions switch sans cas par défaut	N/A
S1200	Class Fan Out Complexity	Nombre d'autres classes dont dépend une classe donnée	20
S1188	Anon Inner Length	Longueur d'une classe interne	20

Le paramétrage de ce composant permet de définir le niveau minimum de toxicité affiché

Toxicity Chart

Toxicity Chart threshold

Défaut: 1

Toxicity Chart threshold.

Clé: threshold

La couverture des tests n'a pas pu être réalisée en raison de la nature des tests. En effet, il s'agit en grande partie de tests d'intégration et non de tests unitaires car le framework utilisé dans Amalfi se prête mal à la conception de tests unitaires.

3.4 Modification de la construction des projets Amalfi

Afin d'obtenir des métriques pertinentes pour les sous projets AmalfiSc, AmalfiSsee, AmalfiGP, AmalfiWar, les Batches, les outils et les tests nous avons dû modifier la hiérarchie des projets. La hiérarchie utilisée est la suivante :

Amalfi

- + AmalfiSc
 - + AmalfiScTransverse
 - + AmalfiScBack
 - + AmalfiScFront
 - + AmalfiScEjb
 - + AmalfiRedevanceEjb
 - + AmalfiRedevanceSSO
- + AmalfiSsee
 - + AmalfiSseeTransverse
 - + AmalfiSseeBack
 - + AmalfiSseeFront
 - + AmalfiSseeEjb
- + AmalfiGp
 - + AmalfiGpEjb
- + AmalfiWar : le projet Web regroupant tout le code des pages JSPs
- + AmalfiSseLra
 - + AmalfiSseLraTransverse
 - + AmalfiSseLraBack
 - + AmalfiSseLraFront
 - + AmalfiSseLraFrontWar
- + Batches
 - + Batch : la librairie cœur des batches
 - + BatchAmalfi : la librairie qui étend le projet précédent pour Amalfi
 - + BatchBaseReference
 - + BatchCalculOJetsAnormaux
 - + BatchCommunRedevance
 - + BatchCreationCommandesRequetes
 - + BatchCreationXMLCopie

- + BatchGenerationBI
- + BatchFusionCommunes
- + BatchDeresev
- + BatchDgi
- + BatchEnvoiCopieElectronique
- + BatchExportBI
- + BatchGenerationJournaux
- + BatchGestionEnum
- + Batchlah
- + BatchImportBI
- + BatchMajScan
- + BatchNettoyageSignatures
- + BatchPavePacs
- + BatchRedevanceCopies
- + BatchReparationPublication
- + BatchRescellementRequete
- + BatchStats
- + BatchStockageGed
- + BatchSupressionCopieAncienne
- + BatchSynchronisationRedevance
- + BatchTests
- + TraitementsBdd
- + Tools : le projet des outils Amalfi
 - + OutilGeneration
 - + OutilsDev
 - + Outils
- + Tests : le projet des tests unitaires et d'intégration

Ces modifications ont été faites sur le poste de développement ayant servi à l'analyse. Elles n'ont pas été publiées dans le référentiel de code source.

4 L'ANALYSE QUANTITATIVE : LES METRIQUES DE CODE

Les résultats ci-dessous correspondent à ceux observés sur la plateforme de développement.

4.1 Métriques de volumétrie

Lignes De Code	Fichiers	Méthodes
<u>476 814</u>	<u>5 623</u>	<u>34 600</u>
Java	388 647	Répertoires
Web	61 908	Lignes
CSS	19 247	<u>771 732</u>
XML	6 639	Classes
JavaScript	373	Instructions
		Accesseurs
		<u>4 983</u>
		<u>198 827</u>
		<u>6 593</u>

Figure 7: Métriques de volumétrie globales

Ce composant nous propose une vue d'ensemble du projet sous la forme d'un dénombrement des différents éléments composant l'application.

Analyse d'Astek : Avec au total près de 500k lignes de codes, AMALFI fait partie de la catégorie des « gros projets ». Cependant Amalfi se décompose en plusieurs sous projets. AmalfiSc avec près de 300k de lignes de code reste un projet de taille importante.

4.2 Documentation

Documentation	Commentaires
<u>72,7%</u>	<u>18,0%</u>
API Publiques	Lignes De Commentaires
28 394	<u>104 579</u>
API Pub. Non Doc.	
7 762	

Figure 8: Métriques de documentation globale

Le code Amalfi est bien documenté avec un taux de 72,7%. En revanche, avec une valeur de 18%, le pourcentage de commentaires est légèrement en deçà du minimum préconisé (la densité de commentaire devrait se situer entre 20% et 40%). Dans le cas des commentaires, il serait plus intéressant d'avoir une donnée qualitative car le volume de commentaires ne préjuge pas de leur qualité.

4.3 Dette technique & défauts

Debt	Défauts	
<u>2 710j</u>	<u>66 079</u>	
		❗ Bloquant
		❗ Critique
		⚠ Majeur
		✅ Mineur
		ℹ Info
		437
		3 318
		35 948
		25 308
		1 068

Figure 9: Dette technique globale et défauts

Ce composant mesure la dette technique (Debt), le nombre de problèmes rencontrés (Défauts) et la répartition de ces problèmes selon les niveaux de sévérité.

Ce composant se base sur SQALE (Software Quality Assessment based on Lifecycle Expectations), une méthode d'évaluation de la dette technique du code source, en utilisant les résultats de l'analyse des règles fournis.

La dette technique, par analogie une dette financière, représente le fait que lorsqu'on code de manière non optimale, on contracte une dette technique que l'on rembourse tout au long de la vie du projet sous forme de temps de développement de plus en plus long et de bugs de plus en plus fréquents.

Les niveaux de sévérités, standard, sont les suivants :

- Bloquante : Devrait être résolue immédiatement.
- Critique : Devrait être inspectée immédiatement et résolue au plus tôt.
- Majeure : Forte probabilité d'avoir un impact significatif ou modéré.
- Mineure : Peut potentiellement avoir un impact modéré ou mineur.
- Info : Ni un bug, ni une faiblesse qualitative, juste une information.

Note SQALE

B

Ratio De Dette Technique

9,1%

Dans ce composant, SQALE donne une note globale au projet (SQALE rating) suivant le ratio suivant : dette technique (exprimé en minutes-hommes nécessaire à la résolution de l'ensemble de la dette technique) / temps global de développement du projet (estimé de façon arbitraire à 30min par ligne de code).

La note associée à ce ratio est déterminé selon le modèle utilisé sur le site de parangonnage de SonarQube de la façon suivante :

- Ratio < 5% : Note A
- 5% <= Ratio < 10% : Note B
- 10% <= Ratio < 25% : Note C
- 25% <= Ratio < 50% : Note D
- Ratio >= 50% : Note E

Analyse d'Astek : Une note correcte pour le projet qui cache cependant des défauts d'architecture qui ne peuvent pas être remontés par ce type d'analyse. L'analyse lexicale et grammaticale d'un projet n'est pas forcément gage de qualité. L'architecture peut être comparée à celle d'un bâtiment : il ne s'agit pas seulement de s'assurer de la qualité de la maçonnerie ou du plafonnage mais aussi de l'agencement par exemple.

4.4 Pyramide de dette technique

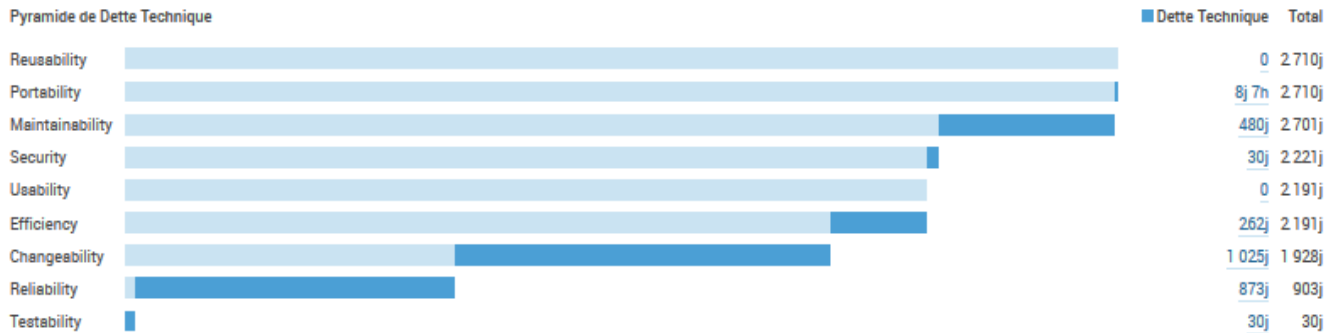


Figure 10: Pyramide de la dette technique

Ce composant nous propose une vue de la dette technique répartie sur neuf grands axes de développement logiciel. Il prend la forme d'une pyramide qu'il faut lire de bas en haut. Chaque étage reprend la dette de la partie inférieure car la pyramide est hiérarchisée : chaque étage est une brique sur laquelle repose l'étage au-dessus de lui. Les différents axes sont :

- Testability (testabilité) : Les parties du code non testables.
- Reliability (fiabilité) : La probabilité que le logiciel fonctionne dans un contexte donné.
- Changeability (versatilité) : Mesure l'effort nécessaire pour la modification de configuration ou le changement d'environnement du logiciel.
- Efficiency (efficacité) : Mesure les performances du logiciel, au niveau du temps d'exécution et le traitement de gros volumes de données.
- Usability (accessibilité) : Mesure l'ergonomie du logiciel.
- Security (sécurité) : Les problèmes de sécurité.
- Maintainability (maintenabilité) : Mesure l'effort pour la correction de bugs ou l'amélioration de performance.
- Portability (portabilité) : comportement du logiciel dans un environnement différent.
- Reusability (réutilisation) : Mesure le degré de modularité du logiciel pour pouvoir utiliser certaines de ces caractéristiques dans les développements futurs.

Analyse d'Astek : On peut observer que les axes les plus critiques sont les axes de fiabilité et de versatilité. En revanche, l'outil SonarQube en version gratuite ne permet pas de remonter les défauts permettant d'améliorer le plus efficacement les axes concernés. La règle reste donc de corriger en priorité les défauts les plus critiques.

4.5 Design des fichiers

Indice D'interdépendance Entre Paquetages

10,3%

Cycles

> 223

Dépendances À Couper

Entre Paquetages

107

Entre Fichiers

669

Figure 11: Métriques de design globales

Ce composant mesure les dépendances cycliques des fichiers et répertoires. Dans une architecture standard, les dépendances sont hiérarchisées, il n'y a pas de dépendances cycliques : il faut éviter d'arriver dans un état de blocage à cause de telles dépendances.

Analyse d'Astek : Le projet AMALFI comporte un certain nombre de dépendances cycliques à inspecter afin de vérifier si elles sont évitables.

4.6 Duplications

Duplications

9,7%

Lignes	Blocs	Fichiers
74 803	4 112	972

Figure 12: Métriques de duplication globales

Ce composant mesure le degré de duplication de lignes de code. Cette duplication est parfois due à une mauvaise architecture logicielle : la notion d'héritage par exemple peut être sous-exploitée. La duplication peut aussi être issue d'un générateur de code, qui à partir d'un modèle, réplique la même logique, ou simplement d'une inadvertance des développeurs.

Quelle que soit la raison, il faut autant que possible éviter les duplications de code, elle entraîne la réplication des bugs (et la réplication de la dette technique associée), augmente le temps de maintenance du logiciel et diminue la compréhension globale du code.

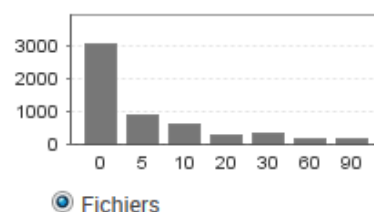
Analyse d'Astek : si d'une manière générale, il est toléré de dupliquer une partie de code, il ne devrait jamais être répliqué plus de deux fois. Malheureusement SonarQube ne dispose pas d'outils permettant de visualiser cette problématique en particulier. Une partie du code de l'application AMALFI est générée ce qui implique inmanquablement une duplication de code. Les duplications devront être inspectées et si possible supprimées.

4.7 Complexité

Complexité

107 445

/Méthode	/Classe	/Fichier
2,3	16,1	19,1



La complexité cyclomatique est le nombre de chemins linéairement indépendants qu'il est possible de suivre. La règle pour java est de l'augmenter de 1 pour chaque if, for, while, case, catch, throw, return, &&, ||, ?. En règle générale, une complexité inférieure à 200 est acceptable pour une classe (SonarQube a trouvé 29 fichiers contenant une classe ayant une complexité supérieure à 200), si elle est couverte par les tests fonctionnels. Au-delà d'une complexité de 200, la classe doit être refactorisée. Pour une méthode, la complexité acceptable est de 10 (SonarQube a trouvé une centaine de fichiers ayant des méthodes avec une complexité supérieure à 10).

Une complexité trop élevée entraîne les problèmes suivants :

- Manque de lisibilité du code.
- Des messages d'erreurs moins précis car ils couvrent plus de code.
- Des tests unitaires plus long à produire, donc une méthode/classe plus difficile à maintenir.

4.8 Règles les plus enfreintes

Règles les plus enfreintes	Bloquant	
Throwable and Error should not be caught	429	
"equals(Object obj)" and "hashCode()" should be overridden in pairs	9	

Most Violated Rules	Critical	
Exception handlers should preserve the original exception	1 521	
Fields in a "Serializable" class should either be transient or serializable	778	
"public static" fields should always be constant	472	
Throwable.printStackTrace(...) should not be called	302	
"static final" arrays should be "private"	72	
"equals(Object obj)" should be overridden along with the "compareTo(T obj)" method	66	
Classes should not be compared by name	59	

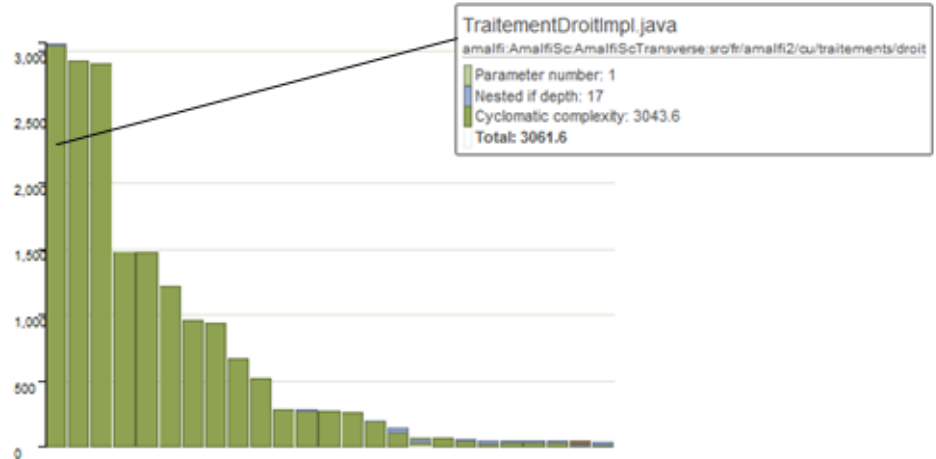
Most Violated Rules	Major	
"@Override" annotation should be used on any method overriding (since Java 5) or implementing (since Java 6) another one	10 651	
Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList"	1 908	
Generic exceptions should never be thrown	1 474	
Method parameters, caught exceptions and foreach variables should not be reassigned	1 465	
Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	1 293	
Strings literals should be placed on the left side when checking for equality	1 253	
Sections of code should not be "commented out"	1 240	
Methods should not be too complex	1 130	
Unused method parameters should be removed	1 092	

Analyse d'Astek : Si les deux défauts considérés bloquants doivent être corrigés, ils ne constituent pas aujourd'hui des éléments susceptibles de produire des dysfonctionnements de l'application. Le projet AMALFI a débuté il y a plus de dix ans, quand la version de java était bien inférieure. De cela résultent quelques problèmes liés à des règles introduites récemment et qui repose sur des outils qui n'existaient pas à l'époque. Comme par exemple l'utilisation des génériques, introduit dans la version 5 du langage java. De plus l'outil ne pouvant « comprendre » le code, certains défauts sont des faux positifs. L'outil permet de les marquer comme tels.

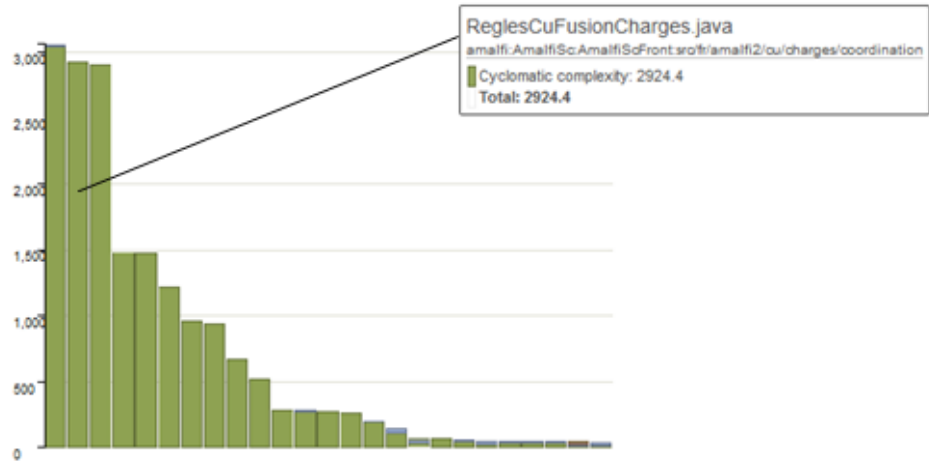
4.9 Toxicity

Les 5 fichiers concentrant le plus haut niveau de toxicité sont les suivants :

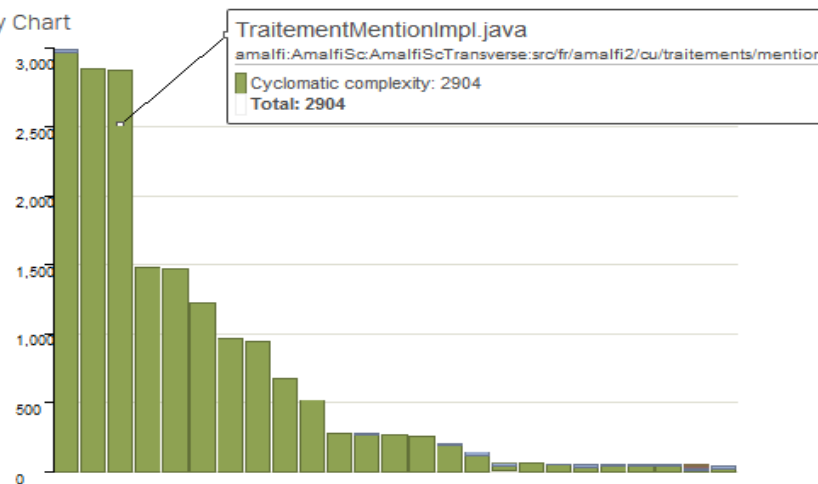
Toxicity Chart



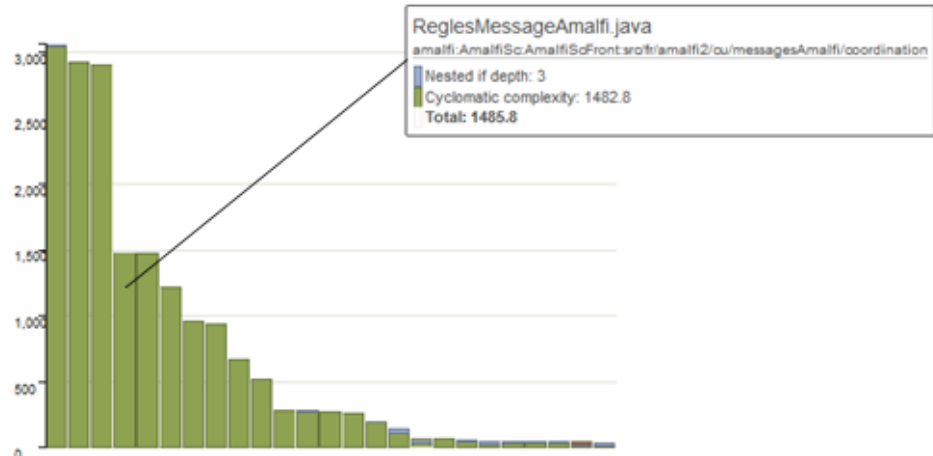
Toxicity Chart



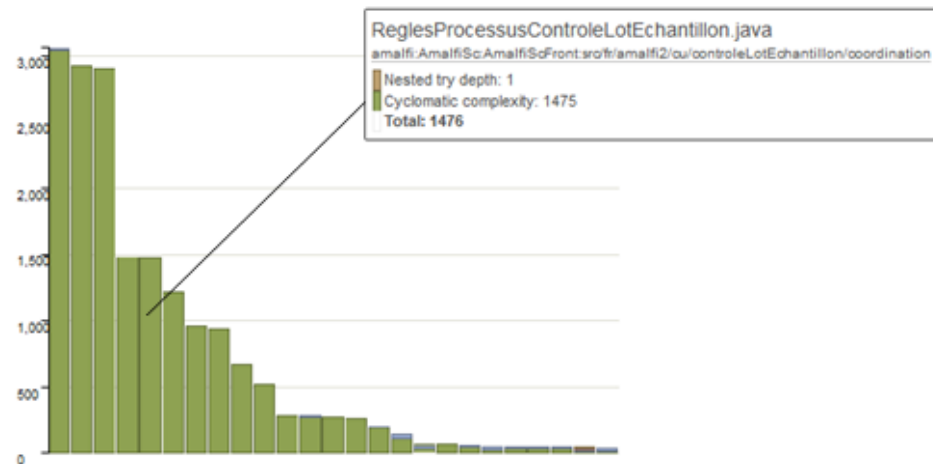
Toxicity Chart



Toxicity Chart



Toxicity Chart



Analyse d'Astek : Une quinzaine de fichiers correspondant essentiellement aux classes de traitement et de définition des règles de validation concentrent les problèmes. Il serait pertinent d'évaluer la possibilité de les découper afin de réduire leur complexité.

4.10 Détail des métriques par « projet »

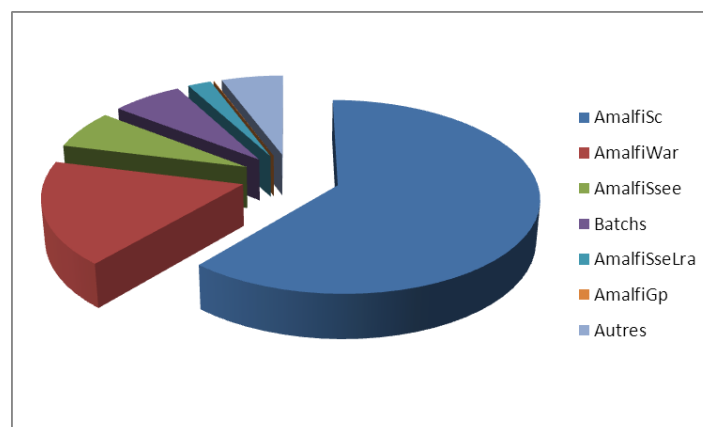


Figure 13: Répartition par taille des projets AMALFI

Dans la suite nous ne considérerons pas les projets Tools et Tests qui sont des projets supports n'ayant aucun objectif métier ni le projet AmalfiGp qui ne fait que reprendre le code du projet AmalfiSc et de ce fait ne contient pratiquement pas de code.

Métriques\Projets	AmalfiSc	AmalfiSsee	AmalfiWar	AmalfiSseLra	Batches
Lignes de code	65,1%	7,0%	18,7%	2,3%	7,0%
Nombre de méthodes	75,8%	7,2%	9,5%	1,2%	6,1%
Nombre de classes	81,8%	6,4%	NA	3,6%	8,2%
Dette technique	48,9%	6,3%	37,8%	0,8%	6,1%
Nombre de défauts	58,6%	5,8%	28,7%	0,8%	6,0%
Nombre de défauts par ligne de code	12,5%	11,6%	21,4%	5,0%	12,1%
Complexité par méthode	2,5	2,6	8,3	3,2	2,6
Complexité par classe	16,3	20,4	NA	7,7	13,6
Indice d'interdépendance	10,7%	2,0%	NA	0,0%	4,7%
Duplications	6,9%	6,8%	31,2%	1,2%	5,0%
Documentation	76,40%	81,50%		50,50%	45,70%
Commentaires	21,90%	23%	2,10%	14,80%	13,80%

Le projet AmalfiSc comporte le plus de lignes de code et de façon naturelle la dette technique et le nombre de défauts les plus importants. Cependant c'est le projet AmalfiWar qui présente le plus grand nombre de défaut par ligne de code, la plus grande complexité par méthode, le plus grand nombre de duplication et le moins de commentaires. La nature particulière de ce projet (code HTML, CSS et Javascript) en est, de toute évidence, la raison. Le projet AmalfiSsee et les Batches ont un pourcentage de dette technique et de défaut en proportion du pourcentage de code. Le projet AmalfiSseLra est le projet qui participe le moins à la dette et qui comporte la plus faible proportion de défauts.

4.10.1 Le projet AmalfiSc : le cœur d'Amalfi

Métriques de volumétrie

Lignes De Code

292 273

Java

XML

Web

CSS

290 723

935

355

260

Fichiers

3 718

Répertoires

424

Lignes

516 358

Méthodes

24 975

Classes

3 892

Instructions

136 301

Accesseurs

4 583

Figure 14: Métriques de volumétrie des projets d'AmalfiSc

Métriques de documentation

Documentation

76,4%

API Publiques

22 528

API Pub. Non Doc.

5 321

Commentaires

21,9%

Lignes De Commentaires

81 773

Figure 15: Métriques de documentation des projets d'AmalfiSc

Dette technique & défauts

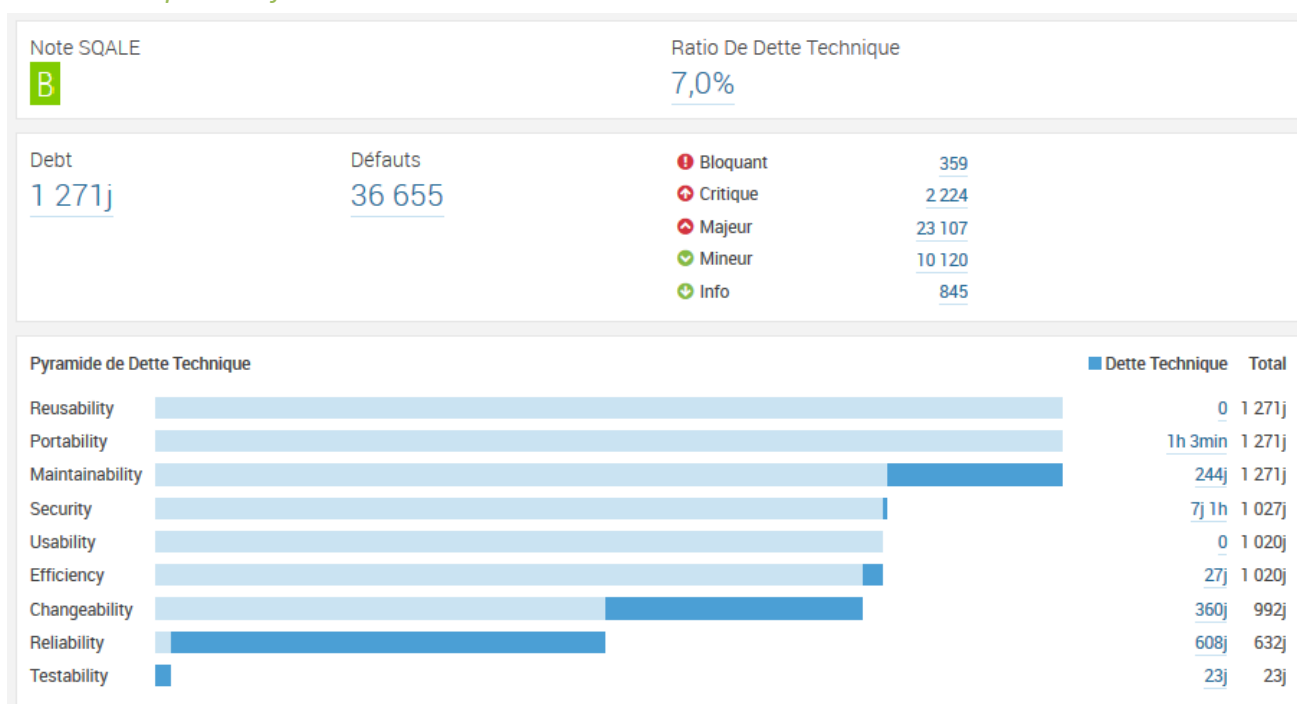


Figure 16: Dette technique des projets AmalfiSc

Design des fichiers

Indice D'interdépendance Entre Paquetages

10,7%

Cycles

> 213

Dépendances À Couper

Entre Paquetages

101

Entre Fichiers

647

Figure 17: Métriques des dépendances des projets d'AmalfiSc

Duplications

Duplications

6,9%

Lignes	Blocs	Fichiers
35 541	2 407	601

Figure 18: Métriques de duplication des projets d'AmalfiSc

Complexité

Complexité

63 278

/Méthode	/Classe	/Fichier
2,5	16,3	17,0

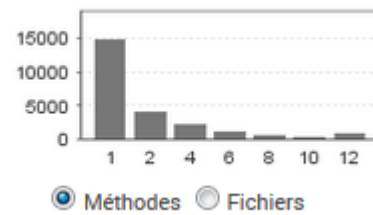


Figure 19: Métriques de complexité des projets d'AmalfiSc

Liste de règles enfreintes toutes sévérités confondues :

Règles les plus enfreintes Toute sévérité

[En savoir plus](#)

⚠ "@Override" annotation should be used on any method overriding (since Java 5) or implementing (since Java 6) another one	8 284	
✅ String literals should not be duplicated	2 170	
✅ Tabulation characters should not be used	2 025	
⚠ Method parameters, caught exceptions and foreach variables should not be reassigned	1 358	
⚠ Insufficient branch coverage by unit tests	1 344	
⚠ Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList"	1 260	
✅ Overriding methods should do more than simply call the same method in the super class	1 233	
⚠ Exception handlers should preserve the original exception	1 120	
⚠ Generic exceptions should never be thrown	1 022	
⚠ Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	954	
⚠ Strings literals should be placed on the left side when checking for equality	879	
⚠ Unused method parameters should be removed	857	
⚠ Sections of code should not be "commented out"	851	
⚠ Methods should not be too complex	837	
⚠ "TODO" tags should be handled	665	
⚠ Fields in a "Serializable" class should either be transient or serializable	632	
✅ Method names should comply with a naming convention	601	
⚠ Duplicated blocks	601	
✅ Statements should be on separate lines	585	
⚠ Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used	574	

Répartition par sous projets :

NOM	LIGNES DE CODE	DETTE TECHNIQUE	DÉFAUTS BLOQUANTS
AmalfiRedevanceEjb	358	6h 24min	4
AmalfiRedevanceSSO	83	1h 29min	1
AmalfiScBack	70 220	403j	251
AmalfiScEjb	664	1j 7h	3
AmalfiScFront	146 651	605j	75
AmalfiScTransverse	74 297	259j	25

Le sous projet AmalfiScBack apparait comme le projet contenant le plus de défauts bloquants et une dette technique importante. AmalfiRedevanceEjb et AmalfiRedevanceSSO sont les projets supportant l'interface d'Amalfi à l'application de Redevance.

4.10.2 Le projet AmalfiSsee : une extension d'AmalfiSc

Métriques de volumétrie

Lignes De Code	Fichiers	Méthodes
31 409	291	2 388
JAVA XML	30 993 416	Répertoires 73
	Lignes 55 687	Classes 304
		Instructions 15 046
		Accesseurs 675

Figure 20: Métriques de volumétrie des projets d'AmalfiSsee

Métriques de documentation

Documentation	Commentaires
81,5%	23,0%
API Publiques 1 931	Lignes De Commentaires 9 403
API Pub. Non Doc. 357	

Figure 21: Métriques de documentation des projets d'AmalfiSsee

Dette technique & défauts

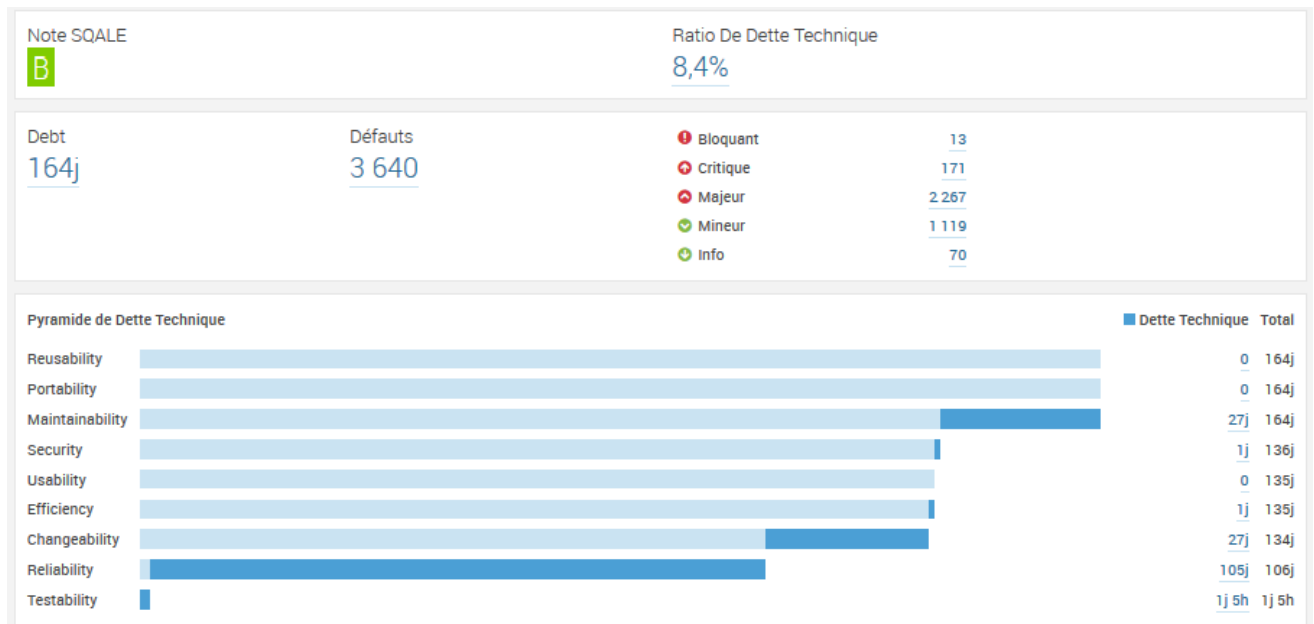


Figure 22: Dette technique des projets d'AmalfiSse

Design des fichiers

Indice D'interdépendance Entre Paquetages

2,0%

Cycles

> 1

Dépendances À Couper

Entre Paquetages

1

Entre Fichiers

3

Figure 23: Métriques de design des projets d'AmalfiSsee

Duplications

Duplications

6,8%

Lignes Blocs Fichiers

3 814 186 86

Figure 24: Métriques de duplication des projets d'AmalfiSsee

Complexité

Complexité

6 201

/Méthode /Classe /Fichier

2,6 20,4 21,3

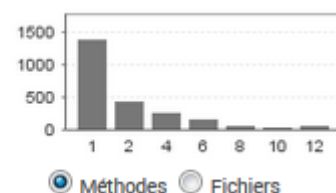


Figure 25: Métriques de complexité des projets d'AmalfiSsee

Liste de règles enfreintes toutes sévérités confondues :

Règles les plus enfreintes Toute sévérité

[En savoir plus](#)

⚠ "@Override" annotation should be used on any method overriding (since Java 5) or implementing (since Java 6) another one	970	
✅ String literals should not be duplicated	426	
✅ Package names should comply with a naming convention	205	
⚠ Unused method parameters should be removed	168	
⚠ Insufficient branch coverage by unit tests	158	
✅ Tabulation characters should not be used	117	
⚠ Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	101	
⚠ Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList"	94	
⚠ Collapsible "if" statements should be merged	92	
⚠ Duplicated blocks	86	
⚠ Fields in a "Serializable" class should either be transient or serializable	86	
✅ Class names should comply with a naming convention	80	
⚠ Sections of code should not be "commented out"	78	
✅ "TODO" tags should be handled	70	
⚠ Methods should not be too complex	67	
✅ Method names should comply with a naming convention	56	
⚠ Strings literals should be placed on the left side when checking for equality	54	
⚠ Collection.isEmpty() should be used to test for emptiness	51	
⚠ Subclasses that add fields should override "equals"	49	
✅ Useless imports should be removed	42	

Répartition par sous-projets

NOM	LIGNES DE CODE	DETTE TECHNIQUE	DÉFAUTS BLOQUANTS
AmalfiSseeBack	761	2j 7h	1
AmalfiSseeEjb	1 116	4j 4h	7
AmalfiSseeFront	29 257	155j	5
AmalfiSseeTransverse	275	6h 43min	0

Liste de règles enfreintes critiques :

Règles les plus enfreintes Critique

[En savoir plus](#)

⚠ Property values should be valid	59	
⚠ "" and "<dt>" item tags should be in "", "" or "<dl>" container tags	2	

Liste de règles enfreintes majeures :

Règles les plus enfreintes	Majeur	En savoir plus
Over-specified selectors should be simplified	1 814	
Name of overqualified element should be removed	1 800	
IDs in selectors should be removed	735	
Attributes deprecated in HTML5 should not be used	599	
Units for zero length values should be removed	405	
Shorthand properties should be used whenever possible	229	
Box model size should be carefully reviewed	221	
Each declaration should end with a semicolon	218	
Duplicated background images should be removed	147	
Images tags and buttons should have an "alt" attribute	69	
 and tags should be used instead of and <i>	64	
Properties that do not work with the "display" property should be removed	60	
Empty declarations should be removed	40	
"===" and "!==" should be used instead of "==" and "!="	35	
Elements deprecated in HTML5 should not be used	31	
Favicons should be used in all pages	22	
A <!DOCTYPE> declaration should appear before the <html> tag	18	
"title" should be present in all pages	14	
Sections of code should not be "commented out"	8	
Unknown CSS properties should be removed	7	

Le projet AmalfiSsee constitue une extension du projet AmalfiSc. Sa qualité globale est équivalente à celle d'AmalfiSc avec une documentation légèrement meilleure et une complexité par classe légèrement moins bonne.

4.10.3 Le projet AmalfiWar : le projet Web détenant la moins bonne note

Métriques de volumétrie

Lignes De Code	Fichiers	Méthodes
83 846	893	3 145
Web	60 867	Répertoires Lignes
CSS	18 987	107 95 681
XML	3 619	Classes Instructions Accesseurs
JavaScript	373	0 13 342 0

Figure 26: Métriques de volumétrie du projet AmalfiWar

Métriques de documentation

Commentaires
2,1%
Lignes De Commentaires
1 820

Figure 27: Métriques de documentation des projets du projet AmalfiWar

Règles les plus enfreintes

Toute sévérité

[En savoir plus](#)

✓ Selectors should follow a naming convention	7 146	
✓ Leading zeros should be removed	2 028	
✓ Source code should comply with formatting standards	1 926	
⚠ Over-specified selectors should be simplified	1 814	
⚠ Name of overqualified element should be removed	1 800	
⚠ IDs in selectors should be removed	735	
⚠ Attributes deprecated in HTML5 should not be used	599	
⚠ Units for zero length values should be removed	405	
⚠ Shorthand properties should be used whenever possible	229	
⚠ Box model size should be carefully reviewed	221	
⚠ Each declaration should end with a semicolon	218	
⚠ Duplicated background images should be removed	147	
✓ Tabulation characters should not be used	77	
⚠ Images tags and buttons should have an "alt" attribute	69	
⚠ and tags should be used instead of and <i>	64	
⚠ Properties that do not work with the "display" property should be removed	60	
⚠ Property values should be valid	59	
✓ Lines should not end with trailing whitespaces	46	
✓ There should be one single declaration per line	46	
⚠ Empty declarations should be removed	40	
⚠ "===" and "!==" should be used instead of "==" and "!="	35	
✓ Source code should be indented consistently	35	
⚠ Elements deprecated in HTML5 should not be used	31	
⚠ Favicons should be used in all pages	22	
⚠ A <!DOCTYPE> declaration should appear before the <html> tag	18	

Un certain nombre de défauts CSS sont liées à du code spécifique pour les navigateurs en raison d'incompatibilité à l'époque de la création de l'application.

Pas de décomposition en sous projet.

4.10.4 Le projet AmalfiSseLra : le projet détenant la meilleure note

Métriques de volumétrie

Lignes De Code	Fichiers	Méthodes
10 406	187	409
Java	8 878	Classes
XML	842	Instructions
Web	686	Accesseurs
	Répertoires	
	25	
	Lignes	
	15 912	

Figure 31: Métriques de volumétrie des projets d'AmalfiSseLra

Métriques de documentation

Documentation

50,5%

API Publiques

416

API Pub. Non Doc.

206

Commentaires

14,8%

Lignes De Commentaires

1 813

Figure 32: Métriques de documentation des projets d'AmalfiSseLra

Dette technique & défauts

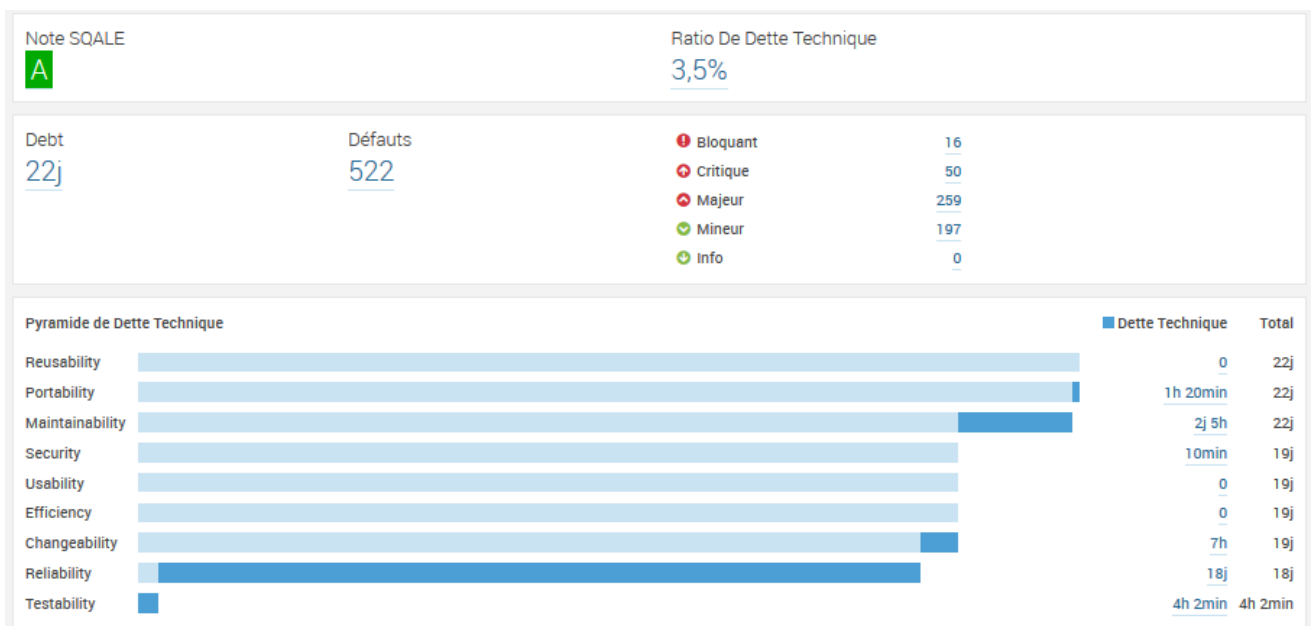


Figure 33: Dette technique des projets d'AmalfiSseLra

Design des fichiers

Indice D'interdépendance Entre Paquetages

0,0%

Cycles

> 0

Dépendances À Couper

Entre Paquetages

0

Entre Fichiers

0

Figure 34: Métriques de design des projets d'AmalfiSseLra

Duplications

Duplications

1,2%

Lignes

183

Blocs

10

Fichiers

7

Figure 35: Métriques de duplication des projets d'AmalfiSseLra

Complexité

Complexité

1 336

/Méthode /Classe /Fichier

3,2 7,7 7,1

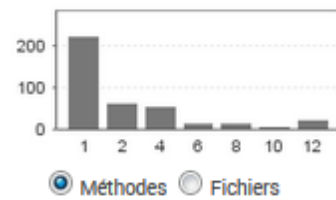


Figure 36: Métriques de complexité des projets d'AmalfiSseLra

Liste de règles enfrentes toutes sévérités confondues :

Règles les plus enfrentes	Toute sévérité	En savoir plus
⚠️ "@Override" annotation should be used on any method overriding (since Java 5) or implementing (since Java 6) another one	83	
✅ Tabulation characters should not be used	75	
⚠️ Insufficient branch coverage by unit tests	30	
✅ String literals should not be duplicated	29	
⚠️ Fields in a "Serializable" class should either be transient or serializable	28	
✅ Modifiers should be declared in the correct order	26	
⚠️ Methods should not be too complex	22	
⚠️ Exception handlers should preserve the original exception	17	
⚠️ Throwable and Error should not be caught	16	
⚠️ A <!DOCTYPE> declaration should appear before the <html> tag	16	
⚠️ Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	16	
⚠️ Favicons should be used in all pages	16	
✅ Local variable and method parameter names should comply with a naming convention	15	
✅ Loggers should be "private static final" and should share a naming convention	13	
⚠️ Sections of code should not be "commented out"	13	
✅ Tabulation characters should not be used	13	
✅ The members of an interface declaration or class should appear in a pre-defined order	9	
⚠️ Collapsible "if" statements should be merged	7	
⚠️ Generic exceptions should never be thrown	6	
⚠️ Utility classes should not have public constructors	6	

Répartition par sous-projets :

NOM	LIGNES DE CODE	DETTE TECHNIQUE	DÉFAUTS BLOQUANTS
AmalfiSseLraBack	4 726	17j	7
AmalfiSseLraFront	2 666	2j 6h	8
AmalfiSseLraFrontWar	909	2h 33min	0
AmalfiSseLraTransverse	2 105	2j 3h	1

Le projet AmalfiSseLra présente moins de duplications de code que les projets AmalfiSc et AmalfiSsee en grande partie par le fait qu'il n'y pas de code généré dans ce projet. Il n'a aucune interdépendance des « paquetages » et aucun défaut bloquant. C'est le projet présentant la meilleure qualité globale avec la plus petite dette technique et le plus faible pourcentage de défauts.

4.10.5 Les projets des Batches

Métriques de volumétrie

Lignes De Code	Fichiers	Méthodes			
31 281	372	2 023			
Java	30 736	Répertoires	Lignes	Classes	Instructions
XML	545	69	47 325	391	14 416
					Accesseurs
					752

Figure 37: Métriques de volumétrie des Batches

Métriques de documentation

Documentation	Commentaires
45,7%	13,8%
API Publiques	Lignes De Commentaires
1 679	4 999
API Pub. Non Doc.	
911	

Figure 38: Métriques de documentation des Batches

Dette technique & défauts

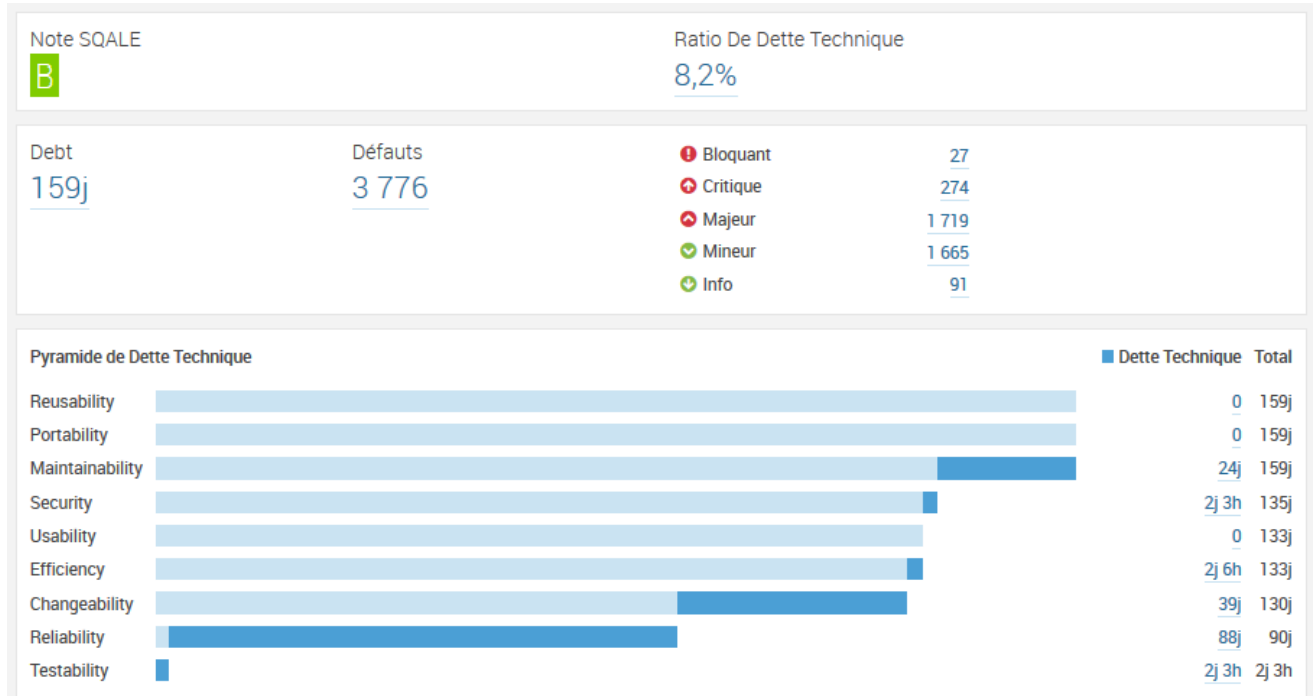


Figure 39: Dette technique des Batches

Design des fichiers

Indice D'interdépendance Entre Paquetages

4,7%

Cycles

> 3

Dépendances À Couper

Entre Paquetages

3

Entre Fichiers

9

Duplications

Duplications

5,0%

Lignes

2 357

Blocs

280

Fichiers

65

Figure 40: Métriques de duplication des Batches

Complexité

Complexité

5 334

/Méthode

2,6

/Classe

13,6

/Fichier

14,3

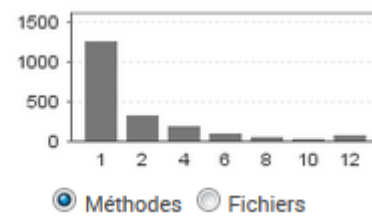


Figure 41: Métriques de complexité des Batches

Répartition par sous-projets :

NOM	LIGNES DE CODE	DETTE TECHNIQUE	DÉFAUTS BLOQUANTS
Batch	821	3j 4h	7
BatchAmalfi	3 151	17j	8
BatchBaseReference	5 179	40j	1
BatchCalculObjetsAnormaux	730	1j 7h	1
BatchCommunRedevance	74	2h 42min	0
BatchCreationCommandesRequetes	563	1j 7h	0
BatchCreationXmlCopie	247	6h 40min	0
BatchDereeserv	195	5h 53min	0
BatchDgi	7 051	40j	1
BatchEnvoiCopieElectronique	865	3j 6h	2
BatchExportBI	806	5j 7h	0
BatchFusionCommunes	802	1j 4h	0
BatchGenerationBI	511	1j 6h	0
BatchGenerationJournauxAF	727	3j	1
BatchGestionEnum	167	3h 27min	0
BatchIah	1 131	6h 12min	1
BatchImportBI	637	3j 2h	1
BatchMajScan	390	1j	0
BatchNettoyageSignatures	292	6h 37min	0
BatchPavePace	2 158	7j 4h	3
BatchRedevanceCopies	539	1j 6h	0
BatchReparationPublication	513	4j	0
BatchRescellementRequete	138	2h 50min	0
BatchStats	2 246	10j	0
BatchStockageGed	250	2h 40min	0
BatchSuppressionXmlCopieAncienne	67	3h 32min	0
BatchSynchronisationRedevance	747	2j 3h	0
BatchTests		0	0
TraitementaBdd	284	1j 4h	0

Liste de règles enfreintes toutes sévérités confondues :

Règles les plus enfreintes Toute sévérité

[En savoir plus](#)

⚠️ "@Override" annotation should be used on any method overriding (since Java 5) or implementing (since Java 6) another one	970	
✅ String literals should not be duplicated	426	
✅ Package names should comply with a naming convention	205	
⚠️ Unused method parameters should be removed	168	
⚠️ Insufficient branch coverage by unit tests	158	
✅ Tabulation characters should not be used	117	
⚠️ Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	101	
⚠️ Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList"	94	
⚠️ Collapsible "if" statements should be merged	92	
⚠️ Duplicated blocks	86	
⚠️ Fields in a "Serializable" class should either be transient or serializable	86	
✅ Class names should comply with a naming convention	80	
⚠️ Sections of code should not be "commented out"	78	
✅ "TODO" tags should be handled	70	
⚠️ Methods should not be too complex	67	
✅ Method names should comply with a naming convention	56	
⚠️ Strings literals should be placed on the left side when checking for equality	54	
⚠️ Collection.isEmpty() should be used to test for emptiness	51	
⚠️ Subclasses that add fields should override "equals"	49	
✅ Useless imports should be removed	42	

Le projet Batches comporte un grand nombre de sous projets de tailles différentes. La qualité globale est similaire à celle des projets AmalfiSc et AmalfiSsee avec toutefois une documentation plus faible.

4.11 Parangonnage des valeurs obtenues

En prenant en compte l'ensemble des projets, toute volumétrie confondue, référencés sur le site de SonarQube, nous avons les chiffres suivants :

	Min.	Moy.	Max.	AMALFI
Ratio de dette technique	0,6%	4,6%	18,1%	9,1%
Commentaires (%)	0,0%	21,2%	22,8%	18,0%
API publique documentée (%)	0,0%	52,7%	99,0%	72,7%

Dans le tableau qui suit, nous comparons les différents projets AMALFI à un sous-ensemble (projets Java de taille supérieure à 1000 lignes de code) des projets disponibles sur le site référentiel de SonarQube (<https://nemo.sonarqube.org/>)

Projet	LDC	Note	Dette technique	Nombre de défauts	Ratio Dette/LDC	Ratio Défaut/LDC
Apache HBase	575 351	A	1,169j	80 333	3,3%	13,96%
Elasticsearch: Parent	554 444	A	997j	36 566	2,9%	6,60%
AmalfiSC	292 273	B	1271j	36 655	7,0%	12,54%
Apache Jackrabbit	276 199	A	777j	29 392	4,5%	10,64%
Apache Tomcat	235 121	A	326j	19 197	2,2%	8,16%
Jetspeed-2 Enterprise Portal	202 695	B	1,164j	33 467	9,2%	16,51%
Wicket Parent	196 947	A	572j	12 595	4,6%	6,40%
SonarQube	176 103	A	56j	2 195	0,5%	1,25%
Cayenne	142 457	A	268j	10 745	3,0%	7,54%
Jenkins main module	140 022	B	806j	29 423	9,2%	21,01%
Struts 2	122 740	B	666j	19 930	8,7%	16,24%
Hudson	122 094	C	878j	21 280	11,5%	17,43%
jOOQ Parent	107 047	B	369j	13 559	5,5%	12,67%
Apache Lucene	90 728	A	128j	7 716	2,3%	8,50%
Apache Tobago	81 739	A	206j	6 008	4,0%	7,35%
lightblue-applications	81 034	C	630j	18 208	12,4%	22,47%
Bonita	80 307	A	164j	4 123	3,3%	5,13%
Atlasboard	79 115	A	140j	8 966	2,8%	11,33%
Activiti	77 098	A	147j	6 204	3,1%	8,05%
Google Cloud Dataflow Java SDK - Parent	58 970	A	82j	3 316	2,2%	5,62%
Jajuk	57 029	A	110j	3 240	3,1%	5,68%
PMD	56 869	A	115j	4 794	3,2%	8,43%
Apache Wink	56 321	A	89j	3 989	2,5%	7,08%
Apache Tika	55 418	A	110j	5 256	3,2%	9,48%
Apache Shindig Project	54 981	A	73j	3 984	2,1%	7,25%
Apache Sirona Incubator	54 219	B	326j	9 432	9,6%	17,40%
Apache OpenNLP Reactor	50 867	A	153j	6 912	4,8%	13,59%
Apache Abdera	50 568	A	112j	3 587	3,5%	7,09%

Maven SCM	50 085	A	93j	2 833	3,0%	5,66%
Fitnessse	49 754	A	82j	3 115	2,6%	6,26%
sonar-persistit	45 690	A	72j	4 355	2,5%	9,53%
Apache Empire-db	43 979	A	101j	3 310	3,7%	7,53%
Doxia Aggregator	42 360	A	53j	3 262	2,0%	7,70%
Apache Hama parent POM	40 361	A	76j	2 626	3,0%	6,51%
cougar-master-pom	36 342	A	54j	2 525	2,4%	6,95%
OPS4J Pax Logging (Build POM)	32 288	A	94j	3 390	4,7%	10,50%
Apache MyFaces CODI	31 877	A	53j	1 752	2,7%	5,50%
AmalfiSsee	31 409	B	164j	3640	8,4%	11,59%
Batches	31 281	B	159j	2023	9,2%	6,47%
TestNG	31 098	A	51j	2 811	2,6%	9,04%
PGJDBC-NG	30 935	A	27j	1 294	1,4%	4,18%
Apache Commons BCEL	30 695	A	66j	3 131	3,4%	10,20%
checkstyle	29 866	A	10j	286	0,5%	0,96%
Apache Commons Collections	28 589	A	39j	1 972	2,2%	6,90%
Apache Commons Configuration	27 362	A	10j	363	0,6%	1,33%
Apache XBean	26 898	A	51j	2 801	3,0%	10,41%
Apache Vysper Parent	26 892	A	56j	2 034	3,3%	7,56%
Apache Commons Lang	26 324	A	36j	1 520	2,2%	5,77%
Apache Pluto	25 965	A	53j	1 554	3,3%	5,98%
WebGoat	25 501	A	56j	2 123	3,5%	8,33%
Kryo Parent	24 228	B	79j	4 820	5,2%	19,89%
XStream Parent	23 869	B	86j	1 946	5,8%	8,15%
Apache Commons VFS	23 568	A	31j	1 215	2,1%	5,16%
lightblue-core	21 932	A	23j	982	1,7%	4,48%
Jackcess	21 706	A	15j	1 377	1,1%	6,34%
Java Microbenchmark Harness Parent	21 468	A	54j	4 570	4,0%	21,29%
pitest-parent	20 949	A	16j	434	1,2%	2,07%
AssertJ fluent assertions	20 909	A	29j	1 703	2,2%	8,14%
Apache Rampart	20 815	A	56j	1 922	4,3%	9,23%
Java Concurrency Stress Tests: Parent	19 836	A	37j	2 031	3,0%	10,24%
Apache Gora	18 624	A	23j	1 129	2,0%	6,06%
project	17 268	A	26j	1 557	2,4%	9,02%
ApacheDS Mavibot Parent	17 222	A	25j	1 357	2,3%	7,88%
sejda	16 356	A	7j 3h	358	0,7%	2,19%
Apache Maven Wagon	15 561	A	28j	881	2,9%	5,66%
hRaven Project	13 771	A	20j	959	2,3%	6,96%
Apache Commons OGNL - Object Graph Navigation Library	13 639	A	18j	758	2,1%	5,56%
Maven-Indexer	13 466	A	17j	1 076	2,0%	7,99%
Apache Commons Digester	12 696	A	12j	630	1,5%	4,96%
AisLib application framework	12 418	A	29j	1 126	3,7%	9,07%

Apache Commons BeanUtils	12 028	A	21j	1 069	2,8%	8,89%
Whirr	11 790	A	11j	465	1,5%	3,94%
Apache Commons DBCP	11 595	A	13j	1 096	1,8%	9,45%
== GreenMail ==	10 625	A	12j	481	1,8%	4,53%
Apache Commons IO	10 594	A	13j	602	2,0%	5,68%
POM Parent	10 494	A	10j	430	1,5%	4,10%
AmalfiSseLra	10 406	A	22j	522	3,4%	5,02%
jGrades Application	10 003	A	2j	70	0,3%	0,70%
Apache Asyncweb Parent	9 949	A	19j	575	3,1%	5,78%
JUnit	9 886	A	11j	560	1,8%	5,66%
Maven Release	9 753	A	19j	701	3,1%	7,19%
Apache Commons SCXML	9 743	A	12j	630	2,0%	6,47%
Moneta (JSR 354 RI)	9 618	A	5j 4h	259	0,9%	2,69%
Apache PhotArk	9 263	A	25j	1 509	4,3%	16,29%
OPS4J - Pax Construct	8 834	A	7j 4h	443	1,4%	5,01%
simple-spring-memcached-parent	8 664	A	10j	235	1,8%	2,71%
Stapler Parent	8 402	A	25j	1 048	4,8%	12,47%
restfiddle	8 243	A	17j	636	3,3%	7,72%
Camus Parent	8 196	A	10j	537	2,0%	6,55%
GMaps4JSF Project	8 184	B	45j	1 157	8,8%	14,14%
sevntu-checks	8 020	A	4j 6h	436	0,9%	5,44%
EclEmma	8 006	B	27j	1 298	5,4%	16,21%
CodeStory - Fluent-http	7 758	A	5j 5h	158	1,2%	2,04%
ObjectLab Kit	7 642	A	9j 3h	214	2,0%	2,80%
ch-smpp	7 158	A	10j	530	2,2%	7,40%
disCoverJ	6 993	A	12j	371	2,7%	5,31%
J2ObjC Gradle Plugin	6 239	A	1j	34	0,3%	0,54%
OPS4J Pax Exam (Build POM)	5 991	A	6j 2h	281	1,7%	4,69%
Apache Commons Chain :: Parent	5 678	A	13j	786	3,7%	13,84%
Apache Commons Pool	5 652	A	15j	508	4,2%	8,99%
Apache MINA 3.0.0-M1-SNAPSHOT	5 599	A	8j 3h	347	2,4%	6,20%
FlatPack	5 477	A	11j	484	3,2%	8,84%
Apache Maven Enforcer	5 183	A	5j 7h	241	1,8%	4,65%
Retrofit (Parent)	5 137	A	10j	376	3,1%	7,32%
lightblue-mongo	5 078	A	5j 3h	230	1,7%	4,53%
hipster-pom	4 939	A	8j	483	2,6%	9,78%
lightblue-rest	4 688	A	5j 2h	371	1,8%	7,91%
HikariCP	4 675	A	3j 7h	164	1,3%	3,51%
Polyforms Framework	4 613	A	6h 46min	55	0,3%	1,19%
mp3agic	4 542	A	10j	561	3,5%	12,35%
Spark	4 477	A	5j 6h	249	2,1%	5,56%
lightblue-client	4 212	A	5j 5h	201	2,1%	4,77%
PuppetLabs Apache	4 212	A	6j	199	2,3%	4,72%
Ambrose	4 132	A	5j 7h	191	2,3%	4,62%

markdown4j	3 697	B	14j	424	6,1%	11,47%
Okio (Parent)	3 521	A	5j 5h	439	2,6%	12,47%
Apache Commons Logging	3 281	A	9j	374	4,4%	11,40%
Language Integrated QERies For ORM/JPA	3 259	A	1j 5h	99	0,8%	3,04%
JSR 354 (Money and Currency API)	3 205	A	1j 5h	80	0,8%	2,50%
Apache Commons Exec	2 032	A	3j 4h	161	2,8%	7,92%
AssertJ Assertions Generator	1 934	A	1j 6h	122	1,4%	6,31%
petclinic	1 877	A	1j 4h	107	1,3%	5,70%
cucumber-reporting	1 841	A	5h 9min	66	0,6%	3,59%
silencio	1 332	A	3h	31	0,5%	2,33%
Spojo	1 152	A	1j 3h	84	1,9%	7,29%
FEST Util	1 075	A	6h 54min	100	1,3%	9,30%

Le parangonnage n'a pas été réalisé pour le projet AmalfiWar car il n'est constitué que de pages Web, de feuilles de style CSS et de code Javascript, et ne peut donc pas se comparer aux projets Java.

4.12 Analyse des défauts

Le tableau ci-dessous présente une analyse des défauts relevés, avec en rouge les défauts bloquants, en orange les critiques et en noir les majeurs. La sévérité (bloquant/critique/majeur) est celle propre à SonarQube (voir le chapitre 4.3 - Dette technique & défauts), elle est dé-corrélée de celle définie dans le CCTP.

Libellé du défaut	Nbre	Difficulté de correction	Action
Throwable and Error should not be caught	429	Simple	L'application ne doit pas traiter les exceptions de bas niveau telles que : InternalError, OutOfMemoryError, StackOverflowError, Remplacer les occurrences de « Throwable » et « Error » par « Exception »
"equals(Object obj)" and "hashCode()" should be overridden in pairs	9	Moyen	Régénérer les méthodes « equals » et « hashCode » avec l'IDE Eclipse. Il faut vérifier que les corrections n'entraînent pas d'anomalies dans Amalfi
Throwable.printStackTrace(..) should not be called	266	Simple	Utiliser l'API de logging pour tracer l'exception
"static final" arrays should be "private"	71	Simple	Transformer la visibilité à « private » et ajouter un accesseur
Floating point numbers should not be tested for equality	5	Simple	Utiliser Float.floatToRawIntBits
"BigDecimal(double)" should not be used	3	Simple	Remplacer new BigDecimal(double) par BigDecimal.valueOf(double)
"for" loop incrementers should modify the variable being tested in the loop's stop condition	14	Moyen	Il faut remplacer la mauvaise variable incrémentée par la bonne variable. Il faut cependant vérifier que le code répond bien à la fonction implémentée

Short-circuit logic should be used in boolean contexts	4	Simple	Utiliser && et pour plus de clarté
Identical expressions should not be used on both sides of a binary operator	10	Simple/ Moyen	Supprimer l'expression inutile ou modifier l'expression selon le cas métier
"equals(Object obj)" should be overridden along with the "compareTo(T obj)" method	66	Moyen	Implémenter la méthode « equals »
Classes should not be compared by name	58	Moyen	Transformer la comparaison des noms par l'utilisation du mot clé « instanceof »
Property values should be valid	61	Simple	défaut css dû à l'utilisation de valeur spécifique au navigateur. Il suffit de remplacer par la valeur standard ou ajouter l'unité. Nécessite cependant de tester l'affichage.
Exit methods should not be called	28	Simple	Vérifier que l'utilisation de cette méthode se fait dans un contexte approprié
Values passed to SQL commands should be sanitized	18	Complexe	Demande à modifier la construction de la requête SQL. Cependant le défaut peut être un faux positif
"Cloneables" should implement "clone"	16	Moyen	Il faut implémenter la méthode « clone ». Peut s'avérer complexe selon la classe (« deep copy » ou « shadow copy »)
Objects should not be created to be dropped immediately without being used	12	Simple	Supprimer la construction ou utiliser une méthode statique
"hashCode" and "toString" should not be called on array instances	9	Simple	Remplacer « array.toString() » par « Arrays.toString(array) » et « array.hashCode() » par « Arrays.hashCode(array) » Dans le cas d'Amalfi ce n'est pas critique car utilisé uniquement dans pour les logs
Servlets should never have mutable instance fields	8	Simple	Rendre le champ final
Thread.run() and Runnable.run() should not be called directly	7	Simple	Appeler « Thread.start »
Methods named "equals" should override Object.equals(Object)	5	Simple	Il suffit de renommer la méthode ou respecter la signature de la méthode « equals »
Interfaces should not have "public static" mutable fields	5	Simple	Ajouter la directive « final »
Non-serializable objects should not be stored in "HttpSessions"	4	Simple	Modifier la signature des méthodes pour imposer l'utilisation de classe dite « serialisable »
Conditions in related "if/else if" statements should not have the same condition	3	Simple	Retirer le code mort
Null should not be returned from a "Boolean" method	1	Simple	La nullité de l'objet retourné est semble-t-il une fonctionnalité souhaitée
Fields in a "Serializable" class should either be transient or serializable	777	Moyen à complexe	Transformer les champs non « serialisable » en « transient » ou les rendre « serialisable »
"public static" fields should always be constant	467	Simple	Transformer les champs en constantes ou rendre les champs privés

Exception handlers should preserve the original exception	1384	Simple	Passer l'exception originale dans le constructeur de la nouvelle exception
"==" and "!=" should be used instead of "=" and "!="	35	Simple	Remplacer == par === et != par !==
Variables should always be declared with "var"	2	Simple	Ajouter la déclaration « var »
Collection.isEmpty() should be used to test for emptiness	515	Simple	Remplacement de la méthode
Class variable fields should not have public accessibility	663	Simple	Changer l'accessibilité à « private » ou « protected »
"@Override" annotation should be used on any method overriding (since Java 5) or implementing (since Java 6) another one	1000 1	Simple	Peut être réalisé automatiquement par l'IDE Eclipse lors de la sauvegarde du fichier
Name of overqualified element should be removed		Moyen	Nécessite de modifier le CSS et éventuellement la page Html afin de supprimer l'utilisation du nom de l'élément
Over-specified selectors should be simplified	1816	Simple	Nécessite de modifier le css et vérifier le bon affichage de la page
Insufficient branch coverage by unit tests	1732	Complexe	Ajouter des tests unitaires pour les cas non couverts
Generic exceptions should never be thrown	1456	Moyen	Nécessite la création d'une exception spécifique et de ce fait change la signature de la méthode
Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList"	1448	Simple	Cependant cela peut entrainer des modifications en chaines en raison du changement de signature de la méthode
Method parameters, caught exceptions and foreach variables should not be reassigned	1434	Simple	Utiliser une variable temporaire
Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	1227	Moyen	Décomposer le flux de contrôle
Strings literals should be placed on the left side when checking for equality	1188	Simple	Intervertir les membres de l'égalité
Sections of code should not be "commented out"	1133	Simple	Nettoyage du code
Unused method parameters should be removed	1079	Simple	Mais change la signature de la méthode ce qui peut affecter beaucoup de code par rebond. La solution consistera pour les cas compliqués à dupliquer la méthode et à déprécier l'ancienne méthode
Methods should not be too complex	1064	Complexe	Remplacement du code de la méthode
IDs in selectors should be removed	778	Moyen	Retirer l'utilisation des identifiants dans les sélecteurs css
Duplicated blocks	773	Moyen	Il faut vérifier que les blocs sont bien des duplications et les supprimer

Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used	694	Simple	Transformation de « Vector » en « ArrayList » ou « LinkedList », « Hashtable » en « HashMap », « Stack » en « Deque », « StringBuffer » en « StringBuilder ». Peut entraîner des modifications en cascade en raison d'un changement de signature
Collapsible "if" statements should be merged	686	Simple	Fusionner le code
Attributes deprecated in HTML5 should not be used	609	Moyen	Nécessite de migrer les attributs vers HTML5
Useless parentheses around expressions should be removed to prevent any misunderstanding	560	Simple	Suppression des parenthèses
Subclasses that add fields should override "equals"	379	Complexe	Amalfi compare les objets et les processus métiers uniquement sur l'identifiant. Si une comparaison est faite entre deux versions d'un même objet, ils seront considérés comme égaux. Ces défauts peuvent être considérés comme des faux positifs
String literals should not be duplicated	3126	Simple	Dans le cas de chaîne de caractères représentant des énumérations métier remplacer la chaîne par une constante.

5 SYNTHÈSE DES RESULTATS DE L'AUDIT

L'application Amalfi est une application dont le code offre une qualité correcte légèrement en-deçà de la qualité moyenne des projets Open Source référencés sur le site de SonarQube.

Les classes les plus complexes se trouvent dans les sous projets AmalfiScTransverse et AmalfiScFront, en particulier dans les classes de traitement et les classes regroupant les règles de validation.

Le projet AmalfiSc concentre la plus grande partie du code. Le projet AmalfiWar détient la plus grande dette technique. Cependant ce projet ne contient que les définitions des pages HTML et du code Javascript. L'essentiel des défauts concerne les styles CSS et le code HTML. Le risque de dysfonctionnement de l'application reste donc faible. Seuls deux défauts majeurs Javascript devraient être corrigés : il s'agit de l'utilisation de « == » et « != » à la place de « === » et de « !== », et l'oubli du mot clé var lors de la déclaration d'une variable locale.

Pour le reste des projets, un grand nombre de défauts de qualité pourraient être aisément corrigés.

Astek souhaiterait également attirer l'attention sur la duplication des chaînes de caractères dans AMALFI. Ce défaut mineur référencé dans SonarQube sous le terme « String literals should not be duplicated » apparaît plus de 3000 fois. Dans AMALFI, ce défaut couvre plusieurs cas plus ou moins gênants. Le plus sensible est l'utilisation de chaînes de caractères pour définir des types d'objet (Type de charge, Type de mention, Type dossier, ...) et des états (Etat cadastre, Etat Dossier, ...). Le fait de ne pas utiliser de constantes diminue considérablement la lisibilité et augmente ainsi la possibilité de se tromper et d'engendrer des impacts dans le fonctionnement de l'application.

Astek tient à rappeler que l'analyse quantitative d'un projet à l'aide de SonarQube ne fournit qu'un aperçu de la qualité globale dudit logiciel.

6 PRECONISATIONS ET PLAN D'ACTIONS

Astek préconise de corriger, dans un premier temps, le premier défaut considéré comme bloquant par SonarQube à savoir « Throwable and Error should not be caught » ainsi qu'un lot de défauts critiques listés ci-dessous et susceptibles de provoquer des dysfonctionnements.

Liste des défauts sélectionnés :

Libellé du défaut	Nombre d'occurrences	Difficulté de correction	Action
Throwable and Error should not be caught	429	Simple	Remplacer « Throwable » et « Error » par « Exception »
Floating point numbers should not be tested for equality	5	Simple	Utiliser Float.floatToRawIntBits
"BigDecimal(double)" should not be used	3	Simple	Remplacer new BigDecimal(double) par BigDecimal.valueOf(double)
"for" loop incrementers should modify the variable being tested in the loop's stop condition	14	Moyen	Il faut remplacer la mauvaise variable incrémentée par la bonne variable. Il faut cependant vérifier que le code répond bien à la fonction implémentée
Values passed to SQL commands should be sanitized	18	Complexe	Demande à modifier la construction de la requête SQL. Cependant le défaut peut être un faux positif
Exit methods should not be called	28	Simple	Vérifier que l'utilisation de cette méthode se fait dans un contexte approprié

Remarque : Le second défaut bloquant « equals(Object obj)" and "hashCode()" should be overridden in pairs » n'a pas été retenu car si le risque d'un dysfonctionnement actuel d'Amalfi est faible sa correction pourrait entraîner des problèmes.

Dans un second temps, Astek propose de réaliser une étude sur les défauts restants afin de maîtriser les risques de corrections.

Identifiant	Libellé
ACT-01	Corriger les défauts susceptibles de créer des dysfonctionnements.
ACT-02	Améliorer la qualité globale de l'application en réalisant une étude sur les défauts restants de type bloquants, critiques et majeurs.

7 GLOSSAIRE

SQALE : (Software Quality Assessment based on Lifecycle Expectations) : méthode d'évaluation de la dette technique du code source, en utilisant les résultats de l'analyse des règles fournis.

Dette technique : par analogie une dette financière, représente le fait que lorsqu'on code de manière non optimale, on contracte une dette technique que l'on rembourse tout au long de la vie du projet sous forme de temps de développement de plus en plus long et de bugs de plus en plus fréquents.

Complexité cyclomatique : c'est le nombre de chemins linéairement indépendants qu'il est possible de suivre.

8 ANNEXES

8.1 Classification des règles

Etiquette	Explication
misra	Référencé dans le guide de développement pour le langage C par l'association MISRA (Motor Industry Software Reliability Association)
convention	Convention de codage
bad-practice	Mauvaise pratique
cert	Référencé dans le guide CERT
clumsy	Code maladroit
java8	Concerne Java 8
design	Concerne le design
error-handling	gestion des erreurs
unused	Code non utilisé
serialization	Concerne la sérialisation
obsolete	Utilisation de code obsolète
confusing	Code confus
multi-threading	Concerne le multi-threading
performance	Concerne la performance
bug	Défaut
junit	Concerne Junit
cwe	Référencé dans le guide Common Weakness Enumeration de l'association MITRE
pitfall	Problème
sql	Concerne SQL
security	Concerne la sécurité
owasp-a1,	OWASP Top Ten 2013 Category A1

owasp-a2	OWASP Top Ten 2013 Category A2
owasp-a6	OWASP Top Ten 2013 Category A6
denial-of-service	Peut permettre un déni de service
leak	Fuite de ressource
unpredictable	Code au comportement imprévisible
owasp-top10	Référencé dans le guide OWASP Top Ten 2013
sans-top25	SANS Top 25
spring	Concerne la librairie Spring
injection	Permet l'injection de code
struts	Concerne la librairie Struts
brain-overload	Trop compliqué
lock-in	Verrouillage vers une implémentation spécifique non souhaitable
browser-compatibility	Compatibilité des navigateurs
format	Défaut de formatage
html5	Compatibilité HTML5

8.2 Règles de codage Java

Défauts bloquants

Libellé	Etiquettes
" <code>.equals()</code> " should not be used to test the values of "Atomic" classes	bug
" <code>Double.longBitsToDouble</code> " should not be used for "int"	bug
" <code>equals(Object obj)</code> " and " <code>hashCode()</code> " should be overridden in pairs	bug, cert, cwe
" <code>equals(Object obj)</code> " should test argument type	bug
" <code>Iterator.hasNext()</code> " should not call " <code>Iterator.next()</code> "	bug
"Lock" objects should not be "synchronized"	clumsy, multi-threading
" <code>Object.wait(...)</code> " should never be called on objects that implement " <code>java.util.concurrent.locks.Condition</code> "	bug, pitfall
" <code>PreparedStatement</code> " and " <code>ResultSet</code> " methods should be called with valid indices	bug, sql
" <code>read</code> " and " <code>readLine</code> " return values should be used	bug
" <code>return</code> " statements should not occur in "finally" blocks	bug, cwe
" <code>runFinalizersOnExit</code> " should not be called	bug, cert, security
" <code>ScheduledThreadPoolExecutor</code> " should not have 0 core threads	bug
A "for" loop update clause should move the counter in the right direction	bug
Conditions should not unconditionally evaluate to "TRUE" or to "FALSE"	bug, cwe, misra
Methods " <code>wait(...)</code> ", " <code>notify()</code> " and " <code>notifyAll()</code> " should never be called on Thread instances	bug, multi-threading

Null pointers should not be dereferenced	bug, cert, cwe, owasp-a1, owasp-a2, owasp-a6, security
Reflection should not be used to check non-runtime annotations	bug
Resources should be closed	bug, cert, cwe, denial-of-service, leak, security
super.finalize() should be called at the end of Object.finalize() implementations	bug, cert, cwe
Synchronisation should not be based on Strings or boxed primitives	bug, cert
Throwable and Error should not be caught	cert, cwe, error-handling

Défauts critiques

Libellé	Etiquettes
"BigDecimal(double)" should not be used	bug, cert
"Calendars" and "DateFormats" should not be static	bug, multithreading
"Cloneables" should implement "clone"	bug
"compareTo" should not return "Integer.MIN_VALUE"	bug
"ConcurrentLinkedQueue.size()" should not be used	performance
"equals(Object obj)" should be overridden along with the "compareTo(T obj)" method	bug
"for" loop incrementers should modify the variable being tested in the loop's stop condition	bug
"hashCode" and "toString" should not be called on array instances	bug
"HttpServletRequest.getRequestSessionId()" should not be used	owasp-top10, security
"indexOf" checks should not be positive numbers	pitfall
"Object.wait(...)" and "Condition.await(...)" should always be called inside a "while" loop	bug, cert, multi-threading
"public static" fields should always be constant	cert, cwe, security
"ResultSet.isLast()" should not be used	performance, pitfall
"Serializable" inner classes of non-serializable classes should be "static"	bug, serialization
"static final" arrays should be "private"	cwe, security
"switch" statements should not contain non-case labels	misra, pitfall
"Threads" should not be used where "Runnable"s are expected	multi-threading, pitfall
"toString()" and "clone()" methods should not return null	bug, cwe
"URL.hashCode" and "URL.equals" should be avoided	performance
"wait(...)" should be used instead of "Thread.sleep(...)" when a lock is held	multi-threading, performance
"wait(...)", "notify()" and "notifyAll()" methods should only be called when a lock is obviously held on an object	bug, multi-threading
Classes should not be compared by name	bug, cwe
Collections should not be passed as arguments to their own methods	bug
Conditions in related "if/else if" statements should not have the same condition	bug, cert, pitfall, unused
Cookies should be "secure"	cwe, owasp-top10, security
Credentials should not be hard-coded	cwe, owasp-top10, sans-top25,

	security
Cryptographic RSA algorithms should always incorporate OAEP (Optimal Asymmetric Encryption Padding)	cwe, owasp-top10, security
Custom serialization method signatures should meet requirements	bug
Dissimilar primitive wrappers should not be used with the ternary operator without explicit casting	bug
Exception handlers should preserve the original exception	error-handling
Execution of the Garbage Collector should be triggered only by the JVM	unpredictable
Exit methods should not be called	cwe
Fields in a "Serializable" class should either be transient or serializable	bug, cwe, serialization
Floating point numbers should not be tested for equality	bug, misra
Identical expressions should not be used on both sides of a binary operator	bug, cert
IllegalMonitorStateException should not be caught	bug, multi-threading
Inappropriate "Collection" calls should not be made	bug
Instance methods should not write to "static" fields	bug, multi-threading
Interfaces should not have "public static" mutable fields	unpredictable
Ints and longs should not be shifted by more than their number of bits-1	bug
Invalid "Date" values should not be used	bug
JUnit assertions should not be used in "run" methods	junit, pitfall
JUnit test cases should call super methods	bug, junit
Locks should be released	bug, multi-threading
Loop conditions should be true at least once	bug
Methods named "equals" should override Object.equals(Object)	pitfall
Methods should not be named "hashCode" or "equal"	bug, pitfall
Neither "Math.abs" nor negation should not be used on numbers that could be "MIN_VALUE"	bug
Non-public methods should not be "@Transactional"	bug, spring
Non-serializable classes should not be written	pitfall, serialization
Non-serializable objects should not be stored in "HttpSessions"	bug
Null should not be returned from a "Boolean" method	pitfall
Objects should not be created to be dropped immediately without being used	bug
Printf-style format strings should not lead to unexpected behavior at runtime	bug, pitfall
Relational operators should be used in "for" loop termination conditions	bug, cert, cwe, misra
Servlets should never have mutable instance fields	bug, cert, multi-threading, struts
Short-circuit logic should be used in boolean contexts	bug
Switch cases should end with an unconditional "break" statement	cert, cwe, misra, pitfall
The non-serializable super class of a "Serializable" class must have a no-argument constructor	bug, serialization
The Object.finalize() method should not be called	cert, cwe, security

The Object.finalize() method should not be overridden	cert, unpredictable
The value returned from a stream read should be checked	bug
Thread.run() and Runnable.run() should not be called directly	cert, cwe, multi-threading
Throwable.printStackTrace(...) should not be called	error-handling
Values passed to SQL commands should be sanitized	cwe, hibernate, injection, owasp-top10, sans-top25, security, sql
Values should not be uselessly incremented	bug

Défauts majeurs

Libellé	Etiquettes
"@Override" annotation should be used on any method overriding (since Java 5) or implementing (since Java 6) another one	bad-practice
"compareTo" results should not be checked for specific values	bug
"entrySet()" should be iterated when both the key and value are needed	performance
"finalize" should not set fields to "null"	clumsy, performance
"FIXME" tags should be handled	
"for" loop stop conditions should be invariant	misra, pitfall
"instanceof" operators that always return "true" or "false" should be removed	bug, cwe
"Iterator.next()" methods should throw "NoSuchElementException"	bug
"java.lang.Error" should not be extended	error-handling
"Object.finalize()" should remain protected (versus public) when overriding	cert, cwe, security
"readObject" should not be "synchronized"	confusing
"StringBuilder" and "StringBuffer" should not be instantiated with a character	pitfall
"switch case" clauses should not have too many lines	brain-overload
"switch" statements should end with a "default" clause	cert, cwe, misra
"switch" statements should not have too many "case" clauses	brain-overload
"toString()" should never be called on a String object	clumsy, pitfall
A field should not duplicate the name of its containing class	brain-overload
Assignments should not be made from within sub-expressions	bug, cwe, misra
Boxing and unboxing should not be immediately reversed	clumsy
Branches should have sufficient coverage by unit tests	bad-practice
Case insensitive string comparisons should be made without intermediate upper or lower casing	clumsy
Child class members should not shadow parent class members	confusing
Class names should not shadow interfaces or superclasses	pitfall
Class variable fields should not have public accessibility	cwe
Classes extending java.lang.Thread should override the "run" method	multi-threading, pitfall
Classes from "sun.*" packages should not be used	lock-in, pitfall
Classes named like "Exception" should extend "Exception" or a subclass	convention, pitfall
Classes should not be empty	clumsy
Classes should not be too complex	brain-overload
Classes that override "clone" should be "Cloneable" and call "super.clone()"	cwe

Classes with only "static" methods should not be instantiated	clumsy
Collapsible "if" statements should be merged	clumsy
Collection.isEmpty() should be used to test for emptiness	clumsy
Collections.emptyList(), emptyMap() and emptySet() should be used instead of Collections.EMPTY_LIST, EMPTY_MAP and EMPTY_SET	obsolete, pitfall
Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	brain-overload
Declarations should use Java collection interfaces such as "List" rather than specific implementation classes such as "LinkedList"	bad-practice
Deprecated elements should have both the annotation and the Javadoc tag	bad-practice
Empty arrays and collections should be returned instead of null	cert
Enumeration should not be implemented	obsolete
Exception classes should be immutable	error-handling
Exception types should not be tested using "instanceof" in catch blocks	error-handling
Exceptions should not be thrown in finally blocks	error-handling
Expressions should not be too complex	brain-overload
Future keywords should not be used as names	obsolete, pitfall
Generic exceptions should never be thrown	cwe, error-handling
Generic wildcard types should not be used in return parameters	pitfall
Inner class calls to super class methods should be unambiguous	pitfall
IP addresses should not be hardcoded	cert, security
Labels should not be used	confusing
Lambdas and anonymous classes should not have too many lines	java8
Lambdas containing only one statement should not nest this statement in a block	java8
Local variables should not shadow class fields	pitfall
Loops should not contain more than a single "break" or "continue" statement	brain-overload
Math operands should be cast before assignment	bug, cwe, sans-top25
Method parameters, caught exceptions and foreach variables should not be reassigned	misra, pitfall
Methods should not be empty	bug
Methods should not be too complex	brain-overload
Methods should not have too many parameters	brain-overload
Nested "enum"s should not be declared static	clumsy
Nested blocks of code should not be left empty	bug
Nested code blocks should not be used	bad-practice
Non-constructor methods should not have the same name as the enclosing class	pitfall
Object.finalize() should not be overloaded (by adding method parameters)	pitfall
Objects should not be created only to "getClass"	performance
Octal values should not be used	misra, pitfall
Only static class initializers should be used	pitfall
Package declaration should match source file directory	pitfall
Parsing should be used to convert "Strings" to primitives	performance

Primitive wrappers should not be instantiated only to perform a "toString" conversion	clumsy
Primitives should not be boxed just for "String" conversion	performance
Public methods should throw at most one checked exception	error-handling
Sections of code should not be "commented out"	misra, unused
Silly bit operations should not be performed	bug
Silly equality checks should not be made	bug, unused
Silly math should not be performed	clumsy
Source files should not have any duplicated blocks	pitfall
Standard outputs should not be used directly to log anything	bad-practice
Strings literals should be placed on the left side when checking for equality	bad-practice
Subclasses that add fields should override "equals"	bug
Synchronized classes Vector, Hashtable, Stack and StringBuffer should not be used	multi-threading, performance
TestCases should contain tests	junit, unused
Try-catch blocks should not be nested	confusing
Unused labels should be removed	misra, unused
Unused local variables should be removed	unused
Unused method parameters should be removed	misra, unused
Unused private fields should be removed	unused
Unused private method should be removed	unused
Unused type parameters should be removed	unused
Useless parentheses around expressions should be removed to prevent any misunderstanding	confusing
Utility classes should not have public constructors	design

Défauts mineurs

Libellé	Etiquettes
"switch" statements should have at least 3 "case" clauses	misra
A close curly brace should be located at the beginning of a line	convention
Array designators "[]" should be located after the type in method signatures	convention
Array designators "[]" should be on the type, not the variable	convention
Class names should comply with a naming convention	convention
Constant names should comply with a naming convention	convention
Constants should not be defined in interfaces	bad-practice
Empty statements should be removed	cert, misra, unused
Field names should comply with a naming convention	convention
Fields in non-serializable classes should not be "transient"	serialization, unused
Interface names should comply with a naming convention	convention
Literal boolean values should not be used in condition expressions	clumsy
Local variable and method parameter names should comply with a naming convention	convention

Local Variables should not be declared and then immediately returned or thrown	clumsy
Loggers should be "private static final" and should share a naming convention	convention
Long suffix "L" should be upper case	convention
Method names should comply with a naming convention	convention
Modifiers should be declared in the correct order	convention
Overriding methods should do more than simply call the same method in the super class	clumsy
Package names should comply with a naming convention	convention
Parentheses should be removed from a single lambda input parameter when its type is inferred	java8
Public constants should be declared "static final" rather than merely "final"	convention
Redundant casts should not be used	clumsy
Return of boolean expressions should not be wrapped into an "if-then-else" statement	clumsy
Statements should be on separate lines	convention
String literals should not be duplicated	design
String.valueOf() should not be appended to a String	clumsy
Tabulation characters should not be used	convention
The default unnamed package should not be used	convention
The members of an interface declaration or class should appear in a pre-defined order	convention
Throws declarations should not be redundant	error-handling
Type parameter names should comply with a naming convention	convention
Useless imports should be removed	unused

Informations

Libellé	Etiquettes
"TODO" tags should be handled	
Deprecated code should be removed eventually	obsolete

8.3 Règles de codage Javascript

Défauts bloquants

Libellé	Etiquettes
"NaN" should not be used in comparisons	bug
A "for" loop update clause should move the counter in the right direction	bug
Results of operations on strings should not be ignored	bug
Short-circuit logic should be used to prevent null pointer dereferences in conditionals	bug
Trailing commas should not be used	cross-browser

Défauts critiques

Libellé	Etiquettes
"delete" should not be used on arrays	bug
"eval" and "arguments" should not be bound or assigned	bug
"for" loop incrementers should modify the variable being tested in the loop's stop condition	bug
"future reserved words" should not be used as identifiers	lock-in, pitfall
"new" operators should be used with functions	bug
"switch" statements should not contain non-case labels	misra, pitfall
Code should not be dynamically injected and executed to prevent Eval Injection vulnerability	cwe, owasp-a3, security
Debugger statements should not be used	cwe, security, user-experience
Function argument names should be unique	pitfall
Function call arguments should not start on new line	pitfall
Function calls should not pass extra arguments	bug, cwe, misra
Identical expressions should not be used on both sides of a binary operator	bug, cert
Local variables should not shadow "undefined"	pitfall
Non-empty statements should have at least one side-effect	bug, cwe, unused
Property names should not be duplicated within an object literal	bug, pitfall
Related "if/else if" statements and "cases" in a "switch" should not have the same condition	bug, cert, pitfall, unused
Relational operators should be used in "for" loop termination conditions	bug, cert, cwe, misra
Setters should not return values	bug
Switch cases should end with an unconditional "break" statement	cert, cwe, misra, pitfall
The global "this" object should not be used	pitfall
Unary operators "+" and "-" should not be used with objects	bug
Values should not be uselessly incremented	bug

Défauts majeurs

Libellé	Etiquettes
"===" and "!== " should be used instead of "==" and "!="	bug

"alert(...)" should not be used	cwe, security, user-experience
"FIXME" tags should be handled	JavaScript
"for...in" loops should filter properties before acting on them	bug
"switch" statements should end with a "default" clause	cert, cwe, misra
"with" statements should not be used	bug
Array and Object constructors should not be used	bug
Bitwise operators should not be used	pitfall
Collapsible "if" statements should be merged	clumsy
Console logging should not be used	owasp-a6, security
Constructor functions should not be called purely for side-effects	pitfall
Control flow statements "if", "for", "while", "switch" and "try" should not be nested too deeply	brain-overload
Dead Stores should be removed	bug, cert, cwe, unused
Expressions should not be too complex	brain-overload
Files should not have too many lines	brain-overload
Function declarations should not be made within blocks	cross-browser, user-experience
Functions should not be defined inside loops	bug
Functions should not be too complex	brain-overload
Functions should not have too many lines	brain-overload
Functions should not have too many parameters	brain-overload
HTML-style comments should not be used	bug
Internet Explorer's conditional comments should not be used	cross-browser
Jump statements should not be followed by other statements	cert, cwe, misra, unused
Loops should not contain more than a single "break" or "continue" statement	brain-overload
Multiline string literals should not be used	bad-practice
Named function expressions should not be used	cross-browser, user-experience
Nested blocks of code should not be left empty	bug
Octal values should not be used	cert, misra, pitfall
Only "while", "do" and "for" statements should be labelled	pitfall
Sections of code should not be "commented out"	misra, unused
Two branches in the same conditional structure should not have exactly the same implementation	bug
Unused function parameters should be removed	misra, unused
Unused local variables should be removed	unused
Useless "if(true) {...}" and "if(false){...}" blocks should be removed	bug, cwe, misra, security
Variables and functions should not be redeclared	bug, pitfall
Variables should always be declared with "var"	pitfall
Variables should be declared before they are used	pitfall
Variables should not be self-assigned	bug, cert

Wrapper objects should not be used for primitive types	pitfall
--	---------

Défauts mineurs

Libellé	Etiquettes
"switch" statements should have at least 3 "case" clauses	misra
A "while" loop should be used instead of a "for" loop	clumsy
Each statement should end with a semicolon	convention
Function names should comply with a naming convention	convention
Lines should not end with trailing whitespaces	convention
Literal boolean values should not be used in condition expressions	clumsy
Return of boolean expressions should not be wrapped into an "if-then-else" statement	clumsy
Statements should be on separate lines	convention
Tabulation characters should not be used	convention

Informations

Libellé	Etiquettes
"strict" mode should be used with caution	cross-browser, user-experience
"TODO" tags should be handled	
Comments should not be located at the end of lines of code	convention

8.4 Règles de codage Web

Libellé	Etiquettes	Niveau
"" and "<dt>" item tags should be in "", "" or "<dl>" container tags	bug	critique
"title" should be present in all pages	user-experience	majeur
 and tags should be used instead of and <i>	accessibility	majeur
A <!DOCTYPE> declaration should appear before the <html> tag	user-experience	majeur
Attributes deprecated in HTML5 should not be used	html5, obsolete	majeur
Elements deprecated in HTML5 should not be used	html5, obsolete, user-experience	majeur
Favicons should be used in all pages	user-experience	majeur
Files should not have too many lines	brain-overload	majeur
Flash animations should be embedded using both the <object> and <embed> tags	cross-browser	majeur
Images tags and buttons should have an "alt" attribute	accessibility	majeur
Links should not directly target images	accessibility, user-experience	majeur
Meta tags should not be used to refresh the page nor for redirection	user-experience	majeur
Sections of code should not be "commented out"	misra, unused	majeur
Server-side image maps ("ismap" attribute) should not be used	accessibility	majeur

8.5 Règles de codage CSS

Défauts critiques

Libellé	Etiquettes
"!important" annotation should be placed at the end of the declaration	bug
CSS parser failure	bug
Property values should be valid	bug
Stylesheets should not "@import" too many other sheets	browser-compatibility, bug
Stylesheets should not contain too many selectors	browser-compatibility, bug, design

Défauts majeurs

Libellé	Etiquettes
"!important" annotation should not be used	design, pitfall
"@import" rule should not be used	performance
BOM should not be used for UTF-8 files	pitfall
Box model size should be carefully reviewed	pitfall
Deprecated Internet Explorer static filters should be removed	browser-compatibility
Deprecated system colors should not be used	browser-compatibility, convention, pitfall
Duplicated background images should be removed	design, performance
Duplicated properties should be removed	pitfall
Each declaration should end with a semicolon	convention, pitfall
Empty declarations should be removed	pitfall
Empty rules should be removed	pitfall
Experimental properties should not be used	browser-compatibility, convention
Gradient definitions should be set for all vendors	browser-compatibility
IDs in selectors should be removed	design
Missing vendor prefixes should be added to experimental CSS properties	browser-compatibility
Name of overqualified element should be removed	design, performance
Obsolete properties and properties not on W3C Standards track should not be used	browser-compatibility
Over-specified selectors should be simplified	design
Properties that do not work with the "display" property should be removed	performance, pitfall
Regular expression like selectors should not be used	performance
Shorthand properties should be used whenever possible	convention, performance
Standard properties should be specified along with vendor-prefixed properties	browser-compatibility
Star hack should not be used	convention
Stylesheets should not contain too many rules	design, performance
The number of web fonts should be reduced	performance
Underscore hack should not be used	convention

Units for zero length values should be removed	convention, performance
Universal selector should not be used as key part	performance
Unknown CSS @-rules should be removed	pitfall
Unknown CSS functions should be removed	pitfall
Unknown CSS properties should be removed	pitfall

Défauts mineurs

Libellé	Etiquettes
Leading zeros should be removed	convention
Lines should not be too long	convention
Properties, functions and variables should be lower case	convention
Selectors should follow a naming convention	convention
Source code should comply with formatting standards	format
Tabulation characters should not be used	convention
There should be one single declaration per line	format

Informations

Libellé	Etiquettes
"FIXME" tags should be handled	
"NOSONAR" tags should not be used to switch off issues	
"TODO" tags should be handled	

8.6 Règles de codage XML

Libellé	Etiquettes	Niveau
XML files containing a prolog header should start first with "<?xml" characters		Majeur
Newlines should follow each element		Mineur
Source code should be indented consistently	convention	Mineur
Tabulation characters should not be used	convention	Mineur

FIN DU DOCUMENT