UNIVERSITÉ DU QUÉBEC À MONTRÉAL

MÉMOIRE DE SYNTHÈSE SUR LA MAINTENABILITÉ D'UN LOGICIEL

JENKINS

MGL7460-90 – RÉALISATION & MAINTENANCE DE LOGICIELS

VERSION FINALE

PAR LILIAN OLOIERI - OLOL21017102

HIVER 2020

Introduction

Jenkins est un logiciel libre, multi plateforme, d'intégration et de livraison continue (CI/CD), ce qui permet aux développeurs de créer et tester les projets logiciels en continue, d'intégrer plus facilement les modifications au projet et facilite aux utilisateurs d'obtenir une nouvelle version du logiciel. Il vous fournissent des moyens puissants pour définir vos pipelines de construction et en s'intégrant à un grand nombre de technologies de test et de déploiement. A ce jour, c'est un application principale de type open-source pour cette tâche (vs Travis, Azure DevOps).

L'application a été fondée en 2006. L'auteur principal et fondateur du projet est Kohsuke Kawaguchi - (koshuke). Le nom initial du projet était Hudson, qui a été ensuite transformé dans Jenkins. Le premier commit dans Git date du 5 novembre 2006. Donc on a fait un analyse de développement du logiciel à partir de cette date.

Dimension "Équipe De Développement"

Qui Sont Les Développeurs Principaux Du Projet ?

Suite à l'analyse des résultats de nombre des commits par développeur on a sorti 10 principaux participants au projet. On a du faire la somme par nom et nom utilisateur (les logs de git contient les deux).

Nom	Commits	%	Première commit	Dernière commit
Kohsuke Kawaguchi	17033	57.96	2006-11-05	2020-04-05
Jesse Glick	7509	25.55	2006-11-20	2020-03-31
Daniel Beck	1277	4.35	2013-07-18	2020-04-07
Oleg Nenashev	1249	4.24	2013-08-12	2020-04-08
Olivier Lamy	1097	3.73	2010-01-19	2019-11-08
Seiji Sogabe	786	2.67	2008-05-08	2015-03-27
Stephen Connolly	663	2.25	2007-05-16	2018-12-04
Christoph Kutzinski	620	2.1	2010-06-26	2013-08-18
mindless - alainharder	593	2.01	2008-12-16	2011-08-18
Oliver Gondza	582	1.98	2012-11-22	2020-03-19

L'équipe De Développement Est-elle Stable ?

Kohsuke Kawaguchi et Jesse Glick sont les deux principaux auteurs qui ont mis les bases du projet en 2006, et sont resté actives jusqu'à présent.

Ces deux, avec le proéminence de Kohsuke Kawaguchi, ont été les principaux commiteurs jusqu'à la fin de l'année 2008. Le troisième, Visizikov, a commencé à travailler avec eux, mais a cessé ces activités peu après.

Comment Les Développeurs Communiquent Entre Eux ?

Côté communication, la communauté Jenkins est assez bien organisée. Il y a des cannaux divisés par sujet. L'équipe de développement organise des rencontres prédéfinies et ouvertes pour ses membres toutes les deux semaines. J'ai bien apprécié la quantité de documentation sur tous les aspects du système.
J'ai mis un question dans leur chat et j'ai reçu une réponse dans 10 minutes.

mydroid1 @mydroid1_gitlab
17:23
Hi, sorry I am new on jenkins dev ... I'am trying to buid core in intelliJ + maven , but without skipping tests .... how to use light-test option, please?
Alex Earl @slide
17:33
HI @mydroid1_gitlab that is probably best asked in the normal jenkins gitter chat
https://gitter.im/jenkinsci/jenkins

https://wiki.jenkins.io/#recently-viewed **https://jenkins.io/chat/**

Dimension "Exigences"

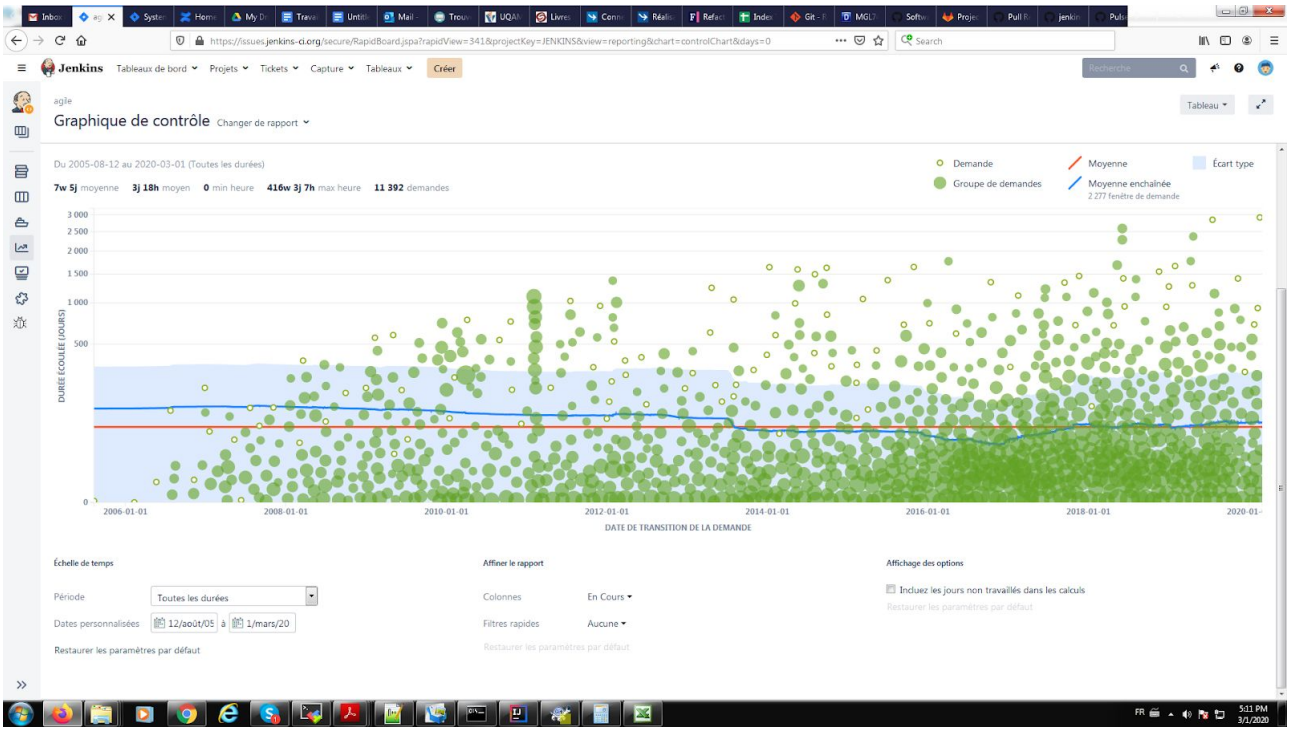
Comment Le Code Source Est-il Tracé Aux Exigences ?

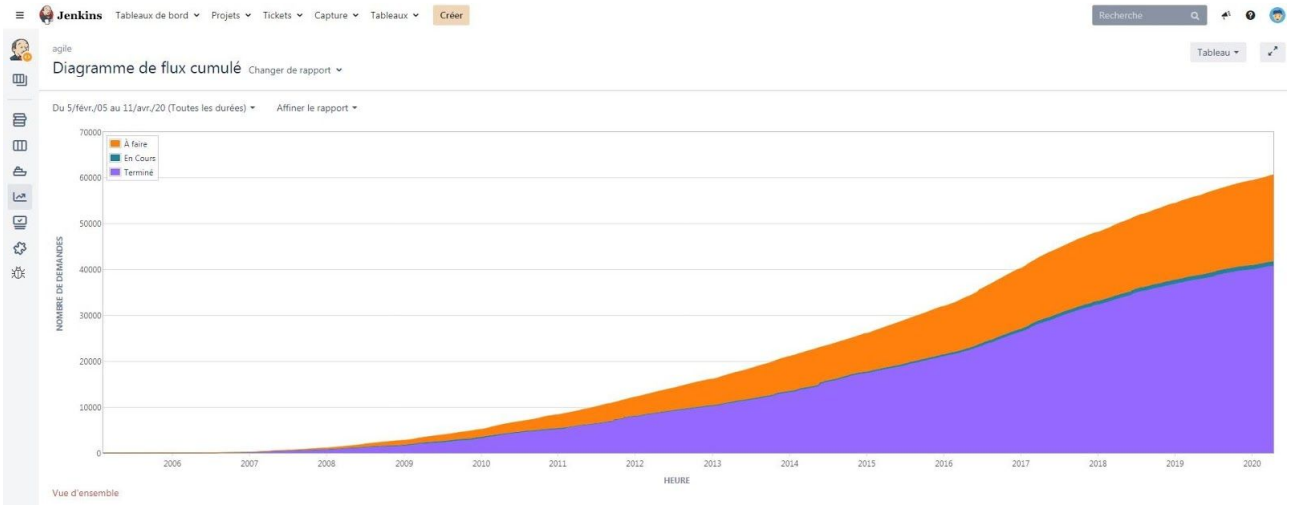
L'équipe de Jenkins utilise un JIRA assez bien configuré. Je me suis fait un compte pour pouvoir extraire des données et analyser leur dynamique.

https://issues.jenkins-ci.org/secure/Dashboard.jspa

Quelle Est La Vélocité De L'équipe De Développement Sur La Maintenance Corrective ? Quel Est Le Volume D'exigence Traité Dans Le Projet ?

Pour la toute la période d'existence du projet ils ont traité 11 392 demandes, avec un moyen de 7 semaines 5 jours par demande, avec un écart type plus grand de 223,5 jours et plus petit de 88.6 jours , que c'est assez bien pour un équipe open source.





Dimension "Architecture Logicielle"

Quelles Sont Les Composantes Principales De L'application ?

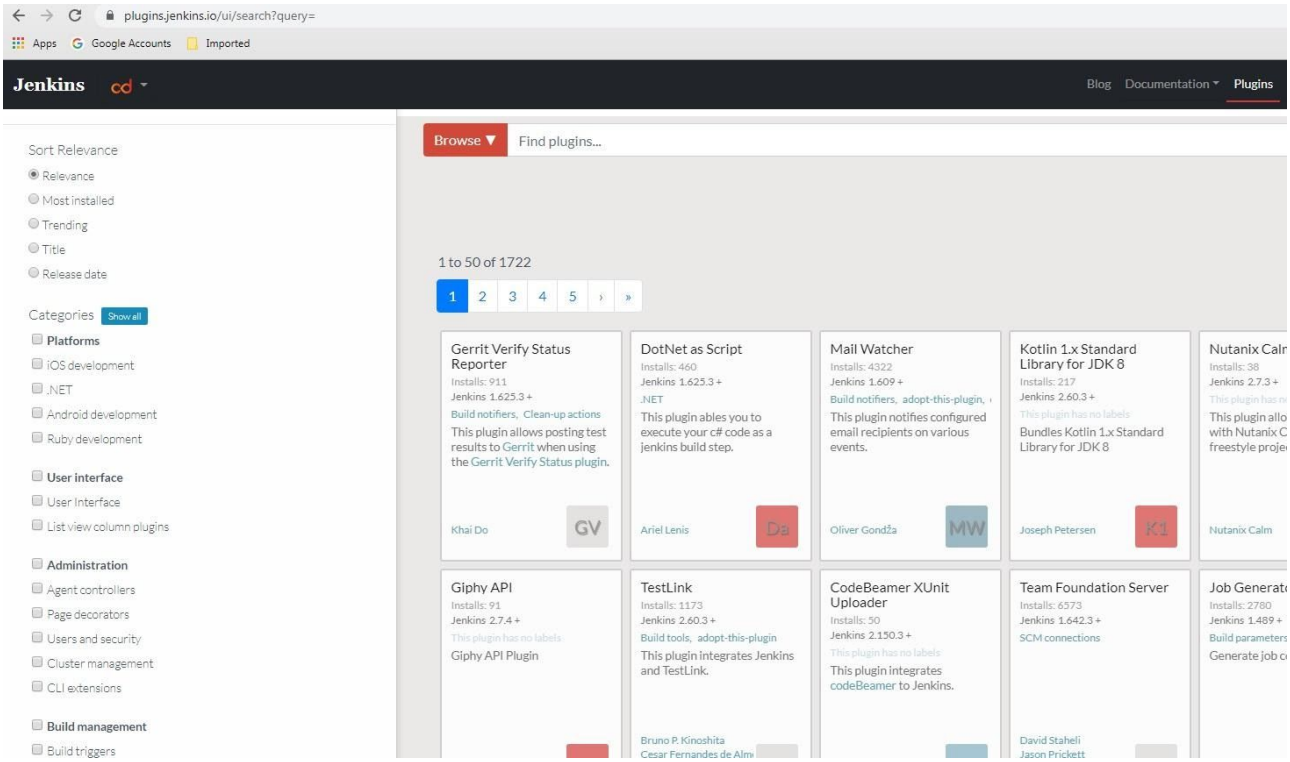
Il y a trois composantes principales de l'application : Module Core, module client et module web. Il y a aussi un module spéciale - test, qui contient les tests fonctionnels pour le module core. J'en trouvé un autre module pour client écrit récemment avec GO.
https://github.com/jenkins-zh/jenkins-cli

C'est un point en plus pour la modularité, vue que le client est interchangeable.

Partie gestion des requêtes dans la partie web est basé sur Stapler qui est un fruit du travail du même Kohsuke Kawaguchi
Les vues sont écrites en Jelly ou Groovy et peuvent être composées d'un certain nombre de vues partielles différentes (ou fragments de vues).
Coté infrastructure Jenkins est basé sur plusieurs services qui peuvent être complétés avec les données provenant des plugins (autre point pour la modularité).
https://jenkins.io/projects/infrastructure/

Comment Est Gérée La Modularité De L'application ?

L'architecture du Jenkins lui permet d'être très extensible et modulaire. Il permet l'ajout des nouveaux fonctionnalités avec l'ajout des nouveaux plugins. Même à l'intérieur du module core, les classes sont réparties de façon à séparer différentes fonctionnalités. Il y a à présent une très bonne documentation sur le développement des plugins pour Jenkins. ce que a donner ces fruits. A ce jour Jenkins a 1722 plugins pour différents besoins d'intégration.



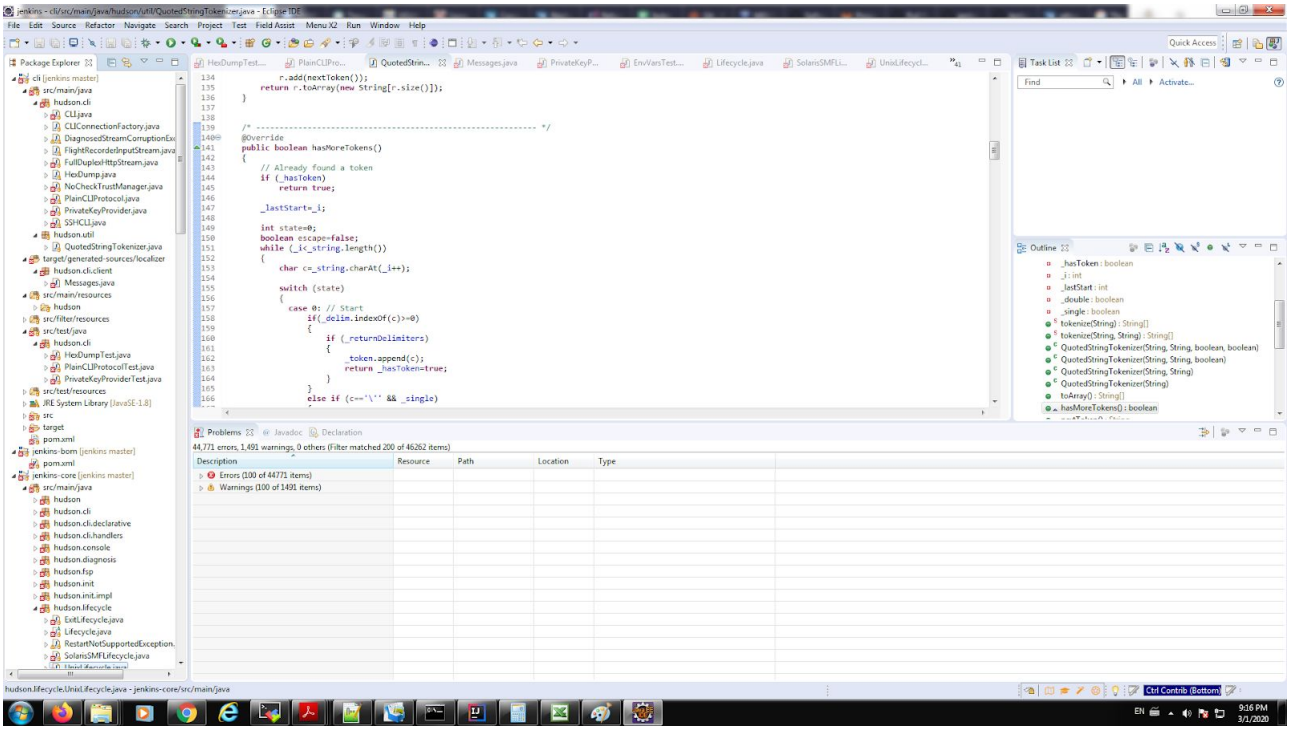
Comment Est Documentée L'architecture De L'application ?

La documentation sur l'architecture de Jenkins est un sujet ouvert. Il y a des lacunes sur différents points de architectures, mais j'en trouvés des discussions récents dans la communauté sur les taches conséquentes.
https://jenkins.io/doc/developer/book/

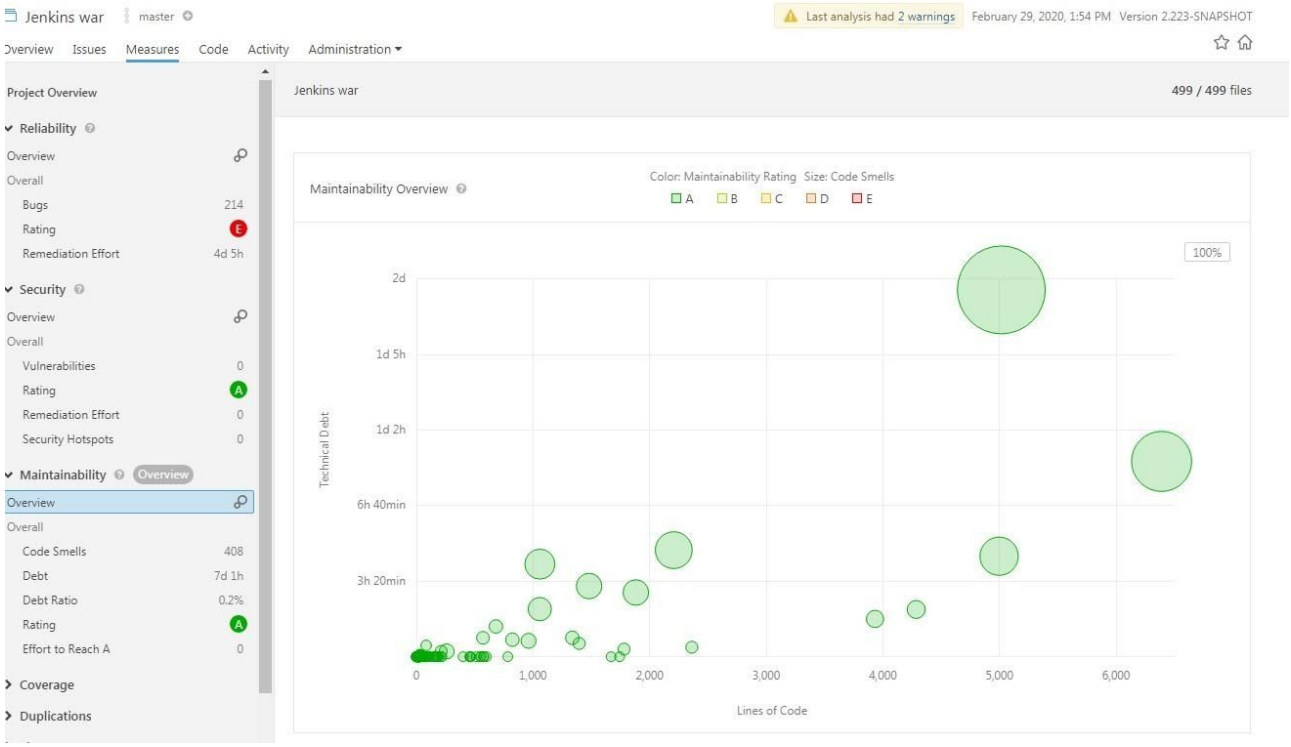
Dimension "Code Source"

Comment Qualifiez Vous La Qualité Du Code Source ?

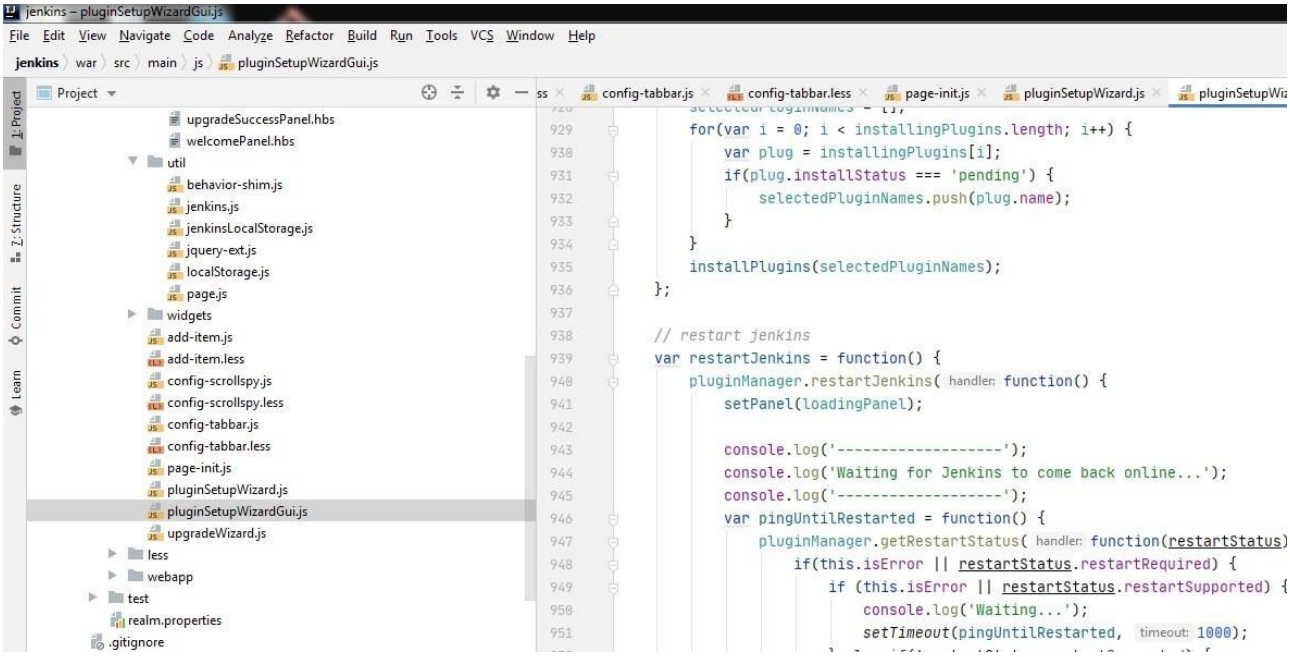
Même si j'en trouve des places ou il faudra du refactoring (noms des variables moins intuitives, méthodes trop longs), de façon général la qualité du code est assez bonne à mon avis. Ce qui est confirmé par les données sorties dans SonarQube. Il y a même un analyse Findbugs qui se fait au chaque build(intégré dans pom.xml).



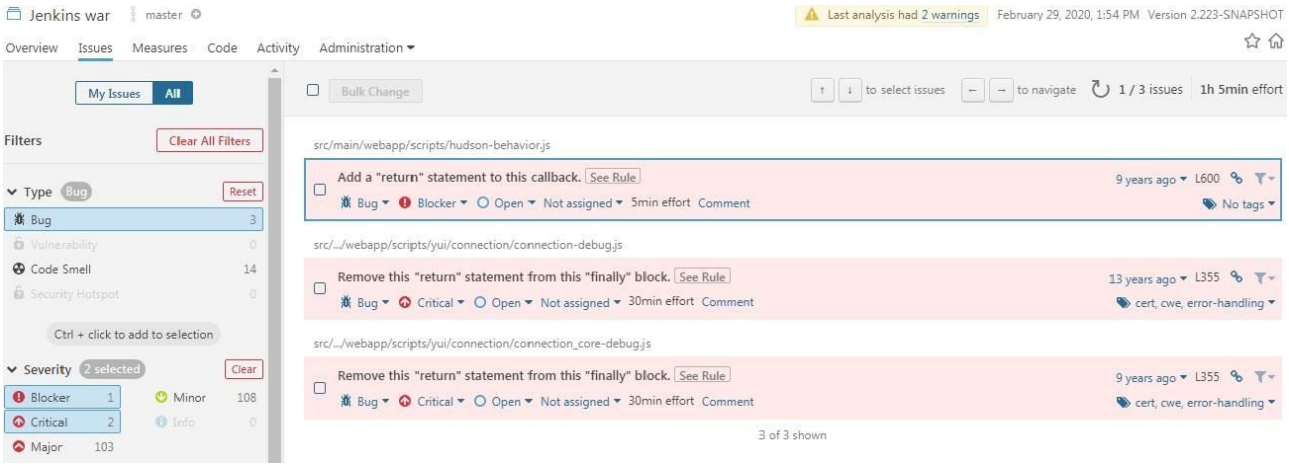
Analyse SonarQube - Module war - contient js, css etc.
Dans Sonar ce module a un cote **A** pour la maintenabilité, confirmé par mon analyse du code.



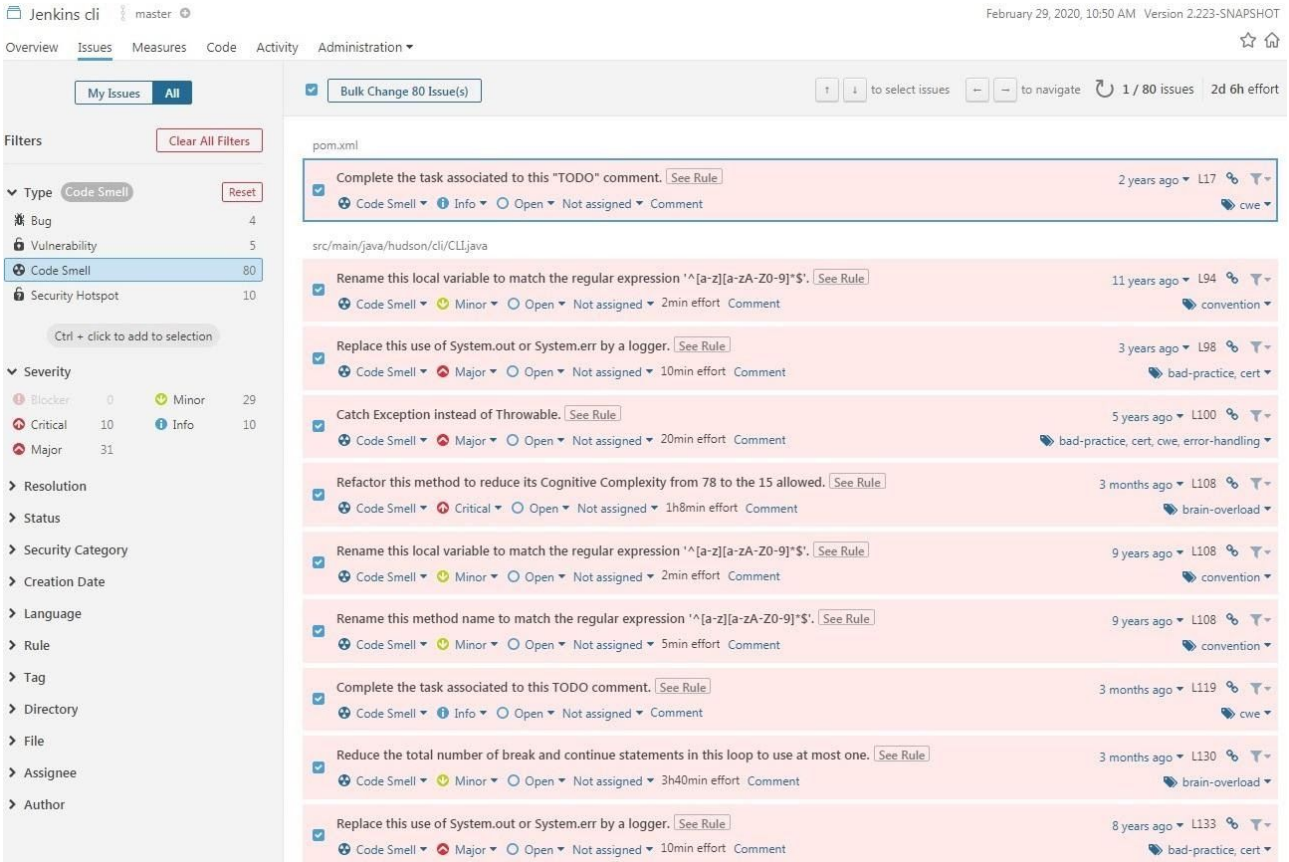
Le code js est assez propre et bien écrit, a exception de quelque console.log. P-e sera bien d'en forcer l'utilisation des namespaces et 'use strict', mais en générale est d'un qualité acceptable. On note aussi l'utilisation de Node.js dans ce module.



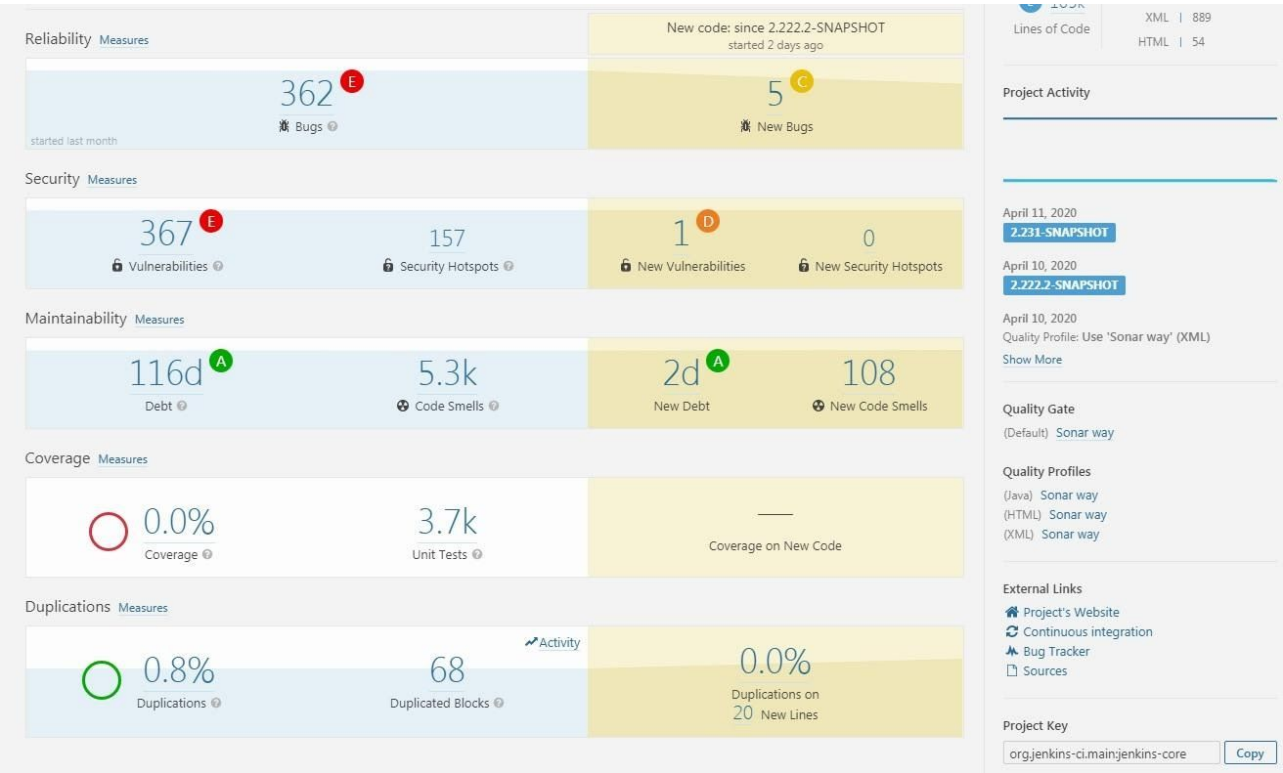
J'ai analysé la cote **E** pour les bugs et a mon avis n'est pas confirmé. Les 211 de 214 'bugs' represent des erreurs mineurs relié a deprecated - Remove this deprecated "tt" element. Il y a seulement 3 vrais erreurs:



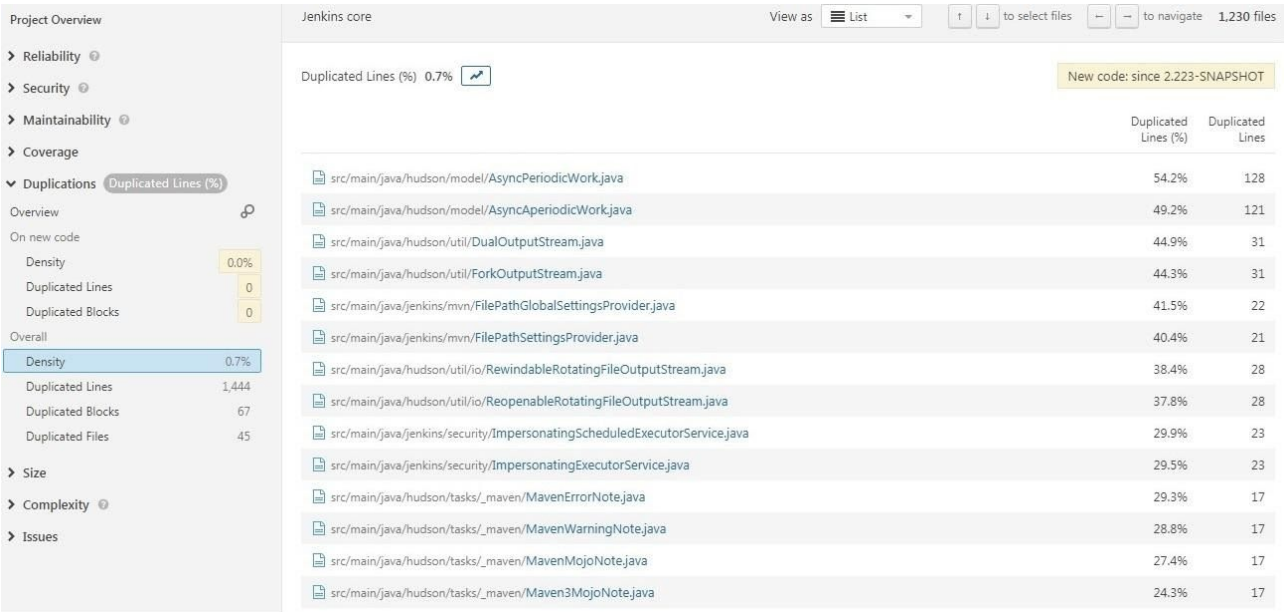
Analyse SonarQube - Module client - contient js, css etc. Je ne confirm pas la cote **A** pour la maintenabilité du ce module. Un chance qu' il n'y a pas trop grand. Seulement 9 fichiers java. Pourtant pour ces 9 on a 80 'code smells' et analyse du code le confirme - code a mon avis assez moche. En plus j'ai vu d'autres comme classes trop grands, méthodes trop grands (confirmé par analyse codeMR), seulement quelques tests d'intégration et pas des tests unitaires.



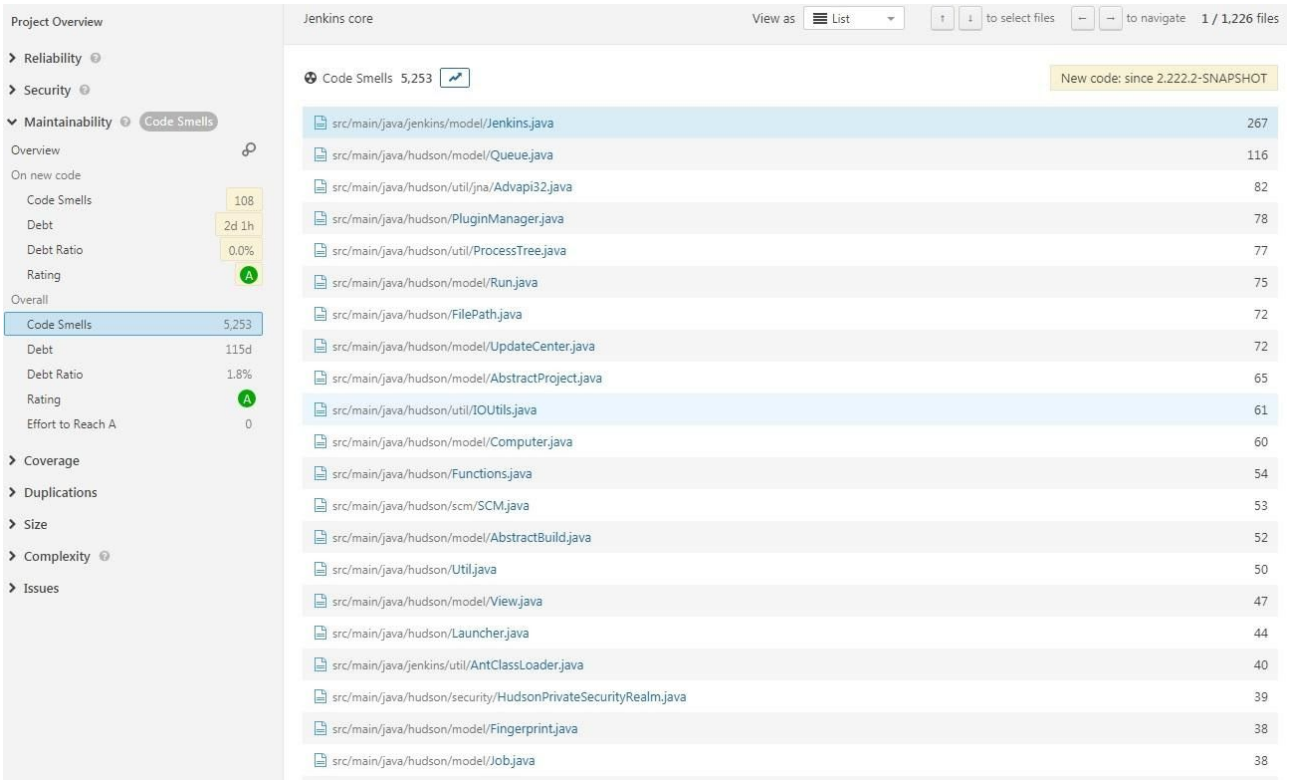
Analyse SonarQube - Module core



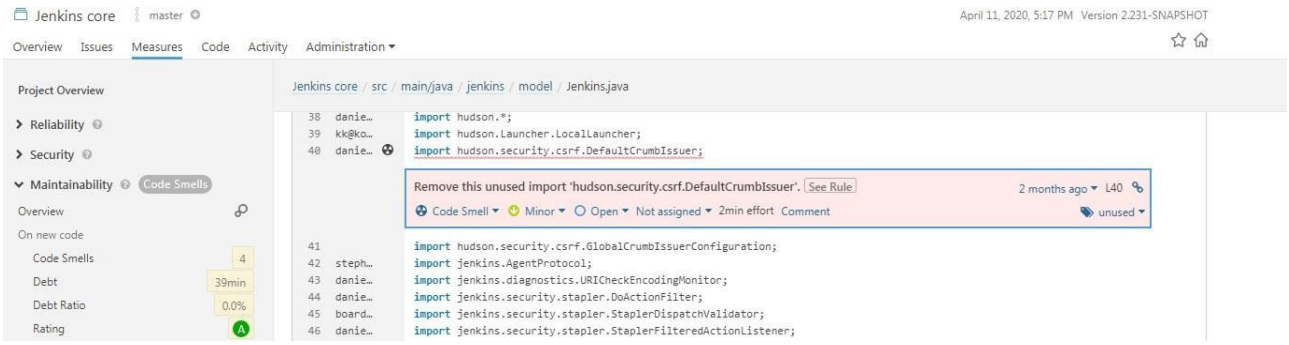
Le pourcentage de 0.8% pour la duplication est bien justifié. Si analyser les classes en question, on vois clairement un duplication des méthodes entre AsyncPeriodicWork.java et AsyncAPeriodicWork.java ou DualOutputStream.java et ForkOutputStream.java, qui pourrait etre gerer simplement avec inheritance ou encore mieux avec composition.



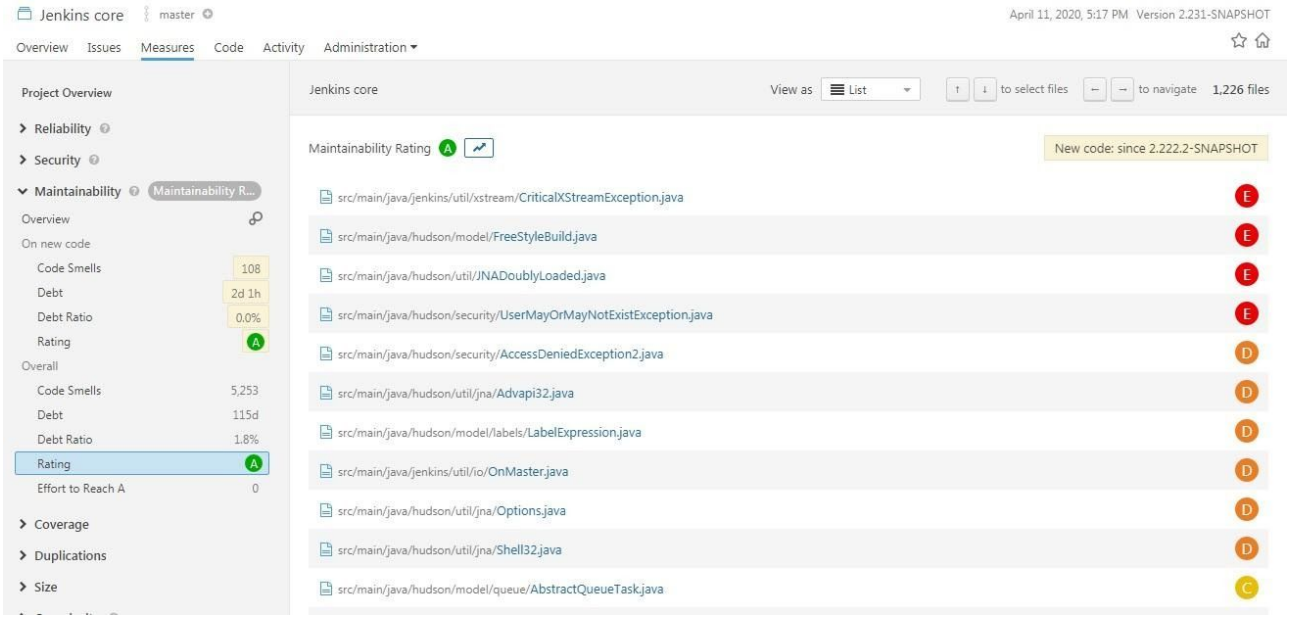
Le module core prends un cote A pour la maintenance, avec 5227 ‘code smells’ sur le code.



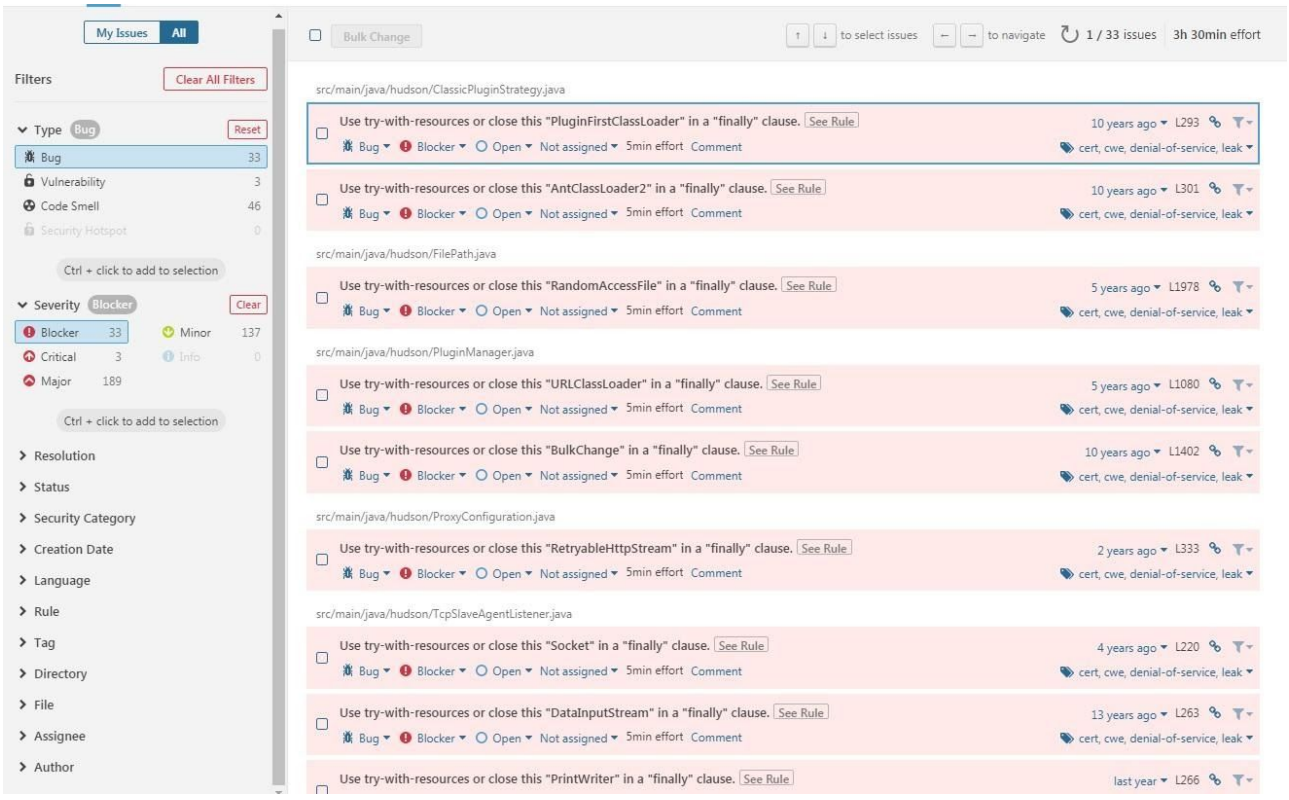
Analyse des dernières confirme la côte, vue que dans la plupart des cas on a des choses non critique et simple a régler.



Les plus compliqués sont quelques classes avec un hiérarchie d'inhérence plus grands que 5.



Côté fiabilité on a un **E** avec 362 problème sur un total dès 1226 classes java. La côte est confirmé suite à l'analyse du code source.



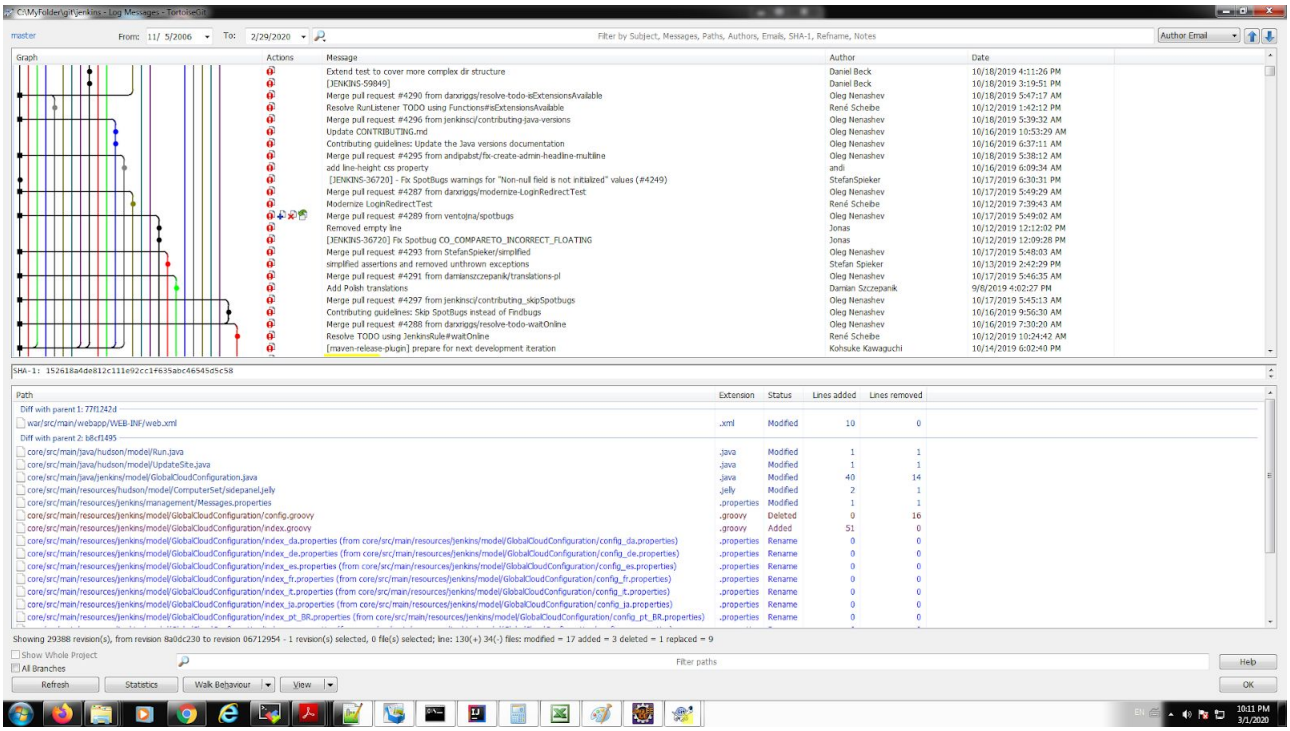
Analyse FindBugs - Les rapports de FindBugs sont propre. Se vois qu l'équipe les utilise pour corriger les défauts du système.

Quels Outils De Construction (P.-ex. Make, Maven) Sont Utilisés ?

Maven est utilisé comme projet outil de construction. L'utilisation est assez facile et intuitive … ce qui montre le point sur la facilité de construction du projet - un des points importants sur la maintenabilité. Le projet s'installe facilement dans IntelliJ, par contre j'ai encore des petits soucis sur Eclipse.

Quel Modèle De Branche Est Utilisé Dans Le Développement Du Projet ?

Au début du projet l'équipe utilisait plutôt un model 'Multiple topic branches'. Avec le temps il est transformé plutôt dans le model linear 'Long-Running Branches'.



Dans Quelle Mesure La Compilation Du Code Est-elle Reproductible ?

La compilation ne pose aucun problème avec la configuration actuelle de Maven. Très bonne et détaillée documentation pour installation et participation au projet. Voir :

https://github.com/jenkinsci/jenkins/blob/master/CONTRIBUTING.md

A Quel Point Le Code Est-il Dépendant De Bibliothèques Externes ?

Le projet au complet références bibliothèques 278 externes.

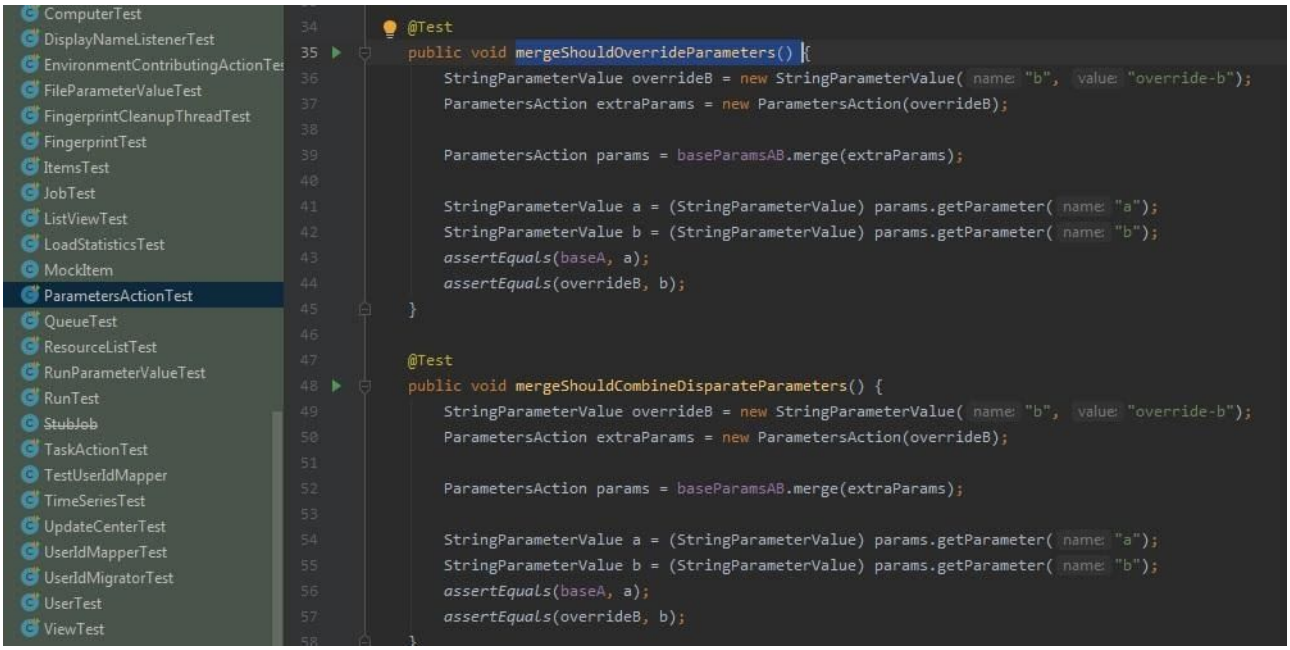
Dimension "Tests"

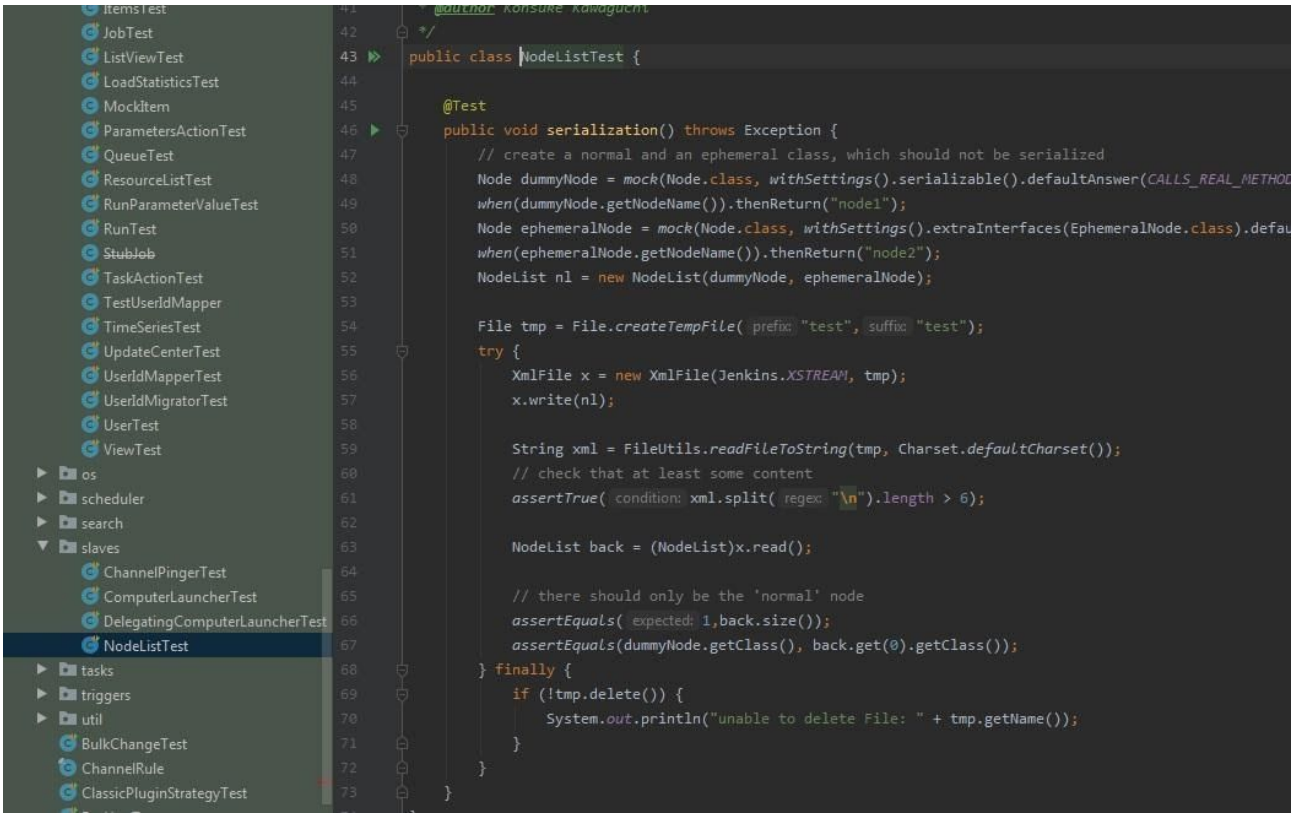
Quelles Sont Les Méthodes De Tests Mise En Place Dans Le Projet ?

Ils sont présentés des tests unitaires, d'intégration et même JavaScript. Module core contient des tests unitaires et fonctionnels. Il y a aussi un module 'test' qui contient autre partie des tests fonctionnels. Les tests fonctionnels sont assez lourd et lancer par les pipelines d'intégration. Par contre à la compilation par Maven du core j'ai du utilisé -DskipTests, pour pouvoir builder le projet. Même dans leur documentation il y a un indicateur de lancer le build de ce façon. https://jenkins.io/doc/developer/building/intellij/

Comment Qualifiez Vous La Qualité Des Tests Mis En Oeuvre ?

Dans le module client il n'y a pas des tests unitaires. Dans le core ça dépends grandement dû auteur des tests. Je m'en doute par exemple de l'utilité du test ComputerLauncherTest.java. Autre cas - par exemple je ne vois pas de différence entre le mergeShouldCombineDisparateParameters() et mergeShouldOverrideParameters() dans ParametersActionTest.java. Pratiquement on a deux tests nommé différemment, mais qui test la même chose.





Dans Quelle Mesure Les Tests Sont-ils Reproductible ?

Sur le core, avec mvn test ou mvn clean install j'ai une partie des tests qui échues. Dans le module core j'ai trouvé aussi des scripts groovy pour l'exécution. Vue la manque de documentation claire et un mécanisme un peu complexe pour les tests unitaires sur un poste de développeur, je note ceci comme un défaut sur la côté testabilité. En travaillant sur l'ajout de jacoco au projet, je me suis rendu compte, que l'utilisation du Sonar n'est pas un pratique courante dans l'équipe Jenkins.

Dimension "Déploiement & Livraison"

Quels Outils D'intégration / Déploiement Continus Sont Mis En Place ?

Jenkins utilise ces propres outils d'intégration et déploiement pour ces besoins.

Comment Sont Gérées Les Dépendances Logicielle Dans Le Projet ?

Jenkins utilise le mechanism BOM de Maven pour la gestion des dépendances.

Quelle Méthodologie De Publication (Releasing) Est Mise En Place ?

Pour la publication des plugins est utiliser Maven :

https://jenkins.io/doc/developer/publishing/releasing/

Par contre pour le core je n'en pas trouver d'information, vu que probablement seulement les leads d'équipe ont la possibilité de le faire.

Conclusion

Caractéristiques de la maintenabilité selon ISO25000 :

Modularité - Prenant en considération toutes les points sur la modularité on cote à 90% ce point.

Réutilisabilité - Un bon utilisation des interface et classes pas trop grands - on cote à 90% ce point.

Potentiel d'analyse - Code lisible, mais ce vois l'empreinte de open source - il n'y a pas un certains règles de codage. Pas trop grand volume de duplication. Aussi certains complexité relié au mélange des differents technologies. Ça va demander certains investissements dans la formation d'équipe de maintenance. Autre point négatif - l'utilisation de Stapler comme Framework WEB, qui est développé par un petit équipe (38 devs). J'irai préférer un framework plus reconnu et largement utiliser. Certains lacunes dans documentation. Prenant en considération les points on cote à 75% ce point.

Potentiel de changement - Bon potentiel en prenant en considération quantité minime de duplication, code pas trop complexe et en couplage minime entre les modules. Un équipe stable - on cote à 90% ce point.

Stabilité des modifications - Manque de pratique d'utilisation de Sonar dans l'équipe comme minus. En faveur - un large communauté. Code pas trop pourri. Je note à 85% ce point

Potentiel de test - Manque des tests unitaires. Parfois la qualité n'est pas à la hauteur. Par contre, un large mécanisme des tests fonctionnel automatiser. 75%

On obtient un côté de 84.2, sans faire un différentiation entre le poids des caractéristiques (certains devraient prendre plus vs autres)

De façon générale je suis favorable au pris du contrat pour ce logiciel. Un assez bonne qualité et stabilité pour un logiciel open-source. Avec ajustement des certains pratiques, correction sur les tests unitaires, ajout du pratique d'utilisation de Sonar, il est possible assez facilement de maintenir cette application.