UNIVERSITÉ DU QUÉBEC À MONTRÉAL

MGL7460-90 HIVER 2020

REALISATION ET MAINTENANCE DE LOGICIELS

PROJET INDIVIDUEL

PRÉSENTÉ PAR

MAJED ABIDI

7 FÉVRIER 2020

Équipe Enseignante :

Professeur : Sébastien Mosser (UQAM)

Laboratoires : Jean-Philippe Gélinas (SFL, Lévis), Jonatan Cloutier (SFL, Montréal)

Table des matières

1. Introduction 3
2. Objectifs..... **Erreur ! Signet non défini.**
3. Cadre d'analyse de maintenance 3
3.1. Dimension "Équipe de développement"..... 3
3.2. Dimension "Exigences" ..... **Erreur ! Signet non défini.**
3.3. Dimension "Architecture logicielle"..... 4
3.4. Dimension "Code source" 5
3.5. Dimension "Tests" 6
3.6. Dimension "Déploiement & Livraison" 6

1. Introduction

Il y a plus d'une dizaine d'années, le framework Java de tests unitaires JUnit avait connu une forte évolution avec l'introduction de JUnit 4. A partir troisième trimestre 2017 une nouvelle ère approche avec JUnit 5. L'équipe JUnit a livré en début avril 2017 une quatrième version *milestone* de JUnit 5 avec une architecture complètement différente des versions précédentes et comprenant une documentation complète.

Dans cet article, je présenterai l'architecture de JUnit 5, la qualité du code source ainsi une analyse sur la stabilité de l'équipe de développement.

Je présenterai ensuite un bref état des lieux sur le support de JUnit 5 par les outils de développement les plus populaires et je terminerai par les méthodes de test mises en place dans le projet.

2. Cadre d'analyse de maintenance

2.1. Dimension "Équipe de développement"

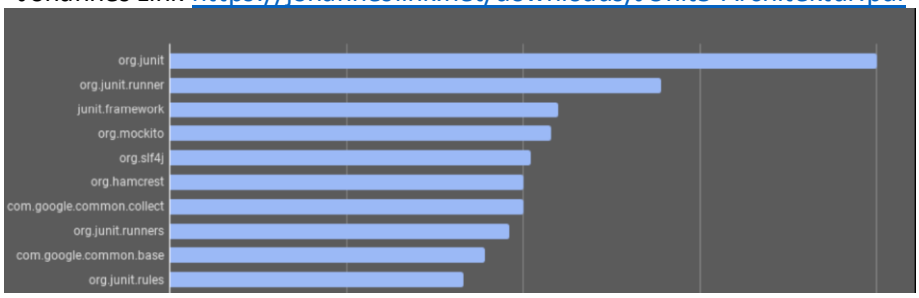
2.1.1. Qui sont les développeurs principaux du projet ?

1- Sam Brannen - Hardcore Software Developer : Conférencier international pour tout ce qui est JAVA, Spring et Junit; L'auteur du spring tesContext Framework et un core committer pour JNUIT5.

2- Marc Philipp - Software engineer at @gradle, @junit-team lead, open-source enthusiast.

3- Christian Stein : Single `junit-jupiter` Aggregator Artifact

4- Johannes Link https://johanneslink.net/downloads/JUnit5-Architektur.pdf



2.1.2.L'équipe de développement est-elle stable ?

Selon l'outil insight de Github, les développeurs/ingénieurs principaux de JUnit5 sont les mêmes depuis 2017, ce qui nous permettre de conclure que l'Equipe est stable.

2.1.3.Comment est répartie la paternité du code source dans l'équipe ?

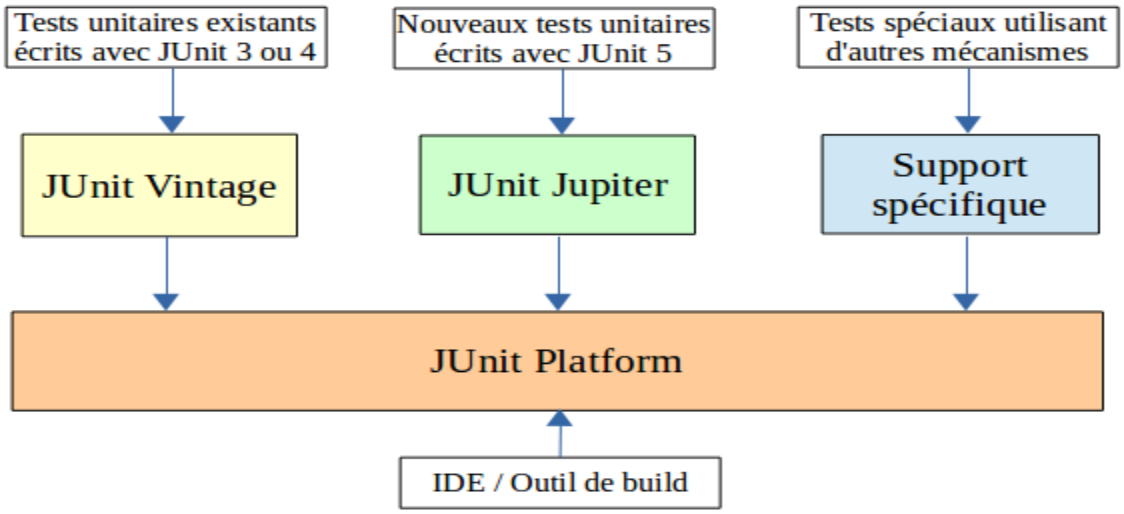
2.1.4.Comment les développeurs communiquent entre eux ?

Les canaux principaux de communication entre les développeurs sont StackOverflow et Gitter

2.2. Dimension "Architecture logicielle"

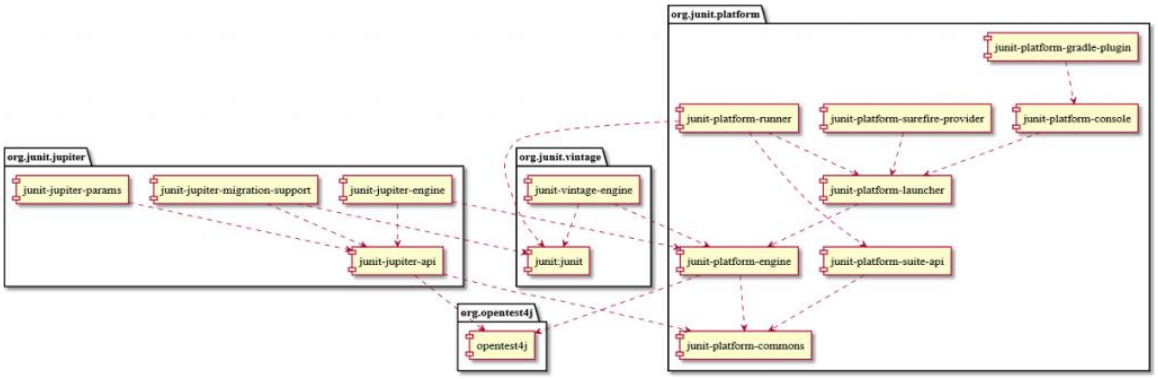
2.2.1.Quels sont les composants principaux de l'application ?

- JUnit Platform : Fondation pour le lancement de frameworks de test sur la JVM.
- JUnit Jupiter : Une nouvelle API et un mécanisme d'extension pour écrire des tests Unitaire sur JUnit5.
- JUnit Vintage : C'est un TestEngine pour exécuter des tests existant sur JUnit3&4



2.2.2.Comment est gérée la modularité de l'application ?

JUnit 5 est devisé en 3 modules et se présentait sous forme d'une architecture monolithique :



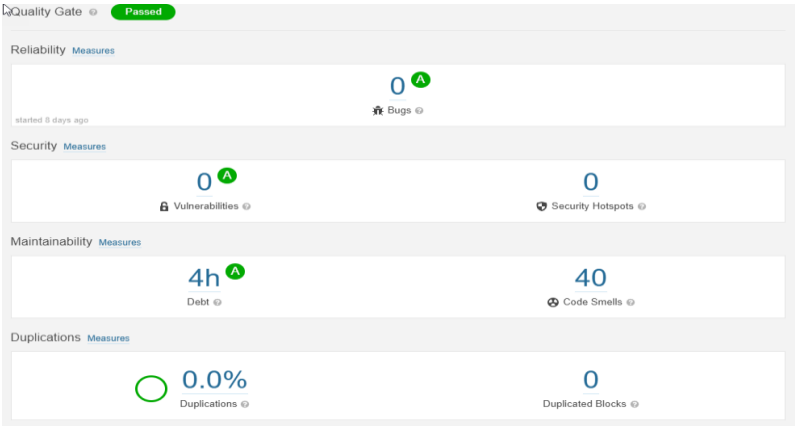
1.1.1. Comment est documentée l'architecture de l'application ?

La documentation de l'architecture est très bien détaillée sous le site Github ainsi que dans le guide utilisateur de JUnit5.

1.2. Dimension "Code source"

1.2.1. Comment qualifiez-vous la qualité du code source ?

L'outil qui sera utiliser pour faire l'analyse du code sera SonarCloud



1.2.2. Quels outils de construction (p.-ex. make, maven) sont utilisés ?

Gradle (https://scans.gradle.com/s/bl3pw4mrbgsao/timeline), Sonarcloud.

1.2.3. Quel modèle de branche est utilisé dans le développement du projet ?

Le modèle de branche utilisé dans ce projet est le modèle Supportingbranches de Github, qui est composée principalement de 5 branches :

- Master
- Develop
- Feature branches (Expriment dans le cas JUnit5)
- Release Branches
- Hotfix branches (Issues et Refactor dans le cas de JUnit5)

(Shema du flow à venir)

1.2.4. Dans quelle mesure la compilation du code est-elle reproductible ?

1.2.5. À quel point le code est-il dépendant de bibliothèques externes ?

Le code ne dépend d'aucune bibliothèque externe (Analyse pas encore terminé)

1.3. Dimension "Tests"

1.3.1. Quelles sont les méthodes de tests mise en place dans le projet ?

1.3.2. Comment qualifiez-vous la qualité des tests mis en œuvre ?

1.3.3. Dans quelle mesure les tests sont-ils reproductibles ?

1.4. Dimension "Déploiement & Livraison"

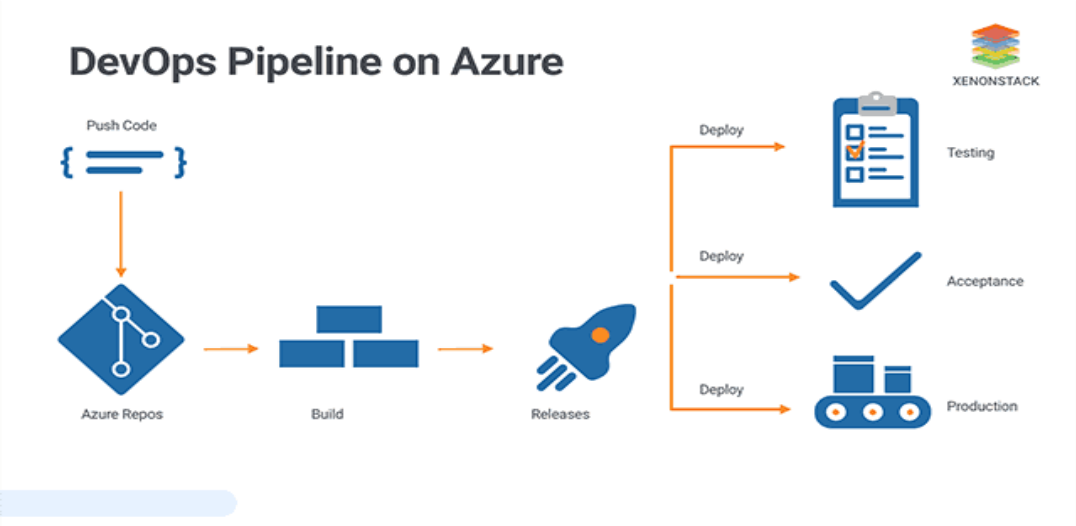
1.4.1. Quels outils d'intégration / déploiement continus sont mis en place ?

L'outil d'intégration / Déploiement continus utiliser est Azure Pipelines.

1.4.1.1 C'est quoi Azure Pipelines **:**

Azure Pipelines est un service cloud que vous pouvez utiliser pour créer et tester automatiquement votre projet et le mettre à la disposition d'autres utilisateurs. Il fonctionne avec à peu près n'importe quel langage ou type de projet.

Azure Pipelines combine l'intégration continue (CI) et la livraison continue (CD) pour tester et créer votre code en permanence et de manière cohérente et l'expédier à n'importe quelle cible.



1.4.1.2 Pourquoi Azure Pipelines :

- Se déploie sur différents types de cibles en même temps
- S'intègre à GitHub
- S'intègre aux déploiements Azure
- Construit sur des machines Windows, Linux ou Mac
- Fonctionne avec des projets open-source.

1.4.2.Comment sont gérées les dépendances logicielles dans le projet ?
1.4.3.Quelle méthodologie de publication (releasing) est mise en place ?