UQAM

PROJET INDIVIDUEL

PAR

ERIC GAGNON - GAGE22037705

TRAVAIL PRÉSENTÉ

À

SÉBASTIEN MOSSER

MGL-7460
RÉALISATION ET MAINTENANCE DE LOGICIELS

29 FÉVRIER 2020

Introduction

Ce travail est pour l'analyse du logiciel ReactiveX / RxJs afin d'estimer le risque d'accepter ce projet de tierce maintenance pour la compagnie.

ReactiveX est un API pour la programmation asynchrone en utilisant la combinaison des meilleurs concept du patron Observable, du patron Iterator et de la programmation fonctionnelle. ReactiveX est utilisé partout dans le développement logiciel, du backend au frontend, et il est multi-plateforme (Java, Scala, C#, C++, Clojure, JavaScript, Python, Groovy, JRuby, et autres).

Pour notre analyse c'est l'implémentation en Javascript (RxJs) qui sera étudié.

Cadre d'analyse de maintenance

Pour analyser la maintenance du logiciel RxJs je vais porter attention aux trois dimensions suivantes:

1. Dimension "Équipe de développement"
2. Dimension "Code source"
3. Dimension "Tests"

La dimension "Équipe de développement"

L'équipe de développement d'un logiciel est une partie importante de la pérennité d'un projet. Pour déterminer si l'équipe de RxJs est un enjeu majeur ou un atout pour le futur de ce projet, je vais répondre aux questions suivantes:

- Qui sont les développeurs principaux du projet ?
- L'équipe de développement est-elle stable ?

La dimension "Code source"

La qualité du code source d'un logiciel est très importante. Si la qualité du code est mauvaise elle peut mener à des pertes financières ou pertes de temps dû aux maintenances, modifications ou ajustements. Pour valider la qualité du code je vais répondre aux questions suivantes:

- Comment qualifiez vous la qualité du code source ?
- Dans quelle mesure la compilation du code est-elle reproductible ?

La dimension "Tests"

Les tests sont la partie où l'on valide le bon fonctionnement de notre code. Il est donc important pour un logiciel d'avoir des bonne méthodes de tests. Pour valider la qualité des tests je vais répondre aux questions suivantes:

- Quelles sont les méthodes de tests mise en place dans le projet ?
- Comment qualifiez vous la *qualité* des tests mis en oeuvre ?
- Dans quelle mesure les tests sont-ils reproductible ?

Dimension “Équipe de développement”

Les développeurs principaux

Outils utilisés

La section “Team” du site web de RxJs (https://rxjs.dev/team).
Pour les statistiques de contributions j’ai utilisé l’outils CodeScene (https://codescene.io/) et la section “contributors” de GitHub .
Pour les informations supplémentaire j’ai utilisé le site https://rxjs.dev/.

L’Équipe “Core”

Ben Lesh est le principal contributeur et qui est en fait le “Lead” de Rxjs. Il a un très bonne expérience pour avoir travailler dans le projet React, Angular et chez Netflix. Pour bien comprendre l’importance de son implication je vous invite à consulter cette analyse:

- https://codescene.io/projects/7238/jobs/20801/results/social/knowledge/individuals.

Les autres membre de l’équipe “core” sont:

- David Driscoll, Web & .NET Developer, Microsoft MVP
- Jan-Niklas Wortmann, Software Developer
- Nicholas Jamieson, front-end developer at Rexlabs
- OJ Kwon, Senior Software Engineer
- Paul Taylor, Software Engineer at nvidia
- Tracy Lee, Google Developer Expert

Nous pouvons voir que l’équipe principale est constituée de plusieurs développeurs d’expériences, ce qui est très encourageant.

Voici un graphique qui démontre l’implication de l’équipe “core” :

- https://codescene.io/projects/7238/jobs/20801/results/social/knowledge/teams

La relève

Ce qui est intéressant c’est qu’il y a une équipe “Learning Team” qui est là pour s’assurer de la bonne continuité du projet dans le futur. Les connaissances des contributeurs principaux doivent être transmise pour que le projet évolue de la bonne façon.

Les autres contributeurs

Rxjs est un projet très populaire et il y a beaucoup de contributeurs, voici une liste complète avec statistiques de contribution:

- https://codescene.io/projects/7238/jobs/20801/results/social/authors

La stabilité de l'équipe de développement

Dimensions "Code source"

La qualité du code source

Pour répondre à la question "Comment qualifiez vous la qualité du code source ?", je vais utiliser les mesures statiques suivantes:

- Le nombre de "code smells".
- La dette technique.
- Le ratio de la dette technique.
- La côte de maintenabilité.

Le nombre de "code smells" est le nombre de problèmes provenant du code source. Un "code smell" est une indication en surface qui corresponds habituellement à un problème plus profond dans le système.

La dette technique, quant à elle, est l'effort requis pour corriger tous les "code smells". Cette mesure est en temps (minutes) et présume qu'une journée de travail est de 8 heures.

Le ratio de la dette technique est le ratio entre le coût de remédiation et le coût de développement. Selon SonarQube une ligne de code coûte 0.06 jour (Remediation cost / (Cost to develop 1 line of code * Number of lines of code)).

La côte de maintenabilité est obtenue à partir du ratio de la dette technique. Selon SonarQube cette côte est évaluée de A à E selon cette grille: A=0-0.05, B=0.06-0.1, C=0.11-0.20, D=0.21-0.5, E=0.51-1.

Outils utilisés

Pour obtenir les mesures précédentes, j'ai utilisé l'outils d'analyse de code statique Sonarqube (https://www.sonarqube.org/).

Les liens pour consulter les résultats de l'analyse sont:

- https://sonarcloud.io/dashboard?id=r_x_j_s.

Métriques

Dans la base de code de RxJs y a au total 77 “code smells”. De ceux-ci 2 sont bloquant, 0 critique, 44 majeurs et les 31 restants sont mineurs. Cette dette technique représente environ 4H40 d’effort pour un développeur logiciel pour tous les corriger. L’analyse statique nous donne comme ratio de la dette technique 0.1%. Selon SonarQube, ces ratios nous donne pour une cote de maintenabilité de A.

Aussi selon le “Hotspot Code Health” de CodeScene:

- https://codescene.io/projects/7238/jobs/20801/results/code/biomarkers

Il y a 6 fichiers sources sur 227 qui ont des problèmes dont en exemples:

- Deeply Nested Logic
  - Nested conditional depth of 6
  - In addition, there are 1 other functions with deep conditional logic.
- Bumpy Road Ahead
  - The code is complex to read due to its nesting with multiple logical blocks.
- Brain Method Detected
  - The function has a McCabe complexity of 19 with 93 lines of code.
- Many conditionals
  - The average function complexity is 4.55, and the total complexity in the file is 91 (McCabe).

Conclusion

Selon l’analyse statique, la qualité du code en général semble assez intéressante sauf pour certaines parties mais c’est trop mineur pour être un enjeu.

Reproductabilité de la compilation

Dimension "Tests"

Méthodes de tests

La *qualité* des tests

Reproductabilité des tests