Projet Individuel MGL7460-90

Rapport final

Réalisé par

Gary Rivera

(RIVG23057109)

Table de matières

Introduction

J'ai choisi d'analyser la dimension de l'équipe de développement du projet Angular. Ce dernier est un cadriciel libre disponible dans GitHub. La période analysée est du 14 septembre 2014 au 17 janvier 2020. Dans l'analyse, je ne remets pas en question l'importance ou la pertinence du code fusionné dans la branche master. Je prends les informations du message du *commits* et du ticket créé dans GitHub.

Aussi, dans certaines extractions, j'ai corrigé les noms ou les adresses courriels des contributeurs car dans certains cas, il était évident que c'est la même personne qui avait changé de courriel.

Méthodologie d'analyse

Je procède de la façon suivante; durant l'analyse de la dimension, je fais des extractions du log du projet angular de la branche master et non les branches intermédiaires qui ont été fusionnées et supprimées à la suite d'un processus de fusion.

Les extractions sont transposées en question. Elles sont travaillées avec l'aide des scripts PowerShell. Par la suite, j'importe les données dans Excel pour être avalisées grâce aux tableaux croisés dynamiques.

Pour chaque réponse aux questions, je fais référence à l'extraction utilisée pour y répondre. Ainsi, tous les requêtes et les fichiers Excel d'analyse se trouvent dans un répertoire identifié question/réponse.

De plus, pour éviter de travailler en vain, j'établi des paramètres pour réduire le nombre de variable. Au début de chaque réponse, on trouve la définition des paramètres qui explique son utilité.

Pour chaque question il y a un répertoire avec les scripts PowerShell et les fichiers Excel.

Choix d'outils d'analyse

Pour l'extraction des informations dans le log, j'utilise les commandes git pour interroger, mais aussi il y a de l'information exploitable dans un format présentable sur le site web:

https://github.com/angular/angular/graphs/contributors

Ensuite, toutes les données seront analysées et compilées grâce aux logiciels Excel et code-maat. Ce dernier est utilisé et mentionné dans le livre *Code as a Crime Scene*. Il permet de résumer certaines données.

Dimension prise en charge par l'analyse

La dimension abordée dans ce projet est "l'Équipe de développement". Je vais répondre aux questions sur l'équipe des contributeurs qui a participé à la période analysée. L'analyse est concentrée davantage sur les contributeurs qui ont réalisé un nombre de *commits* assez élevé durant la période.

Voici les questions que j'essaie de répondre:

1-Qui sont les développeurs principaux du projet ?

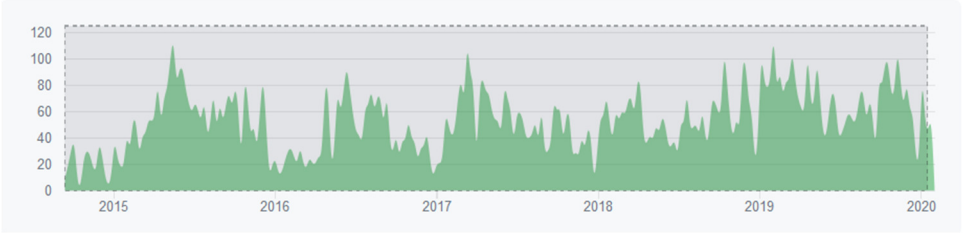
Paramètres: j'analyse les développeurs qui ont réalisé dix *commits* et plus par année ainsi que les contributeurs de 500 *commits* et plus durant la période.

Les développeurs principaux selon ma recherche sont : Peter Bacon Darwin, George Kalpakas, Victor Savkin, Victor Berchet, Igor Minar, Tobias Bosch, MiÅ¡ko Hevery, Alex Eagle, Alex Rickabaugh et Paul Gschwendtner.

Le tableau suivant contient le nombre de *commits* réalisé par les développeurs à ce jour.

	À partir du git log	**À partir du site github**
Peter Bacon Darwin	1240	1255
George Kalpakas	1102	1118
Victor Savkin	952	952
Victor Berchet	916	916
Igor Minar	896	912
Tobias Bosch	789	788
MiÅ¡ko Hevery	615	631
Alex Eagle	559	559
Alex Rickabaugh	531	542
Paul Gschwendtner	535	536

Si je prends les développeurs qui ont fait plus de dix *commits*, j'ai une moyenne de 153 *commits*. Alors mon top 10 sont de loin les plus productifs. Ces *commits* sont répartis durant la période analysée.



https://github.com/angular/angular/graphs/contributors?from=2014-09-14&to=2020-01-17&type=c

Le tableau précédent montre les *commits* des contributeurs dans la branche master. La vraie question est si les développeurs principaux ont été toujours les mêmes. La réponse est, à la logique d'un bon informaticien; ça dépend de la façon à laquelle nous faisons parler les *commits*. Si l'on réparti la période analysée par année, on remarquera qu'entre 2014 et 2016 Peter Bacon n'était pas le premier dans les tops 10.

On observe qu'au début Peter était au bas du peloton. Mais avec le temps il a augmenté sa contribution. Un autre exemple est George Kalpakas, il n'était pas de la première moitié du projet et il est arrivé au top de la liste à ce jour.

Tableau de *commits* par développeurs entre 2014 et 2016 et la performance de Peter Bacon :

Les développeurs principaux ont tous en commun un grand nombre de *commits* sur la branche master, des propositions *et des revues* de *pull request* et des ouvertures de problèmes. Ils participent activement au projet[1].

En dernier, ce petit ensemble de développeurs ont aussi le plus grand nombre d'ajout et de suppression de lignes de code dans le projet durant la période analysée. Voir le tableau suivant:

Nom	Nombre de lignes affectées
Peter Bacon Darwin	473,293 ++ 260,197 --
Victor_Savkin	165,431 ++ 117,393 --
Victor Berchet	160,889 ++ 182,997 --
Igor Minar	576,017 ++ 529,026 --
Tobias Bosch	285,203 ++ 245,331 --
Misko Hevery	158,030 ++ 153,748 --
Alex_Eagle	403,091 ++ 261,394 --
Pawel Kozlowski	38,858 ++ 26,165 --
Marc Laval	37,792 ++ 16,413 --

[1] https://github.com/angular/angular/graphs/contributors?from=2014-06-14&to=2015-12-31&type=c

Ceci veut dire, que ces développeurs sont les joueurs importants dans l'équipe du projet. Ils participent à tous les niveaux du projet: ajout des fonctionnalités, corrections des problèmes et autres. Aussi, je peux dire qu'ils possèdent la connaissance du produit.

2-L'équipe de développement est-elle stable ?

Oui, l'équipe est stable, malgré une baisse en 2017, l'équipe n'a pas cessé d'augmenter. Il y a eu un phénomène particulier, les premiers développeurs ont eu une baisse de productivité et le deuxième groupe de développeurs, qui était moins productif au début de la période, sont passés les premiers du peloton à la fin de la période.

La première analyse est depuis le début de la période 2014 et 2015. Il ne reste alors que quatre personnes: Misko Hevery, Marc Laval, Rado Kirov et Jeremy Elbourn. Seulement deux parmi les quatre apparaissent dans le top dix des développeurs.

En revanche, à partir de 2017, les choses ont changé. On a vingt-six développeurs qui forment le cœur de l'équipe. Ils sont : Peter Bacon Darwin , Igor Minar, Matias NiemelÃ¤, George Kalpakas, Alex Rickabaugh, Stefanie Fluin, Alex Eagle, Olivier Combe, Jason Aden, Misko Hevery, Marc Laval, Kara Erickson, MisÌŒko Hevery, Kapunahele Wong,Pawel Kozlowski, Vikram Subramanian, Hans Larsen, Filipe Silva, Trotyl Yu, vikerman, Martin Probst, Fabian Wiles, JiaLi.Passion, Kara, Rob Wormald et Stephen Fluin.

On peut dire que l'équipe est stable depuis les trois dernières années. Une chose intéressante à remarquer est que le nombre d'anomalies a diminué durant la période de transition de l'équipe. Nous pourrions croire qu'un changement dans l'équipe apporterait des anomalies mais ce ne fut pas le cas dans ce projet.

De plus, il y a un ensemble de personnes qui fait partie de l'équipe depuis le début du projet. Ils sont : Alex Eagle, Alex Rickabaugh, Brandon Roberts, cexbrayat, Igor Minar, Jeremy Elbourn, Marc Laval, Martin Probst, Matias NiemelÃ, Misko Hevery Pawel Kozlowsi et Peter Bacon Darwin. Ce qui assure le transfert de connaissance à travers le temps.

3-Comment est répartie la paternité du code source dans l'équipe ?

Dans cette question, j'essaie de valider si le code a changé beaucoup depuis sa création et durant la vie du projet. Aussi, pour bien comprendre la paternité du code, je vais nommer l'équipe A, celle qui a participé de 2014 à 2017 et l'équipe B, celle-ci qui a pris la relève après 2017.

Alors, sur 29 164 *commits* analysés, je trouve une moyenne 29% et une médiane de 25% de changement tout type de fichiers confondu. Je me suis attardé sur les modules et les packages car ces derniers possèdent une grande quantité de codes exécutables.

12 224 commits parmi cet ensemble ont plus que 29% de taux de changement. Ces changements ont eu lieu dans une période de trente-neuf jours (39) en moyenne. J'ai ajouté la notion de jour entre la création et les *commits* subséquents. Cette donnée peut nous informer si le changement a eu lieu rapidement après la création du code ou non. Il y a du code en haut de 80% du taux de changement, mais à l'intérieur de deux mois. On peut dire qu'il y a eu une correction importante dans le code.

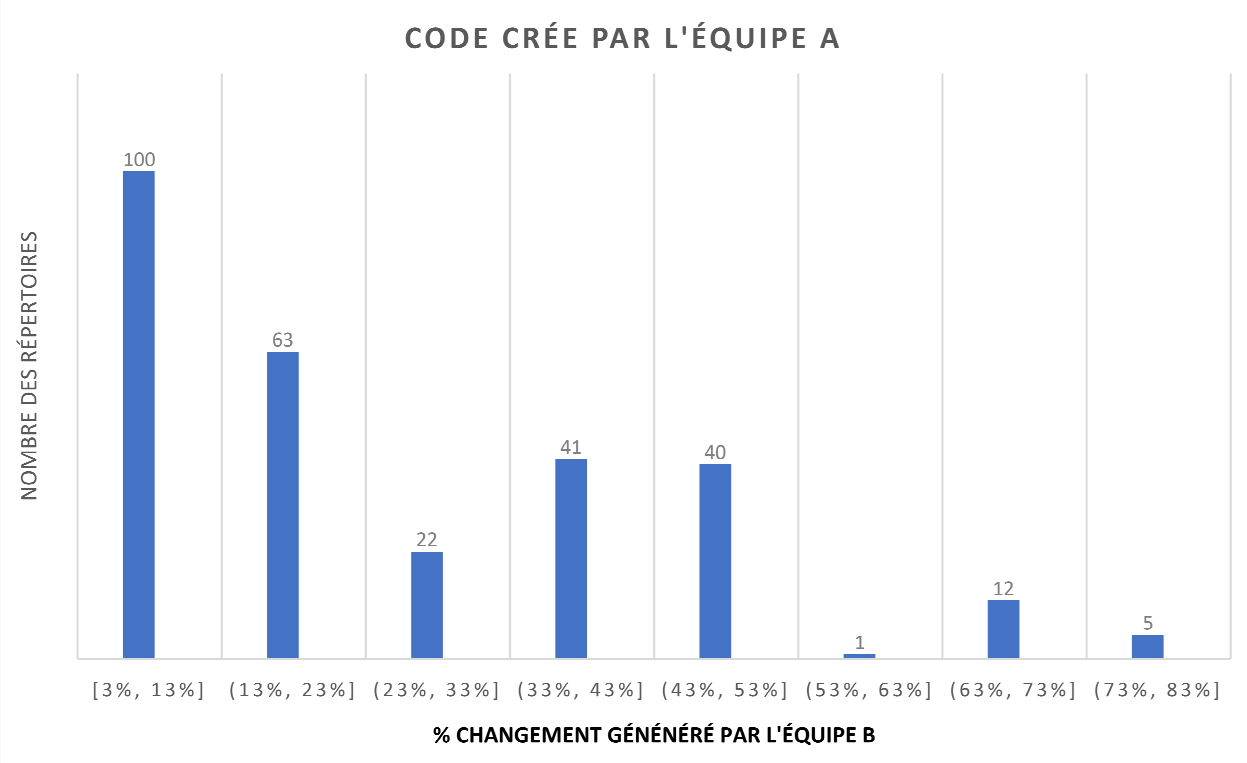
Pour le reste des changements, ceux entre 30% et 79% du code changé, ont en moyenne 46% de changement du code dans ce palier et, nous pouvons reculer presque deux ans entre la création et les *commits*. Nous pouvons affirmer que la paternité de cette portion du code est très touchée car elle a subi beaucoup de changement.

Voici, les paliers de variations depuis le premier *commit* en 2014 :

Palier	**% Changement**	**Commit**
1	0-29	16940
2	30-80	12040
3	80-100	184
Total commits		29164

Si nous prenons en considération les premiers commits faits par l'équipe **A**, les codes ont été modifiés entre 10% et 75% par l'équipe **B**. Dans la recollecte d'information, je me suis assuré que les commits subséquents ont été bien réalisés par un contributeur de l'équipe **B**.

Le tableau suivant montre les changements du code réalisé par l'équipe **B** sur du code créé entre 2014 et 2017.



Il est difficile de dire avec cette répartition des changements si le code a changé complètement de parents. Cependant, je sais que 17 répertoires ont un taux de changement à plus de 63% Dans ce cas la paternité a changé complètement. En dessous de 63%, on est dans un cas de garde partagée, car les créateurs sont encore dans l'équipe, mais l'équipe B prend charge de l'évolution et la maintenance.

4-Comment les développeurs communiquent entre eux ?

Le projet Angular possède un système de tickets assez efficace. Il est enrichi de 138 étiquettes (au moment de l'analyse). Ces dernières permettent d'identifier et qualifier un ticket (github.com/angular/angular). Les types de catégorie possibles des tickets lors d'une ouverture sont:

- ***build**: Changes that affect the build system or external dependencies (example scopes: gulp, broccoli, npm)*
- ***ci**: Changes to our CI configuration files and scripts (example scopes: Circle, BrowserStack, SauceLabs)*
- ***docs**: Documentation only changes*
- ***feat**: A new feature*
- ***fix**: A bug fix*
- ***perf**: A code change that improve performance*
- ***refactor**: A code change that neither fixes a bug nor adds a feature*
- ***style**: Changes that do not affect the meaning of the code (white-space, formatting, missing semi-colons, etc)*
- ***test**: Adding missing tests or correcting existing tests*

Après avoir déterminé le type de ticket, nous pouvons ajouter les attributs comme la sévérité, le *Pull Request*, le risque, la grandeur de l'effort, le composant affecté et son état. Il a y des étiquettes qui s'ajoutent régulièrement dans le projet.

Il y a également une politique de message lors d'un *commit*. Ceci permet d'uniformiser les messages et faciliter la compilation de l'information. Il faut respecter une nomenclature précise pour identifier le message du *commit*. Voir Commit-Message-Guidelines

Le système de tickets compte sur leur système de clavardage, de commentaire et de courriel. Cet ensemble d'outils aide la communication entre les développeurs.

Alors, le système implanté est simple et efficace. Il permet d'être en communication avec toutes les parties impliquées dans le développement du ticket. Aussi, la traçabilité est très présente dans le système. Donc, la communication est fluide.

5-Il n'y a pas de sous-équipes pour corriger des problèmes (tout le monde fait tout)

Afin de répondre à cette état de fait, j'ai choisi de me concentrer sur les catégories de *commits* qui ont plus de 1000 occurrences dans la période analysée. Ces catégories sont : build, chore, doc feat, fix et refactor. Le système des tickets d'Angular, nous a permis de retrouver les tickets par catégorie rapidement.

Selon les données récupérées des *commits*, au-delà de 50 *commits* par contributeur, je remarque qu'un contributeur a participé à presque toutes les catégories mentionnées précédemment.

En dessous de 30 *commits* par contributeur, je trouve une participation dans toutes les catégories de ticket mais elles sont évidemment moindres. Ici, je n'essaie pas d'évaluer l'importance ou la pertinence de l'intervention du contributeur.

En générale, je peux dire qu'il y a une variété d'interventions par les contributeurs. Il est certain qu'il y a une équipe centrale qui est en mesure de tout faire dans le projet mais il n'existe pas une équipe dédiée à la maintenance. Même ceux qui ont moins de *commits*, ont contribué à plusieurs catégories de tickets dans l'ensemble de l'application.

6- Le changement des contributeurs dans l'équipe a-t-il un impact sur l'augmentation des anomalies?

Voici les nombres de *commits* pour les: Feat, Fix et test.

Année	**2014**	**2015**	**2016**	**2017**	**2018**	**2019**	**2020**	Total
Feat	105	633	391	369	322	276	8	**2104**
Fix	50	757	839	**819**	**703**	986	36	**4190**
Test	5	67	74	90	269	367	26	**898**
Total	**160**	**1457**	**1304**	**1278**	**1294**	**1629**	**70**	**7192**

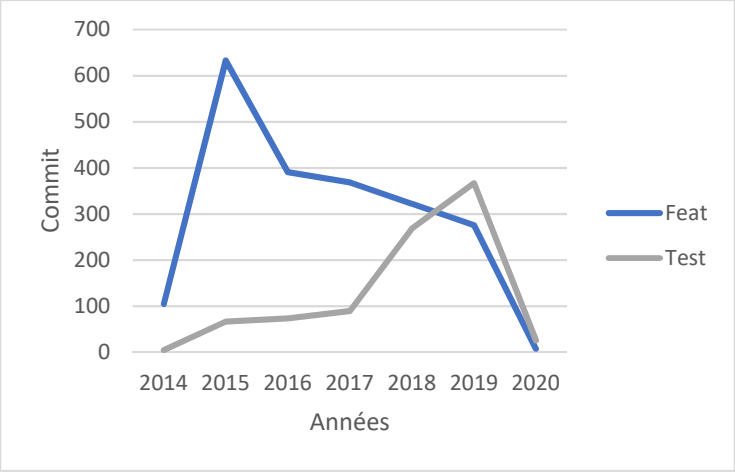
Je dirais que les anomalies sont restées stables malgré le changement de garde dans l'équipe. J'ai remarqué que durant la période 2017 et 2018, les contributeurs plus actifs ont réduit leur participation pour laisser la place à d'autres contributeurs.

Dans le tableau, je me suis permit d'ajouter les *commits* liés aux tests et fonctionnalités. Il est facile de remarquer que les corrections des anomalies sont presque 200% plus élevées que les fonctionnalités ajoutées. Sans trop abuser de l'interprétation des chiffres, je pourrais dire que l'équipe passe plus de temps à corriger des anomalies qu'à ajouter des fonctionnalités.

Alors, pour la maintenance du code, il faut prévoir plus de temps pour des réingénierie (*reafactoring*) et de tests que pour les évolutions. Cette réalité va être un facteur déterminant lors de la planification des activités dans le projet.

7- Est-ce-que l'équipe est axée sur les tests?

Seulement 36 contributeurs sur 132 ont ajouté ou corrigé des tests durant la période analysée. Ceci veut dire que 27% de l'équipe contribue aux tests.



De plus, 2019 est la seule année que le nombre de tests a été supérieur à l'ajout des nouvelles fonctionnalités. Le ratio des tests est très bas dans ce cadriciel qui est très utilisé par l'industrie.

Alors, le manque de tests donne l'impression que seulement un minimum de couverture est présent. Ceci alourdit les tâches des développements pour la maintenance et les évolutions, car le filet de sécurité qui offre les tests n'est pas présent.

8- Où sont les points chauds du cadriciel Angular ?

Du point de vue de la maintenance, je veux savoir où sont les problèmes dans le code et où l'équipe a fait le plus de correction. Ceci va me permettre d'avoir une idée des problèmes courants dans le cadriciel.

Le tableau suivant montre les *commits* liés aux corrections d'Angular dans la période analysée. Étant donné que le tableau est très volumineux, j'ai choisi les 9 *commits* les plus importants. Les détails des fichiers impactés sont dans les fichiers Excel fournis comme les détails de la requête au Git.

Path	Committed file	Sum of added	Sum of deleted
packages/common/locales/	2443	27053	19902
2018	1895	21568	16817
2019	546	5479	3077
2020	2	6	8
packages/core/src/render3/	714	9880	6662
2018	347	5292	3659
2019	364	4504	2993
2020	3	84	10
packages/common/locales/extra/	665	2998	1745
2018	598	2489	1605
2019	67	509	140
packages/core/test/render3/	571	29412	17051
2018	303	16983	6900
2019	264	12335	10132
2020	4	94	19
packages/common/locales/global/	535	4771	3078
2019	535	4771	3078
aio/content/guide/	346	3369	3457
2018	183	2527	2599
2019	162	828	850
2020	1	14	8
packages/compiler-cli/test/compliance/	243	10130	5767
2018	82	3602	1084
2019	158	6448	4681
2020	3	80	2
packages/compiler/src/render3/view/	211	2969	2037

2018	**93**	**1555**	**858**
2019	**116**	**1382**	**1130**
2020	**2**	**32**	**49**
packages/core/test/acceptance/	**193**	**12196**	**3636**
2019	**183**	**11609**	**3538**
2020	**10**	**587**	**98**

On peut dire que les neuf groupes de fichiers les plus corrigés dans la durée du projet se retrouvent dans : **packages/core/, packages/compiler/ et packages/common/locales/.** Il y a eu également les modules de la documentation (**aio/content/guide/**) qui ont été corrigés plus d'une fois. Je vais, toutefois, me concentrer sur les trois autres groupes.

En premier, le **packages/core/** comme son nom l'exprime, fait partie des principaux packages d'Angular. En lisant les commentaires des *commits* de corrections, j'ai remarqué que l'équipe faisait de la réingénierie au même moment qu'une correction. Alors, le code s'est amélioré au fur en mesure que les corrections se faisaient. Dans le carnet de commandes de l'équipe (*backlog*) l'information est facilement retraçable grâce au système de libellés implanté dans le processus de développement.

Ensuite, le packages/compiler/ qui est un composant essentiel, il traite les intrants avant que l'application web finale ne s'en serve. C'est un composant où la parenté n'a pas trop changé, 10% seulement. C'est aussi le composant avec le plus de tests dans le cadriciel. Je peux affirmer que nous avons un filet de protection car les tests peuvent assurer un peu plus de compatibilité à la correction. Je dis cela car, en lissant les commentaires, je trouve des corrections sur IE et les autres navigateurs. De plus, seulement une équipe restreinte joue dans le code. J'ai mis les courriels des développeurs des *commits* analysés car, il est plus facile que d'essayer d'écrire leurs noms. Ce sont les suivants: matias@yearofmoo.com, akushnir@google.com, misko@hevery.com, ben@benlesh.com, igor@angular.io, jasonaden@google.com, karakara@google.com, victor@suumit.com, alexeagle@google.com et igor@angularjs.org.

Pour sa part, le package **packages/common/locales/**, est très corrigé étant donné intégration des cultures dans le cadriciel. Ce qui demande un effort continu car il faut suivre les mises à jour des navigateurs afin de maintenir la comptabilité. Si je compare l'évolution du nombre de *commits* versus celui des corrections, on a 2443 contre 546 évolutions. C'est presque 5 fois plus élevé. Dans ma réponse concernant la paternité du code, ce package figure comme l'un parmi ceux qui ont changé plus qu'à 50%. Il est certain que si le code a changé à cause de fixes et non par des évolutions planifiées, il y aura un risque que des anomalies continuent de sortir à l'avenir.

On peut conclure que la maintenance de ces packages est complexe et elle arrive souvent. Il faudrait donc planifier plus de réingénieries des composants afin d'éviter d'agir en pompier et corriger sans améliorer.

Note de synthèse sur le cadriciel Angular

Est-ce que le cadriciel Angular est maintenable?

Angular est un cadriciel très populaire dans l'industrie. Bénéficiant d'une telle popularité, je m'attendais à avoir un cadriciel facile à maintenir. En analysant la dimension **équipe de développement,** j'ai essayé de comprendre le niveau de maintenabilité du cadriciel à travers l'équipe de développement, ses interactions, le carnet de commandes, ses *commits* et ses livrables.

La note de synthèse sera attaquée sur deux angles; la composition de l'équipe et les livrables produits pendant la période analysée.

La composition de l'équipe

À première vue, le nombre que constitut l'équipe de travail me semble important. Mais, j'ai constaté que seulement un petit groupe parmi eux, est en charge du cadriciel. Les développeurs principaux ont suivi une rotation pendant la période analysée, mais l'équipe est restée à 90% presque la même. Cette constance a comme résultat une paternité du code qui n'a pas trop variée, des modules qui ont beaucoup variés et des créateurs sont demeurés dans l'équipe tout au long de projet. Ce qui est un impact positif pour le projet. En plus, la communication est efficace parmi les membres de l'équipe, elle est gérée avec un outil simple comme le carnet de commandes de GitHub et un système des libellés créé dans le projet où chaque libellé correspond à un composant du cadriciel. Il faut dire, que le carnet de commandes n'arrête pas de croitre.

En terminant, l'équipe est polyvalente et hétérogène. Polyvalente dans le sens que toute les membres participent à tout type de tickets; évolution, correction, test, intégration, documentation et etc. Et hétérogène, car nous avons plusieurs nationalités dans l'équipe. Ceci peut être excellent pour avoir des visions différentes pour résoudre des problèmes. Mais aussi peut être, quelques fois, un obstacle pour la compréhension et pour s'entendre sur une solution. Une chose est certaine, cette multi ethnicité est plus enrichissante qu'un ensemble homogène de caucasiens.

Les produits livrables

Tout d'abord, j'ai trouvé un taux élevé de corrections par rapport aux évolutions réalisées. L'équipe produit beaucoup car elle est nombreuse. Mais, malgré les filets de revue du code et tests, les corrections sont également nombreuses. Ceci donne l'impression que les fonctionnalités sortent avant que tous les tests soient terminés et que l'équipe compte sur le grand public pour trouver les problèmes. Est-ce que la concurrence du marché exerce un peu de pression sur la vitesse de développement ? Il est certain que le modèle d'Angular est particulier, il est axé sur la production. C'est, du moins, l'impression qu'il me donne.

Notamment, les tests dans le cadriciel sont moindres de ce que l'on pense. Alors, pour la maintenance, ceci peut causer des casse-têtes et du fils à retordre. Sans filet de protection pour valider une intervention de correction ou de réingénierie, le risque d'erreurs dans le code est réel. Malgré le nombre réduit de tests, le cadriciel continue à survivre et à se positionner comme l'un parmi les principaux cadriciel dans l'industrie.

Finalement, les points chaud d'Angular sont corrigés et entretenus par les mêmes personnes qui les ont créés. Ceci peut être une arme à double tranchant. Soit que le code est difficile à comprendre ou, sa connaissance n'est pas partagée et seulement les collaborateurs directs de Google y ont accès. Toutefois, dans l'ensemble, je peux dire que les points chauds sont tout de même sous contrôle. Ils sont bien cernés par l'équipe.

Conclusion

La maintenance d'Angular demande beaucoup de code en aval pour mitiger les problèmes amenés par des évolutions ou des intégrations dans le cadriciel. Il est très vaste comme projet. Il y a plusieurs modules et plusieurs packages tous tout aussi important les uns que les autres. Il faut dire qu'Angular subit des pressions du marché au niveau des évolutions et des nouvelles technologies émergentes. Ce qui l'oblige à s'adapter rapidement.

Le mécanisme de gestion de l'équipe et la structure du code me semble corrects. Le fonctionnement de l'équipe est fluide et elle compte sur plusieurs contributeurs. Ces derniers, toutefois, ne le réalisent pas toujours et enregistrent des anomalies ou suggèrent des améliorations. Nous appelons cela la communauté de développeurs. C'est une force en soi, car elle constitue une armé d'analyste assurance qualité à la disposition de projet, et surtout gratuitement. Les outils et langages sur lesquels reposent Angular sont solides : *GitHub, TypeScript, Bazel e*t *JavaScript*. Ils sont tous des outils bien rodés et, par le fait même, une valeur sure.

Malgré la bonne organisation de l'équipe, avec 3 345 *commits* par année en moyenne, il est certain que le code est régulièrement basculé et mis à l'épreuve. Étant donné ce volume, le manque de tests est une faiblesse pour la maintenance du cadriciel. Nous devrions augmenter la couverture des tests pour supporter la maintenance et, également, penser à robotiser des revenues du code pour trouver les erreurs connues et les corriger. De cette façon, on peut libérer du temps aux développeurs seniors pour faire de la réingénierie ou des évolutions.

Avant de connaitre Angular dans mes équipes de développement, les développeurs parlaient de la lourdeur d'implémenter un changement de version du cadriciel. Je comprends bien maintenant, le cadriciel bouge beaucoup et il brise la compatibilité quelques fois. C'est tout de même positif mais il y a un prix à payer par les développeurs qui y travaillent. C'est, par le fait même, un cadriciel difficile à mettre à jour après une nouvelle version.

En ce qui concerne les bonnes pratiques de développement, Angular fait bonne figure. Le cadriciel respecte et utilise des gabarits de conceptions et une approche orientée objet. Il encourage les meilleures pratiques. Jusque-là ça va ! Toutefois, l'adaptabilité aux changements est moins présente dans le cadriciel.

En définitive, admettons que Google décide de vendre Angular et, Desjardins décide d'en évaluer son acquisition. Je ne recommanderais pas cette acquisition car la maintenance serait trop couteuse. En ce moment, Google absorbe les coûts de développement en fournissant des développeurs sur les projets et en profitant quelques contributeurs gratuitement à travers le monde. Le modèle d'affaires n'amènerait pas une valeur chez Desjardins étant donné l'effort et le temps qu'il faut mettre pour garder en vie Angular.

Au final, avec le temps que j'ai passé à vérifier les contributeurs du projet, on dirait que j'étais dans des anomalies et des corrections dans le code source tout le temps. Il se peut que j'aie besoin plus de temps pour mieux comprendre Angular, mais avec l'information acquise, Angular me semble difficile à maintenir.

Bibliographies

Adam Tornhill, Your Code as a Crime Scene: Use Forensic Techniques to Arrest Defects, Bottlenecks, and Bad Design in Your Programs. Pragmatic Bookshelf; 1 edition (April 9 2015) 220 pages

Annexe

Packages Angular

Voici la liste des packages qu'on peut travailler sur le projet Angular. Les requêtes **git log** font références au code source des packages de la liste suivante :

- animations
- common
- compiler
- compiler-cli
- core
- elements
- forms
- http
- language-service
- platform-browser
- platform-browser-dynamic
- platform-server
- platform-webworker
- platform-webworker-dynamic
- router
- service-worker
- upgrade
- zone.js