RÉSUMÉ

Le modèle de développement des logiciels open source attire de plus en plus l'attention des chercheurs et des entreprises par son efficacité et sa structure organisationnelle décentralisée. Dans ce modèle, une communauté de développeurs coordonne les activités de ses membres afin de produire des logiciels libres pour servir des millions d'utilisateurs et dont le code source peut être réutilisé par d'autres développeurs. Cette communauté est considérée comme un système complexe et autoorganisé dont les membres peuvent collaborer et contribuer au même projet bien qu'ils soient de différentes locations, cultures, expériences et contextes. Cette collaboration est possible vu l'existence de certaines plateformes, dont les plus utilisées sont GitHub et Gerrit.

Ces structures virtuelles tirent leur force, non seulement des compétences techniques des membres, mais également des différentes interactions au sein de la communauté. Ainsi, l'objectif principal de cette thèse est d'explorer les interactions au sein des communautés des projets open source, afin d'améliorer à la fois la productivité de ses membres et la qualité des produits logiciels. En premier lieu, nous nous sommes intéressés à la dimension sociale des structures des communautés open source à priori Ad hoc, notre étude sociotechnique a démontré l'existence de formes organisationnelles implicites et efficientes. Ensuite, nous avons suivi l'évolution de ces composantes au fil du temps et nous avons identifié des lignes directrices potentielles pour soutenir l'évolution saine de ces communautés. Puis, nous avons étudié l'impact des sentiments, exprimés dans les commentaires, des développeurs sur leur productivité et sur la qualité de révision du code source. En fin, nous avons mené une analyse de la propagation et de la densité des flux de connaissances au sein des réseaux d'interactions en vue de d'améliorer la performance des développeurs OSS.

Les études empiriques que nous avons mené dans ce travail de recherche s'appuient sur une méthodologie d'extraction des données historiques des activités liées aux projets open source les plus réputés, utilisés, et existants sur les plateformes GitHub et Gerrit. Après le traitement des données, nous avons procédé aux analyses en se basant sur des méthodes d'analyse des réseaux sociaux (SNA), couplées avec des analyses temporelles, des analyses de sentiments et d'autres calculs statistiques appropriés dans le but de répondre à nos questions de recherche prédéfinies.

Mots-clés : Communautés des logiciels Open Source ; Interactions sociotechniques ; Réseaux de collaboration ; Réseaux Dynamiques ; Mapping des réseaux ; Analyse des réseaux sociaux ; Analyse des sentiments ; Transfert des connaissances.

ABSTRACT

Open source software development model is attracting more than ever the attention of researchers and companies through its efficiency and decentralized organizational structure. In this model, a community of developers coordinates its members activities to produce free software to serve millions of users whose source code can be reused by other developers. This community is considered as a complex and self-organized system whose members can collaborate and contribute to the same project despite being of different locations, cultures, experiences, and contexts. This rich collaboration is thanks to the existence of certain platforms, the best known are GitHub and Gerrit.

These virtual structures derive their strength not only from the technical skills of its members, but also from the different interactions within the community. Thus, the main objective of this thesis is to explore the interactions within open source software communities, in order to improve both the productivity of its members and the quality of software products. In the first place, we have been interested in the social dimension of open source community Ad hoc structures, our sociotechnical study has demonstrated the existence of implicit and efficient organizational forms. Then, we tracked these components over time and identified potential guidelines to support the healthy evolution of these communities. Also, we studied the impact of the sentiments, expressed in the comments, of the developers on their productivity and the quality of revision of the source code. Finally, we conducted an analysis of the propagation and density of knowledge flows within interaction networks in order to improve the performance of OSS developers.

Empirical studies conducted in this research are based on a methodology for extracting historical data of activities related to the most famous, used and existing open source projects on GitHub and Gerrit platforms. After data processing, we performed the analyzes based on Social Network Analysis (SNA) methods, coupled with temporal analyzes, sentiment analysis and other appropriate statistical calculations in order to respond to our predefined research questions.

Keywords: Open Source Software Communities; Sociotechnical interactions; Collaborative networks; Dynamic Networks; Network mapping; Social network analysis; Sentiment Analysis; Knowledge Transfer.

ملخص

يستقطب نموذج تطوير البرمجيات مفتوحة المصدر المزيد والمزيد من الاهتمام من الباحثين والشركات من خلال كفاءتها وهيكلها التنظيمي اللامركزي. في هذا النموذج، يقوم مجموعة من المطورين بتنسيق أنشطتهم من أجل إنتاج برنامج مجاني لخدمة ملايين المستخدمين مع إمكانية إعادة استخدام كود المصدر الخاصة بهم بواسطة مطورين آخرين. يعتبر هذا المجتمع نظامًا معقدًا وتنظيمه ذاتيًا يمكن لأعضائه التعاون والمساهمة في نفس المشروع على الرغم من أنهم ينتمون إلى مواقع وثقافات وتجارب وسياقات مختلفة. هذا التعاون ممكن نظرًا لوجود منصات معينة، أكثرها استخدامًا هي GitHub و Gerrit.

تستمد هذه الهياكل الافتراضية قوتها، ليس فقط من المهارات التقنية للأعضاء، ولكن أيضًا من التفاعلات المختلفة داخل المجتمع. الهدف الرئيسي من هذا البحث هو استكشاف التفاعلات داخل مجتمعات المشاريع مفتوحة المصدر، من أجل تحسين إنتاجية أعضائها وجودة منتجات البرمجيات. أولاً، لقد كنا مهتمين بالبعد الاجتماعي لهياكل مجتمعات المصادر المفتوحة، أظهرت دراستنا الاجتماعية والتقنية وجود أشكال تنظيمية ضمنية وفعالة. ثم تابعنا تطور هذه المكونات مع مرور الوقت وحددنا إرشادات محتملة لدعم التطور الصحي لهذه المجتمعات. بعد ذلك، درسنا تأثير المشاعر، المعبر عنها في التعليقات، للمطورين على إنتاجيتهم وعلى جودة مراجعة الكود. أخيرًا، أجرينا تحليلًا لانتشار وكثافة تدفقات المعرفة داخل شبكات التفاعل من أجل تحسين أداء مطوري البرمجيات مفتوحة المصدر.

تستند الدراسات التجريبية التي أجريناها في هذا العمل البحثي إلى منهجية لاستخراج البيانات التاريخية من الأنشطة المتعلقة بمشاريع المصادر المفتوحة الأكثر شهرة الموجودة على منصات GitHub و Gerrit. بعد معالجة البيانات ، أجرينا التحليلات المستندة إلى أساليب تحليل الشبكات الاجتماعية (SNA) ، بالإضافة إلى التحليلات الزمنية ، وتحليلات المشاعر وغيرها من الحسابات الإحصائية المناسبة من أجل الاجابة على أسئلة بحثنا المحددة مسبقا.

الكلمات المفتاحية: مجتمعات البرمجيات مفتوحة المصدر, التفاعلات الاجتماعية, الشبكات الديناميكية, تعيين الشبكة, تحليل الشبكات الاجتماعية , تحليل المشاعر, نقل المعرفة.

REMERCIEMENTS

Je remercie en premier lieu ALLAH le Tout-Puissant pour toute la volonté et le courage qu'il m'a donné pour l'achèvement de de ce travail.

Ce travail n'aurait pas été réalisé sans l'aide de nombreuses personnes. Il n'est pas possible de citer toutes les personnes qui l'ont influencé. Néanmoins, je voudrais exprimer ma gratitude à certaines personnes en particulier.

Tout d'abord, je voudrais remercier mes deux professeurs encadrants :

Professeur Mohammed Abdou Janati Idrissi, d'avoir fait confiance en moi et accepté que je fasse ma recherche au sein de son équipe. Je le remercie également pour ses précieux conseils et son suivi lors de la réalisation de ce travail de recherche.

Professeur Noureddine Kerzazi, qui je ne pourrai jamais assez remercier vu son soutien, sa patience et ses conseils avisés. J'ai beaucoup appris du travail avec lui dans ce domaine. Il a été un conseiller extraordinaire et m'a toujours motivé et encouragé dans les situations critiques.

Un grand merci au professeur Lamia Ben Hiba, pour tout le soutien et l'aide qu'elle m'a apportés durant cette période.

Un merci spécial à l'équipe de l'école polytechnique de Montérial Canada Foutse khomh et Gias Uddin. C'était un réel plaisir, avec beaucoup d'apprentissages, de collaborer avec eux.

J'exprime aussi toute ma gratitude aux membres du jury, pour l'intérêt et l'attention qu'ils ont accordés à ce travail de thèse, et pour avoir accepté de faire partie de ce jury.

Je suis vraiment reconnaissante à Imane Ettahiri et Mohammed Sayagh, qui ont aimablement accepté de réviser avec moi mon rapport de thèse. Je leurs souhaite une bonne chance dans leurs vies professionnelles ainsi que personnelles.

Un grand merci à ma famille. Je remercie mon mari Mohammed Albaz pour sa compréhension et son soutien durant ces années d'études, ma petite fille Salma à qui je dédie ce travail. Mes parents m'ont été d'une grande aide pendant mon doctorat et toutes mes études antérieures. Sans leur amour, soutien et encouragements continus, je ne serais jamais ce que je suis. Je remercie également mes frères et mes sœurs pour leurs soins constants. Mes remerciements vont également à ma belle-famille.

Mes remerciements s'adressent aussi à de nombreuses amies qui m'ont aidé par leurs conseils et leurs prières tout au long de mes travaux de recherche.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ABRÉVIATIONS

API	Application Programming Interface
DSO	Domain Sentiment Orientation
IDE	Environnement de Développement Intégré
JSON	JavaScript Object Notation
LOC	Lines Of Code
MVC	Model-View-Controller
OSS	Open source software
PMS	Propensity Score Matching
REST	Representational State Transfer
SNA	Social Network Analysis
SVM	Supervised Machine Learning
VCS	Version Control System

INTRODUCTION

Les logiciels open source (OSS) évoluent sans cesse depuis plus de deux décennies. Cela a commencé comme une approche révolutionnaire en matière d'ingénierie logicielle, produisant un logiciel librement et ouvertement disponible pour être édité, copié, même utilisé commercialement. Par rapport au développement de logiciels traditionnels, le développement de logiciels open source est unique au sens qu'il est autoorganisé par des développeurs autonomes.

En plus, le développement des projets open source a connu un succès exceptionnel. Plusieurs projets réalisés par la communauté des développeurs des projets open source existent et ont fondamentalement changé plusieurs domaines. Le système d'exploitation Linux et le serveur Web Apache sont les plus connus en raison de leurs intérêts et leurs tailles.

Dans tout système logiciel, certains contributeurs sont plus dans le cœur du projet que d'autres personnes. Ils sont les développeurs qui contribuent le plus au projet, décident des modifications à inclure ou à rejeter. Ce sont aussi l'ensemble des développeurs importants qui aident à faire évoluer le cœur technique d'un système logiciel. Dans cette thèse, nous utilisons le mot emprunté du lexique anglais « Core » pour définir ces développeurs principaux.

L'originalité des communautés de logiciels open source est liée à un mode de production collaboratif basé sur une participation d'acteurs dispersés géographiquement et utilisant l'internet comme le principal moyen de communication (Shirali-Shahreza et al. 2008) et généralement via des plateformes de collaboration comme GitHub et Gerrit. Ces aspects uniques des communautés OSS ont inspiré des études sur les motivations des participants individuels, la gouvernance des projets de logiciels open source (Capra et al. 2008), l'apprentissage organisationnel dans les projets OSS (Huntley 2003) et l'architecture du code source au sein des projets OSS (Conley et al. 2009). Cette originalité ainsi que la popularité et le succès de ce mode de développement a incité notre attention pour investiguer plusieurs aspects sociotechniques sur la collaboration au sein de ces communautés virtuelles. Nous avons ainsi émis une hypothèse globale que les interactions des développeurs open source autour des activités techniques sont un champ fertile pour mieux cerner les caractéristiques de ces communautés virtuelles.

Dans ce travail de recherche, nous avons mené un ensemble d'études empiriques afin d'étudier les interactions au sein de communautés OSS de différents angles de vues. Notre objectif est de

comprendre les spécificités des communautés open source en termes de structure organisationnelle, évolution temporelle, expression des sentiments et transfert des connaissances et ainsi pouvoir mener les actions correctives pour le bienêtre de ces communautés en vue d'améliorer leur productivité.

1. Contributions de la thèse

Les contributions de ce travail de recherche se divisent en quatre axes principaux.

Axe 1, cet axe concerne la structure organisationnelle des communautés OSS. Nous allons proposer une approche basée sur l'analyse des graphes de collaboration, sur l'activité « modification du code », pour classifier les contributeurs open source selon leur position sociale au sein des réseaux sociaux de collaboration. Cette approche a révélé, sur la base des projets étudiés, que les contributeurs open source sont classés en trois catégories (Core, Transitoire, Périphérie) selon leur engagement au projet.

Axe 2, ce deuxième axe s'intéresse particulièrement à la dynamique temporelle associée à l'évolution des rôles de la périphérie au Core et vice versa. Nos principaux objectifs sont d'abord de fournir à la communauté des logiciels libres une meilleure compréhension des activités de collaboration et des directives potentielles pour que les nouveaux arrivants s'engagent davantage dans le projet, ainsi que de comprendre comment la coordination des développeurs de logiciels peut être surveillée et améliorée dans un environnement open source peut aider à empêcher l'exode des contributeurs.

Axe 3, cet axe se focalise sur les interactions textuelles entre les membres de la communauté afin de sensibiliser aux rôles des sentiments au sein du développement open source. Nous étudions de manière empirique l'impact du sentiment exprimé dans les commentaires des développeurs sur la productivité des contributeurs open source. Nous avons constaté que les contributeurs expriment fréquemment des sentiments positifs et négatifs; les sentiments exprimés diffèrent entre les contributeurs en fonction de leur position dans le réseau social de collaboration; les sentiments exprimés par les contributeurs tendent à être neutres à mesure qu'ils passent du statut de nouveau venu dans un projet à celui de contributeur principal. En plus, nous avons constaté que les sentiments exprimés par les développeurs ont un impact sur le temps nécessaire pour fixer une révision et son résultat final.

Axe 4, ce dernier axe traite le transfert et la propagation des connaissances entre les contributeurs open source selon un point de vue social à l'aide de l'Analyse des Réseaux Sociaux. Nous adoptons une approche de chevauchement de réseaux de collaborations relatifs

à plusieurs activités (ex., modification du code, commentaires, révision du code source). Nous avons ainsi pu identifier des modèles de comportement entre l'équipe Core et les contributeurs périphériques liés aux activités dans les projets OSS étudiés. Les résultats montrent que les demandes de révision vont des contributeurs principaux aux contributeurs périphériques ; les commentaires sur les révisions de code sont transmis en boucle continue des équipes Core aux périphériques et inversement ; et les principaux contributeurs tirent parti de leurs connaissances techniques pour accroître leur notoriété en jouant le rôle de courtier en communication.

Dans un contexte plus large, les résultats des études empiriques menées relativement aux quatre axes cités fournissent des informations sur l'organisation de la communauté open source, les modèles temporels communs des communautés de logiciels open source, les effets des sentiments exprimés sur le processus de révision de code, et la propagation des connaissances dans les projets en croissance permettant ainsi d'élargir la compréhension de la collaboration et la dynamique sociotechnique au sein de la communauté open source. Nos analyses ont des implications sur l'amélioration de la collaboration entre les contributeurs au sein des communautés virtuelles de logiciels open source qui poussent les équipes vers la performance et la qualité des logiciels.

2. Approche

Pour entamer nos études empiriques, nous utilisons des données longitudinales de plusieurs projets open source de deux différentes sources de données GitHub et Gerrit. Ces sources de données offrent un environnement de données gratuit, abondant et assez riche pour avoir un aperçu sur plusieurs caractéristiques des communautés open source. Nous avons, en premier lieu, piloté une phase d'extraction et de préparation des données. Ensuite nos traitements et analyses des données sont basées sur l'analyse des réseaux sociaux (SNA) couplée avec des analyses temporelles, des analyses de sentiments et des analyses statistiques appropriées selon la dimension d'analyse envisagée. Une étape d'interprétation des résultats est aussi considérée afin de pouvoir répondre à nos questions de recherches.

3. Organisation de la thèse

Le rapport de cette thèse s'articule autour de cinq chapitres : Le Chapitre 1 définit le contexte global de ce travail de recherche à travers une initiation au développement open source ainsi qu'un aperçu sur les communautés open source. Ensuite, nous détaillons les défis majeurs de recherche relatifs aux communautés open source suivi par une exposition des objectifs de notre

travail de recherche. Puis nous décrivons l'approche suivie tout au long de nos études empiriques. Le chapitre s'achève par un sommaire de nos contributions.

Le chapitre 2 est dédié à notre contribution autour de la structure organisationnelle de la communauté open source. D'abord, nous commençons par définir nos motivations et objectifs suivis par une revue de la littérature afin de mieux positionner notre contribution. Puis, nous décrivons l'extraction et la préparation des données GitHub. Ensuite nous présentons notre modélisation en réseaux sociaux utilisée pour détecter la structure typique des communautés OSS. Enfin nous présentons les résultats de l'étude suivis d'une discussion sur les contributions et une synthèse du travail.

Le chapitre 3 touche la dynamique temporelle des contributeurs open source. Ce chapitre commence d'abord par établir les objectifs de l'étude puis une lecture des travaux antérieurs. Après un rappel sur les données des projets sujets d'études, nous présentons notre modélisation dynamique utilisée pour étudier l'évolution temporelle de la communauté OSS. Enfin, nous présentons les résultats de cette étude suivis par une discussion sur les résultats et une synthèse de l'étude.

Le chapitre 4 présente notre étude sur l'effet des sentiments exprimés par les contributeurs open source sur l'activité de révision du code. D'abord nous présentons nos motivations et objectifs, puis une description du contexte de l'étude suivie par une revue de la littérature. Ensuite, nous présentons la préparation des données Gerrit et le cadre de l'analyse. Enfin, nous détaillions les résultats de cette étude suivis par une discussion sur les résultats et une synthèse du travail.

Le chapitre 5 présente notre contribution autour de l'investigation du transfert des connaissances au sein des communautés open source. Le chapitre est organisé en six parties. D'abord nous présentons nos motivations et objectifs, puis l'état de l'art suivi par l'approche méthodologique. Ensuite, nous présentons les résultats de cette étude suivis par une discussion. Enfin, le chapitre est conclu par une synthèse de l'étude.

Le rapport est achevé par une conclusion générale dans laquelle nous résumons nos résultats et nos contributions, nous discutons leurs implications pratiques, et nous proposons plusieurs perspectives à notre travail de recherche.

CHAPITRE 1. CONTEXTE

Ce chapitre présente le contexte de cette thèse ainsi qu'un aperçu général sur les objectifs attendus et la méthodologie adoptée. En premier lieu, nous présentons le modèle du développement open source (section 1.1) et nous fournissons un aperçu sur les communautés open source (section 1.2) suivi des défis majeurs de recherche au sein des communautés open source (section 1.3). En deuxième lieu, nous explicitons nos objectifs de recherche (section 1.4). Finalement, nous présentons l'approche empirique suivie lors de nos études dans la section 1.5 et listons nos contributions dans la section 1.6.

1.1. Développement open source

Le développement des projets open source est conduit par des communautés de contributeurs répartis à l'échelle du globe. Ce modèle de développement a connu un franc succès suite aux réussites de grands projets tels que Linux, Firefox, Apache, etc. En effet, le développement open source (OSS) est un capital social construit par une communauté de contributeurs distribués à travers le globe (VonHippel et al. 2003, Crowston et al. 2012). Ce capital social est créé et maintenu à travers des échanges techniques et sociaux entre les développeurs qui interagissent au sein d'une communauté pour produire du logiciel (Daniel et al. 2018). La réussite de milliers de projets OSS n'est pas à démontrer, c'est un phénomène qui est là pour rester (Ghosh et al. 2002). Les projets OSS couvrent tous les niveaux de l'environnement informatique : système d'exploitation (ex., Linux, BSD), infrastructure Internet (ex., Apache, Spark, etc.), applications de bureau (ex., OpenOffice, etc.) et applications Internet (ex., Firefox, etc.).

Dernièrement, plusieurs organisations ont tenté de tirer profit des pratiques de développement utilisées dans le cadre des projets OSS réussis. Ceci est soit en appliquant ces pratiques directement au sein de leurs organisations ou en partageant leur code source propriétaire pour plus de qualité et de productivité (Dinkelacker et al. 2001, Melian et al. 2008, Wesselius 2008, Hauge et al. 2010). Le développement open source offre de nouvelles fonctionnalités pour développer et réutiliser des logiciels dépassant les avantages offerts par les approches propriétaires du développement et de la réutilisation de logiciels (Bosch 2009, Franco-Bedoya et al. 2014).

Le modèle de développement des logiciels open source a démontré son succès par rapport au modèle industriel (Paulson et al. 2004, Raghunathan et al. 2005, Khanjani et al. 2011). Le développement open source propose un nouveau modèle pour développer et réutiliser des produits logiciels dépassant les avantages offerts par les approches propriétaires du développement et de la réutilisation de logiciels (Bosch 2009, Franco-Bedoya et al. 2014). Le modèle open source encourage le développement continu qui est indépendant des fonctionnalités (Scacchi 2004, Scacchi et al. 2006). Aussi, l'open source promet une meilleure qualité, un coût quasiment nul et met fin au monopole des grands fournisseurs de logiciels payants (Opensource.org 2018). En plus, dans le livre " The Cathedral & the Bazaar " (Raymond 1999), l'auteur compare le développement de logiciels propriétaires à celui des OSS : Le premier est un développement hautement organisé basé sur une structure caricaturée

de Cathédrale alors que le second a été qualifié de bazar chaotique. Le modèle cathédral est basé sur une planification et une exécution centralisée autour d'un modèle de processus à base de phases et d'itérations. À l'inverse, le modèle « Bazar » repose sur une planification et une exécution décentralisée, où tous les contributeurs peuvent proposer de nouvelles fonctionnalités et corriger des bogues (Raghunathan et al. 2005, Khanjani et al. 2011), ce qui peut favoriser une progression rapide du projet (Paulson et al. 2004). Un facteur indéniable pour la réussite d'un projet OSS est la création d'une communauté capable d'attirer et de retenir de nouveaux contributeurs (Hannemann et al. 2013).

Contrairement aux organisations de développement commerciaux, les développeurs de projets OSS jouissent d'une plus grande liberté pour s'auto organiser. En effet, ces développeurs peuvent décider comment ils souhaitent contribuer à l'évolution d'un projet. Par conséquent, l'activité de contribution, ainsi que les modèles de collaboration et d'interaction diffèrent des modèles et processus de développement classiques (Feller et al. 2002, Zhao et al. 2003, Dietze 2005).

Le nombre croissant de projets de développement logiciels, à grande échelle, exige la participation de groupes de plus en plus importants qui doivent s'organiser au sein de communautés. Toutefois, on connait peu sur la création et l'évolution de ces communautés Ainsi, une compréhension de l'organisation des membres au sein de ces communautés est essentielle pour le développement soutenu de ce modèle.

1.2. Communautés open source

Une communauté open source est un groupe de développeurs qui partagent un intérêt commun pour un projet donné et qui interagissent régulièrement les uns avec les autres pour partager des connaissances et collaborer à l'évolution de ce projet (Stewart et al. 2001, Crowston et al. 2005, Hinds et al. 2006, Ye et al. 2008, Carlson 2015). La connaissance de l'organisation des communautés et des interactions entre ses membres est cruciale pour comprendre la dynamique de collaboration au sein des projets OSS.

L'organisation des membres d'une communauté OSS est souvent décrite dans la littérature comme une structure hiérarchique ou analogue à celle d'un oignon (Cox 1998, Gacek et al. 2004, Moon et al. 2005). Au centre se trouve un noyau de développeurs expérimentés (i.e., Core), qui contribuent à l'essentiel du code et supervisent la conception et l'évolution du projet et qui maintiennent 80% du contenu (Ye et al. 2008) et des contributeurs périphériques qui contribuent occasionnellement au projet (Mockus et al. 2000).

Au premier abord, on peut croire que les nombreux contributeurs périphériques ne sont pas utiles à l'avancement du projet vu leur taux de productivité, leur volatilité et la perte du capital de connaissance qui en découle (Terceiro et al. 2010). Cependant, des études ont rapporté leur utilité et le fait qu'ils soient tout aussi essentiels au succès des projets OSS que les développeurs qui font partie du Core (Raymond 1999). Sans les membres périphériques, la survie et l'évolution des communautés est compromise. De plus, la périphérie est la source de nouveaux membres Core (von Krogh et al. 2006) ainsi, maintenir une forte périphérie est important pour la réussite à long terme d'un projet.

Les interactions sociotechniques entre les développeurs open source est un élément clé de l'efficacité des projets (Barcellini et al. 2014) et de l'amélioration de la participation et de la collaboration de groupes en ligne (Crowston et al. 2012). La communication asynchrone est une exigence générale pour les groupes distribués géographiquement (Crowston et al. 2017) ou les membres Core et périphériques doivent communiquer et synchroniser leurs efforts pour réaliser les tâches (Park 2008).

La littérature reconnait que les interactions entre les contributeurs open source sont la clé de la compréhension de l'organisation de ces communautés, notre travail tente d'investiguer ces interactions selon plusieurs vues d'abstraction.

1.3. Défis de la recherche sur les communautés open source

Notre investigation de la littérature sur les interactions au sein des communautés open source (Chawner 2004, Tuma 2005, Weiss et al. 2006, Stol et al. 2009, Kaur et al. 2014, Brunswicker et al. 2017) a suscité un nombre de questionnements majeurs :

- Comment ces communautés sont-elles organisées ?
- Quel est le mode d'évolution de ces communautés dans le temps ?
- Quel est l'effet des sentiments exprimés dans les interactions textuelles des contributeurs OSS?
- Comment se fait le partage des connaissances au sein de ces communautés ?

Pour répondre à ces questionnements nous avons identifié quatre défis de recherche concernant les communautés open source.

1) Le premier défi concerne l'organisation des membres au sein d'une communauté open source. Les principaux résultats liés à la structure des communautés d'OSS révèlent que le groupe clé (le groupe principal de développeurs) est responsable non seulement de la majorité

des contributions, mais également de la promotion de la participation des autres membres du groupe (Crowston et al. 2006). D'autre part, comprendre la structure organisationnelle est important pour comprendre l'essence des pratiques des équipes (Amrit et al. 2010).

Les recherches antérieurs ont prouvé que des relations prévisibles existent entre la structure du code et la structure sociale de l'équipe de développement (Crowston et al. 2005), aussi une meilleure compréhension de la structure sociale peut améliorer la planification du développement (Scacchi 2005). La structure sociale est également utile dans la gestion des risques, car elle permet d'évaluer des questions telles que "qui sont les développeurs clés dont le retrait pourrait menacer la continuité du projet ?"(Crowston et al. 2005).

Afin de comprendre la structure social d'une communauté, il est utile de considérer trois aspects d'une structure sociale (Hinds et al. 2006): les individus, leurs actions et leurs interactions. La considération de ces trois volets permet d'obtenir une image du groupe : est-il petit, grand, constant, en croissance ou en diminution ? Il est possible d'approfondir la compréhension des schémas dans les actions de ces individus : qui fait le travail ? Est-il réparti uniformément entre les membres du projet ou concentré en sous – groupes ? Existe-t-il une spécialisation, existe-t-il des rôles spécifiques ou les membres du projet contribuent-ils tous de la même manière ?

2) Le second défi, après la compréhension de la structure organisationnelle des communautés OSS, est de comprendre son évolution à travers le temps. L'étude de l'évolution des structures de projet open source au fil du temps permettrait de mieux comprendre comment les projets se développent et comment ils évoluent à travers le temps (Syeed et al. 2013). En générale, l'évolution d'une communauté de logiciels open source dépend des changements de l'engagement de ses membres (Robles et al. 2009). Comparer l'évolution de la structure à intervalle de temps régulier permet de suivre la stabilité et la permanence du groupe de développeurs les plus actifs indispensables pour l'évolution et la durabilité du projet (Robles et al. 2009, Joblin et al. 2016, Joblin 2017). D'autre part, le développement durable d'une communauté exige que le groupe d'utilisateurs actifs soit continuellement nourri avec de nouveaux arrivants qui acquièrent une expertise grâce à leurs participations (Zhang et al. 2001). La dynamique de ce processus doit être caractérisée afin de comprendre l'évolution des communautés à travers l'exploration du changement des rôles au fil du temps.

3) Le développement de logiciels open source est fortement dépendant des efforts et des interactions humaines ainsi plus susceptible aux émotions des praticiens (Guzman et al. 2014). Cette activité hautement collaborative ou les participants communiquent à travers les listes de diffusion, les forums, les référentiels de suivi des changements du code pour évoluer les projets. Ces artefacts de collaboration sont des canaux de communication où les équipes de

développement peuvent exprimer leurs sentiments concernant, à titre d'exemple, leur satisfaction à l'égard du projet ou leurs difficultés lors de l'exécution de certaines tâches, leurs avis sur des modifications proposées. Ainsi, le défi suivant est d'investiguer l'expression des sentiments par les contributeurs dans leurs discussions et l'impact de ces sentiments sur leurs productivités.

4) Le dernier défi concerne le partage des connaissances au sein des communautés OSS. Ces communautés ont été identifiées comme des écosystèmes de partage de connaissances dans lesquels les contributeurs apprennent les uns des autres (Raymond 1999, von Krogh et al. 2003). Dans ces environnements, non commerciaux, les individus partagent librement leurs connaissances avec les autres membres de la communauté afin de développer de nouveaux logiciels ou les améliorés. Les communautés open source fournissent un excellent contexte pour enquêter sur le partage des connaissances. En effet l'accès ouvert au code source et la communication de toute modification du code, ainsi que la collaboration pour améliorer l'existant, assurent un transfert rapide des connaissances et permettent de bénéficier du potentiel des anciens membres plus compétents. Par conséquent, cette distribution instantanée de connaissances diversifiées ne fait qu'accroître la capacité à innover (Kilamo et al. 2014).

Sur la lumière des différents défis de recherche sur les communautés opens source, la section suivante identifiera nos objectifs de recherche.

1.4. Objectifs

Le but de notre thèse est d'améliorer la perception du rôle important des interactions entre les membres de la communauté open source, ceci à travers la compréhension du fonctionnement de cette communauté. Notre objectif principal est de comprendre, à travers des études empiriques, les modes d'interactions au sein des communautés open source puisque le succès des projets open source dépend du succès de ses communautés.

On s'intéresse aux interactions techniques et sociales au sein des dites communautés afin de :

- **Comprendre la structure organisationnelle des communautés OSS** : Notre objectif est d'identifier les différentes composantes qui organisent les communautés open source.
- **Comprendre la dynamique temporelle associée avec l'évolution des rôles de la Périphérie au Core et vice versa** : Suite à la compréhension des différents rôles des collaborateurs à un projet open source, notre deuxième objectif est d'identifier comment ces différentes composantes évoluent dans le temps. L'objective principale de cette

étude est de doter les communautés OSS d'une meilleure compréhension des activités de collaboration qui aident les nouveaux arrivants à atteindre le cœur du projet et ainsi jouer un rôle efficace au sein du projet. Aussi comprendre comment la coordination des développeurs de logiciels peut être surveillée et améliorée dans un environnement open source pour aider à empêcher le désengagement des contributeurs du projet.

- **Sensibiliser aux rôles des sentiments exprimés par les contributeurs open source** : Après avoir identifié les différentes catégories de développeurs de projets open source et leurs évolutions dans le temps, notre objectif suivant est d'étudier la qualité d'interactions entre ces composantes en termes des sentiments exprimés. Notre objectif global est de comprendre l'impact des sentiments exprimés, dans les commentaires, sur le temps et le résultat de la pratique de révision du code source.
- **Investiguer le transfert des connaissances entre les contributeurs open source** : notre objectif pour cette dernière étude est d'explorer la propagation et la densité des flux de connaissances au sein des réseaux de contributeurs open source. Pour une analyse plus fine, nous étudions la relation existante entre la position du contributeur dans le réseau social de collaboration et le partage des connaissances. Ces connaissances sont transférées à travers les interactions entre les membres de la communauté.

Afin de réaliser ces différents objectifs nous avons opté pour une approche empirique décrite dans la section suivante

1.5. Méthodologie

Notre projet de recherche vise à comprendre la dynamique des interactions au sein des communautés open source. L'adoption du modèle de développement open source, qui est de plus en plus répandu, nécessite la compréhension des interactions entre les membres des communautés pour s'assurer du développement durable de ce capitale social (Daniel et al. 2018). Dans cette perspective, nous avons opté pour une approche empirique dans le but de produire de nouvelles connaissances à partir des données empiriques qui permettront aux chefs de projets open source de mieux comprendre leurs contextes, et ainsi définir des stratégies pour améliorer leurs projets.

La Figure 1-1 illustre l'approche générale et les liens entre les études empiriques que nous avons réalisées. Nos sources de données sont deux plateformes, les plus utilisées, de collaboration open source : GitHub et Gerrit. Contrairement au contexte industriel où les entreprises ne souhaitent pas mettre à disposition les dépôts logiciels contenant ces données, les historiques

de développement de tous les projets open source sont accessibles librement en ligne. Ainsi nous tirons profit de l'abondance, la richesse et la disponibilité de ces données pour extraire et préparer les historiques sur les interactions entre les contributeurs open source. Ensuite, nous avons modélisé ces interactions en réseaux sociaux de collaboration ou les nœuds sont les contributeurs et les arcs désignent le nombre d'interactions entre une paire de contributeurs. L'objectif de cette modélisation est d'avoir un aperçu concret de ces interactions et aussi calculer les différentes métriques d'analyse des réseaux sociaux SNA (Wasserman et al. 1994) qui permettent de caractériser le réseau entier ainsi que les nœuds du réseau. Ces métriques ont été ensuite couplées, selon l'objectif, avec d'autres techniques d'analyses en l'occurrence : Mapping des réseaux (c-à-d. l'étude de la connectivité physique de plusieurs réseaux), l'analyse temporelle (c-à-d. une série de valeurs numériques représentant l'évolution d'une quantité spécifique au cours du temps), autres calculs statistiques (ex. mesures des corrélations, tests statistiques... etc.), les outils de détection des sentiments (c-à-d. des algorithmes de détection des sentiments dans le texte) pour mener quatre études empiriques. Par exemple, pour étudier l'organisation des communautés open source (CHAPITRE 2), les métriques SNA sont couplées avec les calculs statistiques. Les chapitres suivants décrivent en détails chacune de nos approches.

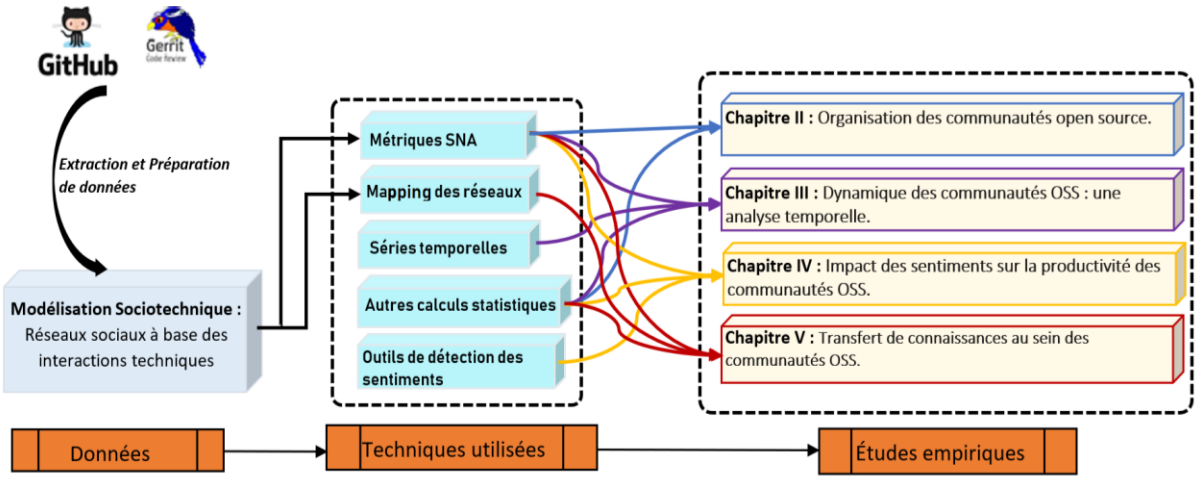


Figure 1-1 : Approche générale d'étude de la dynamique d'interactions entre les contributeurs OSS.

La section suivante décrira un sommaire des contributions de notre projet de recherche.

1.6. Contributions

Notre première étude empirique (CHAPITRE 2) propose une approche pour dévoiler la structure organisationnelle au sein des communautés open source. Cette approche est basée une

méthode de classification et des outils SNA pour catégoriser les contributeurs OSS selon trois positions sociales ; Core, Transitoire et Périphérique. Notre étude a amélioré les résultats des études antérieures (Geldenhuys 2010, Cucuringu et al. 2016) ayant classifié les contributeurs en deux classes. En effet, nous proposons une classification plus raffinée à travers l'introduction de la zone transitoire par laquelle passent les nouveaux arrivants avant d'atteindre le cœur du projet.

La deuxième contribution (CHAPITRE 3) consiste en la compréhension de la dynamique collaborative au sein des projets OSS. En particulier, nous avons quantifié l'évolution de l'engagement des contributeurs open source à travers le suivi du changement de leurs positions au sein des réseaux sociaux de collaboration. Notre analyse a démontré que la modification du code source est l'activité la plus importante pour atteindre le cœur du projet. Aussi, notre analyse permet aux chefs de projets de prédire le désengagement des contributeurs clés dans leur projet.

Le CHAPITRE 4 concerne une étude empirique liée à l'investigation de l'effet des sentiments exprimés par les intervenants open source sur la performance de la pratique de révision de code. Nous nous sommes intéressés aux sentiments exprimés dans les révisions du code. Toutefois, le débat déclenché par lin et al. (Lin et al. 2018) sur la fiabilité des outils de détection des sentiments utilisés dans le contexte de l'ingénierie logiciel nous a amené à faire une étude comparative sur la précision et la fiabilité des outils existants. Ainsi, notre première contribution est une étude comparative de la précision d'un benchmark d'outils présents dans la littérature. Une fois l'outil sélectionné, nous avons mené une étude sur 5 millions de commentaires dans le but d'analyser l'impact des sentiments sur une pratique de développement importante dans l'environnement OSS. Nous avons constaté que l'expression d'un sentiment positif lors de la révision du code source réduit le délai d'achèvement de la révision et augmente les chances que cette révision soit acceptée. En fin, nous avons constaté que les contributeurs open source expriment différents sentiments selon leurs positions dans le réseau social de collaboration particulièrement les contributeurs Core exprimant généralement des sentiments neutres.

La dernière étude empirique (CHAPITRE 5) consiste en l'investigation du transfert de connaissances entre les membres des communautés open source. Étant donné que les réseaux sociaux contiennent de nombreuses informations pouvant être utilisées à d'autres fins, notre étude traite le partage et la propagation des connaissances selon une perspective sociale en utilisant les métriques SNA. Ainsi, nous avons démontré que les contributeurs Core s'appuient sur leurs notoriétés et leurs connaissances techniques pour jouer le rôle de courtiers en

communication à travers leurs commentaires. Aussi, les discussions autour des modifications du code source proposées sont une boucle continue d'échange entre les membres Core et ceux de la périphérie et inversement. Une implication pratique de cette contribution servirait à assigner la tâche de révision du code source aux contributeurs les plus adéquats.

Conclusion

Ce premier chapitre a présenté le contexte général de notre travail de recherche. En effet, le développement open source est un phénomène révolutionnaire qui a changé la vision de construction, de déploiement et d'évolution des projets de logiciel. Le succès d'un projet open source dépend principalement de l'évolution de la communauté qui le développe. L'objectif de ce travail de recherche est d'examiner les interactions au sein des communautés open source en vue d'améliorer à la fois la productivité des contributeurs et la qualité des produits logiciels. Ainsi, nous avons définis quatre défis majeurs de recherche dans les communautés open source : (1) comment ces communautés sont-elles organisées, (2) comment évoluent-elle dans le temps, (3) quel est l'effet des sentiments exprimés dans les interactions textuelles des contributeurs OSS sur leur productivité et enfin (4) comment le partage des connaissances est effectué au sein de ces communautés ?

Pour répondre aux défis identifiés, nous avons mené quatre études empiriques sur des projets sélectionnés dans GitHub et Gerrit. Nous avons analysé les interactions sociotechniques des contributeurs à l'aide de l'analyse des réseaux sociaux ainsi que d'autres techniques d'analyse pour répondre aux questionnements posés. En particulier, nous proposons une approche pour détecter la structure organisationnelle des communautés open source, nous poursuivons l'évolution temporelle de cette structure organisationnelle depuis la création du projet, nous investiguons la qualité des interactions sociotechnique en termes d'expression des sentiments par les contributeurs open source dans leurs discussions autour de l'activité de révision du code source, et enfin, nous avons enquêté sur le partage des connaissances entre les membres d'une communauté open source.

Le chapitre suivant décrira notre première étude empirique concernant la structure organisationnelle des communautés OSS.

CHAPITRE 2. ORGANISATION DES COMMUNAUTÉS OPEN SOURCE

Les informations sur l'organisation structurelle des communautés de logiciels open source sont essentielles pour comprendre la dynamique de collaboration au sein des projets de logiciels. À la différence des organisations de développement de logiciels propriétaires où les structures d'organisation sont explicites et les rôles sont clairement définis (Ex., Architecte, Développeur, Analyste, Testeur, etc.), les structures d'organisation pour la production de logiciels OSS sont implicites. Dans ce chapitre nous nous intéressons à la détection de la structure organisationnelle typique des communautés open source[1]. À travers une étude empirique sur cinq grands projets GitHub, nous proposons une approche basée sur l'analyse des réseaux sociaux pour classifier les contributeurs appartenant à une communauté open source selon leurs positions au sein des réseaux d'interactions. La première section du chapitre explique nos motivations et objectifs, ensuite nous rapportons la revue de la littérature dans la section 2.2. La section 2.3 présente notre source de données à étudier et l'approche adoptée d'extraction de ces données. La section 2.4 décrira notre modélisation en réseaux sociaux. Puis nous présentons nos résultats dans la section 2.5. Enfin la section 2.6 présente nos contributions et les mises en garde pour les limites de cette étude avant de conclure notre étude.

[1] *Le matériel de ce chapitre est la version française d' une section du papier intitulé «**From Periphery to Core: A Temporal Analysis of GitHub Contributors' Collaboration Network**» (El Asri et al. 2017), présenté dans la 18th édition du « Working Conference on Virtual Enterprises -Collaboration in a Data-Rich World. »*

2.1. Motivations et objectifs

Comme nous l'avons souligné dans le premier chapitre, un des grands défis pour le développement durable des communautés OSS est lié à l'aspect organisationnel. Ceci est dû principalement aux caractéristiques inhérentes des projets open source tel que le fait qu'ils soient développés par des contributeurs géographiquement répartis, organisés dans des communautés basées sur internet, et qui collaborent volontairement pour développer des solutions logicielles (Bird et al. 2008). L'efficacité organisationnelle nécessite la collaboration et la coordination entre les membres d'une même communauté et repose à la fois sur les interactions techniques et sociales (Bird 2011).

Les contributeurs Core et périphériques sont impliqués différemment dans les aspects collaboratifs du projet. Il est bien connu que les développeurs principaux (Core) jouent généralement un rôle essentiel dans le développement de l'architecture du système (Geldenhuys 2010). Ils se caractérisent par une participation prolongée, cohérente et intensive au projet. Ils possèdent souvent une connaissance approfondie de la conception du système et une forte influence sur les décisions du projet (Xu et al. 2005). En revanche, les développeurs périphériques se caractérisent par une participation irrégulière et souvent de courte durée au projet. En plus, Terceiro et al. (Terceiro et al. 2010) ont découvert que les développeurs Core apportent des modifications au code source avec moins de complexité que les développeurs périphériques. Par conséquent, une opérationnalisation core-périphérie valide et fiable est cruciale pour tester les preuves empiriques des théories proposées concernant les aspects collaboratifs du développement logiciel (Ex, le partage de connaissances, la coordination, la détection et la résolution des bogues, etc.), qui impliquent les membres Core différemment que les membres périphériques (Herbsleb et al. 2003).

À l'aide d'une étude empirique longitudinale sur la structure organisationnelle de cinq projets open source bien connus, nous cherchons à répondre à la question de recherche suivante :

RQ : Quelles sont les composantes qui organisent les communautés open source ? Les développeurs open source sont souvent classés en fonction de la dichotomie des rôles Core et périphérie (Crowston et al. 2017). Les opérationnalisations basées sur de simples comptes d'activités de développeur (par exemple, le nombre de commits) sont les plus utilisées pour classifier les contributeurs. Cependant cette approche n'est pas toujours valide. Ainsi, l'objectif de cette question de recherche est de proposer une approche basée sur les interactions sociales

ainsi que techniques entre les membres de la communauté pour détecter sa structure organisationnelle.

2.2. État de l'art

Des travaux précédents ont démontré que 80% de fonctionnalités sont développées par 20% des participants (Mockus et al. 2000, Mockus et al. 2002). Ce constat a été validé par Geldenhuys (Geldenhuys 2010) en se basant sur l'analyse des métriques d'activités à savoir : le nombre des contributions, la durée d'activité dans le projet et le nombre de modifications des fichiers code source. L'auteur a constaté que 80% de tous les commits étaient effectués par 3,1% à 8,9% des développeurs. Dans une étude plus vaste, Singh et al. (Singh et al. 2011) ont examiné 205 projets et ont prouvé que, en moyenne, 21% des top développeurs contribuent à 81% du code source. Cette approche est la plus célèbre cependant elle présente la faiblesse de ne pas prendre en considération les interactions sociales entre les contributeurs autour des activités de collaboration.

D'autres études ont porté sur les caractéristiques et le comportement des principaux développeurs du point de vue de l'analyse de réseau social (Mockus et al. 2002, Toral et al. 2010) (c.-à-d., modéliser les interactions entre les développeurs open source au sein du même projet en un réseau social). Joblin et al. (Joblin et al. 2017) ont identifié les développeurs Core selon leurs fortes connexions avec d'autres membres Core et moins fortes interactions avec la périphérie, cependant les contributeurs périphériques ont moins de liens d'interactions puisqu'ils ne sont engagés que dans des modifications mineures dans le projet. Ces résultats ont été prouvés par Madey et al. (Madey et al. 2002) qui ont utilisé les méthodes SNA pour modéliser les développeurs OSS. Ils ont modélisé les communautés de SourceForge.net en tant que réseaux collaboratifs et ils ont constaté que la petite fraction des développeurs ayant un grand nombre de liens de collaboration peut s'expliquer par la tendance des personnes à collaborer avec des membres Core. Bien que ces études aient examiné les interactions sociotechniques des membres de la communauté pour caractériser leurs rôles, elles ne fournissent pas de méthode permettant de distinguer avec précision les membres Core et ceux périphériques.

L'une des études pertinente dans ce contexte est celle de Bosu et Carver (Bosu et al. 2014) qui ont proposé une classification des contributeurs OSS en Core et périphérie basée sur les métriques SNA. Certes, les auteurs dans ce travail ont présenté une approche de distinction

entre les deux composante Core et Périphérie cependant ils ont traité de manière légère la phase de validation de l'approche (voir section 2.5).

Le Tableau 2-1 présente un sommaire des études antérieures ayant traité la structure organisationnelle des communautés OSS avec quelques critiques à leurs égards. Notre approche s'inspire principalement du travail de Bosu et Carver (Bosu et al. 2014). Nous nous basons, comme les études utilisant les méthodes SNA, sur la modélisation de la communauté OSS en réseau social d'interactions vue son utilité pour catégoriser les contributeurs open source. Et contrairement aux travaux de ces études (à l'exception de celui de Bosu et al. 2014) nous caractérisons les rôles des contributeurs moyennant les métriques SNA et non seulement sur la base de visualisation de réseaux. En plus notre approche traite d'une manière approfondie la phase de validation pour remédier aux critiques reportées sur le travail de (Bosu et al. 2014).

Tableau 2-1 : Synthèse sur méthodes de détection de la structure des communautés OSS

Étude	Méthode	Critique
Mockus et al. 2000	Métriques d'activité	Ne prend pas en considération les interactions sociotechniques entre les membres de la communauté.
Mockus et al. 2002	Métriques d'activité	
Geldenhuys 2010	Métriques d'activité	
Singh et al. 2011	Métriques d'activité	
Mockus et al. 2002	SNA	La caractérisation des rôles des contributeurs est basée sur la visualisation et non sur le calcul des métriques SNA.
Toral et al. 2010	SNA	
Madey et al. 2002	SNA	
Bosu et al. 2014	Métriques SNA	Légère validation de l'approche.

2.3. Extraction et préparation des données GitHub

L'objectif de cette étude est la détection de la structure organisationnelle des communautés OSS. Nous avons étudié cinq projets OSS hébergés dans la plateforme GitHub[2] . GitHub est un outil gratuit pour héberger du code open et aussi un outil de développement collaboratif avec plus de 31 millions d'utilisateurs distribués dans 200 pays et plus de 96 millions projets (GitHub 2018). La plateforme GitHub a créé des gains d'efficacité et a aidé à améliorer le fonctionnement des professionnels du logiciel puisqu'elle permet non seulement la traçabilité

[2] https://github.com/

du développement d'un projet open source, mais aussi fournit une plateforme d'échange social entre les intéressés du domaine (Zagalsky et al. 2015).

La plateforme GitHub permet d'accéder aux métadonnées de ses projets hébergés via son API[3]. L'API GitHub est ouverte et très populaire parmi la communauté des développeurs de logiciels ce qui fait de GitHub le meilleur fournisseur de données brutes nécessaires pour analyser et mieux comprendre la dynamique derrière les communautés de développement OSS. La Figure 2-1 ci-dessous illustre notre méthodologie concernant la sélection et l'extraction de données. Elle comporte trois importantes phases : Exploration des projets GitHub, sélection des projets et extraction des données nécessaires.

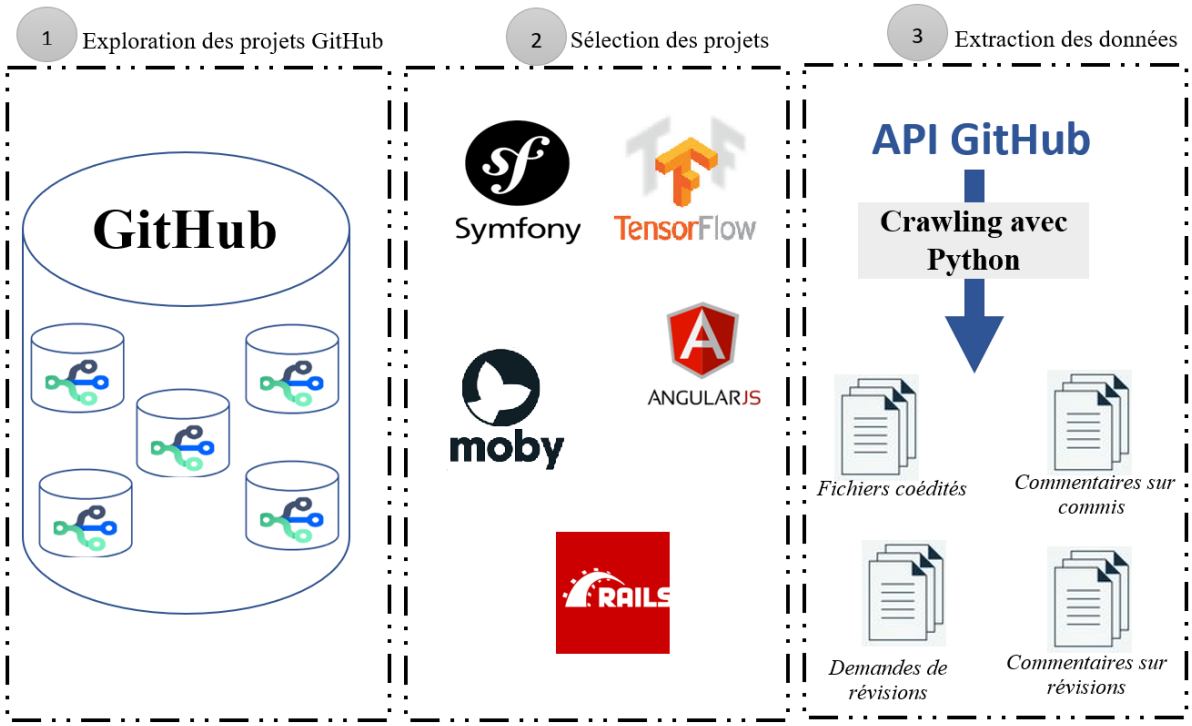


Figure 2-1 : Sélection et extraction des données GitHub

Phase1 : Exploration des projets GitHub

GitHub héberge des millions de projets open source dans une multitude de langages. Une exploration préliminaire des projets GitHub a été menée pour en savoir plus sur chaque projet et décider de l'utiliser ou non dans l'étude. Les informations sur le nombre de contributeurs participant au projet, le nombre de commits, le nombre de versions stables du projet et le nombre de commentaires écrits sur les listes de discutions donnent une idée sur la dimension

[3] https://developer.github.com/v3/

des projets et leur pertinence à utiliser dans l'étude des interactions au sein des communautés open source. Le site GitHub classe les projets selon le nombre d'étoiles de la part des utilisateurs et fournit une liste des projets les plus populaires[4]. Cependant, une exploration manuelle des projets les plus populaires nous permet d'avoir un aperçu sur les paramètres les plus importants de chaque projet à savoir le nombre de contributeurs, le total des commits, le nombre de révisions.

Phase 2 : Sélection des projets

Sur la base des résultats de l'exploration préliminaire, nous déterminons les critères suivants pour le choix des projets : (i) le projet devrait figurer parmi les 100 projets les plus populaires ; (ii) devrait être en cours de développement actif ; et (iii) impliquant au moins 250 contributeurs. Ainsi nous avons choisi cinq grands systèmes open source en évolution rapide de GitHub : AngularJS, Rails, Moby, Symfony et TensorFlow.

Le Tableau 2-2 résume les caractéristiques des projets sélectionnés, y compris la description du projet, le langage de programmation et un aperçue sur la date de création du dit projet.

Tableau 2-2 : Projets OSS sélectionnés

Projet	Description	Langage	Crée le
AngularJs	Un Framework d'applications Web frontales open source basée sur JavaScript, principalement géré par Google, afin de résoudre les nombreux problèmes rencontrés lors du développement d'applications à page unique.	JavaScript	01/2010
Moby	Un projet collaboratif pour l'écosystème de conteneurs pour assembler des systèmes basés sur des conteneurs.	Go	01/2013
Rails	Une infrastructure d'application Web qui inclut tout ce dont vous avez besoin pour créer des applications Web reposant sur une base de données conformément au modèle MVC (Model-View-Controller).	Ruby	01/2005
Symfony	Un Framework PHP pour les applications web et un ensemble de composants PHP réutilisables.	PHP	01/2010
TensorFlow	Une bibliothèque de logiciels open source pour le calcul numérique à l'aide des graphiques de flux de données.	C++	11/2015

Phase 3 : Extraction de données

Nous avons développé un script Python qui utilise les liens d'accès web offerts par l'API pour extraire les données de GitHub (voir ANNEXE). En particulier, nous avons extrait l'information détaillée sur l'activité autour des commits ainsi que l'historique des demandes de révisions. Pour chaque commit nous étions intéressés par le détail commit (auteur, date, message, etc.) ainsi que la liste des fichiers modifiés (nom du fichier, statut, auteur, churn (c.-à-d., l'information sur les lignes de code ajoutées et supprimées) et les commentaires (auteur,

[4] https://github.com/trending/php.

commentaire, date, etc.). D'autre part, lorsqu'un collaborateur ouvre une demande de révision, la conversation autour de la demande est lancée. Nous étions donc intéressés par les contributeurs qui demandent l'examen de leurs codes sources et ceux qui effectuent un examen ou simplement commentent les demandes de révision et finalement ceux qui résolvent la demande révision.

Pour chaque projet dans notre ensemble de données, nous avons interrogé l'API GitHub avec la requête *https://api.github.com/repos/<owner>/<repo>/commits?page=<n>*, où *<owner>* est un compte utilisateur GitHub et *<repo>* est le nom d'un dépôt du projet appartenant à cet utilisateur. La réponse renvoie une liste JSON[5] paginée. La première page liste les 30 commits les plus récentes dans la branche principale du projet. En faisant varier le paramètre *<n>*, nous contrôlons la pagination et pouvons retracer l'historique des commits jusqu'à la toute première. Chaque élément de la liste JSON représente un commit avec toutes les informations pertinentes ; un unique identifiant (noté SHA), les informations sur l'auteur (identifiant, nom, adresse e-mail) et le message du commit. Ensuite le SHA peut être utilisé pour exécuter une requête spécifique au commit. La nouvelle requête *https://api.github.com/repos/<owner> /<repo> /commits/<SHA>* fournit des informations détaillées sur le commit ainsi que sur le détail de tous les fichiers modifiés dans le commit.

La même approche est suivie pour extraire les informations supplémentaires sur les commentaires commits et les détails des demandes de révision. Le Tableau 2-3 décrit une vue d'ensemble des systèmes étudiés, à savoir le nombre total de développeurs, le nombre de versions ; nombre de lignes de code ; nombre de révisions demandées ; et le total commits.

Tableau 2-3 : Vue d'ensemble des projets étudiés

	Total Contributeurs	Total Commits	Total Commentaires sur commits	Total demande de révision	Total commentaires sur révisions	Total Lignes de Code
AngularJs	1,430	8,403	1,292	497	3,013	543,246
Moby	1,633	31,291	298	4,754	23,153	1,039,309
Rails	3,273	61,782	9,986	302	5,028	413,393
Symfony	1,474	30,106	2,309	3,226	17,014	744,619
TensorFlow	700	15,221	147	111	872	1,349, 495

Une fois les données sont récupérées nous les utilisons pour détecter la structure au sein de la communauté open source. La section suivante détaillera notre modélisation en réseaux sociaux de collaboration.

[5] Voir ANNEX A pour un exemple de la réponse Json.

2.4. Modélisation et analyse des graphes de collaboration

Les réseaux de collaboration sont généralement modélisés sous forme de graphiques dans lesquels les nœuds représentent des individus et les arcs représentent des interactions entre ces individus. Par exemple, si deux contributeurs ont modifié le même fichier, il existe une relation de coédition qui peut être représentée comme une arête. De même, si deux développeurs commentent des artéfacts techniques, alors nous pouvons représenter cette interaction (c-à-d. la liste de commentaires) comme une arête entre ces deux nœuds. Notre modélisation des graphes de collaboration comporte trois phases importantes, décrites ci-dessous, la construction des réseaux, la visualisation et l'analyse des réseaux sociaux.

2.4.1. Construction des réseaux

Nous présentons maintenant la modélisation en réseaux sociaux correspondant aux cinq projets open source étudiés dans la première étude empirique afin d'analyser les données des interactions des développeurs et examiner la structure organisationnelle de leurs communautés.

L'unité de travail de base, dans un projet OSS, est un fichier partagé dans le système de contrôle de version (VCS) distribué du projet. Par conséquent, dans cette étude empirique, nous considérons qu'une collaboration se produit lorsque deux développeurs font valider du code dans le même fichier code source. Ainsi, notre graphe de collaboration fait référence au graphe constitué de développeurs open source en tant que nœuds et aux incidences de toute paire de collaborateurs sur le même fichier en tant que lien avec la quantité (c.-à-d., nombre de fichiers co-édités) comme poids de ce lien. La Figure 2-2 présente un exemple de réseau d'interaction entre quatre contributeurs.

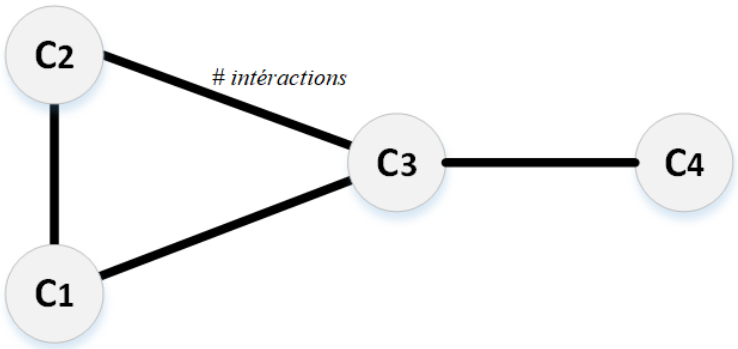


Figure 2-2 : Exemple d'un réseau d'interactions avec quatre nœuds

2.4.2. Visualisation

Il est bien connu que la visualisation fournit des moyens efficaces pour décomposer la complexité de l'information. Elle s'est révélée être aussi un moyen efficace pour étudier la structure et l'évolution des systèmes logiciels (D'Ambros et al. 2009, Goeminne et al. 2010, Stephan 2010). Pour obtenir une image complète des interactions au sein de la communauté open source, les données extraites doivent être filtrées, intégrées et présentées de manière visuelle afin de mettre le point sur les types et les formes de ces interactions. En général, la visualisation permet d'avoir un aperçu sur les relations entre les composantes de la communauté open source. L'analyse visuelle est utile, dans notre contexte, pour comprendre l'organisation des écosystèmes OO, visualiser ce qui se passe en termes de collaboration et valider les résultats de détection des rôles joués par les contributeurs open source.

Tout au long de cette thèse, nous utilisons Cytoscape[6] pour la partie visualisation des réseaux sociaux. Cytoscape (Shannon et al. 2003) est une plate-forme logicielle open source permettant de visualiser les réseaux d'interaction moléculaire et les voies biologiques et d'intégrer ces réseaux à des annotations, des profils d'expression génique et d'autres données d'état. Bien que Cytoscape ait été conçue à l'origine pour la recherche biologique, elle est maintenant une plateforme générale d'analyse et de visualisation de réseaux complexes. Cytoscape fournit un ensemble de fonctionnalités de base pour l'intégration, l'analyse et la visualisation de données.

Figure 2-3 visualise les graphes de collaboration à base de coédition des fichiers pour les cinq projets étudiés. Les contributeurs sont les ronds rouges et les arrêtes sont les liens en gris, par exemple le réseau de collaboration pour le projet AngularJs comporte 1430 nœuds et 57,854 arcs. Cytoscape propose, différents algorithmes permettant d'afficher le réseau de différentes manières dont « Edge-weighted Spring Embedded layout[7]» que nous avons utilisé dans nos études pour une meilleur visualisation des réseaux. Cet algorithme organise les nœuds dans le réseau en fonction de leurs degrés et les poids des arrêtes liées à ces nœuds. Appliqué à l'ensemble du réseau, cet algorithme permet de visualiser les nœuds de grands degrés et d'importantes arrêtes au centre du réseau.

[6] https://cytoscape.org/
[7] http://manual.cytoscape.org/en/stable/Navigation_and_Layout.html

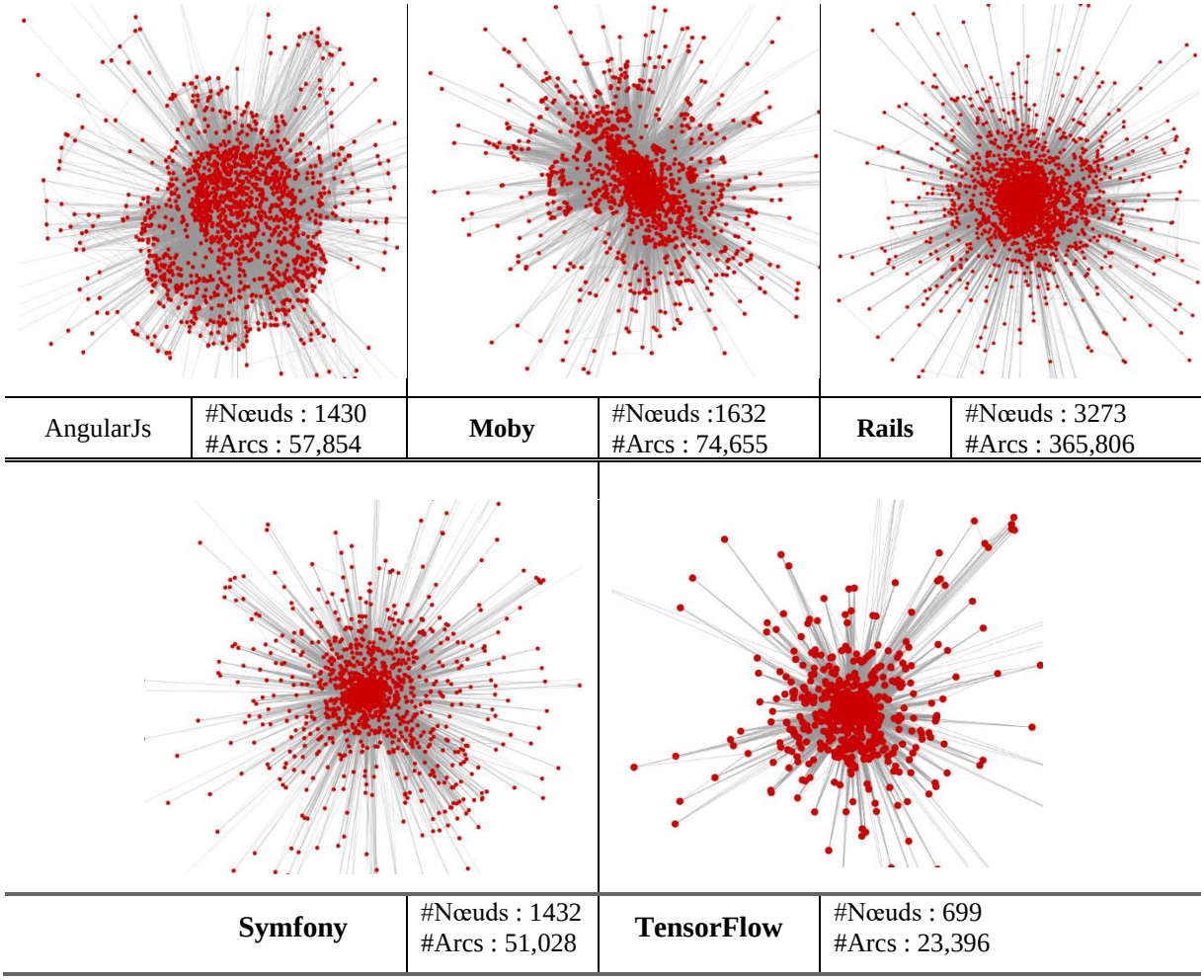


Figure 2-3 : Visualisation des graphes de collaboration des cinq projets.

2.4.3. Analyse des réseaux sociaux (SNA)

L'analyse des réseaux sociaux est une approche basée sur une série de techniques utilisées pour étudier l'échange de ressources entre les acteurs (individus, groupes ou organisations) (Wasserman et al. 1994). L'analyse des réseaux sociaux est fondée sur une approche structurale des relations entre membres d'un milieu social organisé. Elle s'attache à décrire les interdépendances entre acteurs et permet une simplification de leur représentation (Hinds et al. 2006). L'analyse de réseau social fournit un ensemble de métriques graphiques quantitatives puissantes pour comprendre les réseaux et les individus et groupes qui les composent. Celles-ci incluent des mesures de réseau globales, telles que la densité du réseau et le nombre de composants connectés, qui caractérisent le réseau dans son ensemble. Elles incluent également des métriques liées aux nœuds pouvant être utilisés pour identifier des personnes uniques ou importantes au sein d'un réseau. Le Tableau 2-4 décrit les plus importantes métriques SNA selon les définitions de (Wasserman et al. 1994) . Chaque métrique examine des propriétés spécifiques du nœud dans un réseau.

Tableau 2-4 : Mesures SNA de centralité

Métrique	Définition	Signification
Degré Centralité	Nombre d'arêtes incidentes sur un nœud. Le degré de centralité d'un sommet v, pour un graphe donné G est : $C_D(\vartheta) = \deg(v)$	Indique le nombre de contributeurs avec lesquels chaque contributeur interagit. Les développeurs principaux ont une plus grande centralité, car ils interagissent avec un grand nombre d'autres contributeurs.
Betweenness Centralité	Nombre de chemins les plus courts traversant un nœud donné. $C_B(v) = \sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}$	Mesure l'importance d'un nœud dans un graphique. Les contributeurs dont la centralité entre les niveaux est élevée sont des nœuds courtiers pour la communication avec les autres contributeurs. Betweenness est utile en tant qu'indice du potentiel d'un nœud pour le contrôle de la communication.
Closeness Centralité	Distance d'un acteur à tous les autres nœuds du réseau. $C_C = \frac{1}{\sum_{t \in V \setminus \{v\}} dist(v,t)}$	Indique la rapidité avec laquelle un développeur peut atteindre l'ensemble de la communauté. Utilisé pour évaluer la vitesse de transmission des informations d'un nœud donné à d'autres nœuds accessibles du réseau.
Excentricité	Distance maximale d'un nœud à tous les autres nœuds du réseau $c_E(v) = \frac{1}{\max\{\text{dist}(u,v) : \text{u} \in V\}}$	Une excentricité élevée signifie que le sommet le plus éloigné du réseau est très éloigné et une faible excentricité signifie que le sommet le plus éloigné est en réalité assez proche. Un développeur central devrait avoir une excentricité plus faible, car il aura des liens de communication directs avec de nombreux autres développeurs.

2.5. Résultats

Après avoir présenté l'extraction et la préparation des données ainsi que notre modélisation en graphes de collaboration, cette section présente notre réponse à la question de recherche posée dans la section 2.1. Une première sous-section rappelle la motivation derrière cette question de recherche, puis une seconde explique l'approche adoptée pour répondre à cette question et enfin une troisième sous-section présente les résultats obtenus.

La question de recherche de cette étude empirique est :

QR : Quelles sont les composantes qui organisent les communautés open source ?

Motivation. L'objectif de cette question de recherche est d'identifier l'organisation typique d'une communauté open source à travers l'analyse des interactions sociotechniques entre les membres de la communauté.

Approche. La littérature (Freeman 1977, Krebs 2002, Borgatti 2003) a démontré la capacité des mesures de centralité SNA à déterminer la position relative d'un nœud dans un réseau. Ces mesures ont été utilisées dans des travaux de recherche antérieurs pour découvrir des acteurs clés dans les réseaux sociaux. Nous nous basons sur l'approche proposée par Bosu et al. (Bosu et al. 2014) pour modéliser notre approche de détection de la structure organisationnelle des

projets open source. Ces auteurs ont utilisé les métriques SNA avec le classificateur k-means[8] pour classifier les développeurs en deux classes. No critiques sur l'approche de (Bosu et al. 2014) sont principalement liées à la justification de l'utilisation du classificateur k-means avec k égale à 2 et aussi la validation des résultats de classification. Notre approche de classification comporte trois phases : (1) Le calcul des métriques SNA, (2) Classification des contributeurs, et (3) Validation. Dans la sous sous-section suivante nous présentons les résultats de chaque phase.

Résultats.

Étape 1 : Calcul des métriques

La première étape consiste à calculer les métriques SNA des réseaux sociaux pour chaque projet. Nous avons utilisé le paquetage Networkx[9]pour le calcul des métriques de centralité SNA. Les mesures de centralité utilisées pour cette approche sont les suivantes : Degré Centralité, Betweenness Centralité, Closeness Centralité et Excentricité. Chaque métrique calcule la centralité de manière différente et interprète différemment un nœud central (voir Tableau 2-4). Le résultat de cette étape est un ensemble de fichiers CSV (un par projet) qui détaillent les métriques calculées pour chaque nœud.

Étape 2 : Classification des contributeurs

Une fois les métrique récupérées, la deuxième étape consiste à classifier les contributeurs. Notre contribution étant basée sur le travail de (Bosu et al. 2014) nous avons opté pour le classificateur K_means. En effet, l'algorithme k-means est l' un des plus simples algorithmes d'apprentissage non supervisé , appelée algorithme des centres mobiles (Macqueen 1967) , il attribue chaque point dans un cluster dont le centre (centroïde) est le plus proche. Le centre est la moyenne de tous les points dans le cluster, ses coordonnées sont la moyenne arithmétique pour chaque dimension séparément de tous les Points dans le cluster c'est à dire chaque cluster est représentée par son centre de gravité. K-means regroupe les contributeurs du projet en k classes mutuellement exclusives dans lesquelles chaque contributeur appartient à la classe dont la moyenne est la plus proche (mesurée à l'aide des métriques SNA de centralité). K-means traite chaque contributeur comme un objet ayant une localisation dans l'espace. L'objectif de K-means est de trouver une partition dans laquelle les objets de chaque classe sont aussi proches

[8] K-means est une méthode de partitionnement de données. Étant donnés des points et un entier k, le problème est de diviser les points en k groupes, souvent appelés clusters, de façon à minimiser une certaine fonction.
[9] https://networkx.github.io/

que possible et aussi éloignés que possible des objets des autres classes. Nous avons utilisé la fonction k-means () dans R[10], pour classifier les contributeurs.

Pour trouver le nombre approprié de grappes K (c-à-d. nombre de classes), paramètre d'entrée pour l'algorithme k-means, nous avons utilisé la méthode Elbow (Bholowalia et al. 2014). L'algorithme Elbow considère le pourcentage de variance interclasses expliqué en fonction du nombre de clusters. Le principe ici est de choisir un nombre minimum de clusters afin que l'ajout d'un autre cluster n'améliore plus le modèle. Nous avons codé un script R pour calculer et visualiser la courbe Elbow. La Figure 2-4 montre les résultats après avoir effectué un regroupement k-means pour k allant de 1 à 10 sur les données SNA. En effet, la "chute" de la courbe de variance entre 1 et 3 clusters est significativement plus grande que celle entre 4 clusters et 5 clusters. Ainsi, on peut voir un coude assez clair à k = 3, indiquant que 3 est le meilleur nombre de classes.

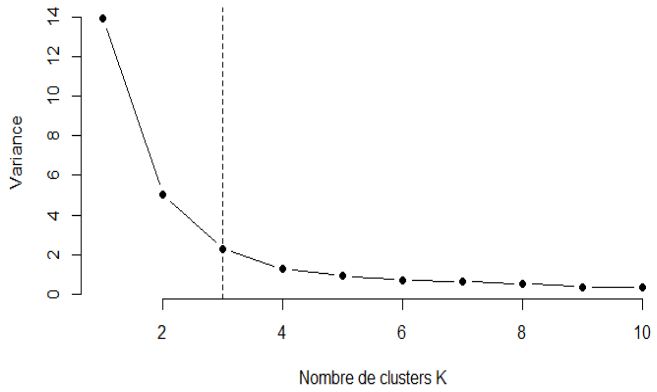


Figure 2-4 : Identification du nombre optimal de clusters avec la méthode Elbow

Nous avons aussi remarqué que la classification en trois groupes fournit un modèle plus précis, comme indiqué dans le Tableau 2-5. La qualité du classificateur k-means, reportée dans Tableau 2-5 mesure de la qualité de la classification trouvé. Par exemple, pour le projet AngularJS, la précision de la classification des contributeurs en deux groupes est de *67%*, tandis que celle basée sur trois groupes donne de meilleurs résultats à *80,1%*.

[10] https://stat.ethz.ch/R-manual/R-devel/library/stats/html/kmeans.html

Tableau 2-5 : Précision du classificateur K-means

	2 Classes (C, P)		3 Classes (C, T, P)	
Projet	**Qualité (%)**	**Taille (#Nœuds)**	**Qualité (%)**	**Taille (#Nœuds)**
AngularJs	67.0	(50, 1379)	80.1	(39, 169, 1221)
Moby	64.3	(92, 1540)	83.7	(22, 156, 1425)
Rails	67.1	(110, 3164)	85.0	(78, 519, 2677)
Symfony	68.7	(133, 1296)	81.7	(26, 172, 1231)
TensorFlow	74.2	(43, 622)	87.6	(24, 56, 585)

Une fois le K est fixé, nous utilisons K-means avec en entré les métriques SNA et K=3 pour classer les contributeurs en trois groupes Core (C), transitoires (T) ou périphériques (P). Le Tableau 2-5, ci-dessus, présente la taille de chaque classe (Core, transitoire et périphérie) en termes de nombre de développeurs dans chaque classe. Nos résultats confirment les recherches préliminaires sur la structure de base (c-à-d. en Oignon) des équipes OSS (par exemple, Cox (Cox 1998), Moon & Sproull (Moon et al. 2005)).

Étape 3 : Validation

Nous avons procédé à plusieurs validations des résultats de classification obtenus en utilisant k-means.

Premièrement, nous avons comparé le Core trouvé par notre approche de classification à celui trouvé par l'algorithme O (m) pour la décomposition des noyaux de réseaux (Batagelj et al. 2003). L'algorithme O(m) prend en entrée un graphe et fournit des sous-groupes d'acteurs cohésifs entre qui existe un lien relativement fort, direct ou fréquent. Pour chaque réseau, nous calculons pour chaque nœud la fonction k_core[11], fournie par le paquetage Networkx, qui implémente l'algorithme O (m). Les nœuds avec le maximum K_core constituent le Core par l'algorithme O(m). La concordance entre notre Core et celui détecté par O(m) est de 68% (par exemple, la partition avec le maximum k_core contient 26 parmi les 39 nœuds identifiés comme Core avec notre approche de classification).

Deuxièmement, pour renforcer notre approche de validation des résultats, nous avons procédé à une inspection manuelle et visuelle de la position des contributeurs Core dans les réseaux de collaboration. La Figure 2-5 visualise les réseaux de coédition des cinq projets étudiés, les nœud jaunes sont les contributeurs classifiés comme Core avec notre approche de classification. La visualisation nous a permis de confirmer que les contributeurs Core forment un groupe dense positionné au centre du réseau de collaboration.

[11] https://networkx.github.io/documentation/networkx-1.7/reference/generated/networkx.algorithms.core.k_core.html

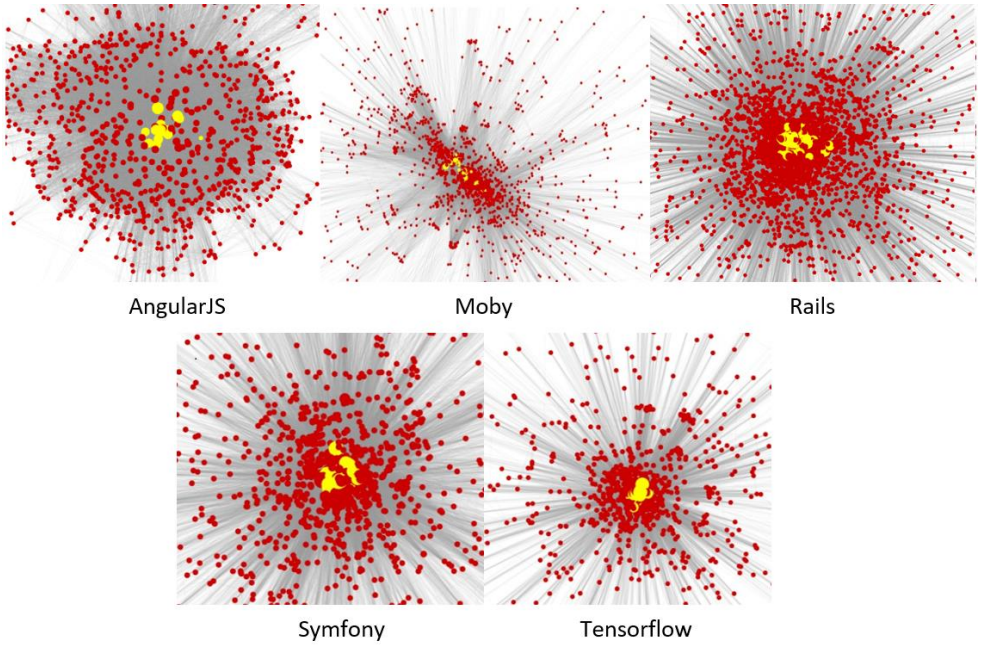


Figure 2-5 : Visualisation des contributeurs Core dans les réseaux de coédition.

Troisièmement, en termes d'activité nous avons vérifié que les contributeurs identifiés comme Core dans notre approche sont responsables de 75% à 95% du total des activités plus précisément des commits. Enfin, nous avons vérifié que les contributeurs identifiés Core figurent parmi la liste des contributeurs les plus actifs dans le projet. Cette liste est fournie par GitHub sur la page principale de chaque projet[12].

2.6. Discussion

L'une des principales contributions de cette première étude empirique consiste en proposer une approche pour identifier les structures qui organisent la communautés open source. En effet, nous avons utilisé une approche basée sur une modélisation sociotechnique pour mieux comprendre les différents rôles des développeurs open source dans un projet. Dans notre étude nous avons pris soin de valider la fiabilité de notre approche.

D'autre part, notre étude démontre que les données stockées dans les référentiels de logiciels, associées à des méthodes d'analyse appropriées, peuvent fournir des informations précieuses, pratiques et valables sur les aspects sociotechniques du développement de logiciels. Les études empiriques présentées dans les chapitres suivants démontrent d'avantage ce point.

Nous reconnaissons quelques limites de validité pour notre étude. D'abord, pour la détection des structures organisationnelles basées sur les réseaux, nous avons utilisé les métriques

[12] Par exemple, pour le projet AngularJs la liste est disponible sur https://github.com/angular/angular.js/graphs/contributors

d'analyse de réseaux sociaux pour établir une base relationnelle pour les développeurs Core et Périphériques. Cela pose le risque que les réseaux et les métriques ne capturent pas avec précision la réalité de la structure des communautés OSS. Cette menace est mineure, car il existe déjà des preuves indiquant que les réseaux et les métriques reflètent de manière authentique la perception des développeurs (Meneely et al. 2011). Ensuite, nous avons appliqué notre approche à cinq projets OSS, ce qui menace la validité externe de notre étude. Nous avons choisi des projets de tailles différentes, mais substantielles, avec des historiques de développement longs et actifs, et provenant de différents domaines d'application pour couvrir une gamme considérable de projets. Malgré la diversité des thématiques de nos projets, ils ne représentent toujours qu'une fraction de la diversité totale des projets OSS. Les projets que nous avons jugés appropriés pour notre étude sont bien établis et ont connu beaucoup de succès. Une structure de communauté significative dans les réseaux de développeurs peut être limitée à ce type de projets. Dans les projets infructueux, les développeurs peuvent être moins conscients des exigences en matière de coordination et sont donc moins représentatifs des interactions réelles entre les développeurs. Ainsi, les résultats peuvent ne pas être pertinents pour des projets immatures ou de très petite taille.

Enfin, notre étude fait l'objet d'une validité de conclusion statistique qui fait référence à la capacité de faire une évaluation précise de la force de la relation entre nos variables indépendantes et dépendantes. Par exemple, dans la section 2.5, nous avons utilisé k-means pour regrouper et répartir les contributeurs dans trois catégories (Core, Transitoire, et Périphérique) selon une série de mesures SNA. Pour gagner en confiance sur notre approche de classification, nous avons justifié le choix du k=3 à l'aide de la méthode Elbow. Ensuite nous avons validé nos résultats en utilisant différentes méthodes telles que l'algorithme O (m), l'information GitHub et l'inspection visuelle des réseaux collaboratifs.

Conclusion

Dans ce chapitre nous avons proposé une approche de regroupement k-means basée sur les métriques SNA de centralité afin de classer de manière dynamique les contributeurs dans les réseaux de collaboration mensuels en trois classes: Core, Transitoire et Périphérique. Notre étude a amélioré les résultats des études antérieures (Geldenhuys 2010, Cucuringu et al. 2016) ayant classifié les contributeurs en deux classes. En effet, nous proposons une classification plus raffinée à travers l'introduction de la zone transitoire par laquelle passent les nouveaux arrivants avant d'atteindre le cœur du projet.

Notre approche concorde avec l'algorithme de décomposition du noyau O (m). En outre, des inspections visuelles ont validé que les principaux contributeurs classés appartenaient effectivement à un bloc dense et cohérent physiquement centré sur le réseau de collaboration.

Ce chapitre a présenté notre contribution autour de la détection de la structure organisationnelle des communautés opens source. Le chapitre suivant complétera notre étude pour explorer l’évolution temporelle de ces communautés.

CHAPITRE 3. DYNAMIQUE DES COMMUNAUTÉS OSS : UNE ANALYSE TEMPORELLE

Dans ce chapitre, nous avons entrepris une analyse[13] sociotechnique de cinq communautés de logiciels open source afin de découvrir la dynamique de croissance et de disparition progressive de ces communautés au fil du temps. Une attention particulière a été accordée à la migration des nouveaux arrivants de la périphérie vers l'équipe principale « Core ». La première section 3.1 décrit les objectifs de notre étude, ensuite nous rapportons la revue de la littérature dans la section 3.2. Le contexte de notre étude est présenté dans la section 3.3, suivie par notre modélisation temporelle en réseaux sociaux dans la section 3.4. Les résultats de notre étude empirique sont listés dans la section 3.5. Enfin, dans la section 3.6 nous discutons les implications pratiques et les limites de l'étude avant de conclure notre étude.

[13] Le contenu de ce chapitre a fait sujet de deux publications. La première est un papier de conférence intitulé «**From Periphery to Core: A Temporal Analysis of GitHub Contributors' Collaboration Network**» (El Asri et al. 2017), présenté dans la 18th édition de « Working Conference on Virtual Enterprises -Collaboration in a Data-Rich World ». La seconde et un papier journal intitulé « **Understanding OSS Project's Collaborative Dynamics : Core and Peripheral Interactions** » publié dans « International Journal of Scientific & Engineering Research -IJSER » (El Asri et al. 2018).

3.1. Motivation et Objectifs

Les communautés OSS se développent et disparaissent avec le temps. Suivre l'évolution de la structure organisationnelle de la communauté open source au fil du temps a d'importantes applications, à savoir suivre son expansion (Weiss et al. 2006), prévoir ces futurs changements (Ye et al. 2008) et prédire son déclin (Ye et al. 2008).

L'évolution d'une communauté de logiciels open source est provoquée essentiellement par les changements de rôles de ses membres. Lorsque les membres de la communauté modifient leurs rôles dans la communauté, ils modifient également la dynamique sociale et modifient la structure de la communauté (Hannemann et al. 2013). Bien qu'un projet de logiciel open source a un chef de projet (souvent celui qui initie le projet), ce responsable n'a pas un plan ambitieux pour le système au départ et ne peut pas dicter son évolution. C'est l'ensemble des développeurs qui pilotent de manière collaborative l'évolution du système. Par conséquent, comprendre le changement du rôle joué par les contributeurs tout au long de l'évolution du projet est essentiel pour comprendre la dynamique de collaboration du projet (Hannemann et al. 2013). Cependant, il existe très peu d'études sur la croissance de ces communautés virtuelles et moins encore su la navigation des nouveaux arrivants de la périphérie d'un projet donné (c'est-à-dire d'une première contribution) au Core (c.-à-d. s'engager, commenter et participer en permanence aux prises de décisions sur l'évolution du projet).

L'objectif de cette étude et d'explorer la dynamique des communautés open source. L'idée principale est d'analyser les schémas temporels selon lesquels les nouveaux arrivants dans le projet de logiciel open source (OSS) passent d'une position de membre périphérique au membre Core. Comprendre ce phénomène au sein d'un projet open source peut aider à mieux comprendre comment maintenir des communautés virtuelles vivantes et en expansion en attirant de nouveaux contributeurs afin d'accélérer le développement des projets logiciels.

À l'aide de cette deuxième étude empirique longitudinale sur l'évolution temporelle de cinq projets open source bien connu, nous aborderons les trois questions de recherche suivantes :

- ***QR1. À quelle fréquence les contributeurs passent-ils de la périphérie à l'équipe Core ?***

L'objectif de cette question est d'identifier les modèles de transition des développeurs à travers différents rôles en observant les changements survenus dans le réseau sociotechnique de collaboration au fil du temps.

- ***QR2.*** **Quelles activités favorisent-elle la transition d'un nouvel arrivant vers le rôle de contributeur principal ?**

Il est bien connu que les développeurs Core obtiennent généralement leurs notoriétés grâce à une implication constante et accumulent des connaissances dans des domaines particuliers du système au cours de leur engagement tout au long de l'évolution du projet. A travers cette question de recherche, nous cherchons à identifier les types d'activités qui favorisent le plus la transition d'un contributeur de la périphérie au Core.

- ***QR3. Est-il possible de prédire si un contributeur principal quittera le projet ?***

L'objectif de cette question de recherche est de comprendre les facteurs influençant la durabilité de participation dans les projets open source. Nous cherchons à identifier les types d'activités qui permettent de prédire le désabonnement des développeurs open source.

Après avoir identifié nos motivations et objectifs la section suivante reportera l'état de l'art sur les travaux antérieurs ayant traité l'évolution temporelle des communautés open source.

3.2. État de l'art

Jensen et Scacchi (Jensen et al. 2007) ont trouvé que la structure organisationnelle des projets étudiés est très dynamique par rapport aux organisations de développement logiciel traditionnelles. L'analyse de l'évolution dynamique de la communauté OSS peut être utilisée pour extraire des indicateurs permettant d'évaluer la stabilité actuelle d'une communauté et de prédire son développement futur (Qiao et al. 2012). Au cours de la dernière décennie, certaines études (voir Tableau 3-1) ont porté sur l'évolution des communautés open source. En particulier, dans la revue systématique de la littérature réalisée par Breivold et al. (Breivold et al. 2012), les chercheurs ont fournis un aperçu des mesures utilisées pour analyser l'évolution de l'OSS au fil du temps cependant les différentes mesures proposées, à savoir les mesures de croissance logicielle, les mesures de croissance du système, etc., ne concernent que les aspects techniques des projets OSS sans prendre en considération les interactions sociales entre les membres de la communauté. Hanneman et al. (Hannemann et al. 2013) ont combiné les mesures démographiques, adaptées pour étudier l'évolution de la communauté, et une analyse sociale pour étudier la dynamique des structures communautaires. La critique de ce travail est, comme annoncé par ses auteurs, leur étude est effectuée sur des projets open source en Bio-informatique, menées principalement par des scientifiques en bio-informatique ou des

doctorants travaillant sur leur thèse. Une fois leur doctorat terminé, ils risquent de perdre l'intérêt et / ou le temps consacré aux contributions.

D'autres études, ayant abordé notre même perception, ont focalisé sur le changement des rôles des développeurs open source pour enquêter la dynamique de la communauté. Ye et al. (Ye et al. 2008) ont trouvé que les systèmes OSS évoluent grâce aux contributions apportées par les membres de la communauté. Ces contributions modifient le rôle joué par le membre dans la communauté, ce qui se traduit par une évolution de la communauté en remodelant la structure et la dynamique de la communauté. (Robles et al. 2006) ont proposé une méthodologie quantitative, basée sur l'analyse de l'activité dans les référentiels de gestion de code source, pour étudier l'impact des processus (développeurs quittant, développeurs rejoignant) sur les projets de logiciels open source. Weiss et al. (Weiss et al. 2007) ont aussi démontré que la communauté open source évolue à travers la participation de nouveaux développeurs à partir d'un groupe de développeurs externes et par la migration interne des développeurs entre projets existants. Loyola et al. (Loyola et al. 2014) ont proposé une méthodologie qui adapte les modèles biologiques basés sur Lotka-Volterra utilisés pour décrire les interactions hôte-parasite, pour fournir une idée quantitative de l'évolution de la population de développeurs.

Le Tableau 3-1 ci-dessous présente une synthèse sur la recherche autour de la dynamique des communautés OSS.

Notre étude complémente les recherches précédentes (voir Tableau 3-1), tout en se basant sur l'analyse sociotechnique pour qualifier l'évolution temporelle de la structure organisationnelle de la communauté open source. En effet, le développement OSS est particulièrement adapté à l'examen des effets combinés du social et du technique dans un contexte de développement de système, car il favorise les interactions entre les développeurs autour les artefacts logiciels par le biais de différents canaux de communication (Begel et al. 2010, Kaur et al. 2014). Ainsi, Notre étude concerne l'évolution des communautés open source en mettant l'accent sur l'évolution des interactions sociotechniques entre les composantes de cette communauté.

Tableau 3-1: Synthèse sur la recherche autour de la dynamique des communautés OSS.

Étude	Approche	Critique
Breivold et al. 2012	Propose et décrit le processus d'analyse d'évolutivité de l'architecture logicielle (AREA), qui fournit plusieurs techniques répétables permettant de prendre en charge l'évolution de l'architecture logicielle	Les différentes mesures proposées, à savoir les mesures de croissance logicielle, les mesures de croissance du système, etc., ne concernent que les aspects techniques des projets OSS
(Hannemann et al. 2013)	Combine les mesures démographiques, adaptées pour étudier l'évolution de la communauté, et une analyse sociale pour étudier la dynamique des structures communautaires.	L'approche est appliquée à l'histoire de la communication et du développement de trois logiciels OSS de Bio-informatique menée principalement par des doctorants en bio-informatique qui risquent de perdre l'intérêt et / ou le temps consacré aux contributions une fois leur doctorat terminé.
(Robles et al. 2006)	Propose une méthodologie quantitative, basée sur l'analyse de l'activité des référentiels de gestion de code source, pour étudier l'impact des processus (développeurs quittant, développeurs rejoignant) sur des projets de logiciels libres.	L'étude se limite à la l'analyse de la décomposition, uniquement, du groupe Core au fil du temps.
(Athey et al. 2014)	Propose un modèle dynamique de l'évolution des projets de logiciels open source.	La dynamique de la communauté open source est étudiée en termes d'évolution de la qualité du produit et le nombre de programmeurs impliqués
(Weiss et al. 2007)	L'utilisation de l'analyse statistique pour explorer l'écologie des communautés open source dans le cadre du projet Apache. L'étude a démontré que la communauté open source évolue à travers la participation de nouveaux développeurs à partir d'un groupe de développeurs externes et par la migration interne des développeurs entre projets existants.	L'étude est effectuée sur un seul projet « Apache » qui comporte plusieurs inters communautés. L'évolution d'une communauté d'un projet open source est vue en termes d'interactions des sous communautés sans prendre en considération les interactions des membres de la même sous communauté.
(Loyola et al. 2014)	Proposer une méthodologie qui adapte les modèles biologiques basés sur Lotka-Volterra utilisés pour décrire les interactions hôte-parasite, pour fournir une idée quantitative de l'évolution de la population de développeurs.	Le modèle biologique standard est-il adapté à un environnement Open Source.

3.3. Préparation des Données GitHub

Afin de comprendre la dynamique des communautés OSS, nous allons étudier l'évolution des interactions sociotechniques dans le temps pour cinq projets OSS de GitHub. Nous avons utilisé les mêmes données extraites dans CHAPITRE 2 section 2.3. Cependant la préparation des données change puisque nous menons dans cette étude une analyse temporelle des données. Le Tableau 3-2 rappelle les informations générales sur les projets OSS étudiés. Bien évidemment, les projets étudiés ont une histoire de plus de cinq ans (à l'exception de Tensorflow, qui est relativement nouveau) et offrent d'excellentes opportunités pour explorer l'évolution temporelle des communautés open source. En outre, ces projets offrent l'occasion d'étudier la dynamique d'un nombre suffisant de contributeurs; près de 8,460 développeurs de logiciels open source.

Tableau 3-2 : Vue d'ensemble des systèmes étudiés

	Langage	#Contrib	Commits	Commentaires sur les commits	Demandes de révision	Commentaires sur les révisions	Lignes de Code	Date de création
AngularJS	JavaScript	1,430	8,403	1,292	497	3,013	543,246	01/2010
Moby	Go	1,633	31,291	298	4,754	23,153	1,039,309	01/2013
Rails	Ruby	3,273	61,782	9,986	302	5,028	413,393	01/2005
Symfony	PHP	1,424	30,106	2,309	3,226	17,014	744,619	01/2010
TensorFlow	C++	700	15,221	147	111	872	1,349,495	11/2015

Les historiques d'activité sont ensuite scindés en des historiques mensuels cumulatifs. En particulier, les historiques de modification des fichiers sont ensuite utilisés pour créer des graphes de collaboration dynamique. Aussi, pour chaque contributeur, les historiques d'activité à savoir les commits, les commentaires, le détail sur les lignes de code modifié et les le nombre de fichiers modifiés sont décomposés en des séries temporelles. La mesure de temps utilisé dans notre cas est un mois cumulatif où l'information liée à un mois m_t et une somme des données du premier mois au mois m_t. Ainsi, chaque contributeur est lié à une suite d'observations cumulatives sur son activité à des dates différentes.

3.4. Modélisation des graphes de collaboration dynamiques

Comme nous l'avons expliqué dans CHAPITRE 2 section 2.4, un réseau social standard peut être modélisé mathématiquement par un graphe G = (N, E) constitué d'un ensemble de nœuds N et d'un ensemble d'arcs E les reliant. Dans cette étude nous travaillons plutôt avec des réseaux dynamiques. Un réseau social dynamique consiste en une série d'observations de réseaux

sociaux à différents intervalles temporels (G1, G2, ..., Gn). Un réseau social dynamique contient non seulement un ensemble de relations entre les nœuds, mais également des informations sur l'évolution de ces relations dans le temps.

Les ensembles de données ont été traités et découpés en mois pour fournir des plages temporelles (PT) pour l'analyse dynamique des données. Pour chaque période, nous avons construit un réseau cumulatif de coédition des fichiers (FCN), en ajoutant progressivement un mois d'activité après un autre. Le FCN peut être modélisé sous la forme d'un graphique $G_t = (N_t, E_t)$ où N_t représente les contributeurs et E_t l'ensemble des interactions entre eux à l'intervalle temporelle t. La coédition du même fichier est la variable dépendante indiquant si une interaction entre deux développeurs a eu lieu ou non. Par conséquent, pour deux nœuds i et j, un lien unidirectionnel est établi entre i et j lorsque les développeurs i et j ont modifié le même fichier, voir Figure 3-1.

Nous considérons G_k comme un graphe pondéré unidirectionnel où le poids d'un lien est un agrégat de la quantité d'interactions entre ces contributeurs, en fonction du nombre de fichiers qu'ils ont tous les deux édités. Nous avons obtenu une série de réseaux de collaboration cumulatifs qui nous ont permis d'étudier l'évolution des structures sociales de chaque communauté ainsi que l'évolution des membres qui la constituent selon la perspective Core/périphérie.

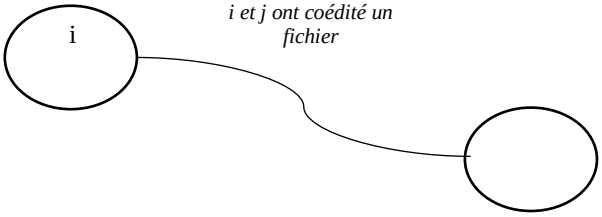


Figure 3-1 : Réseau temporel de coédition des fichiers

3.5. Résultats

Après avoir présenté la préparation des données ainsi que notre modélisation en graphes de collaboration dynamiques, cette section présente notre réponse aux questions de recherche posées dans la section 3.1. Pour chaque question de recherche nous rappelons la motivation, puis l'approche suivie et enfin les résultats obtenus.

QR1. À quelle fréquence les contributeurs passent-ils de la périphérie à l'équipe Core ?

Motivation : Des travaux antérieurs (Jensen et al. 2007, Athey et al. 2014) ont montré que la structure organisationnelle des projets open source est très dynamique par rapport aux organisations de développement de logiciels traditionnels. En effet, la communauté open source évolue avec l'évolution de ses membres en termes d'acquisition d'expertise grâce à l'accumulation de connaissances dans des domaines particuliers du système sur de longues périodes. La Figure 3-2 illustre un exemple concret du changement de position sociale de la périphérie vers le Core pour un développeur choisi dans le projet AngularJs (nœud bleu). Ce contributeur apparait dans la périphérie en Janvier 2013 et atteint le Core en Janvier 2014. Dans cette question de recherche, nous cherchons à quantifier l'évolution de l'engagement des développeurs open source tout au long de l'évolution du projet.

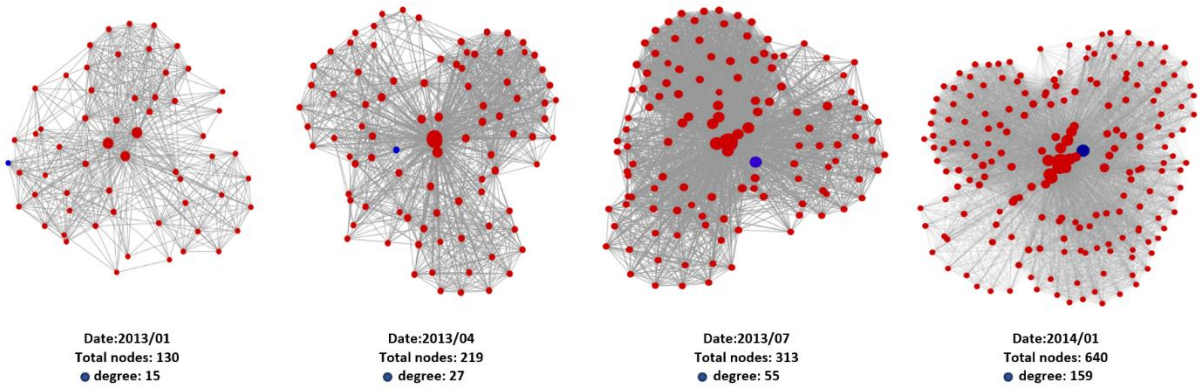


Figure 3-2 : Exemple de suivi d'une migration de la périphérie vers le Core.

Approche : La première étape de notre approche consiste en classifier des contributeurs open source en Core (C), Transitoire (T) et Périphérique (P) à la base des métriques SNA tout en utilisant la même approche proposée dans CHAPITRE 2 section 2.5. La même approche de catégorisation est appliquée cette fois sur une série temporelle des métriques SNA (c-à-d. les métriques SNA des contributeurs par mois). Le résultat, pour chaque contributeur, est aussi une série de classes (C / T / P). Ensuite, nous calculons la stabilité des développeurs en explorant la probabilité mensuelle que les développeurs passent d'une position dans le réseau social à un autre. A travers la séquence de changements de rôles ordonnée dans le temps au cours de son implication dans le projet, nous sommes en mesure de représenter les transitions des développeurs d'un état à l'autre sous la forme d'une matrice de transition N * N, dans laquelle chaque élément indique la probabilité moyenne de transition (présentée en pourcentage) d'un

état à un autre au cours de l'évolution du projet. Cette matrice de transition sera ensuite présentée comme une chaîne de Markov.

Résultats : La Figure 3-3 montre les probabilités de transition entre les états des développeurs sous la forme d'une chaîne de Markov pour les cinq projets étudiés. La principale observation est que les développeurs Core ont beaucoup moins de chances de passer à la zone de transition et ne passent jamais directement à la périphérie. Par exemple, pour le projet AngularJs, nous avons observé que les principaux développeurs restent dans cette zone avec une probabilité de 97%, passent vers l'état transitoire avec une probabilité de 3% et transitent directement vers l'état périphérique avec une probabilité de 0%. Sur la base de ce résultat, nous pouvons confirmer que les développeurs principaux représentent un groupe stable.

En outre, le pourcentage de transition T vers C (de la zone de transition au groupe Core) reste très important. En moyenne, 1% des contributeurs de la zone de transition rejoignent l'équipe centrale. Les nouveaux contributeurs périphériques (P) sont aussi importants que les membres principaux de tout projet de logiciel open source car ils alimentent principalement la classe T. Certains d'entre eux aspirent à devenir des membres essentiels grâce à des contributions continues.

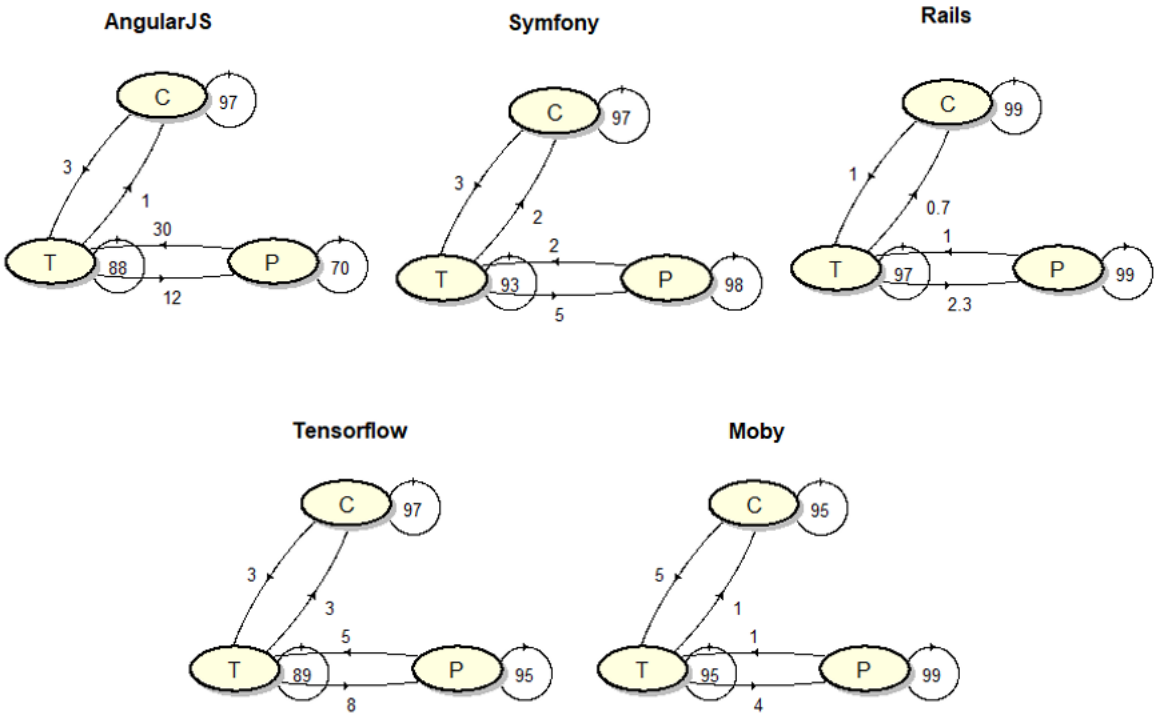


Figure 3-3 : Stabilité du rôle des développeurs

L'analyse mensuelle de l'évolution de la taille de l'équipe principale, sur une échelle mensuelle, dans les projets OSS étudiés révèle une croissance intéressante des équipes principales illustrée en Figure 3-4. Cette constatation montre à quel point le projet est attrayant du point de vue de sa capacité à attirer un grand nombre de contributeurs et les maintenir engagés. L'équipe principale d'AngularJS a commencé avec un seul contributeur et a attiré 26 développeurs dans l'équipe principale en juin 2017. Il est également intéressant d'étudier les transitions d'un groupe à l'autre, en particulier les contributeurs sortant de l'équipe centrale. En effet un taux élevé de contributeurs quittant le Core peut être considéré comme une indication de l'instabilité du projet (Amrit et al. 2010).

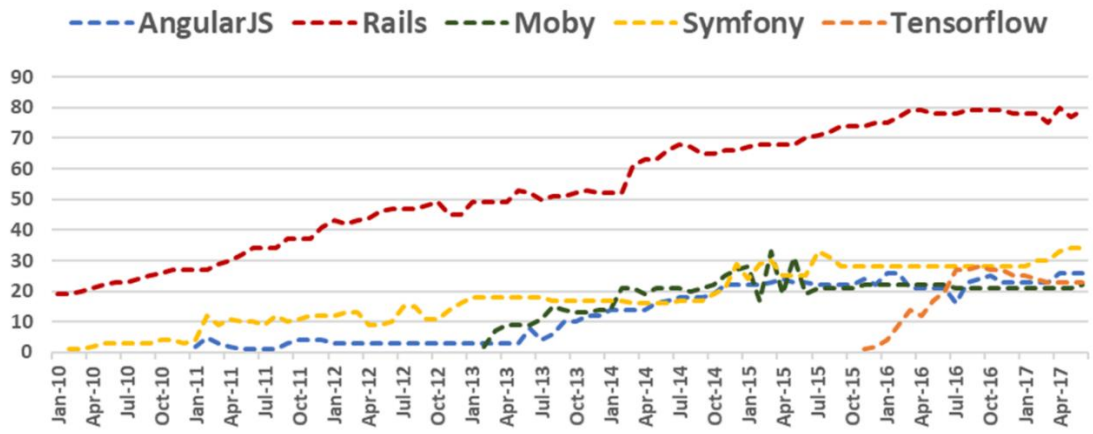


Figure 3-4 : Évolution mensuelle des contributeurs Core.

QR2. Quelles activités favorisent-elle la transition d'un nouvel arrivant vers le rôle de contributeur principal ?

Motivation. Dans la première question de recherche, nous avons bien observé que peu de nouveaux arrivants au sein des OSS finissent par être au cœur du projet. Dans cette question de recherche, nous visons à découvrir quel type de contribution ou de collaboration est le plus pertinent pour favoriser l'évolution d'un nouvel arrivant vers le Core du projet. Nous sommes intéressés à détecter les corrélations existantes entre la position sociale et la participation technique. Notre objectif principal est de doter la communauté OSS d'une meilleure compréhension des activités de collaboration avec le plus valeur et des lignes directrices potentielles pour que les nouveaux arrivants puissent jouer un rôle plus important.

Approche. Nous avons étudié empiriquement l'ascension des 10 top contributeurs pour chaque projet. Ensuite, nous avons retracé l'historique des contributions visant à quantifier les activités de collaboration. Nous avons ensuite considéré les activités des contributeurs sous cinq types de contributions (#commits, # commentaires sur les commits, #commentaires sur les révisions, #Fichiers Modifié, et #Lignes de Code modifiée) puis nous avons mesuré l'activité collaborative la plus corrélée par rapport aux métriques des réseaux sociaux des contributeurs.

Résultats. Le Tableau 3-3 reporte la corrélation entre la métrique de centralité SNA et la mesure de chaque caractéristique d'activité (#commits, #commentaires …). On peut remarquer que les activités liées aux changements de code source sont plus corrélées à la position des contributeurs dans la structure de la communauté. Par exemple, nous avons trouvé pour le projet AngularJs a un facteur de corrélation de 0,76 entre rester dans l'équipe de base et le nombre de commits de contributeur.

Tableau 3-3 : Corrélation entre la métrique SNA de centralité et les activités

	Code source				**Commentaires**	
	Commits	***Fichiers édités***	***Lignes de code ajoutée***	***Lignes de code supprimée***	***Commentaires sur commits***	***Commentaires sur révisions***
AngularJs	0.76	0.77	0.70	0.72	0.28	0.60
Moby	0.73	0.81	0.75	0.72	0.43	0.06
Rails	0.68	0.74	0.62	0.60	0.71	0.31
Symfony	0.77	0.83	0.85	0.84	0.54	0.39
TensorFlow	0.69	0.72	0.73	0.79	0.68	0.27

QR3. Est-il possible de prédire si un contributeur principal quittera le projet ?

Motivation. Des efforts considérables ont été déployés au cours des dernières années pour tenter de comprendre les facteurs influençant la participation et la durabilité des projets open source. La Figure 3-5 illustre le suivi d'un contributeur du projet Docker. Ce contributeur a appartenu à l'équipe Core et a ensuite quitté le projet à la fin de 2013. Si nous pouvions prédire le comportement des contributeurs en fonction de certains aspects que nous sommes en mesure de modéliser et de quantifier alors nous avons les ingrédients pour construire une communauté durable de contributeurs.

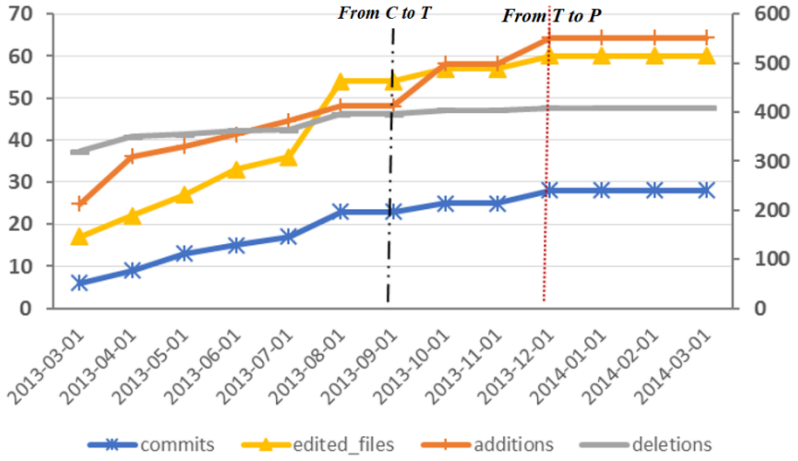


Figure 3-5 : Désengagement des contributeurs

Approche. Pour répondre à cette question de recherche et prédire qui quittera le projet, étant donné les métriques collaboratives et les changement de position des contributeurs au fil du temps, nous avons appliqué l'algorithme de l'arbre de décision *J48,* sous WEKA [14]. Pour les besoins de cette analyse, nous avons filtré les contributeurs qui passent du Core à la zone transitoire (CT) avant la transition finale vers la périphérie (TP), ce qui signifie qu'ils quittent le projet. L'arbre de decision J48 consiste à organiser les métriques d'activités pour les contributeurs étudiées sous forme d'arbre avec une racine, des branches et des feuilles. Le meilleur attribut (c-à-d. type d'activité) est celui qui possède la meilleure corrélation avec la répartition des classes (C / T / P). Généralement un processus récursif est utilisé pour obtenir un résultat optimum de la valeur de la classe faisant l'objet de la prédiction.

Résultats. Le Tableau 3-4 présente les résultats de notre approche d'apprentissage automatique supervisé concernant les cinq projets. À titre d'exemple, nous avons 27 contributeurs au sein du projet AngularJS, passant du Core à la zone Transitoire (CT). Avec l'arbre de décision J48, nous sommes capables de prédire 74,07% (0,93 Sensibilité) du passage de C à T avec seulement 7 cas sur 27 non correctement classés. Un autre point intéressant, nous avons trouvé que le nœud racine de l'arbre de décision était "EditedFiles" <= 871.39. Cela signifie que la quantité de fichiers édités est la mesure la plus précise liée au taux de désabonnement des contributeurs. L'arbre de décision a montré que si le nombre de fichiers édités ne cesse de baisser, alors le contributeur est susceptible de quitter Core.

[14] WEKA est une plateforme d'algorithmes de data mining écrite en Java. WEKA est disponible sur http://www.cs.waikato.ac.nz/ml/weka/downloading.html

Cependant, nous avons remarqué que l'activité racine de l'arbre de décision n'est pas toujours la même pour chaque projet. Nous émettons l'hypothèse que cette différence est due à l'étape de progression de chaque projet. Par exemple, il est plus facile de faire partie de l'équipe de base de TensorFlow, un projet relativement nouveau sur GitHub, en effectuant de nouveaux changements.

Tableau 3-4 : Résultat de l'Arbre de Décision (J48)

	Correctement classifié	Mal classifié	Précision	Sensibilité	Activité Root
AngularJs	20	7	74.07	.93	# Fichiers edités
Moby	43	0	100	1	# commits
Rails	39	14	73.58	.84	CommentairesSurRévisions
Symfony	19	13	59.37	.94	CommentairesSurRévisions
TensorFlow	23	3	88.46	.70	# commits

3.6. Discussion

Dans cette étude empirique nous avons étudié la dynamique temporelle de cinq projets open source. Cette étude a plusieurs implications pratiques.

Comprendre l'implication des contributeurs et leur notoriété peut aider les communautés open source à attirer des individus plus intéressants et plus motivés. De plus, l'analyse de l'historique de l'activité des contributeurs peut aider à construire une communauté durable de contributeurs autour des projets open source.

Fournir des lignes directrices pour ceux qui veulent guider les futures décisions d'un projet open source. Nous avons constaté que l'ascension d'un nouveau venu vers un contributeur Core est associée principalement à la contribution technique. Plus précisément à la quantité de changements de code et les interactions avec le code source existant, ou encore, le nombre de commits et le nombre de lignes de code ajoutées plutôt que d'autres activités telles que « commenter et examiner d'autres travaux ». Cela peut refléter certaines différences inhérentes entre les projets open source et les projets industriels dans lesquels nous avons d'autres contributions collaboratives telles que l'analyse des besoins, les tests, etc.

Prédire qui va quitter et qui va rester est important pour les chefs de projets dans la mesure où ils doivent faire un effort pour ne pas perdre les contributeurs Core et aider les nouveaux arrivants à améliorer leurs comportements. Nous contribuons à ce corpus de connaissances avec notre approche qui peut être utilisée pour construire un modèle prédictif.

En résumé, comprendre le passage des contributeurs périphériques aux contributeurs principaux et ensuite à l'équipe Core dans les projets open source nécessite une compréhension des activités collaboratives ainsi que la motivation derrière l'engagement des développeurs

Nous reconnaissons quelques limites de validité pour nos résultats rapportés. Premièrement, nous n'avons pas vérifié la base de données de bogues pour évaluer la qualité des contributions. Nous comptons plutôt sur les commentaires commits et les révisions de code que les communautés open source utilisent pour améliorer la qualité du logiciel. Notre choix a été justifié, car nous supposons que les équipes Core ont acquis leur notoriété en performant aussi sur la qualité. Deuxièmement, nous avons émis certaines hypothèses en fonction de la façon dont nous avons découpé et construit nos graphiques de coédition, nous avons considéré les données cumulées pour le réseau de coédition par mois, et ainsi, construit des réseaux cumulatifs depuis le début des projets jusqu'au mois étudié. Nous avons considéré la collaboration de code source comme une activité durable dans la mesure où les contributeurs s'appuient sur une succession de modifications pour produire du logiciel.

Conclusion

Dans ce chapitre, nous avons étudié l'évolution temporelle de la dynamique de collaboration de de cinq projets de logiciels open source. Nous avons analysé l'évolution des contributeurs des équipes périphériques aux équipes Core, nous avons présenté une visualisation dynamique basée sur l'analyse de séries temporelles en divisant la longue période du projet en plusieurs périodes cumulatives consécutives (une par mois). Nous avons ensuite utilisé l'approche proposée dans le CHAPITRE 2 de regroupement k-means basée sur les métriques SNA de centralité afin de classer de manière dynamique les contributeurs dans les réseaux de collaboration mensuels en trois états : Core, Transitoire et Périphérie.

Nous nous sommes également intéressés à quantifier le nombre de contributeurs des équipes périphériques en transition. Nous avons observé une évolution mensuelle des contributeurs Core comprise entre [0.4% et 10.4%]. Un pourcentage de 97% à 99% de l'équipe principale reste dans le noyau.

Les informations sur les rôles de développeur sont essentielles pour comprendre la dynamique de collaboration du projet. Nos résultats suggèrent que les activités de collaboration les plus importantes pour rejoindre et rester au sein de l'équipe centrale d'un logiciel open source sont les activités liées aux modifications du code source (#commits, #LOC). Plus un nouveau

contributeur soumet des modifications du code source, plus il sera rapide et susceptible de passer à l'équipe Core.

Enfin, selon l'étape de progression du projet, une baisse de certaines activités collaboratives, telles que le nombre des commits, prédit qui quittera le cœur du projet. Nos travaux futurs seront davantage axés sur une étude qualitative visant à obtenir davantage d'informations des contributeurs qui ont effectué la transition et qui sont devenus des membres Core.

Ce chapitre a présenté notre seconde étude empirique autour de l'évolution temporelle des communautés open source. Le chapitre suivant étudiera les interactions textuelles entre les membres de la communauté open source et plus précisément l'expression des sentiments au cours de la révision du code source des autres contributeurs.

CHAPITRE 4. IMPACT DES SENTIMENTS SUR LA PRODUCTIVITÉ DES COMMUNAUTÉS OSS

Les examens de code modernes sont pris en charge par des outils permettant d'améliorer les interactions des développeurs, ce qui permet aux contributeurs de soumettre leurs opinions pour chaque changement apporté sous forme de commentaires. Bien que les commentaires visent à discuter des problèmes techniques potentiels, le texte pourrait contenir des sentiments préjudiciables qui pourraient éroder les avantages des modifications suggérées. Dans ce chapitre, nous présentons une étude empirique sur l'impact du sentiment[15] *incarné dans les commentaires des développeurs open source sur le temps et les résultats du processus de révision du code. Les sections de ce chapitre sont organisées comme suit : Nos présentons tout d'abord nos motivations et objectives (section 4.1). Ensuite, la section. 4.2 fournit des informations générales sur le contexte de l'étude. Puis, la section 4.3 présente la revue de la littérature. La section 4.5 décrit le modèle de notre étude de cas. La section 4.6 présente nos résultats. Enfin, nous concluons le chapitre par une discussion sur nos contributions ainsi que les limites de notre étude (section 4.7) suivie par une conclusion de l'étude.*

[15] Le contenu de ce chapitre a fait sujet d'un papier journal intitulé « **An Empirical Study of Sentiments in Code Reviews** », soumis à Information and Software Technology-IST. Ce travail est réalisé en collaboration avec SWAT Team, École Polytechnique de Montréal, Canada.

4.1. Motivations et Objectifs

L'examen de code par les pairs est la pratique par laquelle un développeur soumet un morceau de code (c-à-d. des modifications de code) à des pairs afin de juger son admissibilité à être intégré dans la branche principale du projet (Bosu et al. 2013). Des études antérieures ont montré que les listes de diffusion des communautés virtuelles contiennent non seulement des informations utiles telles que des idées d'amélioration, mais aussi des opinions et des impressions des contributeurs sur les changements introduits (Alessandro et al. 2014). En plus, les avis des développeurs jouent un rôle clé dans le processus de prise de décision relative à la révision du code source (Lee et al. 2008, Kucuktunc et al. 2012, Garcia et al. 2013). Toutefois, l'impact des sentiments exprimés sur l'efficacité du processus d'examen est moins investigué.

Notre objectif global est de comprendre l'influence du sentiment exprimé, tout au long des commentaires, sur le temps et les résultats des révisions du code source. Notre étude fournira aux parties prenantes une meilleure compréhension de l'impact des sentiments des contributeurs sur leur productivité

Afin de comprendre la prévalence et l'impact des sentiments dans les révisions de code modernes, nous avons étudié empiriquement les révisions de code de quatre projets logiciels open source de longue durée. En particulier, nous aborderons les questions de recherche suivantes :

- **QR1. Quelle est la précision des outils de détection des sentiments appliqués dans le contexte des révisions du code ?**

Une étude récente (Lin et al. 2018) a bien soulevé des incertitudes liées à l'utilisation des outils d'analyse des sentiments dans le contexte du génie logiciel. Ainsi, nous avons mené une étude comparative basée sur des échantillonnages de trois outils de détection de sentiments largement utilisés dans la recherche en génie logiciel afin de sélectionner l'outil le plus adapté à notre contexte d'étude.

- **QR2. Quelle est la prévalence des sentiments dans les révisions de code ?**

Une fois l'outil le plus performant est sélectionné, nous quantifions l'expression des sentiments dans les discussions entre les contributeurs open source lors de la révision du code source.

- **QR3. Comment les sentiments exprimés contrastent-ils entre les contributeurs Core et périphériques ?**

Nous sommes intéressés à analyser les différences potentielles dans les sentiments exprimés entre les contributeurs Core et ceux périphériques. Notre objectif principal est d'étudier la corrélation entre un gain de réputation des contributeurs et la nature des sentiments qu'ils expriment dans les commentaires sur les révisions de code. Nous allons aussi surveiller l'évolution des sentiments des cinq top contributeurs au fil du temps, pour les quatre projets open source, à mesure qu'ils progressent et acquièrent plus d'expérience, afin de comprendre la corrélation entre la notoriété (expérience) et la tendance des sentiments exprimés dans les interactions textuelles.

- **QR4. Est-ce les sentiments exprimés ont un impact sur la durée et le résultat des révisions de code ?**

L'objectif de cette question de recherche est d'examiner l'effet des sentiments exprimés par les contributeurs open source sur le résultat de la révision en termes de duré et état finale.

4.2. Contexte de l'étude

4.2.1. Analyse des sentiments

Actuellement, l'analyse des sentiments est un sujet d'intérêt et de développement en raison de ses nombreuses applications pratiques (Bo et al. 2002, Lee et al. 2008). Étant donné que les informations disponibles publiquement et à titre privé sur Internet sont en constante augmentation, un grand nombre de textes exprimant des sentiments sont disponibles sur des sites de retours d'avis, des forums, des blogues et des médias sociaux. A l'aide de systèmes d'analyse des sentiments, ces informations non structurées pourraient être automatiquement transformées en données structurées de l'opinion publique sur les produits, les services, les marques, la politique ou tout sujet sur lequel les gens peuvent exprimer des opinions (Piryani et al. 2016). Ces données peuvent être très utiles pour des applications commerciales telles que l'analyse marketing, les relations publiques, les critiques de produits, le scoring des promoteurs de réseau, les commentaires sur les produits et le service client.

En règle générale, l'analyse des sentiments dans les données textuelles peut être calculée à plusieurs niveaux, y compris au niveau d'une phrase, d'un paragraphe ou de l'ensemble du document. Souvent, le sentiment est calculé sur le document dans son ensemble ou certaines agrégations sont effectuées après avoir calculé le sentiment pour des phrases individuelles.

4.2.2. Pratique moderne de révision de code : Gerrit

Gerrit est un outil de révision de code basé sur le Web basé sur le système de contrôle de version « Git ». Il est aussi un outil d'analyse de code utilisé par des projets open source comme Android et Eclipse. Il permet aux équipes de discuter du code, de servir Git en tant qu'expérience intégrée dans le flux de révision de code plus large et de gérer les flux de travail avec des contrôles d'accès profondément intégrés et délégables (Mukadam et al. 2013). L'aspect le plus important de Gerrit est la collaboration autour de la propriété collective du code ou les gens sont encouragés à inspecter le code et se sentent plus à l'aise pour exprimer leurs opinions. Gerrit est conçu pour fournir un cadre léger permettant de réviser chaque commit avant qu'il ne soit accepté dans la base de code. Les modifications sont téléchargées vers Gerrit, mais ne font pas partie du projet tant qu'elles n'ont pas été examinées et acceptées. Plus concrètement il s'agit d'un outil destiné à prendre en charge le processus standard open source de soumission des révisions, qui sont ensuite examinés par les membres du projet avant d'être appliqués à la base de code. Cependant, Gerrit va encore plus loin en simplifiant la tâche de tous les auteurs d'un projet afin de s'assurer que les modifications sont vérifiées avant leur application. Gerrit crée également un compte-rendu durable de la conversation qui peut être utile pour répondre aux questions inévitables telle que "Je sais que nous avons changé cela pour une raison".

La Figure 4-1 illustre le processus global de révision de code dans Gerrit. Gerrit a trois rôles : Auteur, Examinateur et Vérificateur. Les auteurs valident les changements de code dans le système de contrôle des versions (VCS) et demandent une révision. Les examinateurs sont responsables de passer tout au long des changements et de proposer ensuite des ajustements si nécessaire, exprimés sous forme de commentaires, pour que l'auteur puisse adresser les commentaires et produire une nouvelle révision du code. Les vérificateurs sont responsables de l'exécution des tests pour s'assurer que les modifications proposées sont exemptes de bogues et n'entraînent aucune régression du système. Ils peuvent également laisser des commentaires pour décrire les problèmes de vérification qu'ils ont rencontré lors des tests. Une fois les critères d'examen vérifiés, les modifications sont intégrées dans le référentiel principal et signalées « Mergée ». Ce cycle de vie peut avoir une autre transition « Abandonnée » lorsque la révision n'a pas passé l'évaluation et n'est plus active.

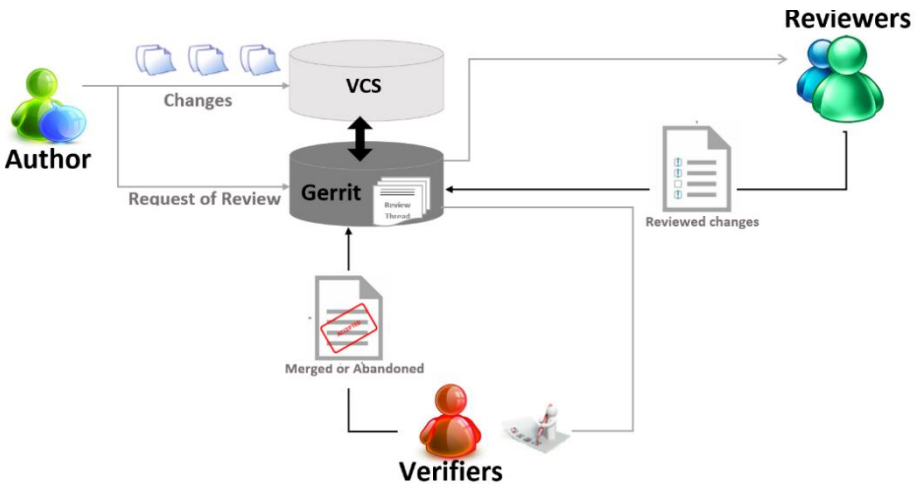


Figure 4-1 : Flux de révision de code dans Gerrit

4.2.3. Facteurs influant la révision du code

L'une des principales préoccupations des développeurs, lors de la soumission d'une proposition pour la révision du code, est de maximiser les chances que leurs suggestions soient examinées dans les plus brefs délais. Cependant, le résultat et la durée du processus de révision du code peuvent être affectés par divers facteurs techniques (Baysal et al. 2015). Ces facteurs peuvent introduire un certain biais lors de l'analyse de l'effet réel des sentiments exprimés par les contributeurs sur les délais de révision et les résultats des examens.

Le facteur le plus intuitif est la taille des patchs (**Churn**) ; Des études antérieures ont montré que les petites parcelles sont plus susceptibles de recevoir des réponses plus rapides (Rigby et al. 2008, Weissgerber et al. 2008, Jiang et al. 2013, Kononenko et al. 2015), car les plus grandes portions de code seraient plus difficiles à revoir et nécessiteraient donc plus de temps. Un autre facteur important est le nombre de fois qu'un développeur a dû soumettre à nouveau son patch[16] pour une révision supplémentaire (**Nombre de Patches**) ; Un patch nécessitant plusieurs révisions et soumissions avant d'être accepté consomme plus de temps. En plus, plus un changement est répandu dans les fichiers **(Nombre de fichiers édités**), plus il touche différentes composantes dans un système, ce qui se traduit souvent par plus de retouches (Beller et al. 2014). Sur la base d'une enquête réalisée auprès de 88 principaux développeurs open source, Kononenko et al. (Kononenko et al. 2016) ont confirmé l'influence importante de ces facteurs techniques sur le processus et les résultats des révisions du code. Les auteurs rapportent aussi que la longueur de la discussion (**Nombre des commentaires**) et le nombre de personnes

[16] Un patch est une section de code ajouté à un logiciel, pour y apporter des modifications : correction d'un bogue. Les termes recommandés en France par la DGLFLF (Délégation générale à la langue française et aux langues de France) sont « retouche » ou bien « correctif » cependant nous préférons de garder le terme anglais « Patch »

impliquées dans la discussion (**Nombre distinct de contributeurs impliqués**) ont été considérés, par les contributeurs interrogés, comme des facteurs influençant.

Pour notre étude, nous sélectionnons ces métriques techniques largement utilisées pour caractériser les révisions. Ensuite, nous utilisons la technique d'appariement des scores de propension (Propensity Score Matching-PSM) pour nous assurer que nos analyses ne sont pas biaisées par les différences entre les révisions examinées (voir la section 4.5.2.

4.3. État de l'art

Plusieurs travaux ont attiré l'attention de la communauté des chercheurs sur l'analyse des sentiments. Ces travaux couvrent de nombreux domaines allant du bonheur sur les lieux de travail (Crowston et al. 2006) aux émotions dans les messages des réseaux sociaux tels que Yahoo et Twitter (Onur et al. 2012, Thelwall et al. 2012) et en ligne comme les messages Stack Overflow (Robertson et al. 2011). Guillory et al. (Guillory et al. 2011) sont allés plus loin et ont examiné la propagation des émotions négatives dans les communautés en ligne. Leurs analyses suggèrent que la contagion des émotions négatives peut se produire dans des groupes de personnes et avoir un impact sur leurs performances. Notre travail se focalise sur les communautés open source avec toutes ses spécificités et en particulier sur l'activité crucial de la révision du code source.

Guzman et Bruegge (Guillory et al. 2011) ont présenté un papier qui décrit la conscience émotionnelle dans les équipes de développement de logiciels. Le travail était motivé par les mêmes préoccupations qui ont motivé notre approche. Leur approche étudie la conscience émotionnelle collective des développeurs dans les équipes distribuées. Ils ont extrait l'état émotionnel d'un millier d'artefacts de collaboration visant à résumer les émotions exprimées dans ces artefacts en extrayant les sujets et en leur attribuant un score d'émotion moyen. Les auteurs ont présenté la fluctuation moyenne des émotions aux chefs de projet, qui ont confirmé la corrélation des pics d'émotion positifs et négatifs avec la performance de l'équipe, la motivation et les échéances importantes. Notre travail améliore leur idée en utilisant les discutions autour des demandes de révision de code au lieu des commentaires des commits.

Auparavant, Baysal et al. (Baysal et al. 2013) ont étudié l'impact de facteurs techniques et non techniques sur la durée des révisions de code source. Ils ont observé que des facteurs non techniques, tels que l'expérience des réviseurs, peuvent avoir un impact significatif sur les résultats de la révision du code. Une compréhension empirique de l'impact des sentiments sur le processus de révision du code peut ajouter une nouvelle dimension aux résultats de Baysal et

al. (Baysal et al. 2013) - notamment pour guider la conception de meilleures approches et outils de révision de code pour faciliter une productivité accrue. Sinha et al. (Sinha et al. 2016) ont analysés l'historique des commits pour un grand nombre de projets GitHub et ont constaté que la majorité du sentiment exprimé par les développeurs était neutre. Ils ont également constaté que les commentaires négatifs sont plus présents que les commentaires positifs (respectivement 18,05% contre 7,17%). De même, Guzman et al. (Guzman et al. 2013) ont examiné les sentiments exprimés par les développeurs dans les commentaires liés aux engagements de 29 projets open source et ont trouvé une répartition à peu près égale des sentiments positifs, négatifs et neutres.

Khan et al. (Khan et al. 2010) ont mené deux études sur l'impact des sentiments sur la performance des développeurs. Ils ont constaté que l'ambiance des programmeurs avait une influence positive sur certaines tâches de programmation telles que le débogage. Similairement Ortu et al. (Ortu et al. 2016) ont étudié l'impact de l'affectivité des développeurs sur la productivité en mettant l'accent sur la corrélation entre les états émotionnels et la productivité en termes de temps de résolution des problèmes. Ils indiquent que plus les développeurs sont heureux, c'est-à-dire qu'ils expriment des émotions telles que la joie et l'amour dans leurs commentaires, plus le temps de résolution des problèmes sera probablement court. Ils signalent également que des émotions telle que la tristesse sont liées à un temps de résolution des problèmes plus long. Nous complétons les travaux existants sur l'impact du sentiment sur la productivité en étudiant l'influence du sentiment exprimé sur la durée et les résultats des examens de code source.

Des études récentes ont examiné les facteurs affectant l'efficacité des commentaires de révision de code. (Rahman et al. 2017). Ils ont extrait un certain nombre trait du texte des commentaires de révision en essayant de prédire l'utilité des commentaires de révision de code à l'aide de l'analyse textuelle. Cependant, leur étude empirique s'est limitée aux caractéristiques structurelles du texte sans prendre en compte les émotions / sentiments exprimés. Efstathiou and Spinellis (Efstathiou et al. 2018) ont étudié la langue (c.-à-d. la communication humaine écrite) des commentaires de révision de code et ont signalé que la langue a de l'importance. Dans cette étude, nous poursuivons cette voie pour enquêter sur le rôle des sentiments exprimés dans les commentaires de révision de code sur les résultats de la révision de code.

4.4. Préparation des données de Gerrit

Notre étude empirique est basée sur des données de révision de code accessibles au public, extraites du système Gerrit et organisées dans un dump de base de données (Yang et al. 2016). Nous avons sélectionné quatre systèmes open source bien connu, *OpenStack*[17], *Eclipse*[18], *Android*[19] et *LibreOffice*[20]. OpenStack est un ensemble de logiciels open source permettant de déployer des infrastructures de cloud computing (infrastructure en tant que service). Eclipse est un environnement de développement intégré (IDE) utilisé dans la programmation informatique. Android est une pile de logiciels gratuits pour un large éventail d'appareils mobiles dirigés par Google. Enfin, LibreOffice est un fork du projet OpenOffice.org. Nous avons sélectionné ces projets, car ils ont été activement développés pendant plus de cinq ans et fournissent donc une riche source de données. En outre, ils proviennent de différents domaines, sont écrits dans différents langages de programmation et ont été étudiés dans d'autres domaines de recherche.

L'ensemble de données[21] est stocké dans une base de données relationnelle (335 626 révisions et plus de 5 millions commentaires) sous le schéma représenté sur la Figure 4-2. En général, un contributeur (ie, personId, name, email) demande une révision caractérisée par un reviewId, une date de création (createdAt), une date de dernière modification (updatedAt), le projet associé et la branche du code source. Une révision inclut un ensemble de patches, lorsque l'auteur met régulièrement à jour la modification en validant de nouvelles soumissions avec le même reviewId, et une liste de fichiers modifiés. L'historique de la discussion lancée autour les modifications proposées est enregistré dans la table « Comment ».

[17] http://openstack.org
[18] https://eclipse.org/
[19] https://source.android.com/
[20] https://www.libreoffice.org/
[21] http://kin-y.github.io/miningReviewRepo/

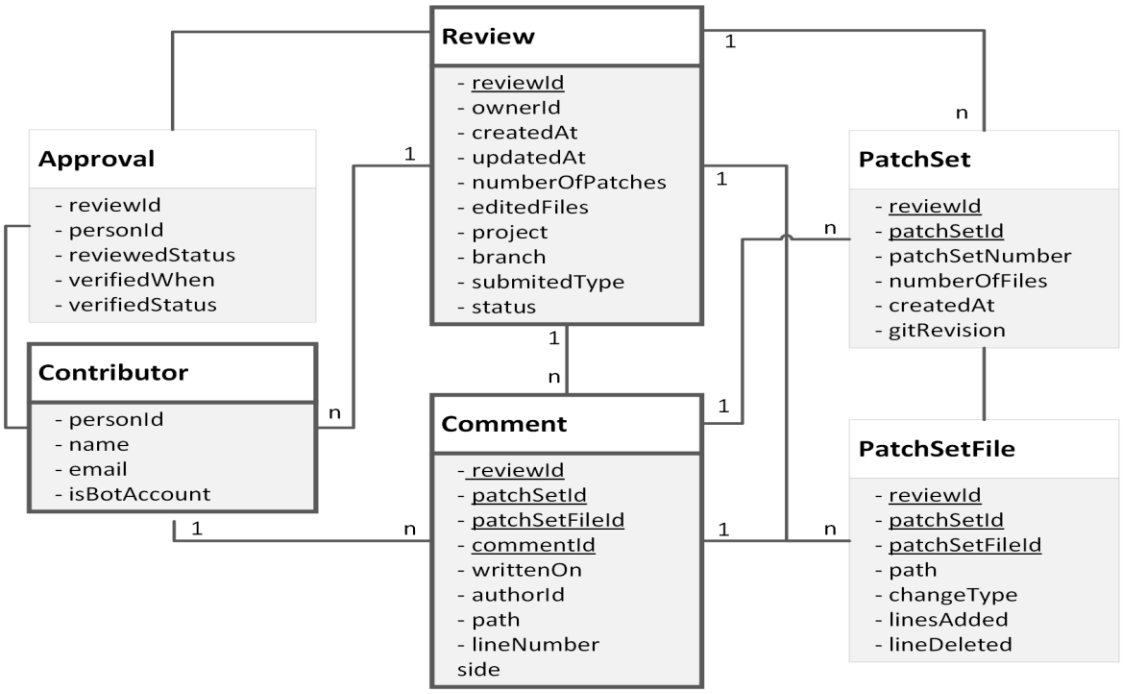


Figure 4-2 : Schéma simplifié de la base de données Gerrit.

Nous avons récupéré et exporté les données requises dans des fichiers csv distincts pour faciliter le prétraitement de nos données. Le Tableau 4-1 présente des statistiques descriptives des projets étudiés.

Tableau 4-1 : Vue d'ensemble des systèmes étudiés

Projet	#Révisions	#Commentaires	#Contributeurs
Openstack	228,099	5,021,264	8,088
Eclipse	15,887	153,176	1,082
Android	63,610	355,765	3,334
LibreOffice	28,030	174,181	634

Afin d'améliorer la qualité de notre ensemble de données par rapport à notre objectif principal, qui est d'étudier les sentiments humains exprimés dans les commentaires de révision de code, nous avons effectué trois étapes de prétraitement sur les données brutes :

1. Nous avons écarté les commentaires générés automatiquement tels que ceux générés par les services d'intégration continue. Ces commentaires contiennent des mots-clés tels que : « Jenkins », « Hudson », « Bot » ou «CI Servers » (environ 29% du total des commentaires ont été exclus). À l'aide d'expressions régulières, nous avons exclu les expressions automatiques (par exemple, « Build succeed », « Build failed », etc.).

2. En outre, nous avons éliminé les révisions avec statut = "Nouveau", car leur statut final reste inconnu (~ 5% du total des révisions). Nous avons limité notre analyse aux révisions clôturées, c'est-à-dire marquées comme "Mergée" ou "Abandonnée", et qui contiennent au moins un commentaire.
3. Pour chaque révision, nous avons recueilli des informations clés telles que l'horodatage de l'ouverture et de la clôture de la révision, le nombre de fichiers édités, le nombre de Patches et le nombre de lignes de code ajoutées et supprimées.

4.5. Cadre d'analyse

La Figure 4-3 présente un aperçu des étapes de notre étude et de leur relation avec nos questions de recherche.

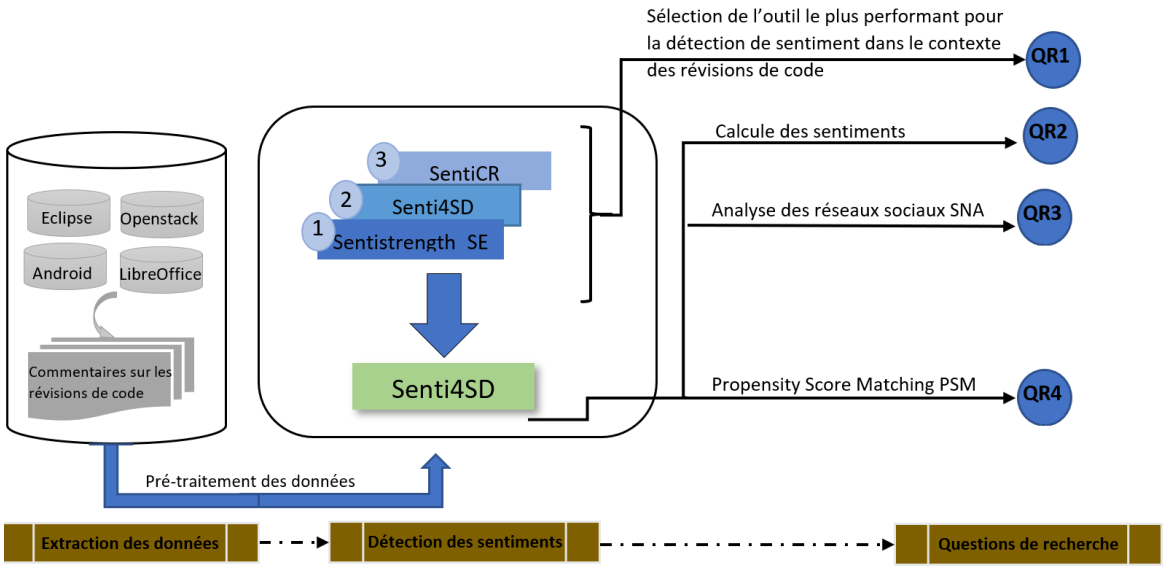


Figure 4-3 : Modèle de l'étude.

Le modèle suivi lors de cette étude comporte trois volets importants. D'abord la préparation des données détaillée dans la section 4.4 ou nous avons expliqué notre approche de sélection des projets étudiés et le pré-traitement des données. Ensuite la détection des sentiments, cette étape a consisté en la sélection de l'outil le plus précis pour la détection des sentiments dans les commentaires de révision de code à travers la comparaison de la performance de trois outils reportés dans la littérature. En fin le dernier volet du modèle présente les questions de recherche.

4.5.1. Outils de détection des sentiments

Cette section présente un résumé des outils d'analyse de sentiment existants conçus et testés dans le contexte d'ingénierie logiciel. La littérature cite quatre outils :

1. **Sentistrength** (Thelwall et al. 2012) : L'outil le plus largement utilisé dans la communauté du génie logiciel (SE) est SentiStrength (Thelwall et al. 2010). SentiStrength utilise une approche lexicale qui exploite une liste de termes liés au sentiment et dispose de règles pour traiter les méthodes linguistiques et sociales standard du Web pour exprimer des sentiments, tels que des émoticônes, une ponctuation exagérée et des fautes d'orthographe délibérées (Thelwall et al. 2010).
2. **SentistrengthSE** (Islam et al. 2017) : Un outil d'analyse de sentiment amélioré spécialement conçu pour une application dans le domaine du génie logiciel. SentiStrength_SE a été développé sur la base de SentiStrength, y compris les nouvelles décisions de conception et les heuristiques.
3. **Senti4SD** (Calefato et al. 2018) : Cet outil introduit un classificateur de polarité supervisé qui est formé sur un jeu de données de 4000 messages (questions, réponses et commentaires) provenant de Stack Overflow. Chaque message a été annoté manuellement pour la polarité du sentiment. Le classificateur est formé pour détecter trois types de polarité : positif, négatif et neutre. Senti4SD utilise des fonctionnalités linguistiques, telles que des ngrams, des lexiques de sentiment et des fonctionnalités sémantiques basées sur les mots incorporés.
4. **SentiCR** (Ahmed et al. 2017) : Un classificateur de polarité supervisé qui est formé sur un jeu de données de 1600 révisions de Gerrit. Contrairement à chacune des techniques ci-dessus qui détecte trois types de polarité (positif, négatif et neutre), le classificateur SentiCR décrit dans (Ahmed et al. 2017) détecte uniquement deux types de polarité: négatif et non négatif.

4.5.2. Propensity Score Matching

Nous utilisons la méthode Propensity Score Matching (PSM)[22] (Thavaneswaran 2008) pour regrouper les révisions de manière homogène en fonction de certaines caractéristiques (par exemple, la taille de la révision, le nombre de lignes de code, le nombre de commentaires, etc.). La PSM est une technique d'appariement statistique largement utilisée pour compresser les covariables en une seule variable (c.-à-d. Comparer des facteurs techniques et générer un score de propension). Le PSM est proposé pour traiter les effets de facteurs de confusion. (Guo et al.

[22] https://en.wikipedia.org/wiki/Propensity_score_matching

2018) présente une évaluation de l'efficacité de la PSM dans l'atténuation des facteurs de confusion.

Les caractéristiques techniques considérées dans notre étude sont :

- **Nombre de commentaires :** nombre de commentaires postés sur chaque demande de révision de code (par exemple, à propos du changement de code proposé).
- **Nombre de Patches :** nombre de patchs soumis avant que le changement de code proposé ne soit accepté ou rejeté.
- **Fichiers modifiés** (nombre discret) : nombre de fichiers modifiés par le changement de code proposé.
- **Contributeurs impliqués** (nombre discret) : nombre de développeurs ayant participé à la révision du code proposé.
- **Churn** (nombre cumulé) : nombre de lignes ajoutées et supprimées dans les modifications de code examinées

Dans ce travail, nous avons utilisé le package R appelé ***Matchit*** pour effectuer les deux premières étapes énumérées ci-dessous, tandis que l'étape 3 nécessitait une vérification manuelle. Les trois étapes sont décrites comme suit :

1) Un modèle de régression logistique est construit sur la base d'un ensemble de caractéristiques de grande dimension. Le sentiment de révision (positif ou négatif) est défini comme variable dépendante et les caractéristiques techniques des révisions ; (i) nombre de commentaires pour la révision; (ii) nombre de Patches, (iii) nombre de fichiers édités, (iv) différents contributeurs impliqués et (v) churn ; sont définis en tant que variables indépendantes. Le résultat du modèle de régression logistique est une valeur ajustée (une valeur de probabilité) appelée score de propension.
2) Les scores de propension sont utilisés pour apparier des paires de points de données. Chaque paire a des valeurs différentes de la variable dépendante. Des valeurs similaires du score de propension impliquent une similitude des caractéristiques techniques des critiques. Pour ce faire, nous avons utilisé l'algorithme d'appariement génétique (Genetic matching) pour apparier les paires de révisions appropriées. Les paires appariées sont ensuite combinées dans un nouvel ensemble de données.
3) La dernière étape consiste à vérifier l’équilibre des caractéristiques des covariables. Pour ce faire, nous avons comparé manuellement les différences de moyennes pour chaque variable au sein du groupe apparié.

Bien que l'étape finale de la PMS consiste à vérifier le l'équilibre des caractéristiques des variables, nous avons effectué une validation manuelle du biais de confusion sur les résultats de la PSM, comme indiqué dans le Tableau 4-2. Par exemple, la différence moyenne entre le total des commentaires pour les révisions positives et négatives est passée de (9.96, 12.6) à (6.2, 6.2), ce qui signifie des groupes plus homogènes. On peut également remarquer de plus grandes p_values[23] avec entre les révisions appariées ce qui signifie une distribution équivalente en ce qui concerne les caractéristiques techniques.

Tableau 4-2 : Ajustement des caractéristiques techniques avec PSM (projet Eclipse).

	Avant PSM			**Après PSM**		
	Révisions Positives	***Révisions Négatives***	***P_value***	***Révisions Positives***	***Révisions Négatives***	***P_value***
Nombre de commentaires	9.96	12.60	9.07e-06	6.20	6.20	0.09
Patchesets	2.60	2.63	2e-03	1.51	1.51	0.06
Fichiers édités	13.25	14.22	0.33	1.66	1.66	0.37
Churn	4993.70	8431.20	0.06	51.49	42.29	0.34
Contributeurs impliqués	2.49	3.27	1.60e-7	2.66	2.66	2e-4

4.6. Résultats

Nous présentons maintenant les résultats de l'analyse des sentiments menée sur les quatre projets open source. Pour chacune de nos quatre questions de recherche, nous présentons notre motivation, notre approche et nos résultats.

QR1. Quelle est la précision des outils de détection des sentiments appliqués dans le contexte des révisions du code ?

Motivation. Pour déterminer si les sentiments négatifs et positifs, exprimés dans les interactions de révision textuelle des développeurs, affectent le processus de révision du code, nous avons besoin d'un outil capable de détecter avec précision les sentiments dans les commentaires de révision de code. Cependant, jusqu'à présent, trois moteurs de détection de sentiment différents ont été proposés dans la littérature en génie logiciel (Ahmed et al. 2017, Islam et al. 2017, Calefato et al. 2018), mais n'ont pas été spécifiquement conçue pour le contexte de la révision de code. Les tentatives précédentes d'analyser les sentiments basés sur le texte pour le génie logiciel ont été soit incomplètes, soit non réutilisables pour d'autres domaines (Efstathiou et al. 2018, Lin et al. 2018). Une raison majeure de cette insuffisance est

[23] Les p_values sont calculées à l'aide du test de Mann-Whitney U.

que le génie logiciel englobe le vocabulaire de divers sous-domaines (Lin et al. 2018). Par conséquent, un outil testé avec succès dans un sous-domaine peut ne pas être assez utile pour un autre sous-domaine. Par conséquent, nous avons été plus prudents quant au choix de notre outil.

Approche. Nous avons comparé les performances de trois moteurs de détection de sentiment (Sentistrength_SE[24] (Islam et al. 2017), Senti4SD (Calefato et al. 2018), et SentiCR (Ahmed et al. 2017)) visant à choisir l'outil le plus adapté au domaine de révision du code source. Ces trois outils ont déjà été utilisés pour détecter des sentiments dans le contexte du génie logiciel en utilisant différents types de jeux de données. Afin de comparer prudemment les performances des trois outils, nous avons effectué une annotation manuelle (par quatre évaluateurs) sur un sous-ensemble de commentaires. Pour renforcer notre échantillonnage, nous avons construit une approche de suréchantillonnage dans laquelle la classe minoritaire (c.-à-d., sentiments négatifs) est également représenté. Concrètement, suivant l'approche utilisée par Novieli et al. (Novielli et al. 2018), nous avons construit quatre sous-ensembles de données en effectuant un échantillonnage opportuniste. Le premier échantillon est créé sur la base de la sortie de SentiStrength_SE, il contient 1200 commentaires équitablement répartis (pour chaque projet, nous avons 100 positifs, 100 négatifs et 100 neutres, ce qui rend 300X4 = 1200). Les deuxième et troisième échantillons extraits respectivement après l'application de Senti4SD et SentiCR contiennent 360 commentaires. Le dernier échantillon contient 300 commentaires aléatoires. Ensuite, chaque commentaire a été annoté manuellement (positif, négatif, neutre) par deux auteurs. L'accord entre les deux codeurs, mesuré à l'aide du kappa de Cohen, allait de 81% à 95% (83% pour l'échantillon Senti4SD, 95% pour l'échantillon SentiCR, 81% pour SentiStrengthSE et 91% pour échantillon aléatoire). Pour résoudre les désaccords entre les rapporteurs, les annotations ont été discutées et le guide d'annotation a été mis à jour par le premier auteur. Par exemple, un exemple de désaccord entre deux annotateurs s'est produit pour la phrase suivante: « 'Patch Set 2: Fails Merges in public tree, but does not build. Please fix and reupload. Thanks!' ». Le premier annotateur a classé ce commentaire comme positif, tandis que le second l'a classé comme négatif. Par consentement mutuel, nous avons décidé de qualifier ce commentaire typique de positif, le commentateur étant très poli et ayant utilisé les termes "Please" et "Thanks" dans son texte.

[24] SentiStrength n'est pas inclus dans cette analyse car SentiStrength_SE est normalement la version de SentiStrength adaptée au contexte du génie logiciel.

Nos sous-ensembles de données ainsi que l'ensemble de données d'origine (5 millions de commentaires) sont disponibles dans l'annexe associée en ligne (El Asri et al. 2018) pour des fins de réplication.

Résultats. Le Tableau 4-3 présente les performances obtenues en termes de rappel (R), de précision (P) et de mesure F1, pour chaque classe de polarité (positive, négative et neutre), ainsi que les performances globales, pour les trois outils dans les différents échantillons de données. Nous mettons en gras les meilleures valeurs pour chaque métrique.

Tableau 4-3 : Fiabilité des outils de détection de sentiment dans le contexte de révision de code

Échantillon de données	Class	Sentistrength_SE			Senti4SD			SentiCR		
		P	R	F1	P	R	F1	P	R	F1
Sentistrength_SE	Positive	**0.86**	0.85	**0.83**	0.81	0.84	0.82	0.59	**0.89**	0.71
	Négative	**0.91**	0.61	**0.73**	0.66	**0.68**	0.67	0.59	0.66	0.62
	Neutre	0.7	**0.98**	0.81	0.83	0.81	**0.82**	**0.88**	0.69	0.77
	Micro-avg	**0.8**	**0.8**	**0.8**	0.79	0.79	0.79	0.72	0.72	0.72
	Macro-avg	**0.82**	**0.8**	**0.79**	0.77	0.77	0.77	0.69	0.75	0.7
Senti4SD	Positive	0.83	**0.92**	0.87	**0.91**	**0.91**	**0.91**	0.3	0.81	0.43
	Négative	0.57	**0.8**	0.67	**1**	0.78	**0.87**	0.42	0.8	0.55
	Neutre	0.89	0.7	0.78	0.79	**0.96**	**0.87**	**0.93**	0.51	0.66
	Micro-avg	0.78	0.78	0.78	**0.88**	**0.88**	**0.88**	0.59	0.59	0.59
	Macro-avg	0.76	0.81	0.77	**0.9**	**0.88**	**0.88**	0.55	0.71	0.55
SentiCR	Positive	0.6	**0.82**	0.7	**0.74**	0.72	**0.73**	0.73	0.7	0.72
	Négative	0.52	0.4	0.45	0.67	**0.46**	**0.55**	**0.91**	0.25	0.4
	Neutre	**0.82**	0.76	0.79	0.76	0.83	0.79	0.53	**0.93**	0.67
	Micro-avg	0.73	0.73	0.73	**0.75**	**0.75**	**0.75**	0.63	0.63	0.63
	Macro-avg	0.65	0.66	0.64	**0.72**	**0.67**	**0.69**	**0.72**	0.63	0.6
Aléatoires	Positive	0.27	**0.95**	0.42	**0.83**	0.89	**0.86**	0.05	0.44	0.09
	Négative	0.11	0.15	0.13	**0.58**	**0.76**	**0.66**	0.05	0.14	0.08
	Neutre	0.95	0.75	0.84	**0.97**	**0.93**	**0.95**	0.96	0.71	0.81
	Micro-avg	0.74	0.74	0.74	**0.92**	**0.92**	**0.92**	0.69	0.69	0.69
	Macro-avg	0.44	0.62	0.46	**0.8**	**0.86**	**0.82**	0.35	0.43	0.33
	Total micro avg	0.7625	0.7625	0.7625	**0.835**	**0.835**	**0.835**	0.6575	0.6575	0.6575
	Total Macro avg	0.6675	0.7225	0.665	**0.7975**	**0.795**	**0.79**	0.5775	0.63	0.545

Senti4SD affiche une performance globale légèrement supérieure à celle des autres outils (F1 = 0,79). Nous utilisons la mesure F1 pour déterminer les classificateurs les plus performants, conformément aux pratiques standard en matière de recherche d'informations (Christopher et al. 2008). En moyenne, Senti4SD a obtenu les meilleures performances (précision 79%, F1 79%) lorsqu'il a été appliqué à nos échantillons de données de révision de code.

Encore une fois, Lin et al (Lin et al. 2018) ont souligné que les outils d'analyse des sentiments devraient toujours être soigneusement évalués dans le contexte spécifique de l'utilisation. Nous vérifions nos résultats deux fois en effectuant un test statistique McNemar[25] (Chen et al. 2009) afin de comparer les résultats de classification des trois outils. Les différences de performance entre Senti4SD et les autres classificateurs se sont révélées statistiquement significatives (p_value < 0,05 et z scores = 11,49> 0), ce qui indique que Senti4SD donne de meilleurs résultats que SentiCR. De plus, en comparant ce résultat avec notre marquage manuel, nous avons constaté que 472 commentaires étaient correctement classés par Senti4SD et classés incorrectement par SentiCR, alors que seuls 178 commentaires étaient correctement classés par SentiCR et classés incorrectement par Senti4SD. Ainsi les résultats de Senti4SD seront utilisés pour répondre à la suite de nos questions de recherche.

QR2. Quelle est la prévalence des sentiments dans les révisions de code ?

Motivation. Les sentiments sont omniprésents dans l'activité humaine. Comme l'ancien proverbe dit « Feeling Good-Doing Good » (George et al. 1992), les contributeurs open source peuvent sous-performer s'ils ne se sentent pas heureux et en sécurité (Khan et al. 2010). Les sentiments négatives comme la colère aident les gens à être moins motivés et donc moins créatifs (Ortu et al. 2015). Dernièrement, Linus Torvalds a envoyé un courrier électronique[26] à la communauté des développeurs Linux, reconnaissant ses abus verbaux dans les communications. Il a ecrit: *["My flippant attacks in emails have been both unprofessional and uncalled for,"*]. Torvalds s'est excusé parce que des gens se plaignaient de son manque de sollicitude dans ses communications, ce qui a blessé certains contributeurs et en a peut-être empêché certains de travailler au développement du noyau [*"I'm going to take time off and get some assistance on how to understand people's emotions and respond appropriately"*.].

Des preuves empiriques de l'effet des sentiments exprimés dans les commentaires sur les révisions de code pourraient aider les développeurs à accorder plus d'attention à la façon dont

[25] https://stat.ethz.ch/R-manual/R-devel/library/stats/html/mcnemar.test.html
[26] 0https://gizmodo.com/linux-founder-takes-some-time-off-to-learn-how-to-stop1829105667

ils commentent le travail des autres, en particulier dans un contexte de communautés virtuelles telles que GitHub caractérisés par des contributeurs multiculturels (Destefanis et al. 2017).

Approche. Nous avons récupéré plus de 5 millions de commentaires sur les révisions de code pour quatre projets open source bien connus et de longue durée: Openstack, Eclipse, Android et LibreOffice. La distribution des commentaires est présentée dans le Tableau 4-4. Ensuite, nous avons effectué une analyse des sentiments sur les commentaires en utilisant des techniques de traitement du langage naturel. Nous avons construit un algorithme entièrement automatisé pour calculer le score de sentiment pour chaque commentaire sur chaque révision en utilisant Senti4SD.

Tableau 4-4 : Distributions de sentiments dans les révisions de code

Projet	Positive	Neutre	Négative	Moyenne	SD	Kurtosis	Skewness
Openstack	16.27%	81.04%	2.68%	0.13	0.41	1.63	0.89
Eclipse	8.31%	89.92%	1.77%	0.06	0.31	6.25	1.53
Android	14.06%	84.09%	1.86%	0.12	0.37	2.43	1.22
LibreOffice	17.14%	80.21%	2.65%	0.14	0.42	0.42	0.87

Résultats. Dans le projet Eclipse, 8,31% des commentaires sont classifié comme positifs (score=1) alors que 89,92% des commentaires sont classifiés neutres (score = 0) et 1,77% sont classifié négatifs (score -1). Le Tableau 4-4 synthétise la distribution des trois types de sentiment pour les projets étudiés et les statistiques descriptives : la moyenne, l'écart-type (SD), le kurtosis et le coefficient d'asymétrie (skewness en anglais). Nous avons remarqué relativement les mêmes distributions concernant la proportion de sentiment exprimé dans les commentaires à travers les quatre projets. Les commentaires neutres sont les plus présents (83,81% en moyenne) ce qui confirme les résultats d'études antérieures (Sinha et al. 2016). La grande quantité de sentiments neutres peut s'expliquer principalement par la présence de vocabulaire technique dans les commentaires.

Nous avons constaté que les commentaires des contributeurs ne sont pas toujours neutres. En effet, 13,94% des commentaires relatifs à tous les projets ont été détecté comme positifs (par exemple, *[Thanks for the most excellent review. :)]*), tandis qu'environ 2.24% des commentaires ont été identifiés comme négatifs (par exemple, *[Horrible :( ])*.

Les valeurs de kurtosis élevées observées dans chaque projet montrent que les sentiments exprimés dans les projets sont variés parmi les contributeurs (rappelons qu'une valeur de Kurtosis de 3 indique que les évaluateurs ont un fort consensus). Globalement, les résultats

révèlent que la distribution est fortement asymétrique (positivement pour Eclipse et Android alors que modérément asymétrique pour Openstack et LibreOffice).

QR3. Comment les sentiments exprimés contrastent-ils entre les contributeurs Core et périphériques ?

Nous formulons les sous questions de recherche suivantes :

- ***QR3.1 : Est-ce que les contributeurs Core et périphériques expriment-ils différents types de sentiments selon leur position dans le réseau social de collaboration ?***
- ***QR3.2 : Comment les sentiments textuels exprimés par les évaluateurs du code source évoluent-ils au fil du temps ?***

Dans la suite, nous répondons aux deux sous-questions.

QR3.1 : Est-ce que les contributeurs Core et périphériques expriment-ils différents types de sentiments selon leur position dans le réseau social de collaboration ?

Motivation. Nous nous intéressons à l'analyse des différences potentielles dans l'expression des sentiments entre les contributeurs Core et ceux périphériques. Les membres Core sont les développeurs qui contribuent de manière intensive et durable au projet OSS, et ainsi, dirigent la communauté, tandis que les périphériques sont des contributeurs occasionnels avec des engagements moins fréquents. Notre objectif principal est d'étudier la corrélation entre un gain de réputation des contributeurs et la nature des sentiments qu'ils expriment dans les commentaires sur les demandes de révision de code. Nous émettons l'hypothèse que les nouveaux arrivants essaient d'imiter les contributeurs avec une certaine réputation, ce qui pourrait affecter la culture du commentaire.

Approche. Pour répondre à QR3 et aux sous-questions de recherche, nous avons construit un réseau social pour chaque projet afin de détecter les contributeurs Core et périphériques. Pour ce faire, nous avons calculé le nombre d'interactions entre chaque paire de développeurs dans chaque projet. Ensuite, nous avons généré des réseaux sociaux sous la forme de graphiques pondérés non dirigés, où les nœuds représentent les développeurs et les poids des arcs représentent la quantité de fichiers co-édités par ces contributeurs. Enfin, pour localiser les contributeurs Cores et périphériques, nous avons utilisé la même approche présentée dans CHAPITRE 2 section 2.4.. Le Tableau 4-5 fournit une description des partitions Core-périphérie obtenue pour les quatre projets de cette étude, ainsi que la précision de l'algorithme de classification.

Tableau 4-5 : Répartition Core-Périphérie dans les projets étudiés

Projet	Taille du Core	Taille du périphérie	Précision
Openstack	1081	4921	82.6%
Eclipse	121	628	82.3%
Android	189	2295	84.2%
LibreOffice	45	437	85.3%

La Figure 4-4 montre la structure du réseau social du projet Eclipse. Les principaux développeurs (en jaune) représentent un petit groupe de contributeurs entre 4,55% et 12,14 % de l'ensemble des contributeurs. Ils sont généralement impliqués dans le projet OSS pendant une période relativement longue et apportent des contributions significatives pour guider le développement et l'évolution du projet. Les développeurs périphériques (indiqués en rouge) sont un ensemble plus important en nombre de contributeurs qui contribuent occasionnellement au projet, interagissant principalement avec les principaux développeurs et interagissant rarement entre eux. Pour rendre le graphique plus lisible, nous avons supprimé les arcs de poids faible et les nœuds isolés.

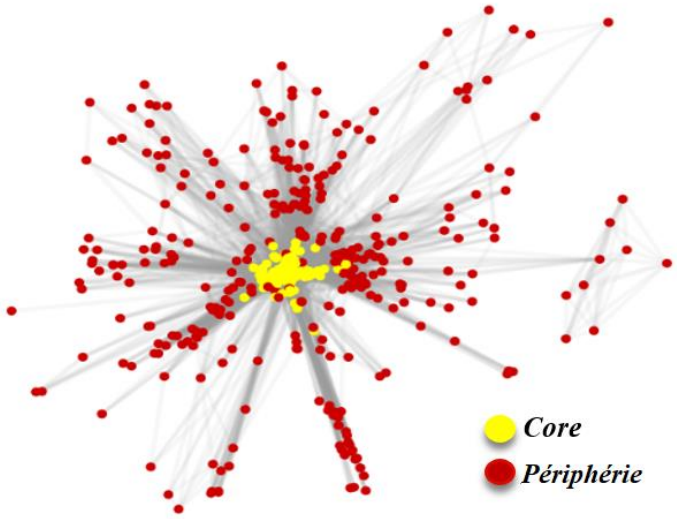


Figure 4-4 : Réseau social des révisions de code pour le projet Eclipse.

Après avoir classé les contributeurs en Core et périphériques, nous calculons un score sentiment pour chaque contributeur sur la base de la moyenne des scores sentiment de tous les commentaires qu'il a écrit. Ensuite, en utilisant le test de Mann-Whitney U[27] (Dmitrienko et al. 2005), nous avons comparé les distributions des moyennes de sentiment entre les groupes de contributeurs principaux et périphériques. Comme nous avons effectué plus d'une comparaison sur le même ensemble de données, afin d'atténuer les risques d'obtenir des résultats faussement

[27] Le test est appliqué en respectant le niveau de confiance couramment utilisé de 95% (c'est-à-dire, $\alpha < 0,05$).

positifs, nous utilisons la correction de Bonferroni (Dmitrienko et al. 2005) pour contrôler le taux d'erreur. Concrètement, nous avons calculé la valeur p ajustée, qui est multipliée par le nombre de comparaisons. Chaque fois que nous avons obtenu des différences statistiquement significatives entre les groupes, nous avons calculé la taille de l'effet Delta de Cliff's (Dmitrienko et al. 2005) pour mesurer l'ampleur de cette différence.

Résultats. La Figure 4-5 montre la comparaison des moyennes des sentiments entre les contributeurs principaux et ceux périphériques. Pour les quatre projets étudiés, la distribution des moyennes de sentiment varie entre [-1, 1]. Le test de Mann-Whitney a révélé une différence significative dans la distribution de la moyenne des sentiments des contributeurs Core et périphériques. Cependant, la valeur de l'effet taille (Effect Size) est faible, sauf pour le projet Libre0ffice où elle est moyenne (voir le Tableau 4-6).

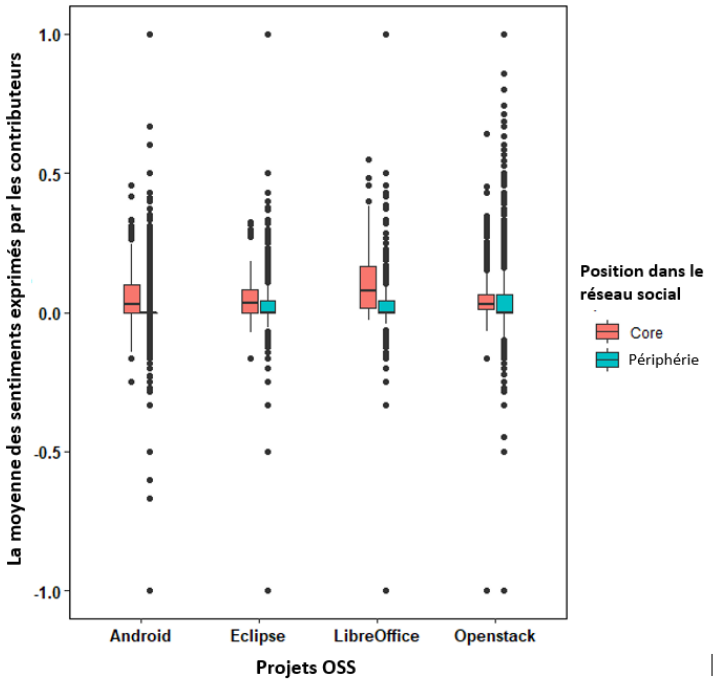


Figure 4-5 : Distribution des sentiments entre les contributeurs Core et périphériques.

Tableau 4-6 : Résultat du test statistique Mann-Whitney U

Projet	U	*p*-value	Effect Size
Openstack	4115100	**2.2e-16**	Faible (0.26)
Eclipse	47683	**1.0e-06**	Faible (0.26)
Android	536920	**2.2e-16**	Paible (0.27)
LibreOffice	22700	**8.48e-12**	Medium (0.47)

Étonnamment, nous avons observé que les contributeurs périphériques dans les quatre projets ont clairement plus de valeurs aberrantes - positives et négatives - par rapport aux contributeurs

Cors dont les sentiments restent concentrés autour des scores neutres (c.-à-d., Valeur égale à zéro). Nous émettons l'hypothèse que le segment aberrant sont des personnes participant à un seul ou à un petit nombre de commentaires, ce qui a une incidence sur les valeurs moyennes, tandis que les développeurs Core restent neutres lorsqu'ils commenteront les révisions du code source. Notre prochaine question de recherche explorera l'évolution des sentiments au fil du temps.

QR3.2 : Comment les sentiments textuels exprimés par les évaluateurs du code source évoluent-ils au fil du temps ?

Motivation. Nous souhaitons comprendre comment les sentiments exprimés par les collaborateurs évoluent au fil du temps à mesure qu'ils gagnent en ancienneté dans un projet. Il y a eu des recherches sur l'évolution des contributeurs de l'OSS au cours du temps (Crowston et al. 2006), en particulier, les chercheurs ont souligné que les analyses empiriques qui mélangent les deux groupes donneront probablement des résultats non valables. Étonnamment, peu de recherches ont examiné l'évolution des sentiments textuels lorsque les contributeurs gagnent en réputation (c'est-à-dire qu'ils appartiennent à l'équipe Core qui dirige le projet). Le sentiment de l'évaluateur du code source peut se détériorer ou croître au fur et à mesure que le projet progresse. Nous dérivons donc la sous question de recherche suivante.

Approche. Pour répondre à cette question de recherche, nous avons procédé comme suit. D'abord, nous avons examiné l'évolution du sentiment de 5% des contributeurs les plus performants pour chaque projet pendant la période complète étudiée. Nous choisissons le top 5% pour nous assurer que nous avons les contributeurs les plus actifs sans discontinuité dans l'activité de révision. Après avoir zoomé sur ce groupe de contributeurs pour les quatre projets, nous avons découvert presque la même tendance dans le temps Figure 4-6. Ensuite, nous avons exploré manuellement en détail les séries temporelles des cinq contributeurs les plus importants pour un projet afin de fonder les modèles d'évolution des sentiments. Nous nous sommes concentrés uniquement sur 5 membres en raison du coût élevé de l'analyse.

Résultats. Nous avons observé une tendance vers des sentiments neutres corrélés avec la progression des contributeurs vers l'équipe Core. Plus un contributeur gagne en réputation, plus il est susceptible d'exprimer des sentiments neutres. La Figure 4-6 montre la moyenne de l'évolution du sentiment par mois des 5% contributeurs les plus performants. Cependant, nous ne pouvons pas conjecturer que cette tendance vers des sentiments neutres est due à un gain de réputation par les contributeurs. Cela pourrait aussi être simplement dû aux changements

culturels dans les projets étudiés. Des analyses complémentaires sont nécessaires pour mieux comprendre l'évolution des sentiments des développeurs dans les projets OSS.

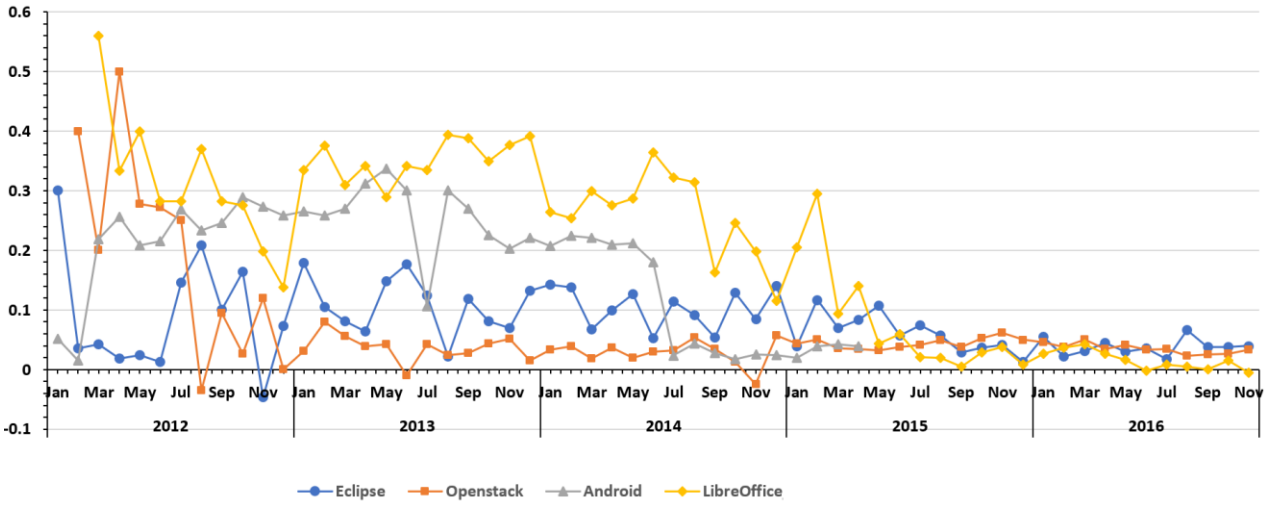


Figure 4-6 : Évolution des sentiments des top 5% contributeurs

Afin de mieux comprendre l'évolution moyenne des sentiments, nous surveillons les cinq principaux contributeurs du projet Eclipse. Nous avons choisi le projet Eclipse parce qu'il est étudié intensivement dans la littérature. La Figure 4-7 montre que les moyennes des sentiments varient considérablement d'une année à l'autre, passant de positives à neutres. Les sentiments des 5 top contributeurs dans Eclipse ont diminué à neutre au fil du temps. Comme mentionné précédemment, une recherche qualitative future intéressante serait d'étudier le comportement des contributeurs les plus productifs.

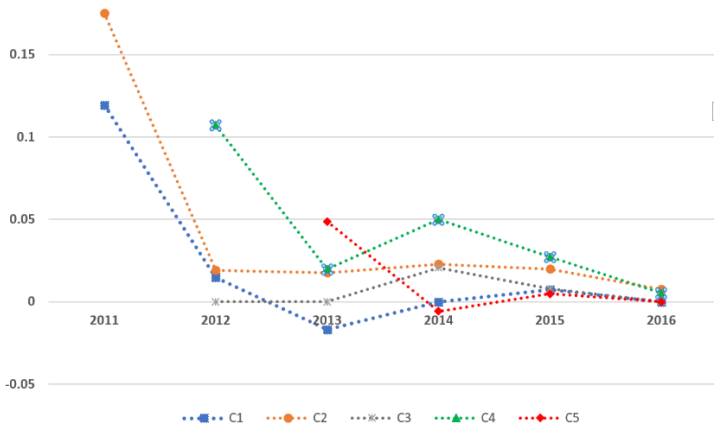


Figure 4-7 : Évolution des sentiments pour les cinq principaux contributeurs du projet Eclipse.

QR4. Est-ce les sentiments exprimés ont un impact sur la durée et le résultat des révisions de code?

L'examen du code est une pratique essentielle pour assurer la qualité à long terme du source code de base. Cette pratique moderne pourrait être influencée par le sentiment exprimé dans les commentaires des contributeurs. Intuitivement, les sentiments positifs peuvent améliorer l'humeur des contributeurs, tandis que les sentiments négatifs peuvent s'avérer nuisibles à leur moral. Un tel changement de moral peut alors avoir une incidence sur le temps et l'issue du processus d'examen. En particulier, il est important de savoir comment les sentiments exprimés peuvent avoir un effet sur les pratiques d'examen du code selon les deux dimensions suivantes : (1) temps de révision du code, et (2) résultat de l'examen du code. La durée d'un examen du code source est un facteur important pour la productivité d'une organisation logicielle (Sudhakar et al. 2012). Nous posons les questions de recherche suivantes :

- ***QR4.1 Les sentiments exprimés dans les révisions ont-ils un impact positif ou négatif sur la durée et le résultat d'une révision de code ?***
- ***QR4.2. La présence de controverses a-t-elle un impact sur les révisions de code qui prennent plus de temps que d'habitude ?***
- ***QR4.3. Les sentiments exprimés par les principaux contributeurs ont-ils un impact différent sur les résultats de l'examen par rapport à ceux exprimés par les contributeurs périphériques ?***

Dans la suite, nous répondons aux trois sous-questions.

QR4.1 Les sentiments exprimés dans les révisions ont-ils un impact positif ou négatif sur la durée et le résultat d'une révision de code ?

Motivation. Lorsqu'un examen de code prend beaucoup plus de temps que prévu, la délivrance du logiciel et la productivité de l'équipe peuvent en souffrir. Un certain nombre de facteurs peuvent contribuer à un tel long temps, tels que l'absence du développeur principal responsable du module particulier lié à la révision demandée. L'objectif de cette sous question de recherche est d'inspecter l'effet des sentiments exprimés par les contributeurs sur la durée et le résultat des révisions de code.

Approche. Un aperçu de la répartition du temps nécessaire pour effectuer les révisions de code dans les projets étudiés a montré que la révision la plus lente révision prenait des centaines de jours alors qu'en médiane la révision nécessitait moins d'un jour. Pour éviter les biais dus aux distributions asymétriques, nous avons utilisé les méthodes Tukey (Tukey 1977) pour la

détection des valeurs aberrantes. Un temps de révision est considéré comme une valeur aberrante si elle est supérieure à un *Upper limit*. Tukey définit cette limite en se basant sur les quartiles inférieur et supérieur [Q1, Q3] (c.à.d. respectivement le 25ème et le 75ème percentiles de la distribution des données) tel que :

$$Upper\ limit\ =\ \mathrm{Q3}\ +\ 1.5\ *\ \mathrm{IQR}$$

Où IQR (Inter-Quartile Range) est l'intervalle entre Q1 et Q3. La méthode de Tukey appliquée à la distribution du temps de révision a détecté des limites supérieures distinctes (en jours) pour chaque projet (13,86 pour Eclipse, 10,02 pour Android, 11,04 pour Openstack et 6,52 pour LibreOffice).

Pour évaluer l'influence des sentiments positifs ou négatifs sur la durée d'un examen du code, nous avons utilisé la méthode de ***Propensity Score Matching*** (PSM) (Thavaneswaran 2008), comme décrit à la section 4.5.2. Pour les applications pratiques, nous ne comparons que les revues qui sont logiquement comparables en termes de caractéristiques techniques : (1) nombre de commentaires sur la demande de révision ; (2) nombre de patchsets, (3) nombre de fichiers édités, (4) nombre de contributeurs impliqués, (5) et churn (c'est-à-dire somme des lignes de code source insérées et supprimées pour mesurer la taille du changement). Aussi, pour évaluer l'influence des sentiments sur les résultats des revues, nous avons cartographié le résumé de sentiment de chaque révision de code (Positive ou Négative) avec son statut final (Accepté ou Abandonné).

Résultats. La comparaison d'un groupe homogène de révisions (obtenus via PSM) révèle que les révisions positives ont nécessité moins de temps à être fermés que les révisions négatives, comme le montre la Figure 4-8. Les révisions négatives nécessitent en moyenne 1,32 jour supplémentaire pour être clôturées. En d'autres termes, la moyenne des durées pour les révisions positives est inférieure à la médiane pour les révisions négatives.

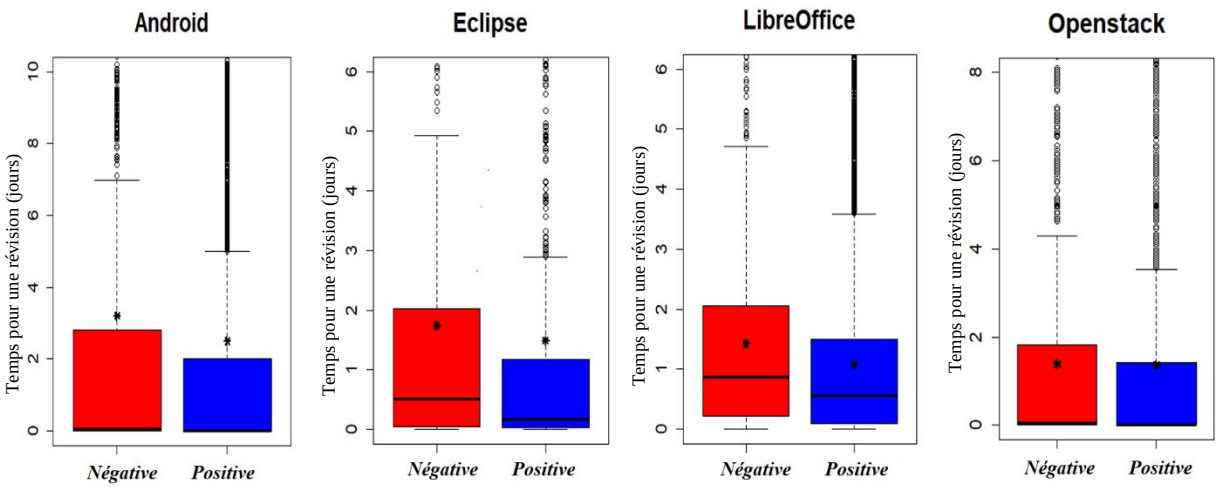


Figure 4-8 : Durées nécessaires pour les révisons Positives et Négatives (*: valeur moyenne).

En outre, comme le montre la Figure 4-8, les révisions positives ont non seulement la minimal moyenne du temps nécessaire de révision. Aussi les révisions positives ont moins de jours maximums pour fermer les révisions par rapport aux évaluations négatives. Par exemple, dans le projet Eclipse, les révisions positives durent au maximum 2,89 jours, tandis que les révisions contenant des commentaires négatifs prennent environ 5 jours.

D'autre part, la Figure 4-9 montre la distribution des types de révisions (positif ou négatif) selon le statut final de la révision (Accepté ou Abandonné). Pour chaque projet, les valeurs ratio présentent le pourcentage de distribution des révisions positives et négatives dans les révisions acceptées (première barre) et abandonnées (deuxième barre). Les résultats montrent que non seulement le sentiment exprimé par les développeurs affecte la durée de la révision du code, mais aussi les résultats finaux. Par exemple, dans le projet Eclipse, plus de 93% des révisions accepté avec succès ont été étiquetés comme positifs, tandis que 55% de tous les demandes de révision abandonnée ont des sentiments négatifs dans leurs commentaires.

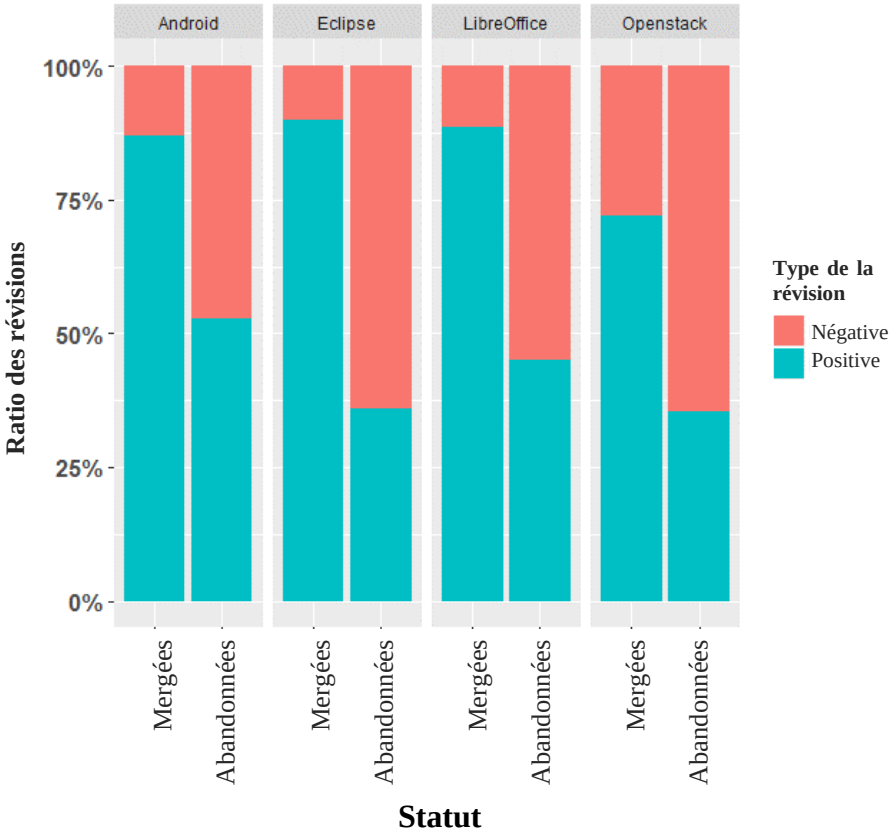


Figure 4-9 : Ratio des révisions positives et négatives selon le résultat de la révision.

QR4.2. La présence de controverses a-t-elle un impact sur les révisions de code qui prennent plus de temps que d'habitude ?

Motivation. Un autre facteur affectant la durée de révision de code pourrait être la présence d'une controverse dans les commentaires. Intuitivement, une révision peut être controverse si la discussion autour contient presque aussi bien les commentaires positifs que négatifs. Une compréhension empirique de la mesure dans laquelle ces controverses peuvent avoir une incidence sur le résultat de l'examen du code peut offrir un gain de sensibilisation à l'égard de l'incidence positive de la pratique d'examen du code. Comme implication pratique, nous pouvons motiver une nouvelle fonctionnalité de Gerrit pour avertir de manière proactive les contributeurs impliqués dans une équipe d'évaluation du risque de retarder l'examen en raison de controverses. Nous enquêtons sur les demandes de révision aberrantes (qui ont pris beaucoup de temps) afin de déterminer si la présence de sentiments controversés dans leurs commentaires pourrait être la cause principale de l'augmentation du temps d'examen.

Approche. Dans cette question de recherche, nous examinons les révisions qui ont pris beaucoup de temps, c'est-à-dire plus que le seuil identifié (Upper limit). Notre objectif est de déterminer si la présence de controverses dans les discussions sur les demandes de révision est la cause principale des longs délais. Le dictionnaire Merriam-Webster (Merriam-Webster 2016)

définit la controverse comme un « fort désaccord parmi un grand groupe de personnes sur quelque chose». Dans notre contexte, nous classons une révision comme controverse si la discussion sur le code source contiennent des commentaires controverses. Nous calculons le degré de controverse en utilisant ControversialMix (Popescu et al. 2010), qui est un score qui estime combien de commentaires positifs et négatifs mixtes sont dans une discussion.

$$\text{ControversialMix} = \frac{(\text{Min}((|\text{Pos}|, |\text{Neg}|)))}{(\text{Max}((|\text{Pos}|, |\text{Neg}|)))} \frac{(|Pos| + |Neg|)}{(|Pos| + |Neg| + |Neu|)}$$

Où Pos, Neg et Neu sont respectivement les sommes de commentaires avec une polarité positive, négative et neutre. ControversialMix prend en considération la quantité de commentaires positifs, négatifs et neutres afin de saisir la diversité des sentiments exprimés au sein de la même révision. Avant de traiter ControversialMix, nous avons effectué quelques prétraitements de l'ensemble de données en éliminant : les révisions avec un seul commentaire ; les révisions où tous les commentaires ont le même tag (négatif, positif, neutre) et les discussions qui ont seulement des commentaires positifs ou négatifs. Enfin, une révision du code source est considérée comme controverse si ControverseMix ≥ 0.5.

Résultats. Le Tableau 4-7 montre la répartition des révisions controverses et le temps d'examen moyen nécessaire pour corriger les révisions controverses et non controverses. Le test de Wilcoxon appliqué sur des révisions controverse et non controverse révèle que les résultats concernant le temps moyen de révision (en jours) étaient significatifs pour un seul projet : Android avec une p_value = 0,0001. Pour ce projet particulier, on peut voir que les revues controverses nécessitent en moyenne plus de jours pour être fermées (+6.33jours). Cependant, un nombre limité de révisions controverses identifiées pour Eclipse, LibreOffice et OpenStack respectivement (4, 31, 95), comme le montre le Tableau 4-7, pourrait expliquer le résultat non significatif obtenu lors du test de Wilcoxon. Par conséquent, nous n'avons pas pu confirmer cette constatation en raison du manque de données.

Tableau 4-7 : Répartition des révisions controversées parmi les révisions aberrantes

Projet	Controverse	#Révisions	Temps moyen de révision (jours)
Android	Non	780	126.38
	Oui	31	148.54
Eclipse	Non	185	104.01
	Oui	4	118.12
LibreOffice	Non	442	49.55
	Oui	34	50.17
Openstack	Non	10201	162.96
	Oui	95	39.54

QR4.3. Les sentiments exprimés par les contributeurs Core ont-ils un impact différent sur les résultats de l'examen par rapport à ceux exprimés par les contributeurs périphériques ?

Motivation. Dans QR3, nous avons observé que les contributeurs Core et périphériques expriment différents types de sentiments. Étant donné que la contribution des Core et périphériques aux activités de révision est probablement différente, les contributeurs Core sont censés être impliqués plus étroitement que les contributeurs périphériques dans les révisions de code. Nous sommes intéressés à examiner si les sentiments exprimés par ces deux groupes de contributeurs affectent également le processus de révision différemment.

Approche. Nous créons deux groupes pour chaque projet, un pour chaque type de contributeurs (c.-à-d., Core et périphérique). Pour chaque type de contributeurs, nous créons trois intervalles de polarité étiquetés positifs, négatifs et neutres. Par exemple, le groupe positif contient tous les temps de révision (en jours) des révisions positives. Le groupe négatif contient tous les temps de révision (en jours) des révisions négatives. Le groupe neutre contient tous les temps de révision (en jours) des révisions neutres. Dans chacun de ces ensembles de données, nous avons exclu : (1) les révisions qui ont duré moins d'un jour ; (2) les révisions qui ont dépassé le seuil déterminé pour chaque projet à l'aide de l'algorithme de détection des valeurs aberrantes Tukey (voir QR4.1). Intuitivement, du point de vue de la productivité, il est inutile d'analyser l'impact des sentiments pour une révision qui a pris moins d'un jour (parce que c'est déjà une performance impressionnante).

Nous divisons ensuite chaque groupe en deux sous-groupes : (1) Mixte. Nous soumettons une révision dans ce segment si des sentiments sont exprimés par les contributeurs principaux et périphériques. (2) Exclusif. Nous mettons une révision dans ce groupe, si elle a des sentiments exprimés par les contributeurs de base ou les contributeurs périphériques, mais pas par les deux dans la même révision. Nous comparons l'impact des sentiments exprimés par des contributeurs Core par rapport aux contributeurs périphériques pour les révisions dans le groupe `Exclusif ' pour chaque projet.

Résultats. Dans le Tableau 4-8, nous montrons les statistiques récapitulatives du temps d'examen (en jours) pris lorsque les contributeurs Core ou périphériques ont offert des révisions positives ou négatives. La colonne « Neutre » sous chaque type de contributeur indique le temps pris pour une révision lorsque tous les contributeurs écrivent des commentaires neutres.

Pour tous les projets, le temps moyen de révision a augmenté lorsque les contributeurs périphériques ont fourni des commentaires négatifs.

Tableau 4-8 : Évaluation de l'impact des sentiments sur le temps de révision du code

		Core			Périphérie		
Projet	**Temps de révision**	**Positive**	**Négative**	**Neutre**	**Positive**	**Négative**	**Neutre**
Eclipse	Moyenne	**4.8**	**5.1**	**4.8**	**4.6**	**4.8**	**4.8**
	Std	3.3	3.2	3.3	3.2	3.4	3.3
	Médian	4	4.9	3.9	3.7	3.9	3.7
Android	Moyenne	**4.1**	**4.3**	**4**	**4.1**	**4.9**	**4**
	Std	2.4	2.4	2.4	2.5	2.3	2.4
	Médian	3.8	4.1	3.3	3.2	3.5	3.5
LibreOffice	Moyenne	**3.1**	**3.3**	**3.1**	**3.1**	**4.1**	**3.2**
	Std	1.6	1.7	1.6	1.6	1.6	1.6
	Médian	2.9	3	2.8	2.9	3.2	2.8
OpenStack	Moyenne	**4.2**	**4.7**	**4.2**	**4.3**	**4.9**	**4.2**
	Std	2.6	1.6	2.6	2.7	2.3	2.6
	Médian	3.2	3.5	3.5	3.5	3.5	3.7

Pour tous les projets, le temps d'examen moyen est plus important lorsque les contributeurs ont fourni des commentaires négatifs que lorsque les contributeurs Core ont fourni des commentaires neutres dans les révisions. Cette tendance est similaire entre les contributeurs Core et périphériques, c'est-à-dire que les commentaires négatifs de tous les contributeurs ont tendance à augmenter le temps de révision. À l'exception d'Eclipse, l'augmentation du temps moyen dû aux commentaires négatifs est statistiquement significative (voir Tableau 4-9[28]). Cependant, nous ne voyons pas un tel impact pour les commentaires positifs. Pour les principaux contributeurs, le temps d'examen moyen est moindre dans un seul projet (Android) lorsque les contributeurs Core ont émis des commentaires positifs. Pour les contributeurs périphériques, l'impact est plus important. Pour deux projets (Eclipse et Android), le temps de révision est moindre lorsque les contributeurs périphériques ont fourni des commentaires positifs.

[28] P_value et la taille de l'effet sont mesurées à l'aide du test Mann-Whitney U et de la taille de l'effet Delta de Cliff, comme expliqué dans QR3.1.

Tableau 4-9 : Validité statistique de l'effet des sentiments sur le temps de révision de code

		Core		Périphérie	
Project	**Temps de révision**	**p-value**	**Delta**	**p-value**	**Delta**
Eclipse	Positive vs Neutre	0.4	0.009	0.17	N/A
	Negative vs Neutral	**0.0002**	0.27	0.48	0.151
Android	Positive vs Neutral	**2.18E-07**	0.18	**1.61E-07**	0.17
	Negative vs Neutral	**2.28E-04**	0.31	**0.002**	0.27
Libreoffice	Positive vs Neutral	**9.60E-05**	0.23	**6.79E-05**	0.24
	Negative vs Neutral	**3.11E-01**	0.38	**8.52E-08**	0.46
Openstack	Positive vs Neutral	**7.82E-09**	0.23	**3.75E-03**	0.27
	Negative vs Neutral	**9.20E-04**	0.42	**6.39E-07**	0.31

Les contributeurs Core et périphériques semblent avoir un impact égal sur le temps de révision lorsqu'ils fournissent des commentaires négatifs dans deux projets (Android et LibreOffice). Pour un projet (Eclipse), les commentaires négatifs des contributeurs Core semblent avoir un impact plus important sur le temps de révision que les commentaires négatifs des contributeurs périphériques. Pour Openstack, la situation est inversée, c'est-à-dire que les commentaires négatifs des contributeurs périphériques semblent avoir un impact sur le temps de révision moyen plus important que les contributeurs principaux. En moyenne, le temps d'examen est beaucoup moins dans les revues où les contributeurs périphériques ont fourni des commentaires positifs. Cette constatation concorde avec notre constatation précédente selon laquelle les contributeurs périphériques offrent plus de sentiments dans les révisions de code, parce que les contributeurs Core ont tendance à devenir plus neutres au fil du temps. Par conséquent, la joie des contributeurs périphériques semble être importante pour réduire le temps de révision du code.

En conclusion, Dans tous les projets à l'exception d'Eclipse, les commentaires négatifs des contributeurs périphériques ont plus d'impact que les commentaires négatifs des contributeurs Core. Dans tous les projets, les délais de révision sont plus longs lorsque les contributeurs périphériques fournissent des commentaires négatifs.

4.7. Discussion

Dans tous nos projets étudiés, les révisions avec des sentiments négatifs ont pris plus de temps à compléter. Cette observation conduit à la question de savoir comment nous pouvons tirer parti de l'analyse des sentiments pour améliorer la productivité du processus de révision de code. Une solution potentielle consisterait à concevoir des moniteurs automatisés basés sur les sentiments qui pourraient guider les contributeurs sur l'acuité dans le texte écrit dans leurs

messages. Bien que ces solutions manquent d'autorité, elles peuvent néanmoins s'avérer utiles pour suivre les contributeurs à travers les différentes phases d'un processus de révision de code en atténuant la négativité dans les commentaires de révision. La haute précision de notre détecteur de sentiment fusionné (QR1) encourage la recherche future sur l'analyse automatisée des sentiments dans les révisions de code pour améliorer la productivité (par exemple, moins de temps de révision).

Dans le but d'améliorer les résultats de l'examen du code et le temps basé sur l'analyse des sentiments, nous offrons les recommandations suivantes en prenant des repères à partir de nos trois questions de recherche (QR2-4) :

1. L'analyse des sentiments peut être appliquée pour trouver des communautés ou des sous-communautés dans un projet qui peuvent affecter la production par des interventions négatives.
2. Les contributeurs nuisibles, tels que les intimidateurs peuvent être détectés pour s'assurer qu'ils n'ont pas d'impact négatif sur le processus de révision du code source.
3. Les révisions controverses (sujets de conflits) peuvent être identifiées pour avertir les chefs de projet sur les composantes ou des équipes potentiellement controverses dans un projet.
4. Les Moteurs de recherche logiciels peuvent être conçus pour avertir les contributeurs participant à une demande de révision lorsque la négativité augmente dans une discussion.

Nos reconnaissons des limites pour notre étude, La première limite est liée à la précision de l'outil utilisé pour l'analyse des sentiments. Ainsi, nous avons renforcé notre approche d'échantillonnage pour l'annotation manuelle en utilisant un échantillonnage opportuniste. Les auteurs ont examiné manuellement 2220 commentaires. En général, nous avons constaté que les sentiments exprimés dans les révisions de code sont faciles à analyser en raison de la nature non ambiguë des sentiments basés sur du texte exprimés dans les commentaires de révision de code, qui ont été compris par les deux codeurs avec une relative facilité.

La seconde limite de notre étude concerne des facteurs internes à notre étude susceptible d'affecter nos résultats. La principale menace pour la validité interne de cette étude concerne la sélection des projets. Une menace possible est que le jeu de données récupéré est trop petit et en quelque sorte pas assez représentatif. Nous avons été prudents en choisissant des projets OSS présentant les caractéristiques suivantes : (1) projets de longue durée avec des communautés dynamiques autour ; (2) la communauté utilise les outils de révision Gerrit pour mener des activités de révision de code. Nous avons également fait attention à ne pas violer les hypothèses des tests statistiques, par exemple, nous avons appliqué des tests non paramétriques qui ne

nécessitent pas de formuler des hypothèses sur la normalité de notre ensemble de données. En outre, nous avons utilisé l'appariement des scores de propension PSM (Zhehui et al. 2010) pour éliminer le biais introduit par les caractéristiques techniques sur le temps de révision. Nous avons comparé la distribution du score de propension estimé entre les évaluations positives et négatives dans l'échantillon apparié et obtenu une moyenne de 96% de chevauchement, ce qui signifie que nous traitons un ensemble de données homogènes basé sur les caractéristiques techniques.

La troisième limite de notre étude concerne la généralisation de nos résultats. Dans le cadre de la première question de recherche QR1, nous avons effectué 2220 classifications manuelles. Nous sommes conscients que la qualité et la taille de l'ensemble annoté peuvent avoir une incidence sur la précision de la classification des sentiments. De plus, notre étude ne concerne que quatre projets. Ainsi, nous devrions reconnaître que nos conclusions ne peuvent pas être généralisées à d'autres systèmes. Nous sommes également conscients que le contexte de chaque projet, y compris la complexité technique et l'organisation, sont des facteurs importants qui peuvent limiter la généralisation. Cependant, ces projets comptent parmi les projets les plus étudiés dans la littérature et les données du système sont accessibles au public. Cependant, la reproduction de nos travaux sur d'autres systèmes open source est souhaitable pour généraliser nos conclusions.

Conclusion

Nous avons analysé les commentaires des développeurs sur les révisions de code à l'aide des données historiques de quatre projets open source. Nous avons cherché à étudier l'influence des sentiments exprimés sous forme de texte sur la durée d'examen du code et son résultat. En utilisant l'outil de détection de sentiments le plus performant, nous avons constaté que les contributeurs expriment des sentiments lorsqu'ils révisent et commentent le code de chacun. Nous avons également examiné l'influence des sentiments exprimés dans les commentaires des développeurs sur le temps et l'issue du processus de révision de code. Nous avons constaté que le fait d'exprimer un sentiment positif lors de la révision du code source avait une influence sur la durée de la révision ; en moyenne, cela pourrait économiser 1,32 jours sur le délai d'achèvement de la révision. De plus, nos conclusions indiquent que les commentaires négatifs sont susceptibles d'augmenter la proportion de révisions infructueuses.

Du point de vue des réseaux sociaux, nous avons utilisé l'approche K-means basée sur les mesures de centralité du SNA, pour distinguer les contributeurs Core et périphériques. Nous

avons constaté que les différents groupes de contributeurs au sein du réseau social de collaboration en matière d'évaluation par les pairs expriment des sentiments différents, les contributeurs principaux exprimant généralement des sentiments neutres.

Notre travail contribue théoriquement et empiriquement au corpus de recherches sur l'OSS et a des implications pratiques sur la prise de conscience des sentiments au sein de l'OSS. Nous espérons que nos travaux inspireront davantage d'études sur la mise au point d'outils efficaces pour aider les contributeurs de logiciels libres à améliorer leur productivité. En tant que travaux futurs, nous prévoyons de compléter cette étude quantitative par une exploration qualitative visant à mieux comprendre l'influence des sentiments exprimés sur le processus de révision du code. Nous prévoyons également d'examiner l'effet de sentiment exprimé par les développeurs sur l'engagement et / ou désengagement du contributeur.

CHAPITRE 5. TRANSFERT DE CONNAISSANCES AU SEIN DES COMMUNAUTÉS OSS

Le développement logiciel est un processus d'acquisition et de cristallisation des connaissances (Roberts 2000, Kilamo et al. 2014, Iskoujina et al. 2015). Ce processus est supporté par les interactions sociales qui alimentent les flux de connaissances entre les contributeurs (Daniel et al. 2018). Ainsi, il y a un besoin d'étudier les interactions dynamiques entre les contributeurs OSS, à un niveau de granularité fin, dans une perspective de comprendre la création et la propagation des connaissances au sein de ces communautés. Ce chapitre, est une étude empirique[29] *qui a pour objectif d'étudier les flux de connaissances entre les membres des communautés OSS. Dans une première section de ce chapitre nous explicitons les motivations et les objectifs de notre étude suivie par une revue de la littérature dans la section 5.2. Puis, la section 5.3 décrit l'approche méthodologique suivie dans cette étude. Ensuite, la section 5.4 présente les résultats de notre investigation en réponse à nos questions de recherche. La section 5.5 discute nos contributions et les mises en garde pour la validité de notre étude avant de conclure.*

[29] *Le contenu de ce chapitre a fait sujet de deux papiers de conférence. Le premier intitulé «**Who Can Help to Review this Piece of Code** »(Kerzazi et al. 2016), présenté dans la 17th édition de « Working Conference on Virtual Enterprises - Collaboration in a Hyperconnected World. Le second intitulé «**Knowledge Flows Within Open Source Software Projects: A Social Network Perspective** »(Kerzazi et al. 2017), présenté dans la 2ème édition de « UNET International Symposium on Ubiquitous Networking ».*

5.1. Motivations et Objectifs

Le développement logiciel est une activité intense de gestion des connaissances (Wiig 2004). La gestion des connaissances doit être coordonnée à travers les artefacts et entre les membres d'une équipe de développement (Crowston et al. 2016). Comprendre la dynamique de partage des connaissances est un sujet de préoccupation centrale, spécialement pour les responsables de projets (Sowe et al. 2006). En effet, la manière dont les connaissances sont transférées et reçues peut avoir un impact direct sur le succès des projets (Phillips et al. 2014). En particulier, plus un contributeur open source apprend de la communauté, plus il est susceptible d'y participer.

Dans le contexte de développement OSS, les membres d'une communauté visent en premier lieu à construire un **capital social** à travers le partage des connaissances (Daniel et al. 2018). Ils interagissent en apportant leurs connaissances aux projets et apprennent des autres (Roberts 2000). **Les interactions entre les contributeurs constituent le support** des flux de connaissances d'où notre intérêt pour cette étude. Concrètement, nous visons à comprendre le rationnel derrière la dynamique des interactions sociotechniques en analysant les flux. Plus particulièrement, les interactions entre les membres expérimentés (i.e., Core) et ceux qui viennent d'intégrer le projet sont d'un intérêt particulier puisque les contributeurs Core ont généralement des connaissances du domaine ainsi que techniques cependant un nouvel arrivant au projet devient progressivement un membre légitime en apprenant des autres au cours d'un processus d'interaction et de communication.

Notre but principal pour cette étude est de comprendre qui est impliqué? qui communique avec qui? et quelle serait la nature de l'interaction? dans le processus du transfert des connaissances. Dans ce sens, nous avons opté pour une approche sociotechnique dans le but d'investiguer l'échange de connaissances entre les contributeurs open source (Core et périphériques). Une attention particulière a été accordée aux interactions sociales à travers les commentaires des contributeurs lors des demandes de révisions du code source pour expliquer la propagation et la densité des flux de connaissances au sein des réseaux de contributeurs open source. Nous avons donc représenté les interactions sous forme de graphes d'interaction dans le but d'analyser des métriques SNA pour visualiser et quantifier les flux de connaissances. Dans cette étude nous cherchons à répondre aux questions de recherche suivantes :

- **QR1. Les flux de connaissances sont-ils dirigés des membres Core vers ceux de la périphérie ou l'inverse ?**

L'objectif de cette question de recherche est de justifier l'existence d'une relation de corrélation entre la position des contributeurs dans le réseau social de collaboration et le partage de connaissances. Pour répondre à cette question, nous avons commencé par identifier les membres Core d'une communauté ensuite nous avons représenté leurs interactions avec les autres membres lorsqu'ils collaborent sur des activités spécifiques. À titre d'exemple, lorsqu'ils coéditent un fichier du code source ou lorsqu'ils commentent sur une demande de révision de code source.

- **QR2. Quel type de connaissances est transféré entre les membres d'une équipe?**

L'objectif de cette question de recherche est de détecter les types de connaissances partagés au sein des communautés open source. Nous cherchons ces types de connaissances dans les interactions entre les contributeurs open source autour des demandes de révisions du code source.

- **QR3. Est-ce qu'il est possible de recommander un membre pour réviser une demande de changement?**

Nous examinons l'utilité de l'analyse sociotechnique pour aider les contributeurs de logiciels open source à identifier des experts aidant à réviser leur code source. Surian et al. (Surian et al. 2011) ont recommandé une liste des principaux développeurs les plus compatibles pour une tâche spécifique en fonction de leurs projets antérieurs sur lesquels ils ont travaillé auparavant. D'autre études (Asundi et al. 2007, Thongtanunam et al. 2015) ont démontré que les contributeurs experts sont les plus susceptibles de réviser les contributions techniques proposées par les membres d'une équipe. Toutefois, ces recherches n'expliquent pas où se trouvent ces experts dans le réseau des contributeurs.

5.2. État de l'art

La communauté de logiciels open source tire sa force essentiellement de la collaboration et du partage des connaissances en vue de produire des logiciels (Lee Endres et al. 2007). Une communauté OSS est une structure dans laquelle des connaissances tacites et explicites sont partagées et échangées entre divers membres afin de créer une valeur collective utile pour faire avancer le projet de développement (Huang 2009, Wenger 2010). De plus, la communauté de logiciels open source est une source continue d'apprentissage pour les participants (Singh et al. 2013). Par conséquent, une communauté open source ne se limite pas au développement de

logiciels, elle fournit également un champ riche pour explorer le processus de création, d'accumulation et de diffusion de connaissances.

Le partage des connaissances nécessite l'engagement actif des individus dans un processus d'interaction et d'apprentissage (Roberts 2000, Butler 2001). Par conséquent, comprendre ce qui motive les individus à participer au partage des connaissances facilitera la conception de stratégies de gestion des connaissances efficaces. Iskoujina et al. (Iskoujina et al. 2015) ont étudié les facteurs qui poussent les participants à partager leurs connaissances au sein des communautés de logiciels open source. Ils ont conclu que la satisfaction individuelle des participants à l'égard de la gestion d'un projet de logiciel libre est un facteur important qui influe sur l'ampleur de leur contribution personnelle à la communauté entraînant réellement un partage des connaissances. En plus, Chen (Chen et al. 2017) ont analysé les facteurs clés affectant le partage des connaissances dans les projets de logiciels open source. Ils ont confirmé que la motivation participative a un effet positif sur le partage des connaissances. Notre étude vient pour enrichir cette littérature par une modélisation du transfert des connaissances basée sur l'analyse des interactions sociotechniques entre les contributeurs open source.

Lakhani et al. (Lakhani et al. 2004) ont étudié les interactions dans le projet Apache au sein de leur système de support où ils fournissent l'assistance aux utilisateurs ayant des difficultés avec le programme. Leur étude c'est focalisé sur la manière dont les demandeurs d'information [connaissances] publient leurs questions et les fournisseurs potentiels d'informations [connaissances] lisent et affichent leurs. Les auteurs ont constaté qu'environ 2% des fournisseurs de connaissances sont responsables d'environ 50% des réponses aux questions posées sur le système de support et que 50% des questions ont été fournies par 24% des fournisseurs de connaissances. Les 100 demandeurs d'informations les plus actifs ont répondu à une moyenne de 10,43 questions et les 100 fournisseurs d'informations les plus actifs à une moyenne de 83,63 réponses au cours des quatre années de leur étude. Cela signifie que seules quelques personnes fournissent activement des réponses aux questions posées dans le système Apache. Cette étude nous a inspiré à enquêter la position sociale des demandeurs et fournisseurs de connaissances au sein des réseaux sociaux d'interactions.

D'autres travaux antérieurs ont abordé une vision sociale sur la création et le partage des connaissances au sein des communautés open source. Cette vision favorise l'idée que les développeurs construisent des connaissances **lorsqu'ils interagissent** dans un contexte social (Daniel et al. 2018). Sowe et al. (Sowe et al. 2006) ont introduit un modèle de partage des connaissances pour développer une compréhension de la dynamique de collaboration et de la

répartition du partage des connaissances entre les équipes de développement de logiciels. Le modèle proposé par les auteurs caractérise les interactions de demandeurs de connaissances et de ceux qui les fournissent en se basant sur l'analyse du contenu des listes de diffusion. Dans notre étude, nous nous sommes inspirés de l'approche suivie par (Sowe et al. 2006) mais nous avons pris en considération plusieurs interactions sociotechniques (commits, code révision et commentaires) pour modéliser le partage des connaissances.

Le Tableau 5-1 ci-dessous présente une synthèse sur la recherche autour du transfert des connaissances au sein des communautés OSS.

Tableau 5-1: Synthèse sur l'étude du transfert des connaissances au sein des communautés OSS.

Étude	Résultat	Observation
(Lakhani et al. 2004)	2% des fournisseurs de connaissances sont responsables d'environ 50% des réponses aux questions posées sur le système de support et que 50% des questions ont été fournies par 24% des fournisseurs de connaissances.	L'étude constitue une source d'inspiration pour enquêter la position sociale des demandeurs et fournisseurs de connaissances au sein des réseaux sociaux d'interactions.
(Sowe et al. 2006)	Les demandeurs de connaissances et les fournisseurs de connaissances interagissent dans les listes de diffusions (Mailing listes). Les courtiers en connaissances ont un double rôle de courtiers en connaissances et de fournisseurs de connaissances.	Les interactions dans les listes de diffusion est la seule activité prise en considération lors de l'investigation du partage des connaissances. Notre travail modélise le transfert des connaissances à la base de plusieurs interactions sociotechniques (commits, code révision et commentaires).
(Hemetsberger et al. 2004)	Les débutants sont encouragés à observer les pratiques courantes et la communication dans le but de favoriser une nouvelle réflexion et une nouvelle expérience des processus avant de devenir des praticiens.	L'étude se focalise uniquement sur les moyens de communication (canaux de transfert des connaissances) sans analyser le contenue (types de connaissances).
(Lanzara et al. 2004)	Le processus de création de connaissances dans les projets de logiciels open source est caractérisé comme étant une écologie d'agents, d'artefacts, de règles, de ressources, d'activités, de pratiques et d'interactions.	L'étude constitue une source d'inspiration pour investiguer les interactions entre les membres de la communauté pour modéliser le partage des connaissances.
(Lee et al. 2003)	Compare le modèle de la création des connaissances pour la communauté Linux par opposition au modèle basé sur les entreprises".	La comparaison entre les projet open source et ceux fermé est faite uniquement en termes d'initiation des processus d'apprentissage aux niveaux individuel et collectif.

5.3. Approche Méthodologique

5.3.1.Données

Nous avons extrait l'historique de la collaboration des membres de trois communautés OSS de GitHub. Notre sélection des projets suit la même approche présentée dans le CHAPITRE 2 section 2.3 . Le Tableau 5-2 résume les projets sélectionnés ainsi que leurs caractéristiques : le langage de programmation, le nombre total de développeurs, le nombre de versions ; le nombre de lignes de code (LOC) ; le nombre de révisions demandées ; et le total des commits.

Tableau 5-2 : Vue d'ensemble sur les projets étudiés

	Langage	#Contributeurs	Versions	LOC	Demandes de révisions	Commits
AngularJs	JavaScript	1403	161	369.574	349	7534
Docker	Go	1314	129	670.722	2399	22318
JQuery	JavaScript	250	134	62.566	10	6050

5.3.2. Modélisation SNA du transfert de connaissances

La Figure 5-1 illustre l'approche méthodologique suivie dans cette étude empirique. Après la sélection des projets ainsi que l'extraction des données, nous avons traité ces données pour la création des réseaux sociaux de collaboration à l'aide de la même approche présentée dans CHAPITRE 2 section 2.4. Pour chaque projet, nous avons construit trois graphes de collaboration : (1) Réseau de collaboration basé sur la coédition des fichiers où les nœuds sont les contributeurs et les arcs sont des liens pondérés et unidirectionnels entre contributeurs ayant modifié les mêmes fichiers ; (2) Réseau de commentaires représentant les contributeurs qui ont commenté sur des artéfacts techniques ; (3) Réseau d'évaluateurs qui ont été responsables de la révision des changements du code source proposés par d'autres contributeurs. Ces trois réseaux d'interactions sociotechniques fournissent un système permettant d'examiner les groupes de personnes qui travaillent ensemble en se basant sur les métriques SNA. En nous basant sur l'analyse sociotechnique de ces interactions, nous cherchons à répondre aux trois questions de recherche, énoncées précédemment en section 5.1, selon le modèle suivant (voir Figure 5-1).

En effet, Pour répondre à la première question de recherche nous nous sommes basés sur la visualisation et le mapping des contributeurs ayant participé à une discussion autour une demande de révision, et les contributeurs ayant résolu et clôturé la révision sur le réseau de collaboration autour des modifications des fichiers. Ensuite, pour répondre à la deuxième question de recherche nous avons mené une annotation manuelle du texte dans les

commentaires pour identifier les types de connaissances transférés. Enfin, pour répondre à la dernière question de recherche nous avons analysé les métriques SNA.

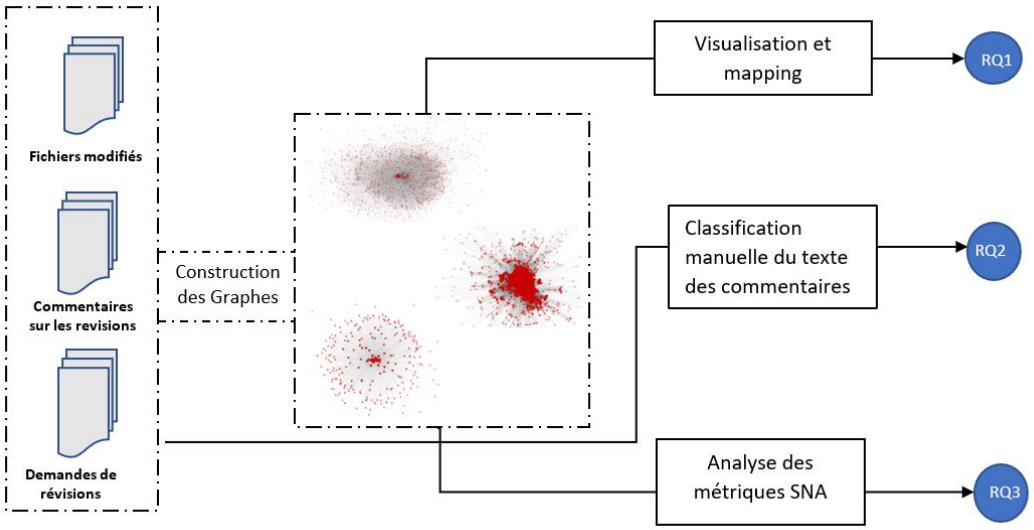


Figure 5-1 : Transfert de connaissances basés sur SNA

5.4. Résultats

Nous présentons maintenant les résultats de l'analyse du transfert des connaissances menée sur les trois projets open source. Pour chacune de nos questions de recherche, nous présentons notre motivation, notre approche et nos résultats.

QR1. Les flux de connaissances sont-ils dirigés des membres Core vers ceux de la périphérie ou l'inverse ?

Motivation. La littérature démontre que le partage des connaissances nécessite l'engagement actif des individus dans un processus d'interaction et d'apprentissage. L'objectif de cette question de recherche est d'investiguer les flux de connaissances entre les membres Core et ceux périphériques.

Approche. La première étape consiste à identifier les membres Core d'une communauté ensuite nous avons représenté leurs interactions avec les autres membres lorsqu'ils collaborent sur des activités spécifiques. En particulier lorsqu'ils coéditent un fichier du code source, commentent sur une demande de révision de code source et résolvent les révisions. Ensuite, nous avons récupéré des métriques SNA que nous avons analysées avec des modèles statistiques dans le but de trouver des corrélations. Aussi nous nous sommes servis d'une approche de mapping des différents réseaux construits.

Résultat. La Figure 5-2 montre le mapping des réseaux appliqué à deux projets AngularJS et Docker. Les réseaux en rouge sont les réseaux de collaboration à base de la modification des fichiers. Nous avons cartographié sur ces réseaux les sous-réseaux suivants : 1) les demandeurs de révision du code, 2) les commentateurs et 3) les résolveurs de révisions. Plus précisément, la Figure 5-2.a1 (respectivement 5-2.a2) illustre le réseau social d'AngularJS (respectivement Docker) en couleur rouge et un sous-réseau de contributeurs qui ont demandé la révision du code en couleur bleue. On peut observer les différences entre les deux projets en termes de densité et de position des demandeurs de révision de code. La Figure 5-2.b montre le mapping des contributeurs qui ont commenté les révisions de code (vert). Et enfin, les Figure 5-2.c. compare la position relative des contributeurs qui ont résolu les révisions de code dans le réseau global.

Notre approche de visualisation et de mapping des réseaux a révélé que les demandeurs de révision du code source sont généralement localisés au centre du réseau de coédition avec un minimum de dispersion vers la périphérie. En revanche, les contributeurs ayant résolu et clôturé une révision de code sont généralement localisés au centre du réseau initial. Et enfin les contributeurs qui commentent les changements sont plus dispersés sur le réseau de coédition comparé au réseau des demandeurs de révisions.

Ainsi nous avons constaté que **les contributeurs Core agissent comme des intermédiaires des flux de connaissances et aussi des intermédiaires des discussions (à travers les commentaires) avec les contributeurs périphériques**. En effet le centre du réseau de collaboration autour des modifications des fichiers comporte initialement les contributeurs ayant des connaissances techniques ainsi que des connaissances de domaine. D’autre part le centre du réseau social des commentateurs comporte initialement les intermédiaires des discussions. Compte tenu de la coïncidence des deux centres alors les contributeurs Core partagent leurs connaissances à travers leurs interactions avec les différents membres de la communauté à l’aide des discussions dans les commentaires ou la validation des propositions des modifications du code source.

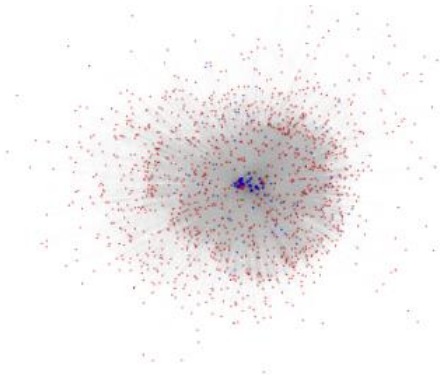


Figure 5-2.a1 Réseau des contributeurs qui ont demandé une révision de code pour AngularJS

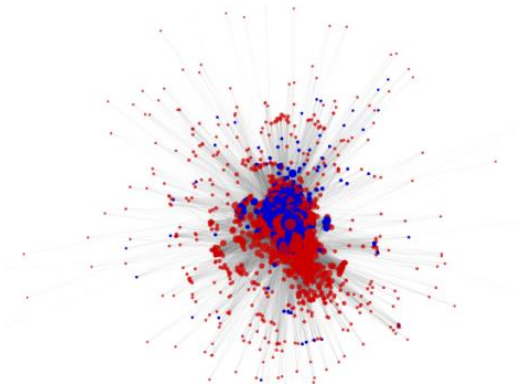


Figure *5-2*.a2 Réseau des contributeurs qui ont demandé une révision de code pour Docker

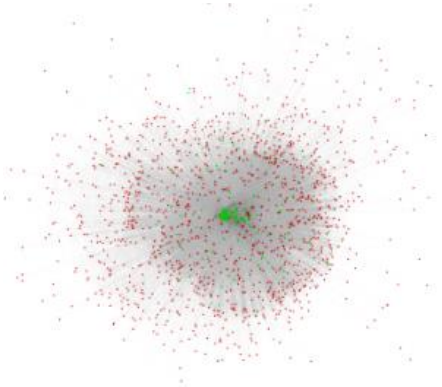


Figure 5-2.b1 Réseau des commentateurs sur la révision du code pour AngularJS

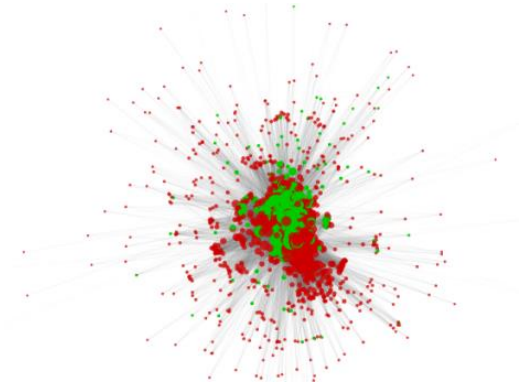


Figure *5-2*.b2 Réseau des commentateurs sur la révision du code pour Docker

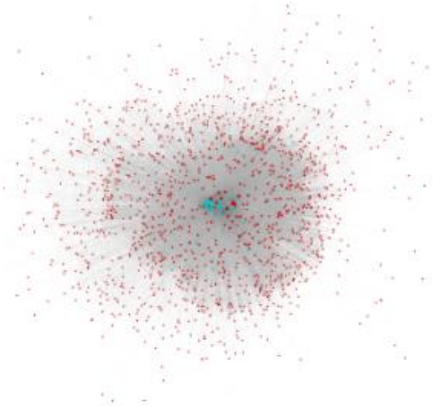


Figure 5-2.c1 Le réseau des contributeurs ayant résolu et clôturé la révision pour AngularJS

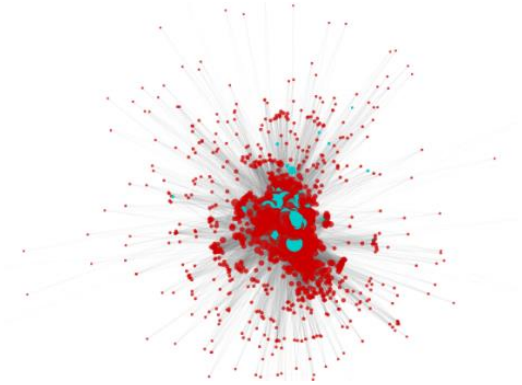


Figure 5-2.c2 Le réseau des contributeurs ayant résolu et clôturé la révision pour Docker

Figure 5-2 : Mapping des réseaux de collaboration

QR2. Quel type de connaissances est transféré entre les membres d'une équipe *?*

Motivation. Plusieurs études antérieurs (Ye et al. 2003, Roberts et al. 2006, Iskoujina et al. 2015) ont abordé les types de motivation qui sous-tendent le partage des connaissances sans entrer dans les détails sur les types de connaissances partagés au sein de communautés open source.

Approche. Nous avons effectué une annotation manuelle (par deux évaluateurs) sur un ensemble de commentaires sur les révisions de code. Notre annotation consiste à déterminer, à partir du texte, le type de connaissances transmis dans un commentaire.

Résultat. Notre classification manuelle des commentaires sur les révisions de code a révélé que les connaissances transmises au sein d'une communauté open source peuvent être classifiés en trois catégories importantes. Premièrement des connaissances techniques, c'est-à-dire lorsque le contributeur intervient pour expliquer ou résoudre un problème technique. Ces commentaires contiennent évidemment des bouts de code. La deuxième catégorie inclut des connaissances du domaine acquises par l'expérience dans un domaine d'activité spécifique. En particulier, les interventions des développeurs connaisseurs du domaine jettent un nouvel éclairage sur les aspects fonctionnels et comportementaux. Enfin, la troisième catégorie concerne la sensibilisation. Cette catégorie est identifiée par un fragment autonome de texte qui fournit uniquement du texte informatif (voir Tableau 5-3).

Tableau 5-3 : Description des types de connaissances

Type de connaissances	Description
Techniques	Fournit des exemples de code sur la façon d'utiliser et de combiner des éléments pour implémenter certaines fonctionnalités ou certains résultats de conception.
De domaine	Explique le but de fournir une modification ou justifie une décision. Par exemple, pourquoi cette composante est conçue de cette façon ? Pourquoi voudrions-nous l'utiliser ?
Sensibilisation	Un fragment autonome de texte qui fournit uniquement du texte informatif.

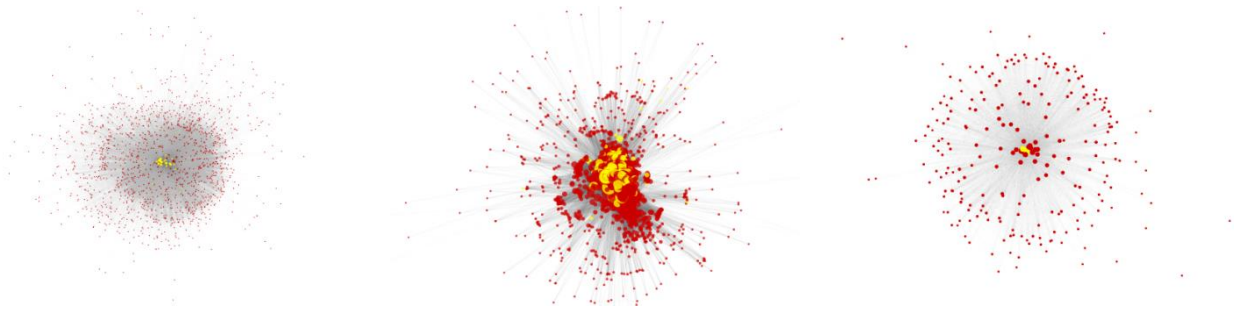
La classification est faite à l'aides des mots clés utilisés dans le texte des contributeurs**. La majorité du transfert de connaissances concerne la sensibilisation (46,3%), puis les connaissances techniques (34,5%), et des connaissances du domaine (19,1%).** Certes, cette étude n'est pas complète, une étude qualitative est nécessaire pour avoir un aperçu complet sur les types de connaissances partagés au sein des communautés open source.

QR3. Est-ce qu'il est possible de recommander un membre pour réviser une demande de changement?

Motivation. Après avoir modélisé le partage des connaissances entre les membres de la communauté open source, nous cherchons à tirer parti de cette modélisation pour proposer les porteurs de connaissances susceptibles d'améliorer le processus de révision du code.

Approche. Pour répondre à cette question de recherche nous utilisons le mapping des réseaux pour visualiser la position des contributeurs assignés aux révisions de code, ensuite à l'aide d'une analyse des métrique SNA nous allons qualifier ces contributeurs.

Résultat. L'investigation du transfert des connaissances entre les membres de la communauté open source nous permet d'identifier les contributeurs les plus susceptibles de réviser le code des autres. En effet, les experts sont les développeurs qui ont les deux connaissances techniques et du domaine. Dans le contexte de la communauté open source ces experts sont les personnes à qui sont assignées les modifications du code source pour être correctement révisées et surtout dans les brefs délais. La Figure 5-3 montre la répartition des contributeurs pré-assignés pour les révisions de code (en jaune) dans le réseau global des contributeurs. La Figure 5-3 confirme que les assignés sont localisés au centre des réseaux de collaboration pour les trois projets étudiés.



AngularJS Docker JQuery

Figure 5-3 : Mapping des contributeurs assignés au réseau de collaboration

La Figure 5-4 montre la distribution des deux plus importantes métriques SNA Degré et Betweeness pour les contributeurs assignés aux révisions (les experts). **Les résultats des projets étudiés montrent que les contributeurs experts sont** les contributeurs Core (Degré > 500 et Betweeness> 600 qui ont suffisamment de connaissances sur le produit pour gérer les contributions des autres développeurs.

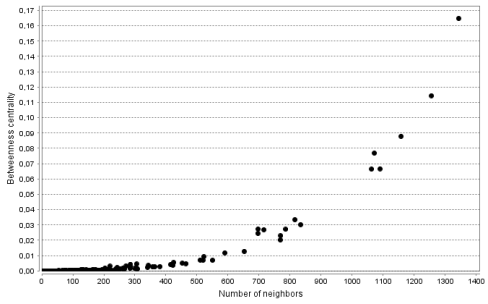


Figure 5-4.a1 Distribution du Betweenness AngularJS

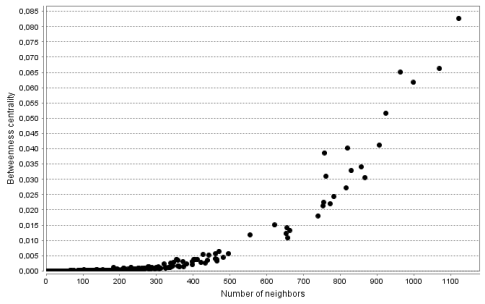


Figure 5-4.a2 Distribution du Betweenness Docker

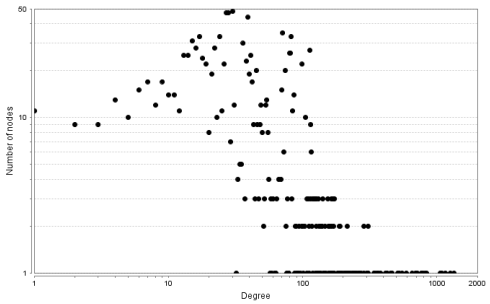


Figure 5-4.b1 Distribution du Degré AngularJS

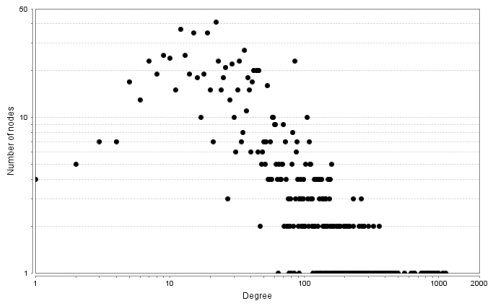


Figure 5-4.b2 Distribution du Degré Docker

Figure 5-4 : Métriques SNA des contributeurs assignés

Néanmoins, nous avons constaté beaucoup de révisions de code effectués par d'autres contributeurs avec différents degrés de centralité, ce qui n'est pas nécessairement problématique mais peut indiquer des domaines où la mission d'évaluation n'était pas soutenue par la connaissance ou la connaissance inter fonctionnelle ou la distribution de connaissance du domaine dans l'équipe de base. Une étude plus avancée est nécessaire pour mieux qualifier les experts dans le contexte open source afin de bénéficier de leurs connaissances.

5.5. Discussion

Cette étude empirique démontre que les communautés OSS surmontent avec succès le problème de la transformation des connaissances tacites grâce à des outils technologiques de communication. Nos résultats ont des implications sur l'amélioration de la collaboration entre les contributeurs au sein des communautés virtuelles OSS, ce qui améliore la performance des équipes et la qualité des produits logiciels. Les principales contributions de ce travail sont :

- L’exploration empirique de l'activité de révision du code source dans trois OSS afin d’enquêter les interactions sociales invisibles dans une structure Core-périphérie ;

- L’utilisation de l’approche de chevauchement de différents réseaux (coédition de fichiers, révisions du code source, commentaires) pour localiser les contributeurs d’un réseau sur un autre ;
- Les collaborateurs ayant un degré de centralité élevé ont plus de chances d’être assignés aux révisions de code.

La génération de nos résultats est limitée vu le choix de seulement trois projets. Cependant, nous avons pris le soin de choisir des projets matures, considérés comme attractifs sur la plateforme GitHub. Ces projets sont écrits avec différents langages de programmation et impliquant un nombre de contributeurs allant de 250 à 1403. Nous avons filtré les projets impliquants plus de 250 contributeurs où plus de 1 000 fichiers édités pour nous assurer qu’il y a suffisamment d’activités et d’interactions sociotechniques à analyser.

Conclusion

Dans cette étude, nous avons adopté l'analyse de réseaux sociaux sur trois projets open source. Nous avons montré comment les connaissances sont transférées entre les contributeurs Core et ceux de la périphérie lors de leurs interactions sociotechniques. Nous avons construit des réseaux de contributeurs basés sur la coédition des fichiers, puis nous avons construit des sous-réseaux pour les contributeurs demandant des révisions de code, des commendataires et ceux effectuant les révisions de code. La visualisation, le mapping et les métriques SNA rendent possible l'identification et l'analyse des interactions structurelles de ces réseaux. Nous avons constaté qu'il existe une forte corrélation entre la centralité des contributeurs et leur implication dans le transfert des connaissances. Ensuite nous avons enquêté les types de connaissances. Enfin, nous avons introduit une approche basée sur le SNA pour identifier les évaluateurs appropriés.

En comprenant les flux de connaissances entre les collaborateurs open source, la structure des interactions sociotechniques, les communautés open source acquièrent une capacité accrue à faciliter les révisions de code dans leurs projets. Nous espérons que cela mènera à des projets logiciels avec un transfert de connaissances plus efficace, moins de frais généraux de révision, et un effet multiplicateur de la qualité du logiciel et de la performance des équipes.

CONCLUSION ET TRAVAUX FUTURS

1. Vue d'ensemble

Dans cette thèse, nous avons mené une série d'études empiriques pour investiguer la dynamique des interactions au sein des communautés open source en vue d'améliorer à la fois la productivité des contributeurs et la qualité des produits logiciels. Notre hypothèse globale de recherche stipule que les interactions sociotechniques des développeurs open source constituent une source riche pour caractériser les communautés OSS. À travers l'analyse des interactions sociotechniques des contributeurs open source, nous avons proposé une approche de détection de la structure organisationnelle des communautés open source. Ensuite, nous avons suivi l'évolution temporelle de cette structure depuis la création du projet. Puis, à travers l'analyse des interactions textuelles entre les composantes de cette structure, nous avons étudié l'impact des sentiments exprimés par les contributeurs open source sur leur productivité et sur la qualité de la révision du code. Enfin, nous avons examiné le partage des connaissances au sein des communautés en vue d'améliorer la collaboration entre les contributeurs et ainsi améliorer la performance des équipes et la qualité des produits logiciels.

Nos quatre études empiriques sont pilotées sur des projets sélectionnés des deux plateformes de collaboration open source les plus utilisées : GitHub et Gerrit. Nous avons gagné une expertise importante en matière d'extraction des historiques de développement et de préparation des données. Nos analyses des interactions sociotechniques des contributeurs open source se basent principalement sur l'analyse des réseaux sociaux qui nous a permis de modéliser les communautés OSS en graphes de collaborations et d'étudier les interactions à travers les métriques SNA. En plus l'analyse des réseaux sociaux, nous avons utilisé plusieurs techniques d'analyse de données, en particulier les séries temporelles pour explorer l'évolution temporelle des communautés OSS et l'analyse des sentiments pour détecter les sentiments dans le texte des contributeurs open source.

2. Contributions

La première contribution de cette thèse concerne l'organisation structurelle des communautés open source. Nous avons adopté une modélisation en réseaux sociaux à base des interactions techniques entre les contributeurs open source. Ensuite, nous nous sommes basés sur l'analyse

des réseaux sociaux (SNA) pour proposer une approche de classification des développeurs open source en fonction de leurs activités et leurs interactions au sein de la communauté. Cette approche classifie les contributeurs open source en trois classes Core, Transitoire et Périphérique. Notre étude empirique a permis de détecter l'organisation structurelle typique pour les communautés OSS. Comprendre les différents rôles des développeurs dans un projet open source nous a suscité le questionnement sur comment ces rôles évoluent dans le temps et comment participent-ils dans la construction du capital social des communautés open source.

La seconde contribution concerne l'analyse temporelle de l'évolution de la structure organisationnelle détectée au sein des communautés open source. Nous nous sommes servis d'une modélisation en réseaux sociaux dynamiques pour capturer la structure organisationnelle sur des intervalles de temps continus et ainsi caractériser le changement temporel des rôles des contributeurs open source. Notre étude propose des lignes directrices pour garantir une évolution saine de la communauté OSS. En particulier, nos résultats ont permis de recommander aux nouveaux arrivants de travailler plus sur les activités liées aux modifications du code source pour accélérer leur passage d'une position de membre périphérique au membre Core, et aussi de prédire qui va quitter le projet en fonction de son historique de contribution au projet.

La troisième contribution concerne l'analyse des sentiments au sein des communautés open source. Nous avons notamment analysé les commentaires des développeurs sur les demandes de révisions. Nous avons étudié l'influence des sentiments exprimés sous forme de texte sur la durée d'examen du code et son résultat. Nos résultats mettent en évidence l'effet des sentiments exprimés par les contributeurs open source sur leur productivité et sur le succès de la pratique crucial de révision de code. Notre étude met l'accent sur une approche d'amélioration de productivité des contributeurs open source et la qualité des projets OSS à travers la détection des contributeurs nuisibles et neutraliser leur impact négatif sur le processus de révision du code source.

La dernière contribution concerne l'investigation des interactions entre les contributeurs open source pour modéliser le partage des connaissances au sein de ces communautés. Nous nous sommes basés sur une modélisation sociotechnique des interactions et nous avons étudié le partage de connaissances entre les deux composantes principales « Core » et « Périphérie ». Nos résultats préliminaires nous ont permis l'exploration et la visualisation des interactions entre les contributeurs Core et ceux périphériques afin d'identifier les modèles de collaborations ainsi que l'important rôle des contributeurs « Core » vu l'existence d'une forte corrélation entre

la position sociale du contributeur et son implication dans le processus de partage des connaissances. Nos résultats ont des implications sur l'amélioration de la collaboration entre les contributeurs au sein des communautés virtuelles OSS, ce qui améliore la performance des équipes et la qualité des produits logiciels

3. Implications pratiques de nos contributions

Notre travail contribue théoriquement et empiriquement au corpus de la recherche sur le développement des logiciels open source et a des implications pratiques sur la dynamique sociotechnique des communautés open source. Nos études empiriques démontrent que les données stockées dans les référentiels de logiciels, associées à des méthodes d'analyse appropriées, peuvent fournir des informations précieuses, pratiques et valables sur les aspects sociotechniques du développement de logiciels open source.

Outre l'aspect scientifique, nos recherches présentent aussi des avantages pour les praticiens. Les résultats de notre recherche sont particulièrement pertinents pour les groupes de parties prenantes suivantes :

- Un développeur qui souhaite contribuer à un écosystème et devenir l'un des acteurs principaux « Core » le plus tôt possible. À travers nos études empiriques chaque contributeur open source devient conscient de l'importance de ces interactions sociales et techniques au sein de la grande communauté.
- Les propriétaires de projets open source ont une vue globale de la dynamique de leurs projets avec des outils pour mieux intervenir en faveur de la maturité de leurs projets. En outre, cette recherche fournit un aperçu sur les rôles de différents contributeurs et les patterns d'engagement et désengagement des contributeurs open source.
- Les chercheurs qui souhaitent approfondir la compréhension de la dynamique des écosystèmes open source pourraient bénéficier de nos investigations sur la dynamique sociotechnique.
- Les chercheurs intéressés par l'analyse de réseaux dans le domaine des écosystèmes logiciels. Notre projet de recherche est une validation sur la commodité de ce champ d'être compris à travers la modélisation des interactions en réseaux sociaux.

4. Orientations futures de la recherche

Prenant en considération toutes les facettes d'interactions techniques que nous avons étudiées, nous pouvons nous engager au développement d'outils pour identifier automatiquement les dépendances techniques entre les contributeurs et fournir une visualisation de la dynamique des

écosystèmes open source. Tel outils pourraient accroître la prise de conscience des besoins de coordination et aider les développeurs à avoir une meilleure vision de l'écosystème entourant leur projet. La prise de conscience de l'écosystème autour d'un projet logiciel peut aider à la réussite des projets OSS.

Une possible amélioration pour les différentes études empiriques, dans ce travail de recherche, serait d'utiliser des enquêtes qualitatives ciblant les développeurs open source pour mieux comprendre le pourquoi de certains comportements. En particulier, des entrevues avec les développeurs open source peuvent être menés en vue de mieux cerner les problèmes liés aux partages des connaissances rencontrés durant leurs activités journalières. L'analyse de ces entrevues permettra de proposer de bonnes pratiques en vue d'améliorer la conduite des projets de logiciels open source.

L'étude empirique sur l'impact des sentiments au sein de la communauté open source nous a permis de détecter un volet important pour une future exploration qui est liée à la diversité (de cultures, de genre, d'appartenance ethnique, de religion, etc.) des membres. Notre approche de modélisation des interactions sociotechniques entre les membres de la communauté open source peut être étendue pour prendre en considération, par exemple, le genre du contributeur comme attribue du nœud pour étudier les interactions entre les hommes et les femmes dans le contexte open source.

Enfin, nous prévoyons de continuer l'automatisation de nos scripts d'extraction et de traitement des données. Ces outils sont d'un grand intérêt pour plusieurs communautés scientifiques qui souhaitent faire des analyses et des études sur les communautés open source.

LISTE DES PUBLICATIONS

Conférences internationales

- Noureddine Kerzazi and **Ikram El Asri**. *Who Can Help to Review This Piece of Code?* Working Conference on Virtual Enterprises. Springer, Cham, 2016, 289-301.
- Kerzazi, Noureddine, and **Ikram El Asri**. *Knowledge Flows Within Open Source Software Projects: A Social Network Perspective.* in Advances in Ubiquitous Networking 2, 2017, 247-258.
- **Ikram El Asri**, N. Kerzazi,L. Benhiba and M. A. Janati Idrissi. *From Periphery to Core: A Temporal Analysis of GitHub Contributors' Collaboration Network*. in Collaboration in a Data-Rich World, 2017, 217-229.
- Kerzazi, Noureddine, and **Ikram El Asri**. *Release Engineering: From Structural to Functional View*. the 12th International Conference on Intelligent Systems: Theories and Applications". 24-25 October 2018 - EMI Rabat – Morocco, 3535–3577.
- **Ikram El Asri,** Noureddine Kerzazi. *Where Are Females in OSS Projects? Socio Technical Interactions.* In: Collaborative Networks and Digital Transformation. PRO-VE 2019. IFIP Advances in Information and Communication Technology, vol 568. Springer, Cham, 37-54.

Journaux internationaux

- **Ikram El Asri,** M. A. Janati Idrissi. *Understanding OSS Project's Collaborative Dynamics: Core and Peripheral Interactions*. International Journal of Scientific & Engineering Research -IJSER. Volume 9, Issue 11 November 2018, Pages 1541-1551.
- **Ikram El Asri,** Noureddine Kerzazi, Gias Uddin, Foutse Khomh, M. A. Janati Idrissi. *An Empirical Study of Sentiments in Code Reviews*. Information and Software Technology-IST. Volume 114, October 2019, Pages 37-54

BIBLIOGRAPHIE

Ahmed, T., A. Bosu, A. Iqbal and S. Rahimi (2017). SentiCR: a customized sentiment analysis tool for code review interactions. The 32nd IEEE/ACM International Conference on Automated Software Engineering, IEEE Press**:** 106-111.

Alessandro, M., T. Parastou, A. Bram and O. Marco (2014). Do developers feel emotions? an exploratory analysis of emotions in software artifacts. Proceedings of the 11th Working Conference on Mining Software Repositories. Hyderabad, India**:** 262-271.

Amrit, C. and J. van Hillegersberg (2010). "Exploring the impact of socio-technical core-periphery structures in open source software development." Journal of Information Technology **25**(2): 216-229.

Asundi, J. and R. Jayant (2007). Patch Review Processes in Open Source Software Development Communities: A Comparative Case Study. The 40th Annual Hawaii International Conference on System Sciences**:** 166-166.

Athey, S. and G. Ellison (2014). "Dynamics of open source movements." Journal of Economics & Management Strategy **23**(2): 294-316.

Barcellini, F., F. Détienne and J.-M. Burkhardt (2014). "A situated approach of roles and participation in Open Source Software Communities." Human–Computer Interaction **29**(3): 205-255 %@ 0737-0024.

Batagelj, V. and M. Zaversnik (2003). "An O(m) Algorithm for Cores Decomposition of Networks." Adv. in Data Analysis and Classification **5**(2): 129-145.

Baysal, O., O. Kononenko, R. Holmes and M. W. Godfrey (2013). The influence of non-technical factors on code review. The 20th Working Conference on Reverse Engineering-WCRE Koblenz, Germany**:** 122-131.

Baysal, O., O. Kononenko, R. Holmes and M. W. Godfrey (2015). "Investigating technical and non-technical factors influencing modern code review." Empirical Software Engineering **21**(3): 932-959.

Begel, A., R. DeLine and T. Zimmermann (2010). Social media for software engineering. FSE/SDP workshop on Future of software engineering research. Santa Fe, New Mexico, USA**:** 33-38.

Beller, M., A. Bacchelli, A. Zaidman and E. Juergens (2014). Modern code reviews in open-source projects: which problems do they fix? Proceedings of the 11th Working Conference on Mining Software Repositories-MSR'14**:** 202-211.

Bholowalia, P. and A. Kumar (2014). "EBK-means: A clustering technique based on elbow method and k-means in WSN." International Journal of Computer Applications **105**(9).

Bird, C. (2011). Sociotechnical coordination and collaboration in open source software. 27th IEEE International Conference on Software Maintenance-ICSM, IEEE**:** 568-573.

Bird, C., D. Pattison, R. D'Souza, V. Filkov and P. Devanbu (2008). Latent Social Structure in Open Source Projects. Proceeding of the 16th International Symposium on Foundations of Software Engineering-FSE' 08. Atlanta, Georgia**:** 24-35.

Bo, P., L. Lillian and V. Shivakumar (2002). Thumbs up?: sentiment classification using machine learning techniques. Proceedings of the ACL-02 conference on Empirical methods in natural language processing Association for Computational Linguistics. **10:** 79-86.

Borgatti, S. P. (2003). The key player problem.

Bosch, J. (2009). From software product lines to software ecosystems. Proceedings of the 13th international software product line conference**:** 111-119.

Bosu, A. and J. C. Carver (2013). Impact of peer code review on peer impression formation: A survey. The ACM / IEEE International Symposium on Empirical Software Engineering and Measurement--ESEM '13, IEEE**:** 133-142.

Bosu, A. and J. C. Carver (2014). Impact of developer reputation on code review outcomes in OSS projects. The ACM / IEEE International Symposium on Empirical Software Engineering and Measurement-ESEM '14. Torino, Italy**:** 1-10.

Breivold, H. P., I. Crnkovic and M. Larsson (2012). "Software architecture evolution through evolvability analysis." Journal of Systems and Software **85**(11): 2574-2592.

Brunswicker, S., A. Armisen and S. Haefliger (2017). "Collaboration in OSS Communities: Who Solves Whose Problems?" SSRN Electronic Journal.

Butler, B. S. (2001). "Membership size, communication activity, and sustainability: A resource-based model of online social structures." Information systems research **12**(4): 346-362.

Calefato, F., F. Lanubile, F. Maiorano and N. Novielli (2018). "Sentiment polarity detection for software development." Empirical Software Engineering **23**(3): 1352-1382.

Capra, E., C. Francalanci and F. Merlo (2008). "An Empirical Study on the Relationship Between Software Design Quality, Development Effort and Governance in Open Source Projects." IEEE Transactions on Software Engineering **34**(6): 765-782.

Carlson, P. E. (2015). Engaging developers in open source software projects: harnessing social and technical data mining to improve software development. Doctor of Philosophy, Iowa State University.

Chawner, B. (2004). Free/open source software: new opportunities, new challenges. 12th VALA Biennial Conference e Breaking boundaries: Integration & Interoperability.

Chen, J., W. Geyer, C. Dugan, M. Muller and I. Guy (2009). Make new friends, but keep the old: recommending people on social networking sites. SIGCHI Conference on Human Factors in Computing Systems. Boston, MA, USA**:** 201-210.

Chen, X., Y. Zhou, D. Probert and J. Su (2017). "Managing knowledge sharing in distributed innovation from the perspective of developers: empirical study of open source software projects in China." Technology Analysis & Strategic Management **29**(1): 1-22.

Christopher, D. M., R. Prabhakar and S. Hinrich (2008). "Introduction to information retrieval." An Introduction To Information Retrieval **151**(177): 5.

Conley, C. A. and L. Sproull (2009). Easier Said than Done: An Empirical Investigation of Software Design and Quality in Open Source Software Development. 42nd Hawaii International Conference on System Sciences**:** 1-10.

Cox, A. (1998). "Cathedrals, bazaars and the town council." Slashdot. Org.

Crowston, K., K. Chudoba, M.-B. Watson-Manheim and P. Rahmati (2016). Inter-team coordination in large-scale agile development: A test of organizational discontinuity theory. Proceedings of the Scientific Workshop Proceedings of XP2016, Edinburgh, Scotland, UK.

Crowston, K. and J. Howison (2005). "The Social Structure of Free and Open Source Software Development." First Monday **10**(2).

Crowston, K. and I. Shamshurin (2017). "Core-periphery communication and the success of free/libre open source software projects." Journal of Internet Services and Applications **8**(1).

Crowston, K., K. Wei, J. Howison and A. Wiggins (2012). "Free/Libre open-source software development: what we know and what we do not know." ACM Computing Surveys **44**(2): 1-35.

Crowston, K., K. Wei, Q. Li and J. Howison (2006). Core and Periphery in Free/Libre and Open Source Software Team Communications. Proceedings of the 39th Annual Hawaii International Conference on System Sciences. **06**.

Cucuringu, M., P. Rombach, S. H. Lee and M. A. Porter (2016). "Detection of core–periphery structure in networks using spectral methods and geodesic paths." European Journal of Applied Mathematics **27**(06): 846-887.

D'Ambros, M., M. Lanza and M. Lungu (2009). "Visualizing Co-Change Information with the Evolution Radar." IEEE Transactions on Software Engineering **35**(5): 720-735.

Daniel, S., V. Midha, A. Bhattacherhjee and S. P. Singh (2018). "Sourcing knowledge in open source software projects: The impacts of internal and external social capital on project success." The Journal of Strategic Information Systems **27**(3): 237-256.

Destefanis, G., M. Ortu, S. Counsell, M. Marchesi and R. Tonelli (2017). "Measuring Affectiveness and Effectiveness in Software Systems." arXiv preprint arXiv:1703.01642.

Dietze, S. (2005). Agile Requirements Definition for Software Improvement and Maintenance in Open Source Software Development. The 13th IEEE Requirements Engineering Conference Paris, France**:** 176-187.

Dinkelacker, J. and P. Garg (2001). Corporate source: Applying open source concepts to a corporate environment (position paper), HPL-2001-135.

Dmitrienko, A., G. Molenberghs, C. Chuang-Stein and W. W. Offen (2005). Analysis of clinical trials using SAS: A practical guide, SAS Institute.

Efstathiou, V. and D. Spinellis (2018). Code review comments: Language Matters. Proceedings of the 40th International Conference on Software Engineering New Ideas and Emerging Results - ICSE-NIER '18**:** 69-72.

El Asri, I. and M. A. Janati (2018). "Understanding OSS Project's Collaborative Dynamics : Core and Peripheral Interactions." International Journal of Scientific & Engineering Research -IJSER **9**(11).

El Asri, I., N. Kerzazi, L. Benhiba and M. Janati (2017). From Periphery to Core: A Temporal Analysis of GitHub Contributors' Collaboration Network. Collaboration in a Data-Rich World-pro-ve'17**:** 217-229.

El Asri, I., N. Kerzazi, G. Uddin and F. Khomh (2018). An Empirical Study of Sentiments in Code Reviews (online appendix).

Feller, J. and B. Fitzgerald (2002). Understanding open source software development, Addison-Wesley Longman Publishing Co., Inc. .

Franco-Bedoya, O., D. Ameller, D. Costal and X. Franch (2014). Protocol for a systematic literature review on open source-software ecosystems Technical report.

Freeman, L. C. (1977). "A set of measures of centrality based on betweenness." Sociometry: 35-41.

Gacek, C. and B. Arief (2004). "The many meanings of open source." IEEE software **21**(1): 34-40.

Garcia, D., M. S. Zanetti and F. Schweitzer (2013). The Role of Emotions in Contributors Activity: A Case Study on the GENTOO Community. Third International Conference on Cloud and Green Computing, IEEE**:** 410-417.

Geldenhuys, J. (2010). Finding the Core Developers. 36th Euromicro Conference on Software Engineering and Advanced Applications. Lille, France**:** 447-450.

George, J. M. and A. P. Brief (1992). "Feeling good doing good: A conceptual analysis of the mood at work organizational spontaneity relationship." Psychological Bulletin **112**(2): 310-329.

Ghosh, R. A., R. Glott, B. Krieger and G. Robles (2002). Free/libre and open source software: Survey and study, International Institute of Infonomics, University of Maastricht and Berlecon Research GmbH.

GitHub. (2018). "The State of the Octoverse." Retrieved 2019-04-01, from https://octoverse.github.com/.

Goeminne, M. and T. Mens (2010). A framework for analysing and visualising open source software ecosystems. Proceedings of the Joint ERCIM Workshop on Software Evolution-EVOL and International Workshop on Principles of Software Evolution-IWPSE. Antwerp, Belgium, ACM**:** 42-47

Guillory, J., J. Spiegel, M. Drislane, B. Weiss, W. Donner and J. Hancock (2011). Upset now?: emotion contagion in distributed groups. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. Vancouver, BC, Canada**:** 745-748.

Guo, L., P. Qu, R. Zhang, D. Zhao, H. Wang, R. Liu, B. Mi, H. Yan and S. Dang (2018). "Propensity Score-Matched Analysis on the Association Between Pregnancy Infections and Adverse Birth Outcomes in Rural Northwestern China." Scientific Reports **8**(1): 5154.

Guzman, E., D. Azocar and Y. Li (2014). Sentiment analysis of commit comments in GitHub: an empirical study. Proceedings of the 11th Working Conference on Mining Software Repositories. Hyderabad, India**:** 352-355.

Guzman, E. and B. Bruegge (2013). Towards emotional awareness in software development teams. Proceedings of the 9th Joint Meeting on Foundations of Software Engineering. Saint Petersburg, Russia, ACM**:** 671-674.

Hannemann, A. and R. Klamma (2013). Community dynamics in open source software projects: Aging and social reshaping. IFIP Advances in Information and Communication Technology, Springer**:** 80-96.

Hauge, Ø., C. Ayala and R. Conradi (2010). "Adoption of open source software in software-intensive organizations – A systematic literature review." Information and Software Technology **52**(11): 1133-1154.

Hemetsberger, A. and C. Reinhardt (2004). Sharing and creating knowledge in open-source communities: the case of KDE. The Fifth European Conference on Organizational Knowledge, Learning, and Capabilities. Innsbruck, Austria.

Herbsleb, J. D. and A. Mockus (2003). Formulation and preliminary test of an empirical theory of coordination in software engineering. Proceedings of the 9th European software engineering conference held jointly with 10th ACM SIGSOFT international symposium on Foundations of software engineering - ESEC/FSE '03.

Hinds, P. and C. McGrath (2006). Structures that work: social structure, work structure and coordination ease in geographically distributed teams. Proceedings of the 20th International Conference on Computer Supported Cooperative Work. Banff, Alberta, Canada**:** 343-352.

Huang, C.-C. (2009). "Knowledge sharing and group cohesiveness on performance: An empirical study of technology R&D teams in Taiwan." Technovation **29**(11): 786-797.

Huntley, C. L. (2003). "Organizational learning in open-source software projects: an analysis of debugging data." IEEE Transactions on Engineering Management **50**(4): 485-493.

Iskoujina, Z. and J. Roberts (2015). "Knowledge sharing in open source software communities: motivations and management." Journal of Knowledge Management **19**(4): 791-813.

Islam, M. R. and M. F. Zibran (2017). Leveraging Automated Sentiment Analysis in Software Engineering. 14th International Conference on Mining Software Repositories-MSR'17, IEEE/ACM 203-214.

Jensen, C. and W. Scacchi (2007). Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study. 29th International Conference on Software Engineering-ICSE'07, IEEE**:** 364-374.

Jiang, Y., B. Adams and D. M. German (2013). Will my patch make it? And how fast? Case study on the Linux kernel. 10th Working Conference on Mining Software Repositories-MSR'13**:** 101-110.

Joblin, M. (2017). Structural and Evolutionary Analysis of Developer Networks, University of Passau.

Joblin, M., S. Apel, C. Hunsen and W. Mauerer (2017). Classifying developers into core and peripheral: An empirical study on count and network metrics. Proceedings of the 39th International Conference on Software Engineering, IEEE Press**:** 164-174.

Joblin, M., S. Apel and W. Mauerer (2016). "Evolutionary trends of developer coordination: a network approach." Empirical Software Engineering **22**(4): 2050-2094.

Kaur, J., A. Kaur and P. Kaur (2014). "Socio-technical Interactions in OSS Development." International Journal of Engineering Research and General Science **2**(3).

Kerzazi, N. and I. El Asri (2016). Who Can Help to Review This Piece of Code? Working Conference on Virtual Enterprises-pro-ve'16. Porto,Portugal, Springer**:** 289-301.

Kerzazi, N. and I. El Asri (2017). Knowledge Flows Within Open Source Software Projects: A Social Network Perspective. Advances in Ubiquitous Networking 2**:** 247-258.

Khan, I. A., W.-P. Brinkman and R. M. Hierons (2010). "Do moods affect programmers' debug performance?" Cognition, Technology & Work **13**(4): 245-258.

Khanjani, A. and R. Sulaiman (2011). The aspects of choosing open source versus closed source. IEEE Symposium on Computers & Informatics**:** 646-649.

Kilamo, T., A. Nieminen, J. Lautamäki, T. Aho, J. Koskinen, J. Palviainen and T. Mikkonen (2014). Knowledge transfer in collaborative teams: experiences from a two-week code camp. 36th International Conference on Software Engineering-ICSE '13. Hyderabad, India**:** 264-271.

Kononenko, O., O. Baysal and M. W. Godfrey (2016). Code review quality: How Developers See It. Proceedings of the 38th International Conference on Software Engineering - ICSE '16**:** 1028-1038.

Kononenko, O., O. Baysal, L. Guerrouj, Y. Cao and M. W. Godfrey (2015). Investigating code review quality: Do people and participation matter? International Conference on Software Maintenance and Evolution-ICSME, IEEE**:** 111-120.

Krebs, V. (2002). "Uncloaking terrorist networks." First Monday **7**(4).

Kucuktunc, O., B. B. Cambazoglu, I. Weber and H. Ferhatosmanoglu (2012). A large-scale sentiment analysis for Yahoo! answers. the Fifth International Conference on Web Search and Web Data Mining-WSDM'12. Seattle, WA, USA, ACM**:** 633-642

Lakhani, K. R. and E. Von Hippel (2004). How open source software works:“free” user-to-user assistance. Produktentwicklung mit virtuellen Communities, Springer**:** 303-339.

Lanzara, G. F. and M. Morner (2004). The knowledge ecology of open-source software projects. the Annual Conference of the American Academy of Management. New Orleans.

Lee Endres, M., S. P. Endres, S. K. Chowdhury and I. Alam (2007). "Tacit knowledge sharing, self-efficacy theory, and application to the Open Source community." Journal of knowledge management **11**(3): 92-103.

Lee, G. K. and R. E. Cole (2003). "From a firm-based to a community-based model of knowledge creation: The case of the Linux kernel development." Organization science **14**(6): 633-649.

Lee, L. and B. Pang (2008). "Opinion Mining and Sentiment Analysis." Foundations and Trends® in Information Retrieval **2**(1–2): 1-135.

Lin, B., F. Zampetti, G. Bavota, M. Di Penta, M. Lanza and R. Oliveto (2018). Sentiment Analysis for Software Engineering: How Far Can We Go? the 40th International Conference on Software Engineering-ICSE '18. Gothenburg, Sweden

Loyola, P. and I.-Y. Ko (2014). Population dynamics in open source communities: an ecological approach applied to Github. Proceedings of the companion publication of the 23rd international conference on World wide web companion, ACM**:** 993-998.

Macqueen, J. (1967). Some methods for classification and analysis of multivariate observations. the fifth Berkeley symposium on mathematical statistics and probability.

Madey, G., V. Freeh and R. Tynan (2002). The open source software development phenomenon: An analysis based on social network theory. Americas Conference on Information Systems-AMCIS**:** 247.

Melian, C. and M. Mähring (2008). Lost and Gained in Translation: Adoption of Open Source Software Development at Hewlett-Packard. Open Source Development, Communities and Quality**:** 93-104.

Meneely, A. and L. Williams (2011). Socio-technical developer networks: should we trust our measurements? Proceeding of the 33rd International Conference on Software Engineering-ICSE '11). Waikiki, Honolulu, USA**:** 281-290.

Merriam-Webster. (2016). "Dictionary and Thesaurus | Merriam-Webster." Retrieved 22/11/2016, 2016, from http://www.merriam-webster.com/.

Mockus, A., R. T. Fielding and J. Herbsleb (2000). A case study of open source software development: the Apache server. Proceedings of the 22nd international conference on Software engineering-ICSE '00 Limerick, Ireland Acm**:** 263-272.

Mockus, A., R. T. Fielding and J. D. Herbsleb (2002). "Two case studies of open source software development: Apache and Mozilla." ACM Transactions on Software Engineering and Methodology **11**(3): 309-346.

Moon, J. Y. and L. Sproull (2005). "Essence of distributed work: The case of the Linux kernel." First Monday.

Mukadam, M., C. Bird and P. C. Rigby (2013). Gerrit software code review data from Android. 10th Working Conference on Mining Software Repositories-MSR'13**:** 45-48.

Novielli, N., F. Calefato and F. Lanubile (2018). A Gold Standard for Emotion Annotation in Stack Overflow. Proceedings of the 15th International Conference on Mining Software Repositories-MSR '18**:** 14-17.

Onur, K., B. B. Cambazoglu, I. Weber and H. Ferhatosmanoglu (2012). A large-scale sentiment analysis for Yahoo! answers. Proceedings of the fifth ACM international conference on Web search and data mining. Seattle, Washington, USA**:** 633-642.

Opensource.org. (2018). "Open Source Initiative." Retrieved 09-10-2018, 2018, from https://opensource.org/about.

Ortu, M., B. Adams, G. Destefanis, P. Tourani, M. Marchesi and R. Tonelli (2015). Are bullies more productive?: empirical study of affectiveness vs. issue fixing time. Proceedings of the 12th Working Conference on Mining Software Repositories. Florence, Italy**:** 303-313.

Ortu, M., A. Murgia, G. Destefanis, P. Tourani, R. Tonelli, M. Marchesi and B. Adams (2016). The emotional side of software developers in JIRA. International workshop on Mining software repositories-MSR'16. Austin, TX, USA, ACM**:** 480-483.

Park, J. r. (2008). "Linguistic politeness and face-work in computer mediated communication, Part 2: An application of the theoretical framework." Journal of the American Society for Information Science and Technology **59**(14): 2199-2209.

Paulson, J. W., G. Succi and A. Eberlein (2004). "An empirical study of open-source and closed-source software products." IEEE Transactions on Software Engineering **30**(4): 246-256.

Phillips, S., T. Zimmermann and C. Bird (2014). Understanding and improving software build teams. Proceedings of the 36th International Conference on Software Engineering. Hyderabad, India**:** 735-744.

Piryani, R., D. Madhavi and V. K. Singh (2016). "Analytical mapping of opinion mining and sentiment analysis research during 2000–2015." Information Processing & Management.

Popescu, A.-M. and M. Pennacchiotti (2010). Detecting controversial events from twitter. Proceedings of the 19th ACM international conference on Information and knowledge management. Toronto, ON, Canada, ACM**:** 1873-1876.

Qiao, S., C. Wang, Y. Lei and G. Wang (2012). Community evolution in dynamic social networks. Proceedings of the 2nd International Conference on Computer and Information Applications-ICCIA

Raghunathan, S., A. Prasad, B. K. Mishra and H. Chang (2005). "Open source versus closed source: software quality in monopoly and competitive markets." IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans **35**(6): 903-918.

Rahman, M. M., C. K. Roy and R. G. Kula (2017). Predicting usefulness of code review comments using textual features and developer experience. Proceedings of the 14th International Conference on Mining Software Repositories-MSR '17. Buenos Aires, Argentina, IEEE**:** 215-226.

Raymond, E. S. (1999). The cathedral and the bazaar.

Rigby, P. C., D. M. German and M.-A. Storey (2008). Open source software peer review practices: a case study of the apache server. International Conference on Software Engineering. Leipzig, Germany, ACM**:** 541-550.

Roberts, J. (2000). "From know-how to show-how? Questioning the role of information and communication technologies in knowledge transfer." Technology Analysis & Strategic Management **12**(4): 429-443.

Roberts, J. A., I.-H. Hann and S. A. Slaughter (2006). "Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects." Management Science **52**(7): 984-999.

Robertson, I. and C. Cooper (2011). Well-being: Productivity and Happiness at Work, PalgraveConnect

Robles, G. and J. M. Gonzalez-Barahona (2006). Contributor turnover in libre software projects. IFIP International Conference on Open Source Systems, Springer. **203:** 273-286.

Robles, G., J. M. Gonzalez-Barahona and I. Herraiz (2009). Evolution of the core team of developers in libre software projects. 6th IEEE International Working Conference on Mining Software Repositories**:** 167-170.

Scacchi, W. (2004). "Free and open source development practices in the game community." IEEE Software **21**(1): 59-66.

Scacchi, W. (2005). Socio-technical interaction networks in free/open source software development processes. Software process modeling, Springer**:** 1-27.

Scacchi, W., J. Feller, B. Fitzgerald, S. Hissam and K. Lakhani (2006). "Understanding Free/Open Source Software Development Processes." Software Process: Improvement and Practice **11**(2): 95-105.

Shannon, P., A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski and T. Ideker (2003). "Cytoscape: A Software Environment for Integrated Models of Biomolecular Interaction Networks." Genome Research **13**(11): 2498-2504.

Shirali-Shahreza, S. and M. Shirali-Shahreza (2008). Various aspects of Open Source Software development. International Symposium on Information Technology**:** 1-7.

Singh, P. V., Y. Tan and N. Youn (2011). "A hidden Markov model of developer learning dynamics in open source software projects." Information Systems Research **22**(4): 790-807.

Singh, V. and L. Holt (2013). "Learning and best practices for learning in open-source software communities." Computers & Education **63**: 98-108.

Sinha, V., A. Lazar and B. Sharif (2016). Analyzing developer sentiment in commit logs. 13th Working Conference on Mining Software Repositories. Austin, TX, USA, IEEE/ACM 520-523.

Sowe, S. K., A. Karoulis and I. Stamelos (2006). A constructivist view of knowledge management in open source virtual communities. Managing learning in virtual settings: the role of context, IGI Global**:** 290-308.

Stephan, D. (2010). Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software, Springer Publishing Company, Incorporated.

Stewart, K. and S. Gosain (2001). An exploratory study of ideology and trust in open source development groups. Proceedings of the International Conference on Information Systems-ICIS'01**:** 63.

Stol, K.-J. and M. Ali Babar (2009). Reporting Empirical Research in Open Source Software: The State of Practice. Open Source Ecosystems: Diverse Communities Interacting**:** 156-169.

Sudhakar, G. P., A. Farooq and S. Patnaik (2012). "Measuring productivity of software development teams." Serbian Journal of Management **7**(1).

Surian, D., N. Liu, D. Lo, H. Tong, E. P. Lim and C. Faloutsos (2011). Recommending People in Developers' Collaboration Network. 18th Working Conference on Reverse Engineering**:** 379-388.

Syeed, M., I. Hammouda and T. Syatä (2013). "Evolution of Open Source Software Projects: A Systematic Literature Review." Journal of Software **8**(11).

Terceiro, A., L. R. Rios and C. Chavez (2010). An empirical study on the structural complexity introduced by core and peripheral developers in free software projects. Brazilian Symposium on Software Engineering, IEEE**:** 21-29.

Thavaneswaran, A. (2008). Propensity Score Matching in Observational Studies, Manitoba Centre for Health Policy.

Thelwall, M., K. Buckley and G. Paltoglou (2012). "Sentiment strength detection for the social web." Journal of the American Society for Information Science and Technology **61**(1): 2544-2558.

Thelwall, M., K. Buckley, G. Paltoglou, D. Cai and A. Kappas (2010). "Sentiment in short strength detection informal text." Journal of the American Society for Information Science and Technology **61**(12): 2544-2558.

Thongtanunam, P., C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida and K. i. Matsumoto (2015). Who should review my code? A file location-based code-reviewer recommendation approach for Modern Code Review. Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on**:** 141-150.

Toral, S. L., M. R. Martínez-Torres and F. Barrero (2010). "Analysis of virtual communities supporting OSS projects using social network analysis." Information and Software Technology **52**(3): 296-303.

Tukey, J. W. (1977). Exploratory data analysis, Reading, Mass.

Tuma, D. (2005). "Open source software: Opportunities and challenges." Journal of Defence Software Engineering 6-10.

von Krogh, G., S. Spaeth and K. R. Lakhani (2003). "Community, joining, and specialization in open source software innovation: a case study." Research Policy **32**(7): 1217-1241.

von Krogh, G. and E. von Hippel (2006). "The Promise of Research on Open Source Software." Management Science **52**(7): 975-983.

VonHippel, E. and G. VonKrogh (2003). "Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science." Organization Science **14**(2): 209-223.

Wasserman, S. and K. Faust (1994). Social network analysis: Methods and applications, Cambridge university press.

Weiss, M. and G. Moroiu (2007). Ecology and dynamics of open source communities. Emerging Free and Open Source Software Practices, IGI Global**:** 46-67.

Weiss, M., G. Moroiu and P. Zhao (2006). Evolution of Open Source Communities. IFIP Working Group 2.13 Foundation on Open Source Software. E. Damiani, B. Fitzgerald, W. Scacchi, M. Scotto and G. Succi. Como, Italy, Springer US**:** 21-32.

Weissgerber, P., D. Neu and S. Diehl (2008). Small patches get in! Proceedings of the 2008 international workshop on Mining software repositories - MSR '08.

Wenger, E. (2010). Communities of practice and social learning systems: the career of a concept. Social learning systems and communities of practice, Springer**:** 179-198.

Wesselius, J. (2008). "The Bazaar inside the Cathedral: Business Models for Internal Markets." IEEE Software **25**(3): 60-66.

Wiig, K. (2004). People-Focused Knowledge Management: How Effective Decision Making Leads to Corporate, Elsevier.

Xu, J., Y. Gao, S. Christley and G. Madey (2005). A topological analysis of the open souce software development community. Proceedings of the 38th Annual Hawaii International Conference on System Sciences. Big Island, HI, USA, USA, IEEE**:** 198-198.

Yang, X., R. G. Kula, N. Yoshida and H. Iida (2016). Mining the modern code review repositories. The 13th International Conference on Mining Software 460-463.

Ye, Y. and K. Kishida (2003). Toward an understanding of the motivation open source software developers. Proceeding of the Int'l Conf. on Software Engineering (ICSE'03)**:** 419-429.

Ye, Y., K. Nakakoji, Y. Yamamoto and K. Kishida (2008). The co-evolution of systems and communities in free and open source software development. Global Information Technologies: Concepts, Methodologies, Tools, and Applications, IGI Global**:** 3765-3776.

Zagalsky, A., J. Feliciano, M.-A. Storey, Y. Zhao and W. Wang (2015). The emergence of github as a collaborative platform for education. Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing, ACM**:** 1906-1917

Zhang, W. and J. Storck (2001). Peripheral members in online communities. Americas Conference on Information Systems-AMCIS'01**:** 117.

Zhao, L. and S. Elbaum (2003). "Quality assurance under the open source development model." Journal of Systems and Software **66**(1): 65-75.

Zhehui, L., J. C. Gardiner and C. J. Bradley (2010). "Applying Propensity Score Methods in Medical Research: Pitfalls and Prospects." Medical Care Research and Review **67**(5): 528-554.

ANNEXE. Extraction des données de GitHub

Dans cette annexe nous présentons un aperçu sur notre approche d'extraction des données GitHub. D'abord, nous décrivons le flux de GitHub. Ensuite nous présentons un aperçu sur un projet open source dans GitHub, puis une présentation de l'API GitHub.

A.1. Flux de GitHub

GitHub est un flux de travail léger, basé sur les branches, qui prend en charge les équipes et les projets dans lesquels des déploiements sont effectués régulièrement (voir **Figure A-** *1*).

Figure A- 1 : Flux dans GitHub

Création d'une branche : Lors du travail sur un projet, le contributeur a un tas de fonctionnalités ou d'idées différentes en cours de développement à tout moment, dont certaines sont prêtes à fonctionner et d'autres qui ne le sont pas. Le branchement existe pour aider les développeurs à gérer ce flux de travail. A travers la création d'une branche dans un projet, le contributeur crée un environnement dans lequel il peut tester de nouvelles idées. Les modifications apportées sur une branche n'affectent pas la branche principale laissant une totale liberté d'expérimenter et de valider des modifications, sachant que la branche ne sera pas fusionnée jusqu'à ce qu'elle soit prête à être examinée par une personne. En conclusion, le branchement est un concept fondamental dans Git, et le flux complet de GitHub est basé sur celui-ci. Une seule règle : tout élément de la branche principale est toujours déployable.

Ajout des commits : Une fois une branche créée, il est temps de commencer à apporter des modifications. Chaque fois un ajout, modification ou suppression d'un fichier, un commit est ajouté à la branche. Les commits créent également un historique transparent du travail que les autres peuvent suivre pour comprendre ce qui est fait et pourquoi. Chaque commit a un message de commit associé, qui est une description expliquant pourquoi un changement particulier a été apporté.

Ouvrir un Pull Request : Les pull requests sont utiles pour contribuer aux projets open source et pour gérer les modifications apportées aux référentiels partagés. Les pull requests fournissent un moyen d'informer les responsables du projet des modifications que souhaitées qu'ils prennent en compte. Lors de l'utilisation d'un modèle de référentiel partagé, les pull requests aident à démarrer la révision du code et la conversation sur les modifications proposées avant leur fusion dans la branche principale.

Discussion et révision : Une fois qu'un pull request a été ouverte, la personne ou l'équipe examinant les modifications peut avoir des questions ou des commentaires. Le contributeur peut également continuer à faire pression sur la branche à la lumière des discussions et des commentaires concernant les commits. Si quelqu'un signale un bogue dans le code, les corrections peuvent être ajoutées dans la branche et faire avancer le changement. GitHub affiche les nouveaux commits et tous les commentaires.

Fusionner et déployer : Avec GitHub, il est possible déployer depuis une branche pour les tests finaux en production avant de les fusionner à la branche principale. Une fois un pull request a été examinée et les tests ont réussi, les modifications sont déployées pour les vérifier en production.

A.2. Aperçu sur un projet GitHub

La **Figure A-*2*** montre la page principale du projet AngularJS dans GitHub. Nous avons pointé en cercle rouge les informations sur le projet à savoir le nom, le nombre d'étoiles, le nombre de commits et le nombre de contributeurs impliqués. Ensuite **Figure A-*3*** montre le détail des commits.

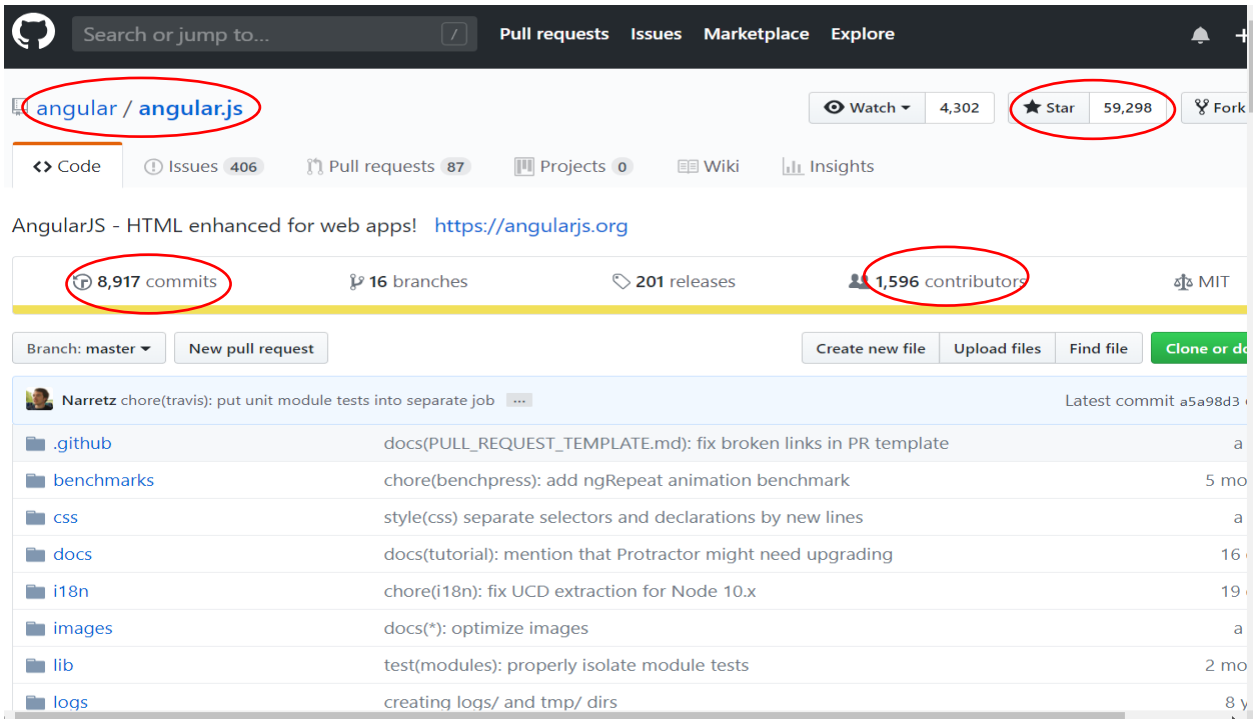


Figure A- 2 : Page principale d'un projet GitHub

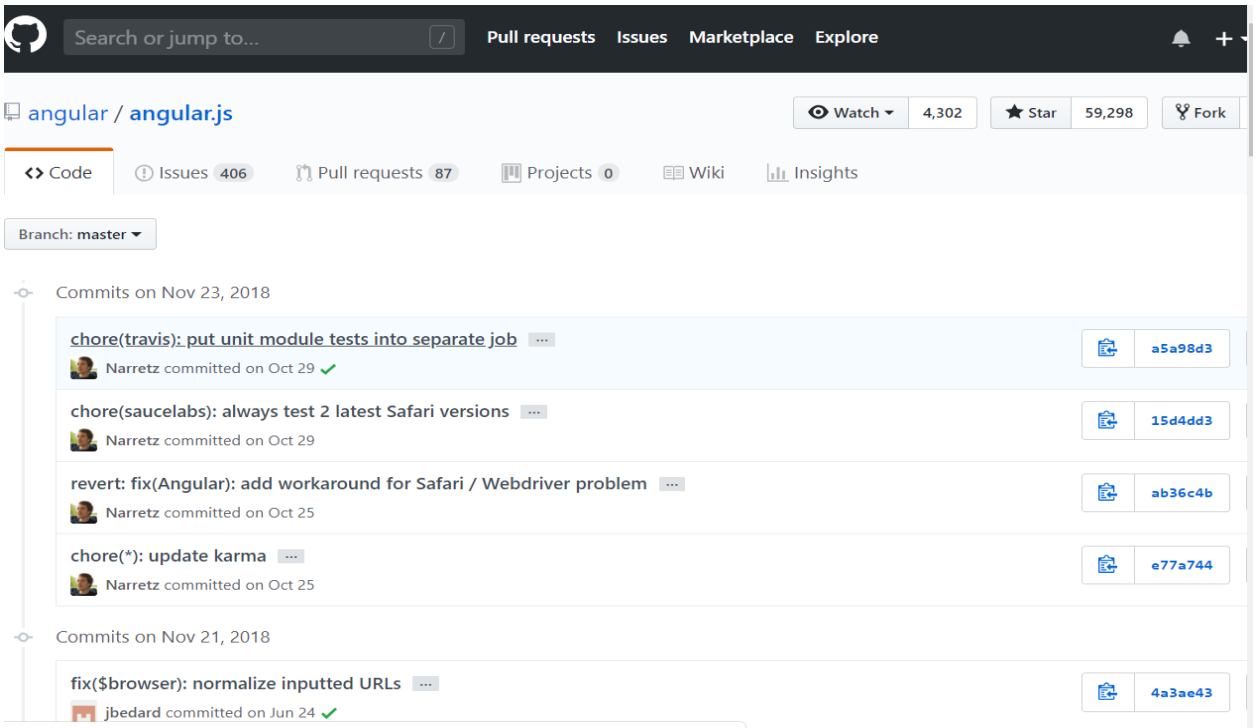


Figure A- 3 : Détails commits d'un projet GitHub

A.3. Présentation de L'API GitHub

GitHub fourni une documentation détaillée est destinée à se familiariser avec l'API GitHub. La documentation couvre tous ce qu'il faut savoir, de l'authentification à la manipulation des résultats, en passant par la combinaison des résultats avec d'autres applications.

Le premier point avant de commencer est l'authentification. En effet Les clients non authentifiés peuvent faire 60 demandes par heure. Pour obtenir plus, il faut s'authentifier (faire quelque chose d'intéressant avec l'API GitHub nécessite une authentification). Le moyen le plus simple de s'authentifier avec l'API GitHub consiste simplement à utiliser un nom d'utilisateur et un mot de passe GitHub via l'authentification de.

Pour nos besoins d'études nous utilisons l'API pour accéder aux détails des projets sélectionnés. Chaque projet est un référentiel dans GitHub.

```
https://api.github.com/repos/angular/angular.js
```

L'API GitHub prend en charge la création, l'affichage et la comparaison des validations dans un référentiel. Dans la suite nous présentons à titre d'exemple l'accès aux commits d'un projet. Plus de détails sur le guide GitHub est disponible en ligne sur https://developer.github.com/v3/

L'accès au détails commits est effectué à travers le lien suivant :

```
/repos/angular/angular.js/commits
```

La réponse est une liste Json comme suit :


```
{
    "sha": "0cdff4273766579fcf897fac7da84346b5a83029",
    "node_id":
"MDY6Q29tbWl0NDYwMDc4OjBjZGZmNDI3Mzc2NjU3OWZjZjg5N2ZhYzdkYTg0MzQ2YjVhODMwMjk=",
    "commit": {
      "author": {
        "name": "Eirik Blakstad",
        "email": "eirik@wolftech.no",
        "date": "2018-11-16T13:01:32Z"
      },
      "committer": {
        "name": "George Kalpakas",
        "email": "kalpakas.g@gmail.com",
        "date": "2018-12-02T11:15:39Z"
      },
      "message": "fix(aria/ngClick): check if element is `contenteditable`
before blocking spacebar\n\n`ngAria`'s `ngClick` blocks spacebar keypresses
on non-blacklisted\nelements, which is an issue when the element is
`contenteditable`.\n\nCloses #16762",
```

```
      "tree": {
        "sha": "3108a3c8b0c953022d310a1529b993c5500d1fe5",
        "url":
"https://api.github.com/repos/angular/angular.js/git/trees/3108a3c8b0c95302
2d310a1529b993c5500d1fe5"
      },
      "url":
"https://api.github.com/repos/angular/angular.js/git/commits/0cdff427376657
9fcf897fac7da84346b5a83029",
      "comment_count": 0,
      "verification": {
        "verified": false,
        "reason": "unsigned",
        "signature": null,
        "payload": null
      }
    },
    "url":
"https://api.github.com/repos/angular/angular.js/commits/0cdff4273766579fcf
897fac7da84346b5a83029",
    "html_url":
"https://github.com/angular/angular.js/commit/0cdff4273766579fcf897fac7da84
346b5a83029",
    "comments_url":
"https://api.github.com/repos/angular/angular.js/commits/0cdff4273766579fcf
897fac7da84346b5a83029/comments",
    "author": {
      "login": "Eblax",
      "id": 22151370,
      "node_id": "MDQ6VXNlcjIyMTUxMzcw",
      "avatar_url":
"https://avatars0.githubusercontent.com/u/22151370?v=4",
      "gravatar_id": "",

    },
    "committer": {
      "login": "gkalpak",
      "id": 8604205,
      "node_id": "MDQ6VXNlcjg2MDQyMDU=",
      "avatar_url": "https://avatars2.githubusercontent.com/u/8604205?v=4",
      "gravatar_id": "",
      "url": "https://api.github.com/users/gkalpak",

      "site_admin": false
    },
    "parents": [
      {
        "sha": "a5a98d36c0acd7a02614f5ccf18582e07f412e76",
        "url":
"https://api.github.com/repos/angular/angular.js/commits/a5a98d36c0acd7a026
14f5ccf18582e07f412e76",
        "html_url":
"https://github.com/angular/angular.js/commit/a5a98d36c0acd7a02614f5ccf1858
2e07f412e76"
      }
    ]
  },
```


……

Cet Annexe a présenté un aperçu sur notre approche d'extraction des données de la plateforme de collaboration GitHub.