L'identification des projets de logiciel libre accessibles aux nouveaux contributeurs

Mémoire de Master Sciences de l'Éducation, parcours SYNVA

Soutenu par
Paul HERVOT

le 8 septembre 2022

Mémoire présenté en vue de l'obtention du Grade de Master

Commission de soutenance composée par :

Benoît CRESPIN	directeur
Richard NGU LEUBOU	assesseur

INSPÉ Institut national supérieur du professorat et de l'éducation
Université de Strasbourg

Université de Strasbourg

INSPÉ Institut national supérieur du professorat et de l'éducation
Université de Strasbourg

ATTESTATION D'AUTHENTICITE

Ce document rempli et signé par l'étudiant(e) doit être inséré dans tous les documents soumis à évaluation, après la page de garde.

Je, soussigné(e) : *(nom et prénom)* Paul HERVOT

Etudiant(e) de : *(Formation et année)* Master 2 SYNVA, 2021-2022

- certifie avoir pris connaissance du « Guide du Mémoire » de Master de l'INSPÉ et en particulier des pages consacrées au plagiat,
- certifie que le document soumis ne comporte aucun texte ou son, aucune image ou vidéo, copié sans qu'il soit fait explicitement référence aux sources selon les normes de citation universitaires.

Fait à Strasbourg le 24/08/2022

Signature de l'étudiant(e) :

Tout plagiat réalisé par un étudiant constitue une fraude au sens du décret du 13 juillet 1992 relatif à la procédure disciplinaire dans les Établissement public à caractère scientifique, culturel et professionnel (EPSSCP). La fraude par plagiat relève de la compétence de la section de discipline de l'Université. En général la sanction infligée aux étudiants qui fraudent par plagiat s'élève à un an d'exclusion de tout établissement d'enseignement supérieur.

Tout passage ou schéma copié sans qu'il soit fait explicitement référence aux sources, selon les normes de citation universitaires sera considéré par le jury ou le correcteur comme plagié.

Colophon

Ce document a été composé avec l'aide de KOMA-Script et LaTeX en utilisant la classe kaobook.

Un grand merci à Antoine Pietri pour son aide précieuse lors de la collecte des données sur le graphe de Software Heritage.
Deux grands mercis à Marc Trestini et Benoît Crespin, ayant par la force des choses tous deux endossé le rôle de directeur de ce mémoire à différentes étapes de son écriture, leurs expertises complémentaires m'a permis d'arriver au bout de ce travail si nouveau pour moi.

Table des matières

Table des figures

Liste des tableaux

1. Introduction

1.1. Le logiciel libre

Au cours des quelques décennies précédentes, le numérique s'est développé pour intégrer et assister la plupart des domaines de recherche, des secteurs industriels, ainsi que des aspects de nos vies. Son champ est aujourd'hui très vaste et connaît en son sein de nombreuses dynamiques différentes ayant un impact sur ses domaines d'applications, parmi lesquelles celle du logiciel libre.

En essence, les logiciels, applications, services numériques et autres programmes sont des ensembles d'instructions compréhensibles par un ordinateur et lui indiquant comment utiliser ses ressources afin de réaliser un certain nombre de tâches. Pour être utilisables par un ordinateur, ces instructions doivent être formulées en « code machine », un langage souvent représenté en binaire et extrêmement difficile à manipuler par les humains, même les plus spécialisés, aussi bien en ce qui concerne son écriture que sa compréhension. C'est pourquoi lors de la création d'un logiciel, les développeurs décrivent d'abord le comportement souhaité dans un autre langage, textuel et raisonnablement maîtrisable pour un être humain spécialisé, que l'on appelle un « langage de programmation ». Cette description textuelle du logiciel est ce que l'on appelle son « code source », il est ensuite traduit en code machine par un autre programme (généralement appelé un « compilateur ») avant d'être livré aux personnes souhaitant utiliser le logiciel.

Pour utiliser un logiciel, nous n'avons donc besoin que de son code machine, mais c'est seulement en lisant son code *source* que l'on peut vraisemblablement comprendre son fonctionnement, corriger ses éventuelles erreurs, l'adapter à d'autres besoins, vérifier la présence de comportements malveillants, etc. La question de rendre publiquement disponible le code source d'un logiciel est un enjeu faisant souvent intervenir plusieurs intérêts divergents, allant de la transparence du fonctionnement de certains services et leur gestion démocratique au secret industriel.

L'expression « logiciel libre » désigne en français un logiciel disponible publiquement et gratuitement dont le code source est lui aussi disponible publiquement et gratuitement, ainsi que sa modification et sa redistribution par toutes et tous. Cette expression désigne aussi et surtout un mouvement visant à développer la part de logiciel libre dans le numérique, un mouvement dans lequel se développe toute une culture, des pratiques, des fondations et des événements internationaux. L'expression équivalente anglaise, « *Free*

and Open Source Software » (FOSS) est souvent utilisée dans les références citées dans ce mémoire.

1.2. Mon parcours personnel

Étant issu d'une formation d'ingénieur spécialisée en informatique, mon domaine d'expertise se situe principalement dans le numérique, et en particulier dans le développement logiciel. Mon parcours m'a rapidement amené au logiciel libre, notamment via les technologies utilisées au sein de mon école d'ingénieur, l'EPITA, mais surtout via mon implication pendant mes études dans des associations comme Prologin[1] et GConfs[2] : l'une organisant un concours francophone d'algorithmique ainsi que des stages d'initiation à l'informatique pour lycéennes[3], l'autre œuvrant au partage de connaissance entre pairs en organisant des conférences par les élèves et pour les élèves au sein de l'école. Ces deux associations ont produit des sites web ainsi que de nombreux outils informatiques, tous sur le principe du logiciel libre, ce qui a beaucoup contribué aux échanges avec leurs publiques cibles en permettant à tous de participer, commenter et améliorer les outils développés, en plus d'accroitre la transparence de leurs structures. Plusieurs des outils développés par ces associations ont d'ailleurs été réutilisés dans d'autres contextes par d'autres personnes qui leurs sont externes. Cette implication dans le mouvement du logiciel libre m'a aussi amené à assister régulièrement au *FOSDEM*[4], la conférence principale du logiciel libre en Europe, aussi bien l'occasion de s'informer sur l'évolution des nombreuses communautés du logiciel libre qu'un prétexte pour y retrouver de vieilles connaissances lors d'un week-end à Bruxelles.

1 : https://prologin.org
2 : https://gconfs.fr
3 : https://girlscancode.fr/
4 : https://fosdem.org

Mon parcours s'est ensuite enrichi d'une expérience dans l'enseignement lorsque j'ai rejoint pendant ma cinquième et dernière année à l'EPITA la petite équipe des « assistants », qui encadre l'enseignement de la principale matière informatique pour les étudiants de troisième année. J'ai écrit et évalué au sein de cette équipe des sujets de travaux pratique et encadré de nombreuses séances de travail sur ordinateur, ce qui a développé chez moi une forte appétence pour l'enseignement. J'ai souhaité explorer cette appétence après la fin de mes études lors d'un service civique au collège et lycée Jean Monnet de Strasbourg, où j'y ai donné des cours de soutient. Ayant confirmé mon intérêt pour cette voie professionnelle, je suis revenu à l'EPITA pour y devenir enseignant des matières informatiques, à l'occasion de l'ouverture de son nouveau campus à Strasbourg.

Enfin, de nombreuses discussions avec des amis chercheurs et passionnés de méthodologie scientifique, ainsi qu'une tentative per-

sonnelle pour me familiariser avec la recherche en éducation, m'ont poussé à entreprendre une formation m'aidant d'une part à développer mes connaissances en éducation, et me fournissant d'autre part une introduction à la recherche scientifique, tout en exploitant mes acquis en informatique. C'est ainsi que je me suis inscrit au Master SYNVA et que je cherche, à travers ce mémoire, à explorer et contribuer à la recherche touchant ces domaines de l'informatique, de l'enseignement, ainsi que de la libération de la connaissance et de l'information.

1.3. Les projets de logiciel libre dans l'enseignement de l'informatique

La pédagogie de projet est une méthode d'enseignement qui consiste à inviter les apprenants à appliquer leurs connaissances et à en acquérir de nouvelles au travers d'un projet plus ou moins concret et imitant plus ou moins fidèlement une situation réelle. Réalisé soit individuellement soit en groupe, les projets aboutissent généralement à une production évaluée par l'enseignant, parfois au travers d'une présentation donnée par les apprenants. L'efficacité de cette méthode a fait l'objet de nombreuses recherches au cours des années précédentes. Plusieurs méta-analyses concluent que cette méthode a des effets mesurables, positifs et importants sur les résultats académiques des apprenants en sciences sociales et en sciences naturelles (C.-H. Chen & Yang, 2019 ; Balemen & Keskin, 2018).

C.-H. Chen et Yang (2019) : « Revisiting the effects of project-based learning on students' academic achievement: A meta-analysis investigating moderators »
Balemen et Keskin (2018) : « The effectiveness of Project-Based Learning on science education: A meta-analysis search. »

Les projets proposés aux étudiants sont le plus souvent factices, imaginés par les enseignants dans le seul but de servir d'exercice. Utiliser au contraire des projets réels, qui ne cessent pas d'exister après la fin de la séquence pédagogique, sur lesquels faire travailler les étudiants donne des résultats encore meilleurs, mais est aussi plus difficile et incertain à encadrer pour les enseignants (Detmer *et al.*, 2010 ; Buckley *et al.*, 2004). Il est notamment très difficile pour l'enseignant de prévoir à l'avance les problèmes que les apprenants risquent de rencontrer, c'est pourquoi d'autres efforts dans cette direction ont tenté un compromis, consistant à créer des projets de logiciel libre aussi proches des situations réelles que possible, mais dédiés à l'éducation (Meneely *et al.*, 2008). Les projets en question ne sont cependant plus accessibles aujourd'hui, une explication possible à leur disparition étant leur portée réduite à l'éducation et l'absence d'une communauté persistante autour d'eux.

Detmer *et al.* (2010) : « Incorporating Real-World Projects in Teaching Computer Science Courses »
Buckley *et al.* (2004) : « Benefits of Using Socially-Relevant Projects in Computer Science and Engineering Education »
Meneely *et al.* (2008) : « ROSE: A Repository of Education-Friendly Open-Source Projects »

Bien qu'elles soient plutôt rares, quelques initiatives existent aussi pour enseigner spécifiquement les pratiques du logiciel libre dans l'éducation supérieur et la formation tout au long de la vie. Certaines sont des initiatives venant des communautés du logiciel libre,

comme le *Professional Certificate in Open Source Software Development, Linux and Git*[5] de la *Linux Foundation*, et en particulier la première séquence : *Open Source Software Development: Linux for Developers*[6] ; ou les séminaires de l'*open source initiative*[7]. Enfin, certaines universités proposent des cursus en informatique ayant des éléments visant spécifiquement les pratiques du logiciel libre, c'est le cas notamment de l'Université de l'État de Caroline du Nord aux États-Unis, qui a expérimenté plusieurs façons d'inclure la contribution aux projets de logiciel libre dans leurs cursus d'informatique (Meneely *et al.*, 2008 ; Gehringer, 2007) ; mais aussi de l'université de Calais qui propose actuellement un « Master Informatique - Ingénierie du logiciel libre »[8], sous la forme d'une formation en alternance.

5 : https://www.edx.org/professional-certificate/linuxfoundationx-open-source-software-development-linux-and-git

6 : https://www.edx.org/course/open-source-software-development-linux-for-developers

7 : https://opensource.org/osi-open-source-education

Meneely *et al.* (2008) : « ROSE: A Repository of Education-Friendly Open-Source Projects »
Gehringer (2007) : « Open source for homework: Real projects, peer reviewed »

8 : https://www.univ-littoral.fr/formation/offre-de-formation/masters/master-informatique-ingenierie-du-logiciel-libre/

Partie théorique

2. Revue de littérature

2.1. L'intérêt professionnel et personnel dans la contribution aux projets de logiciel libre

L'industrie informatique s'intéresse de plus en plus au logiciel libre et à son inclusion ou ses interactions avec ses activités, au point de ressentir un besoin de plus en plus important en terme de formation sur ce sujet, ainsi qu'une volonté à participer aux projets utilisés au sein des entreprises (Kim *et al.*, 2012). Même quand les logiciels libres ne sont pas directement impliqués dans les activités d'une entreprise, d'autres auteurs avancent que les personnes produisant des contributions à de tels projets envoient un signal positif à leurs pairs et augmentent leurs chances de succès professionnel dans l'industrie informatique (Lerner & Tirole, 2002, p. 218).

Kim *et al.* (2012) : « Step-by-Step Strategies and Case Studies for Embedded Software Companies to Adapt to the FOSS Ecosystem »

Lerner et Tirole (2002) : « Some Simple Economics of Open Source »

Kim *et al.* (2012) identifient cinq grandes étapes dans les interactions que les entreprises ont généralement avec les logiciels libres :

l'identification – évaluer la qualité d'un logiciel libre, ses garanties et les éventuels brevets utilisés ;
l'adoption – utilisation du logiciel ;
la conformité – examen des licence attachées au logiciel, de leur compatibilité avec les activités de l'entreprise, et des contraintes qu'elles apportent dans les activités futures ;
la contribution – partage avec la communauté.

Les motivations poussant certaines personnes à contribuer au logiciel libre sont aussi personnelles. Oreg et Nov (2008) distinguent l'importance de ces motivations en fonction du contexte de la contribution. Lorsqu'il s'agit de contribuer au code source d'un projet de logiciel libre, les contributeurs semblent principalement poussés par un intérêt de développement personnel (apprendre à utiliser de nouvelles technologies ou consolider son expertise par la pratique), puis par une volonté altruiste d'aider le projet, puis en dernier lieu par la publicité des contributions, dont ils se servent pour se construire une réputation. Lorsqu'il s'agit de contribuer à un autre type de contenu que le code source en revanche (les articles de l'encyclopédie Wikipédia par exemple), c'est la volonté altruiste qui semble être le moteur principal amenant à une contribution, puis le développement personnel et, là encore, l'établissement d'une réputation.

Oreg et Nov (2008) : « Exploring motivations for contributing to open source initiatives: The roles of contribution context and personal values »

2.2. Les nouveaux contributeurs dans le logiciel libre

Les barrières d'entrée

Mendez *et al.* (2018) notent la difficulté des nouveaux volontaires à rejoindre une communauté de logiciel libre, citant comme exemple extrême le projet « Apache Hadoop » voyant 82% de ses nouveaux volontaires quitter le projet après leur première contribution (Steinmacher *et al.*, 2013). Les études qu'ils citent mentionnent deux approches différentes de la résolution de problèmes observées dans les projets informatique : l'une consiste à d'abord rassembler le plus d'informations possibles sur le problème avant de tenter en un deuxième temps de le résoudre (approche statistiquement plus commune chez les femmes que chez les hommes), l'autre consiste à agir sur la première information ou piste prometteuse trouvée, quitte à revenir en arrière et chercher de nouvelles informations si elles n'étaient pas concluantes (approche statistiquement plus commune chez les hommes que chez les femmes) (Darley & Smith, 1995 ; Meyers-Levy & Loken, 2015). De façon plus précise concernant les logiciels, on observe deux façons d'apprendre les fonctionnalités d'un logiciel statistiquement préférées de façon inégales selon le genre : l'une, statistiquement plus représentée chez les femmes, consiste à les apprendre méthodiquement en suivant des processus d'apprentissage clairs. L'autre, statistiquement plus représentée chez les hommes, consiste à expérimenter à la manière d'un jeu avec ces fonctionnalités (Burnett *et al.*, 2010 ; Hou *et al.*, 2006). Une des tendances trouvées par Mendez *et al.* (2018, p. 1008) est que la majorité (58%) des barrières rencontrées dans leur étude sont de nature sociale (« *community-oriented* ») et non techniques. Concernant les aspects plus techniques, il semble que les outils et la documentation représentent la majorité des barrières rencontrées, alors même que ces éléments ont spécifiquement pour objectif d'aider les nouvelles contributions. C'est un type de barrière qui semble amplifié lorsque les méthodes d'apprentissage et de traitement de l'information de la personne consistent à rassembler le plus d'information possible avant de commencer à agir. Ces méthodes étant sur-représentées chez les femmes, ces barrières deviennent discriminantes selon le genre, et peuvent potentiellement participer à expliquer la sous-représentation des femmes dans les communautés de logiciel libre.

Mendez *et al.* (2018) : « Open Source Barriers to Entry, Revisited: A Sociotechnical Perspective »

Steinmacher *et al.* (2013) : « Why do newcomers abandon open source software projects? »

Darley et Smith (1995) : « Gender Differences in Information Processing Strategies: An Empirical Test of the Selectivity Model in Advertising Response »
Meyers-Levy et Loken (2015) : « Revisiting gender differences: What we know and what lies ahead »

Burnett *et al.* (2010) : « Gender Differences and Programming Environments: Across Programming Populations »
Hou *et al.* (2006) : « "Girls Don't Waste Time": Pre-Adolescent Attitudes toward ICT »

Plus précisément en ce qui concerne les contributions, selon Steinmacher *et al.* (2013), les facteurs susceptibles de provoquer un abandon des nouveaux contributeurs semblent être les réponses inadéquates proposées à leurs questions, notamment lorsqu'un autre

Steinmacher *et al.* (2013) : « Why do newcomers abandon open source software projects? »

nouveau contributeur répond à la place d'un membre expérimenté du projet.

Il a été théorisé et souvent observé de façon générale que la diversité de genre, d'origines et d'ancienneté au sein d'une équipe augmente sa productivité (voir par exemple Horwitz & Horwitz, 2007). C'est ce que Vasilescu *et al.* (2015) ont pu confirmer dans le cas précis des projets de logiciel libre hébergés sur GitHub. Ils mettent cette observation en perspective avec celle de la proportion de femmes dans ce type d'équipe, en très forte minorité, et concluent en suggérant que des efforts supplémentaires d'inclusivité et de réduction des barrières d'entrée pour ces population en particulier permettraient probablement d'augmenter la valeur globale de ce type de projets.

Horwitz et Horwitz (2007) : « The Effects of Team Diversity on Team Outcomes: A Meta-Analytic Review of Team Demography »

Vasilescu *et al.* (2015) : « Gender and Tenure Diversity in GitHub Teams »

La littérature sur le sujet identifie un grand nombre de barrières d'entrée que rencontrent les nouveaux arrivants dans un projet de logiciel libre, les plus importantes semblent être :

- le manque d'interaction sociale avec les membres du projet (Steinmacher, Graciotto Silva *et al.*, 2015 ; Steinmacher, Conte *et al.*, 2015) ;
- le manque de réponse (dans un temps raisonnable) à leurs requêtes (Steinmacher, Graciotto Silva *et al.*, 2015) ;
- les connaissances techniques initiales (Steinmacher, Graciotto Silva *et al.*, 2015) ;
- l'identification d'une tâche par laquelle commencer (Steinmacher, Conte & Gerosa, 2015) ;
- la mise en place de l'environnement propre au projet abordé et permettant de faire une première contribution (Steinmacher *et al.*, 2016).

Steinmacher, Graciotto Silva *et al.* (2015) : « A systematic literature review on the barriers faced by newcomers to open source software projects »

Steinmacher, Conte *et al.* (2015) : « Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects »

Steinmacher, Conte et Gerosa (2015) : « Understanding and Supporting the Choice of an Appropriate Task to Start with in Open Source Software Communities »

Steinmacher *et al.* (2016) : « Overcoming Open Source Project Entry Barriers with a Portal for Newcomers »

Mesure de l'accessibilité d'un projet pour les nouveaux contributeurs

Ces modèles de barrières d'entrée permettent d'identifier les freins que rencontrent les nouveaux contributeurs dans un projet de logiciel libre. Un autre point de vue sur la question consiste à mesurer *a posteriori* la capacité d'un projet à accueillir les nouveaux contributeurs et à leur permettre de mener leurs contributions à terme.

Une première approche à cela, qualitative, est de proposer à un ensemble maîtrisé d'étudiants d'essayer de contribuer à un projet et de leur soumettre un questionnaire avant et/ou après l'expérience afin de recueillir leur ressenti concernant les problèmes

qu'ils pensent rencontrer et ceux qu'ils ont effectivement rencontré, puis d'éventuellement compléter avec des entretiens individuels afin d'explorer les mécanismes liés à ces problèmes (Steinmacher *et al.*, 2016 ; Stanik *et al.*, 2018 ; voir aussi G. Chen, 2005).

Une autre approche, peut être plus adaptée aux recherches quantitatives, consiste à compter automatiquement le nombre de contributeurs accumulés depuis le début de la vie du projet et jusqu'à différents points dans le temps, afin d'étudier la progression de ce nombre. C'est ce que font par exemple Lerner *et al.* (2006), mais sans préciser la méthode de comptage plus en détail que « en utilisant plusieurs outils d'édition de texte » (p. 116, traduit).

Lerner *et al.* (2006) : « The Dynamics of Open-Source Contributors »

Qiu *et al.* (2019) traitent d'une question similaire en essayant de déterminer quels sont les signaux que les potentiels nouveaux contributeurs regardent (et comment) pour choisir un projet de logiciel libre auquel contribuer. Ils s'intéressent pour cela spécifiquement aux projets disponibles publiquement sur GitHub, ce qui leur permet, après une approche exploratoire pour dégager des hypothèses, d'étudier empiriquement l'effet des signaux que la plateforme met en évidence.

Qiu *et al.* (2019) : « The Signals That Potential Contributors Look for When Choosing Open-Source Projects »

Qiu *et al.* (2019) suggèrent qu'une mesure possible de l'accessibilité pour les nouveaux contributeurs (« *newcomers openness* » dans l'article) est le pourcentage de *pull requests* créées par des contributeurs externes. Ils définissent les « contributeurs externes » par opposition aux contributeurs principaux (les « *core contributors* ») qu'ils identifient, eux, comme étant les personnes auteurs de plus de 5% des *commits* du projet. En réalité cependant, leur étude quantitative simplifie la mesure en ne retenant que la présence ou non de *pull request* venant de nouveaux contributeurs (p. 16), ce qui leur permet de n'étudier qu'une variable binaire.

2.3. L'analyse automatique (« minage ») de projets de logiciel libre

Pour leur analyse quantitative, Qiu *et al.* (2019) commencent par collecter des données sur la plateforme d'hébergement de code source GitHub via son API. Ce choix est cohérent avec un grand nombre d'études de ces dernières années (Cosentino *et al.*, 2017), la plateforme est très populaire dans le milieu du logiciel libre et a connu une croissance très importante ces dernières années. Ces mêmes études notent cependant parfois que se restreindre à cette seule plateforme est une limite diminuant la représentativité de l'échantillon qu'elles étudient (Rousseau *et al.*, 2019). Les limites et erreurs courantes liées au « minage » de GitHub font d'ailleurs l'objet d'une littérature grandissante et remettant potentiellement

Cosentino *et al.* (2017) : « A Systematic Mapping Study of Software Development With GitHub »

Rousseau *et al.* (2019) : « Growth and Duplication of Public Source Code over Time: Provenance Tracking at Scale »

en question les résultats des études utilisant cette technique sans détailler suffisamment leur méthodologie de collecte des données (Kalliamvakou *et al.*, 2014 ; Trujillo *et al.*, 2022).

Kalliamvakou *et al.* (2014) : « The Promises and Perils of Mining GitHub »
Trujillo *et al.* (2022) : « The penumbra of open source: projects outside of centralized platforms are longer maintained, more academic and more collaborative »

Le nombre de projets et d'artéfacts qui leur sont lié est par ailleurs en augmentation rapide, voir exponentielle. Rousseau *et al.* (2019) estiment que le nombre de *commit*s uniques originaux double environ tous les trente mois, le nombre de fichiers uniques originaux doublerait quant à lui tous les vingt-deux mois. Cette explosion du nombre de *commit*s et de fichiers s'accompagne d'un important phénomène de duplication : de nombreux fichiers se retrouvent dans de nombreux projets différents, comme les notices de licence par exemple. Les *commit*s eux-même peuvent se trouver dans plusieurs projets différents, c'est ce qui arrive lorsqu'un projet débute en tant que *fork* d'un autre. Ces deux points rendent de plus en plus difficile l'analyse efficace et à grande échelle de l'historique de développement des projets de logiciel libre.

Rousseau *et al.* (2019) : « Growth and Duplication of Public Source Code over Time: Provenance Tracking at Scale »

L'archive de logiciels de Software Heritage

Les techniques de compression de graphe sont aujourd'hui suffisamment avancées pour permettre une nouvelle approche à cette analyse automatique des projets, en rendant un très grand nombre de ceux-ci disponibles au sein d'une même structure de donnée unifiée et optimisée (Boldi *et al.*, 2020). Cette approche permet par exemple de grandement faciliter la détection des « clones » ou « *fork*s » de projets, c'est à dire des projets ayant une part de leur historique de développement en commun. La structure de donnée proposée par Boldi *et al.* (2020) est un graphe orienté où chaque *commit* est représenté par un nœud et où chaque arc $u \xrightarrow{succ} v$ (« v est un successeur de u », autrement dit pour les *commit*s « v est un ancêtre immédiat de u ») possède aussi un arc inverse $u \xleftarrow{pred} v$ (« u est un prédécesseur de v »). Une telle structure permet donc de trouver tous les « *fork* » d'un projet avec un classique calcul de composante fortement connexe, ce qui peut se faire avec un simple parcours profondeur ou largeur du graphe à partir de n'importe lequel de ses *commit*s.

Boldi *et al.* (2020) : « Ultra-Large-Scale Repository Analysis via Graph Compression »

Software Heritage[1] est une initiative exploitant ces techniques de compression afin de construire une archive aussi complète que possible du code source actuellement disponible publiquement dans le monde. L'archive est elle aussi publique et un de leurs objectifs est de la rendre exploitable pour la recherche (Di Cosmo & Zacchiroli, 2017). Il s'agit du corpus que Boldi *et al.* (2020) ont utilisé comme exemple pour leur modèle de compression. L'archive comptait en 2018 plus d'un milliard de *commit*s uniques archivés depuis quatre-vingt-cinq millions d'origines différentes (telles que des *remotes*

1 : https://www.softwareheritage.org/

Di Cosmo et Zacchiroli (2017) : « Software Heritage: Why and How to Preserve Software Source Code »

Git ou des versions de paquets Python publiés sur PyPi[2]). L'archive est stockée dans une structure en « ajout seul » : elle grandit de façon continue en archivant petit à petit de nouveaux projets et en ajoutant de nouveaux artéfacts à chaque évolution des projets déjà archivés, ce qui en fait l'un des corpus les plus complets et exploitable par la recherche scientifique (Rousseau *et al.*, 2019 ; Pietri *et al.*, 2019).

2 : https://pypi.org/

Rousseau *et al.* (2019) : « Growth and Duplication of Public Source Code over Time: Provenance Tracking at Scale »
Pietri *et al.* (2019) : « The Software Heritage Graph Dataset: Public Software Development Under One Roof »

2.4. Méthodologies de recherches autour de ces questions

Méthodes qualitatives d'identification des barrières d'entrée

Mendez *et al.* (2018, p. 1006) analysent les réponses écrites lors d'entretiens via un codage qualitatif suivant un modèle de Steinmacher *et al.* (2014), avec pour objectif de répondre à trois questions de recherche que l'on pourrait traduire ainsi : « Quels problèmes peuvent être révélés en regardant les logiciels libres au travers des outils et de l'infrastructure ? », « Les outils et l'infrastructure participent-ils à créer des barrières d'entrée pour les nouvelles personnes souhaitant contribuer ? Si oui, comment ? » et « Existe-t-il des barrières d'entrées pour ces personnes qui soient biaisés sur la question du genre ? Si oui, de quelles façons sont-elles biaisées ? ». L'entretien lui-même se fait en suivant un processus appelé « *GenderMag* », lui-même un type de « *Cognitive Walkthrough* » qui consiste à présenter aux sujets interrogés un personnage fictif avec ses traits de caractères, ses compétences et ses préférences (une « *persona* ») et à leur demander d'imaginer comment ce personnage se comporterait dans une situation donnée. (Burnett *et al.*, 2016 ; Spencer, 2000)

Steinmacher *et al.* (2014) : « Preliminary Empirical Identification of Barriers Faced by Newcomers to Open Source Software Projects »

Burnett *et al.* (2016) : « GenderMag: A Method for Evaluating Software's Gender Inclusiveness »
Spencer (2000) : « The Streamlined Cognitive Walkthrough Method, Working around Social Constraints Encountered in a Software Development Company »

Sélection des projets étudiés

Dans une méta-analyse concernant l'étude des barrières d'entrée, Steinmacher, Graciotto Silva *et al.* (2015) remarquent un biais courant dans la sélection des projets de logiciel libre retenus dans la littérature (p. 83) : les auteurs ont tendance à plutôt étudier les projets importants, matures et populaires, car ils sont plus susceptibles d'apporter un volume important de données. Bien que les résultats semblent cohérent avec les plus rares études sélectionnant de petits projets, ils invitent la communauté scientifique à s'y intéresser plus en profondeur et à constituer des échantillons plus variés.

Steinmacher, Graciotto Silva *et al.* (2015) : « A systematic literature review on the barriers faced by newcomers to open source software projects »

Quand Kalliamvakou *et al.* (2014) analysent les *pull requests* sur

Kalliamvakou *et al.* (2014) : « The Promises and Perils of Mining GitHub »

GitHub, ils ne retiennent que les projets ayant au moins deux auteurs de *commit*s uniques, ce qu'ils considèrent comme des projets « vraiment collaboratifs ». Ce choix vient de leur observation qu'une très grande partie des projets hébergés sur la plateforme (71%) sont des projets « personnels », utilisés par leurs auteurs à des fins d'expérimentation ou de stockage, sans réelle ouverture à la collaboration.

Partie expérimentale

3. Méthodologie

3.1. Problématique

La question de l'accessibilité des projets de logiciel libre pour les nouveaux contributeurs est encore assez peu explorée de façon quantitative dans la littérature. Qiu *et al.* (2019) se sont intéressés aux signaux que les potentiels nouveaux contributeurs utilisent afin de sélectionner les projets auxquels ils *essayent* de contribuer, c'est à dire les signaux formant l'attractivité de ces projets. Nous nous attacherons dans notre étude à déterminer si certains de ces mêmes signaux sont, de surcroit, prédictifs d'une réelle accessibilité de ces projets, c'est à dire à quel point de nouveaux contributeurs *réussissent* à produire une contribution apparaissant dans l'historique de développement du projet.

Comme nous l'avons vu, la recherche quantitative portant sur les historiques de développement des logiciels libre a récemment eu tendance à limiter sa population étudiée aux projets hébergés sur la plateforme GitHub, ce qui occulte une part importante des projets de logiciel libre (Kalliamvakou *et al.*, 2014 ; Trujillo *et al.*, 2022). Pour améliorer la représentativité de nos résultats, nous utiliserons donc dans notre étude l'archive de Software Heritage, celle-ci ne se limitant pas aux projets hébergés sur GitHub, ni même au système de versionnement Git.

L'usage de cette archive nous limitera en revanche dans le type de données que nous pourrons collecter. Il sera par exemple impossible de mesurer le nombre d'*issues* publiées par de nouveaux intervenants dans le *bug tracker* des projets, ou le nombre de *pull requests*. Nous devrons nous servir exclusivement des données disponibles dans l'historique de développement en tant que tel, comme l'ensemble des *commits*, leurs auteurs, leur date, ainsi que les noms et contenus des fichiers qu'ils manipulent.

3.2. Mesure de l'accessibilité choisie

La variable mesurée qui sera utilisée comme proxy pour l'accessibilité d'un projet pour les nouveaux contributeurs est le nombre de contributeurs apparaissant pour la première fois dans l'historique de développement du projet au cours de la période de référence étudiée : du premier juin 2019 au premier septembre 2019 (voir section 2.2, ainsi que Qiu *et al.*, 2019, p. 13, 16). Cette période de référence a été choisie sur recommandation de Software Heritage, la désignant comme un bon compromis entre le nombre de projets pour lesquels les données sont disponible dans l'archive à cette période et le caractère récent des observations.

3.3. Hypothèses

Hypothèse H1 : *les projets possédant des instructions de contribution (fichier type « `CONTRIBUTING.md` » ou section type «* Contributing *» dans un fichier type `README.md`) sont plus accessibles pour les nouveaux contributeurs que ceux n'en ayant pas (voir Qiu* et al.*, 2019, p. 11).*

Hypothèse H2 : *le nombre de contributeurs uniques récents (au cours des six mois précédents la période de référence étudiée) d'un projet est positivement corrélée à son accessibilité pour les nouveaux contributeurs (voir Qiu* et al.*, 2019, p. 12-13, 16).*

Hypothèse H3 : *le nombre de* commit*s récents (au cours des six mois précédents la période de référence étudiée) au sein d'un projet est positivement corrélé à son accessibilité pour les nouveaux contributeurs (voir Qiu* et al.*, 2019, p. 13, 16).*

3.4. Constitution de l'échantillon

L'échantillon de départ est constitué de la totalité des projets archivés dans le graphe de Software Heritage. Plusieurs critères d'exclusion ont ensuite été appliqués :

- quand deux projets ou plus ont un ou plusieurs *commit*s en commun et sont donc des *fork*s les uns des autres, seul celui qui a reçu le plus d'activité (mesuré par le nombre d'arêtes maximal entre le *commit* source et un des *commit*s initiaux) a été retenu comme représentant du groupe, afin d'éviter de compter plusieurs fois les mêmes historiques ;
- les projets n'ayant enregistré aucune activité (aucun *commit*) au cours de la période de référence étudiée sont considérés inactifs et n'ont pas été retenus (voir Kalliamvakou *et al.*, 2014) ;
- les projets ayant vu moins de deux contributeurs uniques récents (au cours des six mois précédent la période de référence) sont considérés comme des projets individuels, non réellement collaboratifs et n'ont pas été retenus (voir Kalliamvakou *et al.*, 2014).

3.5. Collecte initiale des données

Comme présenté en section 2.3, l'archive de Software Heritage se présente sous la forme d'un graphe orienté. Ses nœuds représentent diverses entités de l'historique de développement, parmi lesquelles les *commit*s (appelés *revisions* au sein du graphe), les origines (l'URL à partir de laquelle a été archivé un projet), les *snapshots* (point de départ d'un archivage précis et daté, beaucoup de projets étant ré-archivés régulièrement), les dossiers et les fichiers (contenu du projet). Un arc $u \xrightarrow{succ} v$ peut donc signifier, en fonction des types de u et v :

- que u est un *commit* créé immédiatement après le *commit* v ;
- que v est un archivage (*snapshot*) du projet disponible à l'origine u ;
- que v est le dernier *commit* d'une des branches visibles lors de l'archivage (*snapshot*) u ;
- que v est le dossier racine du contenu du projet en l'état du *commit* u ;
- etc.

Afin de découvrir et sélectionner les projets dans lesquels nous collecterons les données, une première exploration est lancée à partir de chaque nœud de type « origine » du graphe avec deux objectifs : identifier tous les *fork*s du projet de départ et sélectionner un représentant pour le groupe. Pour ce faire, l'exploration prend la forme d'un premier parcours largeur sur les arcs « successeur » des nœuds « révision » afin d'identifier tous les *commit*s initiaux accessibles depuis l'origine de départ, puis d'un deuxième parcours largeur partant de chacun de ces *commit*s initiaux afin d'identifier tous les autres nœuds « origine » de la composante connexe. Des marqueurs de niveaux lors de ce deuxième parcours largeur permettent de mesurer la distance qui éloigne chaque origine ainsi découverte de son *commit* initial le plus éloigné. Pour chaque composante connexe, seule le *snapshot* le plus récent de l'origine la plus éloignée d'un *commit* initial est retenu pour la collecte.

Pour tous les projets ainsi retenu, un nouveau parcours largeur est démarré à partir de sa branche principale afin de récolter les données de recherche. Le nombre de contributeurs uniques récents et de *commit*s récents se compte facilement en accédant à la date et aux auteurs de chaque nœud « révision » rencontré, mais la vérification de la présence d'instructions de contribution demande un peu plus de travail. Le graphe possède le nom et la hiérarchie de chacun des fichiers du projet pour chaque *commit*, mais pas leur contenu. Ce parcours se contente donc initialement de vérifier la présence d'un fichier nommé `CONTRIBUTING.md` ou assimilé, nous considérons que le projet possède effectivement des instructions de contribution si un tel fichier existe. Si aucun fichier de ce type n'est trouvé, nous recherchons alors la présence d'un fichier nommé `README.md` ou assimilé, l'absence d'un tel fichier nous permet de conclure que le projet ne possède *pas* d'instructions de contribution. Si un tel fichier est trouvé, en revanche, nous devons vérifier son contenu avant de conclure, nous sauvegardons alors l'identifiant unique du contenu du fichier afin de pouvoir l'analyser lors d'une phase de collecte ultérieure. Enfin, le nombre de nouveaux contributeurs est calculé en comptant les contributeurs uniques de la période de référence qui n'apparaissent dans aucun *commit* antérieur à cette période (et non seulement au cours des six mois précédents la période de référence).

Voir aussi l'annexe A.1 pour plus de détails.

3.6. Collecte complémentaire des données

Il nous faut maintenant vérifier le contenu du fichier `README.md` (ou assimilé) retenu pour chaque projet ne possédant pas de fichier `CONTRIBUTING.md` (ou assimilé) afin de déterminer s'ils possèdent ou non des instructions de contribution, et ainsi compléter notre jeu de donnée. Le contenu des fichiers est bien archivé par Software Heritage mais n'est pas disponible directement dans le graphe, il est en revanche possible de télécharger ces contenus de deux autres façons différentes : soit via des requêtes HTTP sur le site web https://archive.softwareheritage.org/, accessible publiquement, soit en téléchargeant les contenus directement depuis leur « *registry* » Amazon S3 [1], lui aussi accessible publiquement. Le site https://archive.softwareheritage.org/ limite le nombre de requêtes autorisées par adresse IP à quelques dizaines par heure. Le *registry* Amazon S3 n'impose pas ce genre de limite, mais le contenus de certains fichiers n'y est pas encore disponible. Nous utiliserons donc en priorité le *registry* Amazon S3 pour télécharger le contenu des fichiers `README` nécessaires, puis le site web https://archive.softwareheritage.org/ pour tous ceux que nous ne parviendrons pas à y trouver.

Une fois le contenu des `README` téléchargés, nous y cherchons une section dont le nom contient le mot « *contributing* » ou un mot proche. Pour identifier les noms de section dans les fichiers téléchar-

1. https://registry.opendata.aws/software-heritage

gés, nous supposons que ceux-ci sont formatés suivant un des deux standards les plus répandus de formatage de text brut : le Markdown (dont l'extension courante est `.md`) ou le reStructuredText (dont l'extension courante est `.rst`). Le format Markdown autorise deux façon de définir une section [2], la première consiste à démarrer une ligne avec un ou plusieurs caractères « croisillon » (« # ») puis d'écrire le nom de la section sur la fin de la ligne, l'autre consiste à écrire directement le nom de la section, puis à la « souligner » en écrivant plusieurs fois le caractère « égal » (« = ») ou « tiret » (« - ») sur la ligne suivante. Le format reStructuredText n'autorise qu'une seule façon de définir des sections, en soulignant une ligne de la même façon que la deuxième méthode du Markdown, à ceci près que d'autres caractères sont aussi acceptés pour composer la ligne [3].

Pour déterminer si un fichier `README` contient des instructions de contribution, nous identifions donc tous les noms de section le composant et vérifions si l'un d'eux contient le mot « *contributing* » ou assimilé.

Voir aussi l'annexe A.2 pour plus de détails.

3.7. Reproduction des travaux

Afin d'améliorer la transparence méthodologique de ce mémoire et d'aider l'éventuelle reproduction des résultats qui y sont présentés, un *replication package* contenant le code source et les bibliothèques utilisés dans ce mémoire, ainsi que les données brut collectées, a été publié sur Zenodo (Hervot, 2022).

2. voir https://www.markdownguide.org/basic-syntax/#headings
3. voir https://docutils.sourceforge.io/docs/ref/rst/restructuredtext.html#sections

4. Résultats et discussion

L'analyse du graphe de Software Heritage a permis de déterminer les quatre variables de recherche pour 60 966 projets distincts dont les historique de développement sont eux aussi entièrement distincts deux à deux (aucun projet analysé n'est un *fork* d'un autre).

4.1. Forme et distribution des données collectées

Table 4.1. – Aperçu statistique des données collectées

	hasContrib
count	60966
unique	2
top	no
freq	52394

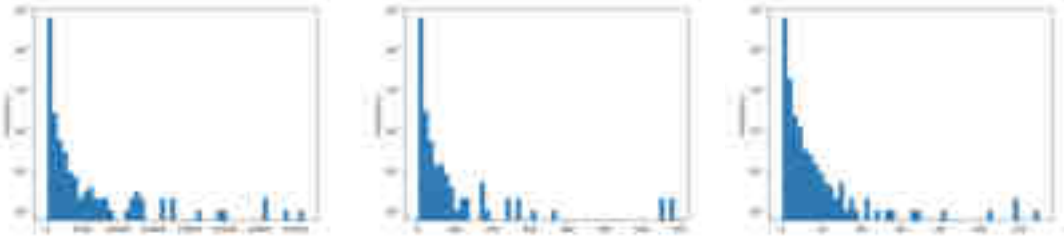
	recentCommitCount
count	60966.000000
mean	61.791474
std	408.956494
min	2.000000
25%	8.000000
50%	19.000000
75%	49.000000
max	36176.000000

	recentContributorCount
count	60966.000000
mean	3.246268
std	6.983897
min	2.000000
25%	2.000000
50%	2.000000
75%	3.000000
max	688.000000

	newContributorCount
count	60966.000000
mean	0.522209
std	1.663200
min	0.000000
25%	0.000000
50%	0.000000
75%	1.000000
max	130.000000

Un aperçu initial des données collectées (table 4.1) révèle quelques propriétés intéressantes de la population étudiée. Les projets possédant des instructions de contribution sont significativement moins nombreux (8 572, soit environ 14% des projets) que ceux n'en possédant pas. L'écart entre la moyenne et la médiane du nombre de *commit*s récents, ainsi que sont écart type, indiquent une forte variation de ce nombre au sein de la population étudiée, mais aussi la présence de quelques individus extrêmes, avec un maximum à 36 176 *commit*s récents observés dans un seul projet. Cette dernière observation se retrouve aussi dans le nombre de contributeurs récents, bien que de façon moins spectaculaire. Rappelons à la vue des quantiles de cette dernière valeur que les projets ayant vu moins de deux contributeurs distincts récents ont été exclus de l'étude (voir section 3.4 p. 21), ce qui explique que la valeur minimale soit de deux.

Une visualisation de la distribution de la population selon chaque variable numérique (figure 4.1) semble donner une forme d'exponentielle décroissante pour chacune d'elles, ce qui signifie par exemple que les projets ayant peu de nouveaux contributeurs sont les plus courants, alors que ceux en ayant plus deviennent rapidement beaucoup plus rare à mesure que le nombre augmente. Les

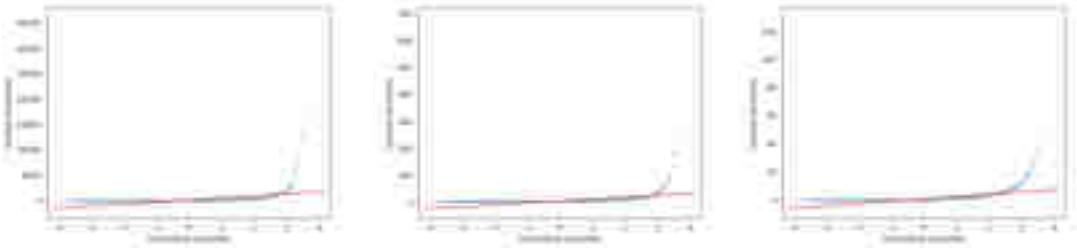


(a) Nombre de *commit*s récents (ordonnées logarithmiques)

(b) Nombre de contributeurs récents (ordonnées logarithmiques)

(c) Nombre de nouveaux contributeurs (ordonnées logarithmiques)

Figure 4.1. – Distribution des individus selon chaque variable numérique



(a) Nombre de *commit*s récents

(b) Nombre de contributeurs récents

(c) Nombre de nouveaux contributeurs

Figure 4.2. – Diagrammes quantile-quantile pour chaque valeur numérique

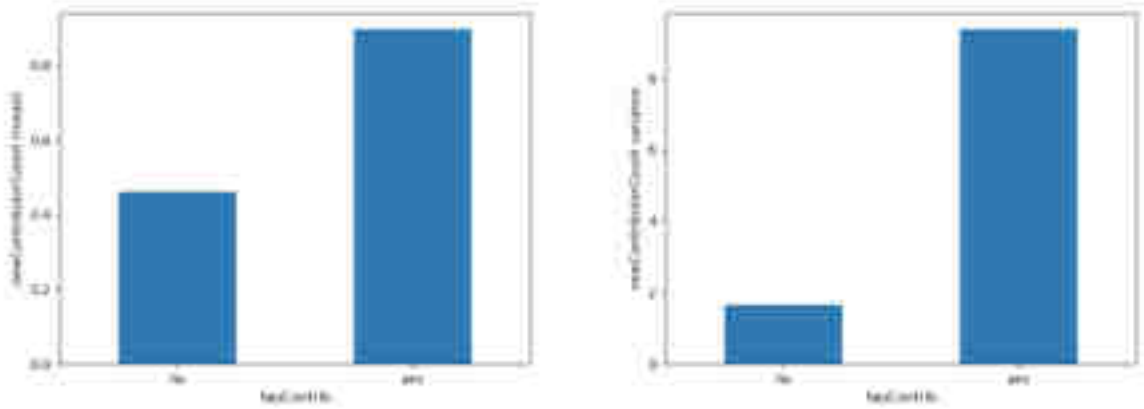
diagrammes quantile-quantile (figure 4.2) de ces variables confirment de façon un peu plus formelle que leur distribution ne suit pas une loi normale, ce qui sera important pour l'interprétation des modèles obtenus plus loin dans cette analyse.

4.2. Lien entre la présence d'instructions de contribution et le nombre de nouveaux contributeurs

La figure 4.3 montre la moyenne du nombre de nouveaux contributeurs au sein des projets possédant des instructions de contribution d'une part, et ceux n'en possédant pas d'autre part.

Ces deux catégories de projets ont été comparées avec un test de Wilcoxon-Mann-Whitney. Celui-ci donne une taille d'effet $\rho \approx 0.57$, ce qui signifie en langage courant que si l'on choisi au hasard un projet A possédant des instructions de contribution et un projet B n'en possédant pas, il y a environ 57% de chances que le projet A ait vu un plus grand nombre de nouveaux contributeurs durant la période étudiée que le projet B. Le test confirme en outre avec un degré de confiance quasi certain ($p \approx 0$) que la distribution des valeurs au sein de ces deux catégories (projets avec instructions de contribution ou sans) est bien différente. Le test ayant été réalisé avec l'hypothèse que les projets « avec instructions de contribution » ont un meilleur score que les autres [1], nous pouvons formellement affirmer que cette différence est à l'avantage des projets possédant des instructions de contribution, ce qui valide l'hypothèse H1.

1. hypothèse plus spécifique que la version « *two-tailed* » du test.



(a) Moyenne du nombre de nouveaux contributeurs pour chaque catégorie

(b) Variance du nombre de nouveaux contributeurs pour chaque catégorie

Test de Wilcoxon-Mann-Whitney : $U = 2.549736 \times 10^8$ ($p = 4.4591222 \times 10^{-135}$, $\rho = 0.56771648$)

Figure 4.3. – Moyenne du nombre de nouveaux contributeurs pour chaque catégorie

Discussion

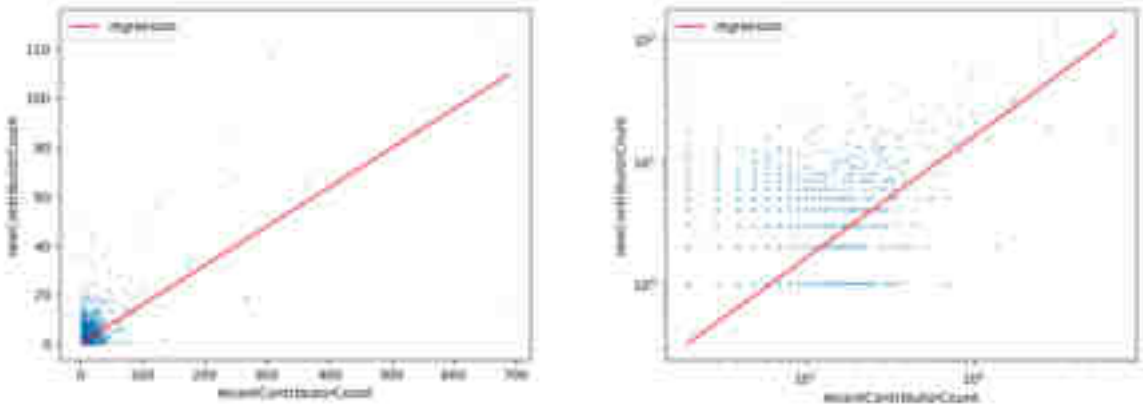
Ce résultat est cohérent avec ceux de Qiu *et al.* (2019, p. 11) qui avaient déterminé que la présence d'instructions de contribution au sein d'un projet était un signal important dans le processus de décision d'un contributeur potentiel qui cherche un projet auquel contribuer. De futures recherches pourraient essayer de déterminer si ce plus grand nombre de nouveaux contributeurs *ayant réussi* à contribuer (observé dans notre étude) est dû uniquement à ce plus grand nombre de nouveaux contributeurs *essayant* de contribuer, ou si la présence d'instructions de contribution a un réel rôle dans le succès d'une tentative de contribution, en plus d'être un signal positif attirant plus de contributeurs potentiels.

Il est tout de même important de noter que, bien que le test de Wilcoxon-Mann-Whitney ne soit pas paramétrique et puisse donc s'appliquer à ces données (contrairement à un test de Student nécessitant une distribution normale, par exemple), sa robustesse diminue significativement lorsque la distribution des données ne suit pas une loi normale *et* que les groupes comparés ont une variance significativement différente (Zimmerman, 1998), ce qui est le cas ici (voir figures 4.2c et 4.3b). Ce point, ainsi que la petite taille d'effet obtenue, justifieraient une attention plus particulière à la préparation des données afin d'approcher une distribution normale et d'en homogénéiser la variance.

4.3. Lien entre le nombre de contributeurs récents et le nombre de nouveaux contributeurs

S'agissant ici de comparer deux valeurs numériques dont l'éventuelle relation n'est pas connue, le modèle choisi est celui de la régression linéaire, visualisé en figure 4.4.

Un premier modèle utilisant la méthode des moindres carrés ordinaires a été calculé, puis soumis à un test d'homoscédasticité de White ayant conclu, à l'inverse, à une forte hétéroscédasticité des données de la régression ($p = 0$). Cette hétéroscédasticité signifie que les données ont une variance



(a) Échelle linéaire **(b)** Échelle logarithmique

$$newContributorCount = recentContributorCount \times 0.15949774 + 0.0044366175$$
$$(R^2 = 0.44855516)$$
Test d'homoscédasticité de White : $LM = 16280.789$ $(p = 0)$

Figure 4.4. – Nombre de nouveaux contributeurs en fonction du nombre de contributeurs récents uniques

fortement hétérogène au sein du problème, ce qui implique que la méthode OLS n'est pas la plus statistiquement efficace pour modéliser la relation entre les deux variables (Taboga, 2021).

Le modèle retenu a donc finalement été calculé par la méthode des moindres carrés généralisée [2], il suggère effectivement que plus le nombre de contributeurs récents d'un projet est élevé, plus son nombre de nouveaux contributeurs l'est aussi. Le coefficient de détermination $R^2 \approx 0.45$ du modèle indique que le nombre de contributeurs récents explique environ 45% de la variation du nombre de nouveaux contributeurs, ce qui valide l'hypothèse H2, avec une taille d'effet tout de même modérée, voir faible.

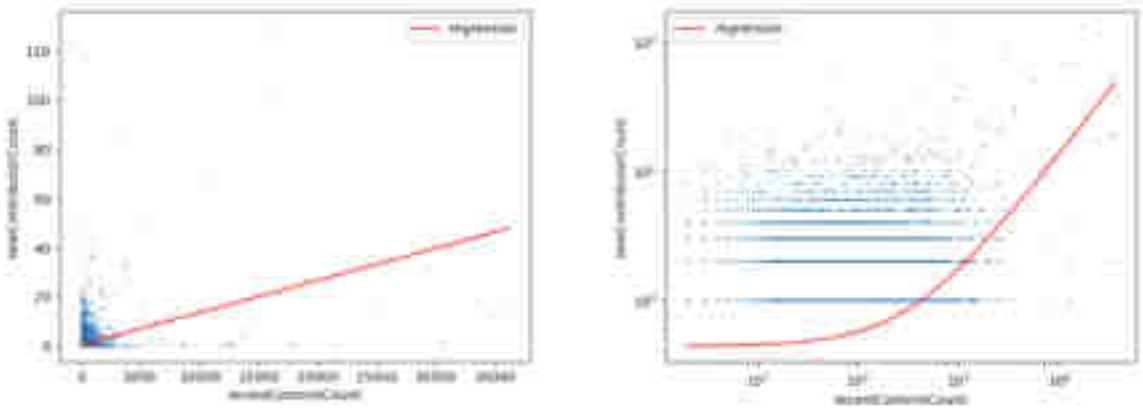
Discussion

Ce résultat est aussi cohérent avec ceux de Qiu *et al.* (2019, p. 12-13, 16) qui observaient que le nombre de contributeurs uniques récents était positivement corrélé au nombre de *tentatives* de contribution faites par de nouveaux contributeurs (mesuré par le nombre de *pull requests*). Là aussi de futures recherches pourraient s'intéresser à la causalité de cette relation. Tout ce que notre étude peut affirmer sur un éventuel lien de causalité est que, le nombre de contributeurs récents étant mesuré sur une période antécédente à celle sur laquelle est mesuré le nombre de nouveaux contributeurs, cette deuxième variable ne peut pas influencer la première, du moins tel que celles-ci ont été définies ici. Il reste à déterminer si une chaîne causale lie directement ces deux variables (si oui, laquelle) ou si elles ne sont liées que par un ancêtre causal commun (et si oui, lequel).

2. Modèle `GLS` du paquet Python `statsmodels` : https://www.statsmodels.org/dev/generated/statsmodels.regression.linear_model.GLS.html

4.4. Lien entre le nombre de *commits* récents et le nombre de nouveaux contributeurs

Suivant la même approche que pour l'analyse du nombre de contributeurs récents, une première modélisation par régression linéaire suivant la méthode des moindres carrés ordinaires a d'abord été faite. Un test d'homoscédasticité de White a, là aussi, conclu à une forte hétéroscédasticité des données, c'est donc une régression linéaire utilisant la méthode des moindres carrés généralisée qui a été retenue (figure 4.5).



(a) Échelle linéaire **(b)** Échelle logarithmique

$$newContributorCount = recentCommitCount \times 0.0013121815 + 0.44112747$$
$$(R^2 = 0.10410059)$$
Test d'homoscédasticité de White : $LM = 8150.885$ $(p = 0)$

Figure 4.5. – Nombre de nouveaux contributeurs en fonction du nombre de *commit*s récents

Le modèle suggère ici aussi une corrélation positive entre le nombre de *commit*s récents d'un projet et son nombre de nouveaux contributeurs, mais son coefficient de détermination est trop faible ($R^2 \approx 0.10$) pour considérer que le modèle représente fidèlement des données du problème. Nous ne pouvons donc conclure à une relation entre ces deux variables et devons rejeter l'hypothèse H3.

Discussion

Ce résultat ne rejoint cette fois pas ceux de Qiu *et al.* (2019, p. 13, 16), qui avaient là encore trouvé une corrélation positive entre le nombre de *commit*s récents d'un projets et le nombre de tentatives de contribution faites par de nouveaux contributeurs. Cela pourrait potentiellement signifier que le nombre de *commit*s récents d'un projet est un signal que les nouveaux contributeurs potentiels utilisent *à tort* pour choisir un projet auquel essayer de contribuer. Nos résultats ne permettent pas de conclure qu'un grand nombre de *commit*s récents, bien qu'il attire les nouveaux contributeurs, est un indicateur de l'accessibilité du projet pour ces nouveaux contributeurs.

Conclusion

Conclusion

Les problématiques liées au numérique prennent de plus en plus d'importance dans notre monde, la question du logiciel libre devient de ce fait un enjeu de société déterminant pour la transparence des technologies, l'ouverture de l'information et la collaboration international. Rendre disponible publiquement et gratuitement le code source et son processus de développement ne suffit cependant pas à le rendre réellement accessible, de nombreuses barrières existent pour les personnes souhaitant se familiariser avec les projets de logiciel libre et y contribuer. Ces barrières représentent même un frein pour les enseignants souhaitant introduire leurs étudiants à ce milieu, même lorsque ces enseignants lui sont eux-mêmes déjà familiers.

Dans ce mémoire, nous avons commencé la construction d'un outil permettant aux enseignants de répondre à l'un de ces freins : la sélection de projets de logiciel libre réels susceptibles d'être un bon support de travaux pratiques. Ce travail préliminaire se base sur une littérature florissante d'identification des barrières aux contribution dans le logiciel libre, sa méthodologie s'inscrit quant à elle dans le domaine de l'analyse automatique des dépôts logiciels (*Mining Software Repositories*).

Par l'analyse de l'archive de Software Heritage, l'un des corpus de développement logiciel les plus représentatifs et exploitables dans un contexte de recherche, nous avons tenté de déterminer si trois signaux facilement observables pour un enseignant ou un nouveau contributeur sont prédictifs de la capacité d'un projet à suffisamment accompagner les nouveaux contributeurs pour leur permettre de mener leur première contribution à terme.

Nos résultats suggèrent que la présence d'instructions de contribution au sein d'un projet (typiquement au travers d'un fichier « `CONTRIBUTING.md` »), ainsi que le nombre de contributeurs uniques ayant récemment contribué au projet, sont positivement corrélés au nombre de *nouveaux* contributeurs étant parvenus à ajouter leur pierre à l'édifice. Nous n'avons en revanche pas trouvé de corrélation, positive ou négative, permettant d'affirmer que le nombre de *commit*s récents au sein d'un projet était prédictif du nombre de nouveaux contributeurs allant au bout de leur contribution.

Quelques précautions mériteraient cependant d'être prises dans l'interprétation de ces résultats. La limite la plus importante de ce travail est l'absence d'information permettant d'inférer un quelconque lien de causalité entre les éléments étudiés. Rien dans nos résultats ne nous permet par exemple de dire que la présence d'instructions de contribution *améliore* l'accessibilité d'un projet de logiciel libre pour les nouveaux contributeurs, la corrélation que nous observons entre ces deux variables pourrait s'expliquer par le fait que les mainteneurs d'un projet qui ont tendance à efficacement guider les nouveaux contributeurs ont aussi tendance à écrire des instructions de contribution. Par ailleurs, plusieurs propriétés statistiques des données collectées rendent difficile leur analyse. Nous avons tenté traiter ces difficultés au mieux de nos capacités, mais une étude et une préparation plus rigoureuse des données pourraient, au mieux, améliorer la fiabilité des conclusions que nous en avons tiré, ou au pire, les invalider.

Les perspectives de recherche concernant l'accessibilité des projets de logiciel libre sont encore nombreuses, le sujet étant à la fois très changeant et relativement nouveau, la littérature existe

mais est encore très qualitative. De futures études quantitatives pourraient s'intéresser aux éventuels mécanismes causaux responsables de l'accessibilité observée des projets. Le domaine de l'analyse automatique des dépôts logiciels pose encore beaucoup de questions intéressantes, l'archive de Software Heritage par exemple apporte une nouvelle méthode d'observation particulièrement puissante, mais celle-ci est encore jeune et peu d'étude se consacrent aux bénéfices et limites de ce qu'elle peut apporter à la science. Enfin, si nos résultats suggèrent que la présence d'instructions de contribution ainsi que le nombre de contributeurs uniques récents sont de bons indicateurs à observer pour un enseignant souhaitant trouver un projet sur lequel faire travailler ses étudiants, de futures recherches mériteraient d'être conduites afin de déterminer si cette accessibilité se traduit effectivement dans la qualité de l'apprentissage, ou si le contexte est trop différent pour que les étudiants soient réduit à la catégorie générique des « nouveaux contributeurs ».

Annexes

A. Détails techniques de la collecte et de l'analyse des données

A.1. Collecte initiale depuis le graphe de Software Heritage

Le code de la collecte initiale prend la forme d'une classe Java utilisant la bibliothèque `swh-graph` [1] et est disponible sur le dépôt GitHub de ce mémoire au lien suivant : https://github.com/Dettorer/synva-dissertation/blob/main/experiment/data_collection/CollectData.java, ainsi que dans son « *replication package* » (Hervot, 2022).

La version de `swh-graph` utilisée est cependant une version modifiée pour rendre la plupart des fonctions d'accès aux données du graphe « *thread safe* », c'est à dire utilisables dans un contexte parallélisé sans besoin de synchronisation des fils d'exécution pour éviter les appels concurrents. Cette version de la bibliothèque permet l'implémentation d'algorithmes parallélisés beaucoup plus rapides, car utilisant beaucoup moins de points de synchronisation, elle est disponible dans le *replication package* sous la forme d'un fichier `swh-graph-1.0.1.jar`.

Dans un premier temps, la fonction `discoverProject` est appelée sur tous les nœuds de type `ORI` (point de départ de l'archivage d'un projet par Software Heritage), cette fonction identifie tous les projets étant des *fork* de celui-ci via un double parcours largeur sur la composante connexe du nœud de départ formée par le sous-graphe des nœuds `REV` (*revision*, terme générique pour les *commit*s) et `SNP` (*snapshot*, le point d'entrée d'un archivage du projet). Cette détection se fait en deux parcours largeur au lieu d'un seul afin de calculer en même temps la taille de la plus longue chaîne de *commit*s accessible depuis chaque nœud `ORI` de la composante connexe (donc depuis chaque *fork* du projet initial). Le premier parcours remonte les ancêtres du nœud de départ pour trouver les révisions racines (les « *initial* commits ») du projet, puis un deuxième parcours est lancé dans l'autre sens avec des marqueurs de niveau depuis chacune de ces révisions racines afin de trouver tous les nœuds `ORI` qui peuvent les atteindre et sont donc des *fork*s les uns des autres. Pour chaque composante connexe, seul un nœud `SNP` est retenu pour l'analyse en deuxième étape : celui étant le point de départ de la plus longue chaîne de *commit*s possible, donc ayant le plus de données exploitables. Cette partie du code étant parallélisée, il est possible qu'à un instant donné, plusieurs fils d'exécution soient en train d'explorer la même composante connexe (en y étant entrés par différents nœuds `ORI`). La sélection du représentant de la composante est en revanche déterministe, les différents fils d'exécution sélectionneront donc systématiquement le même projet pour une même composante. Afin d'éviter la duplication des projets retenus liée à cette parallélisation, ceux-ci sont donc simplement stockés dans une structure d'ensemble dont l'API garanti l'unicité de ses éléments (en l'occurrence le conteneur `LongOpenHashSet`).

Dans un deuxième temps, la fonction `collectProject` est appelée sur chaque projet retenu afin d'en extraire les données de recherche. Cette fonction commence par identifier la branche ayant le plus de chance d'être la branche principale du projet (voir la table de priorité `mainBranchScore`), puis démarre un parcours largeur à partir de cette branche dans lequel elle compte :

1. https://docs.softwareheritage.org/devel/swh-graph/java-api.html

— le nombre de contributeurs pendant la période de référence n'ayant jamais contribué dans ce projet avant (variable expliquée) ;
— le nombre de contributeurs uniques lors des six mois précédant la période de référence (variable explicative de hypothèse H2) ;
— le nombre de *commit*s enregistrés lors des même six mois précédant la période de référence (variable explicative de hypothèse H3).

La présence d'instructions de contribution (variable explicative de l'hypothèse H1) est plus compliquée à vérifier. Le graphe possédant le nom et la hiérarchie des fichiers disponibles à chaque *commit*, mais pas leur contenu, l'analyse se contente de vérifier si un fichier dont le nom est une variante du classique `CONTRIBUTING.md` existe. Si ce fichier existe, l'analyse conclue que le projet possède effectivement des instructions de contribution, sinon, elle vérifie la présence d'un fichier dont le nom est une variante du classique `README.md`. Si ce fichier existe, l'analyse conclue que le projet possède *peut être* des instructions de contribution et construit l'URL à laquelle un traitement ultérieur pourra télécharger le contenu du fichier `README` et y confirmer ou non la présence d'instructions de contribution. Si aucun de ces deux types de fichier n'existe, l'analyse conclue que le projet ne possède pas d'instructions de contribution.

Enfin, les données collectées sont affichées sur la sortie standard sous la forme d'un fichier CSV.

A.2. Collecte complémentaire : analyse des fichiers `README`

Le code de cette collecte complémentaire prend la forme d'un script Python utilisant plusieurs bibliothèques comme `requests` pour les requêtes sur https://archive.softwareheritage.org/, `boto3` pour les requêtes Amazon S3, ou `charset_normalizer` pour le décodage des fichiers brut. Le script complet est disponible sur le dépôt GitHub de ce mémoire au lien suivant : https://github.com/Dettorer/synva-dissertation/blob/main/experiment/data_collection/check_readme_contents.py ainsi que dans son « *replication package* » (Hervot, 2022).

Une difficulté de la récolte du contenu des fichiers `README` est une incohérence dans le type d'identificateur utilisé sur https://archive.softwareheritage.org/ et le *registry* Amazon S3. Pour télécharger le contenu d'un fichier depuis le site web, il faut l'identifier en utilisant son hachage cryptographique `sha1_git`, qui est celui renseigné dans le graphe. Le *registry* Amazon S3, en revanche, identifie les contenus en utilisant un hachage différent appelé `sha1`. Pour pouvoir télécharger le contenu d'un fichier sur Amazon S3 à partir de données collectées dans le graphe, il faut donc traduire le `sha1_git` obtenu en un `sha1`, ce qui est impossible si l'on ne possède pour seule information que le `sha1_git`. Pour contourner le problème, une table de correspondance des deux méthodes de hachage peut être construite préalablement à partir de la représentation Apache ORC du graphe [2], qui contient les deux types de hash.

Une table de correspondance permettant de traduire tous les `sha1_git` des fichiers README strictement nécessaire à l'étude de ce mémoire n'étant pas déjà disponibles sur le *registry* Amazon S3 se trouve dans le *replication package*, elle se présente sous la forme d'un fichier CSV appelé `sha1_git_to_sha1.csv`.

2. voir https://docs.softwareheritage.org/devel/swh-dataset/graph/dataset.html

A.3. Analyse des résultats

Le code utilisé pour produire les analyses du chapitre 4 prend la forme d'un script Python utilisant plusieurs bibliothèques de traitement des données et d'analyse statistique comme `pandas`, `numpy`, `scipy`, `statsmodels`, ainsi que `matplotlib` pour la production des visualisations. Le script complet est disponible sur le dépôt GitHub de ce mémoire au lien suivant : https://github.com/Dettorer/synva-dissertation/blob/main/experiment/data_analysis/data_analysis.py ainsi que dans son « *replication package* » (Hervot, 2022). Ce dernier contient de plus un fichier `2022-08-18_-completed.csv` contenant les données collectées et complétées lors de la phase précédente, il s'agit du fichier exact utilisé pour la présentation des résultats du chapitre 4.

Bibliographie

Références utilisées, présentées par ordre de première citation.

Chen, C.-H., & Yang, Y.-C. (2019). Revisiting the effects of project-based learning on students' academic achievement: A meta-analysis investigating moderators. *Educational Research Review*, *26*, 71–81. https://doi.org/https://doi.org/10.1016/j.edurev.2018.11.001 (cf. p. 9)

Balemen, N., & Keskin, M. Ö. (2018). The effectiveness of project-based learning on science education: A meta-analysis search. *International Online Journal of Education and Teaching*, *5*(4), 849–865 (cf. p. 9).

Detmer, R., Li, C., Dong, Z., & Hankins, J. (2010). Incorporating real-world projects in teaching computer science courses. *Proceedings of the 48th Annual Southeast Regional Conference*. https://doi.org/10.1145/1900008.1900042 (cf. p. 9)

Buckley, M., Kershner, H., Schindler, K., Alphonce, C., & Braswell, J. (2004). Benefits of using socially-relevant projects in computer science and engineering education. *SIGCSE Bull.*, *36*(1), 482–486. https://doi.org/10.1145/1028174.971463 (cf. p. 9)

Meneely, A., Williams, L., & Gehringer, E. F. (2008). Rose: A repository of education-friendly open-source projects. *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education*, 7–11. https://doi.org/10.1145/1384271.1384276 (cf. p. 9, 10)

Gehringer, E. (2007). Open source for homework: Real projects, peer reviewed. *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education, Covington, KY* (cf. p. 10).

Kim, S., Yoo, J., & Lee, M. (2012). Step-by-step strategies and case studies for embedded software companies to adapt to the foss ecosystem. In I. Hammouda, B. Lundell, T. Mikkonen, & W. Scacchi (Eds.), *Open source systems: Long-term sustainability* (pp. 48–60). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-33442-9_4. (Cf. p. 12)

Lerner, J., & Tirole, J. (2002). Some simple economics of open source. *The Journal of Industrial Economics*, *50*(2), 197–234. https://doi.org/https://doi.org/10.1111/1467-6451.00174 (cf. p. 12)

Oreg, S., & Nov, O. (2008). Exploring motivations for contributing to open source initiatives: The roles of contribution context and personal values [Including the Special Issue: Internet Empowerment]. *Computers in Human Behavior*, *24*(5), 2055–2073. https://doi.org/https://doi.org/10.1016/j.chb.2007.09.007 (cf. p. 12)

Mendez, C., Padala, H. S., Steine-Hanson, Z., Hilderbrand, C., Horvath, A., Hill, C., Simpson, L., Patil, N., Sarma, A., & Burnett, M. (2018). Open source barriers to entry, revisited: A sociotechnical perspective. *Proceedings of the 40th International Conference on Software Engineering*, 1004–1015. https://doi.org/10.1145/3180155.3180241 (cf. p. 13, 17)

Steinmacher, I., Wiese, I., Chaves, A. P., & Gerosa, M. A. (2013). Why do newcomers abandon open source software projects? *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 25–32. https://doi.org/10.1109/CHASE.2013.6614728 (cf. p. 13)

Darley, W. K., & Smith, R. E. (1995). Gender differences in information processing strategies: An empirical test of the selectivity model in advertising response. *Journal of Advertising*, *24*(1), 41–56. https://doi.org/10.1080/00913367.1995.10673467 (cf. p. 13)

Meyers-Levy, J., & Loken, B. (2015). Revisiting gender differences: What we know and what lies ahead. *Journal of Consumer Psychology*, *25*(1), 129–149. https://doi.org/https://doi.org/10.1016/j.jcps.2014.06.003 (cf. p. 13)

Burnett, M., Fleming, S. D., Iqbal, S., Venolia, G., Rajaram, V., Farooq, U., Grigoreanu, V., & Czerwinski, M. (2010). Gender differences and programming environments: Across programming populations. *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. https://doi.org/10.1145/1852786.1852824 (cf. p. 13)

Hou, W., Kaur, M., Komlodi, A., Lutters, W. G., Boot, L., Cotten, S. R., Morrell, C., Ozok, A. A., & Tufekci, Z. (2006). "girls don't waste time": Pre-adolescent attitudes toward ict. *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, 875–880. https://doi.org/10.1145/1125451.1125622 (cf. p. 13)

Horwitz, S. K., & Horwitz, I. B. (2007). The effects of team diversity on team outcomes: A meta-analytic review of team demography. *Journal of Management*, *33*(6), 987–1015. https://doi.org/10.1177/0149206307308587 (cf. p. 14)

Vasilescu, B., Posnett, D., Ray, B., van den Brand, M. G., Serebrenik, A., Devanbu, P., & Filkov, V. (2015). Gender and tenure diversity in github teams. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, 3789–3798. https://doi.org/10.1145/2702123.2702549 (cf. p. 14)

Steinmacher, I., Graciotto Silva, M. A., Gerosa, M. A., & Redmiles, D. F. (2015). A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology*, *59*, 67–85. https://doi.org/https://doi.org/10.1016/j.infsof.2014.11.001 (cf. p. 14, 17)

Steinmacher, I., Conte, T., Gerosa, M. A., & Redmiles, D. (2015). Social barriers faced by newcomers placing their first contribution in open source software projects. *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, 1379–1392. https://doi.org/10.1145/2675133.2675215 (cf. p. 14)

Steinmacher, I., Conte, T. U., & Gerosa, M. A. (2015). Understanding and supporting the choice of an appropriate task to start with in open source software communities. *2015 48th Hawaii International Conference on System Sciences*, 5299–5308. https://doi.org/10.1109/HICSS.2015.624 (cf. p. 14)

Steinmacher, I., Conte, T. U., Treude, C., & Gerosa, M. A. (2016). Overcoming open source project entry barriers with a portal for newcomers. *Proceedings of the 38th International Conference on Software Engineering*, 273–284. https://doi.org/10.1145/2884781.2884806 (cf. p. 14, 15)

Stanik, C., Montgomery, L., Martens, D., Fucci, D., & Maalej, W. (2018). A simple nlp-based approach to support onboarding and retention in open source communities. *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 172–182. https://doi.org/10.1109/ICSME.2018.00027 (cf. p. 15)

Chen, G. (2005). Newcomer adaptation in teams: Multilevel antecedents and outcomes. *Academy of Management Journal*, *48*(1), 101–116. https://doi.org/10.5465/amj.2005.15993147 (cf. p. 15)

Lerner, J., Pathak, P. A., & Tirole, J. (2006). The dynamics of open-source contributors. *American Economic Review*, *96*(2), 114–118. https://doi.org/10.1257/000282806777211874 (cf. p. 15)

Qiu, H. S., Li, Y. L., Padala, S., Sarma, A., & Vasilescu, B. (2019). The signals that potential contributors look for when choosing open-source projects. *Proc. ACM Hum.-Comput. Interact.*, *3*(CSCW). https://doi.org/10.1145/3359224 (cf. p. 15, 20, 21, 26–28)

Cosentino, V., Cánovas Izquierdo, J. L., & Cabot, J. (2017). A systematic mapping study of software development with github. *IEEE Access*, *5*, 7173–7192. https://doi.org/10.1109/ACCESS.2017.2682323 (cf. p. 15)

Rousseau, G., Di Cosmo, R., & Zacchiroli, S. (2019). *Growth and Duplication of Public Source Code over Time: Provenance Tracking at Scale* [working paper or preprint]. (Cf. p. 15–17).

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., & Damian, D. (2014). The promises and perils of mining github. *Proceedings of the 11th Working Conference on Mining Software Repositories*, 92–101. https://doi.org/10.1145/2597073.2597074 (cf. p. 16, 17, 20, 21)

Trujillo, M. Z., Hébert-Dufresne, L., & Bagrow, J. (2022). The penumbra of open source: Projects outside of centralized platforms are longer maintained, more academic and more collaborative. *EPJ Data Science*, *11*(1), 31 (cf. p. 16, 20).

Boldi, P., Pietri, A., Vigna, S., & Zacchiroli, S. (2020). Ultra-large-scale repository analysis via graph compression. *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 184–194. https://doi.org/10.1109/SANER48275.2020.9054827 (cf. p. 16)

Di Cosmo, R., & Zacchiroli, S. (2017). Software Heritage: Why and How to Preserve Software Source Code. *iPRES 2017 - 14th International Conference on Digital Preservation*, 1–10 (cf. p. 16).

Pietri, A., Spinellis, D., & Zacchiroli, S. (2019). The software heritage graph dataset: Public software development under one roof. *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 138–142. https://doi.org/10.1109/MSR.2019.00030 (cf. p. 17)

Steinmacher, I., Chaves, A. P., Conte, T. U., & Gerosa, M. A. (2014). Preliminary empirical identification of barriers faced by newcomers to open source software projects. *2014 Brazilian Symposium on Software Engineering*, 51–60. https://doi.org/10.1109/SBES.2014.9 (cf. p. 17)

Burnett, M., Stumpf, S., Macbeth, J., Makri, S., Beckwith, L., Kwan, I., Peters, A., & Jernigan, W. (2016). GenderMag: A Method for Evaluating Software's Gender Inclusiveness. *Interacting with Computers*, *28*(6), 760–787. https://doi.org/10.1093/iwc/iwv046 (cf. p. 17)

Spencer, R. (2000). The streamlined cognitive walkthrough method, working around social constraints encountered in a software development company. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 353–359. https://doi.org/10.1145/332040.332456 (cf. p. 17)

Hervot, P. (2022). Replication package: L'identification des projets de logiciel libre accessibles aux nouveaux contributeurs. https://doi.org/10.5281/zenodo.7023495. (Cf. p. 23, 33-35)

Zimmerman, D. W. (1998). Invalidation of parametric and nonparametric statistical tests by concurrent violation of two assumptions. *The Journal of Experimental Education*, *67*(1), 55–68. https://doi.org/10.1080/00220979809598344 (cf. p. 26)

Taboga, M. (2021). *"generalized least squares", lectures on probability theory and mathematical statistics*. https://www.statlect.com/fundamentals-of-statistics/generalized-least-squares. (Cf. p. 27)

Glossaire

API
: Acronyme pour « *Application Programming Interface* », les *APIs* sont des collections d'outils logiciel permettant à un programme d'interagir avec un service logiciel sans utiliser l'interface utilisateur du service en question, on parle donc d'interface logicielle. Par exemple, l'*API* du site GitHub (détaillée ici : https://docs.github.com/en/rest) permet aux développeurs et chercheurs d'écrire des programmes collectant automatiquement des données sur la plateforme (comme le nom des projets hébergés, le nombre de *pull request*s d'un projet spécifique, les auteurs de *commit*s, etc.), cela sans utiliser de navigateur web. 15

bug tracker
: Outil se présentant généralement sous la forme d'un site web (ou d'une partie d'un site web) servant initialement à répertorier les *bugs* connus d'un projet de logiciel libre, à organiser les discussions autour de la résolution de ces bugs et à suivre la progression de leurs correctifs. L'usage de cet outil s'est aujourd'hui élargi et sert en plus à organiser et suivre de la même façon les discussions et changements du code liés au développement de nouvelles fonctionnalités, à la réorganisation du code, aux réflexions plus génériques sur la conception du projet, voir parfois le support utilisateur. 20, 40

clone
: En développement logiciel, voir *fork*. 16

commit
: Une contribution au code. Un *commit* est l'enregistrement d'un petit ensemble de modifications (suppression, ajout ou modification de une ou plusieurs lignes, dans un ou plusieurs fichiers) apporté au code source d'un projet. 6, 15, 16, 18, 20–22, 24, 25, 28, 30, 33, 34, 39–41

dépôt
: Voir Git. 40

EPITA
: https://www.epita.fr/
École d'ingénieurs privée, acronyme de « École pour l'Informatique et les Techniques Avancées ». 8

fork
: En développement logiciel, un *fork* est un projet dont le début de l'historique de développement est confondu avec celui d'un autre projet. Il s'agit en quelque sorte d'une version alternative d'un projet existant, devenue autonome par la suite et pouvant avoir sa propre équipe de développement totalement distincte, jusqu'à éventuellement (mais pas obligatoirement) n'avoir plus rien à voir avec le projet d'origine. Un exemple célèbre de ce phénomène est LibreOffice [1], un logiciel libre de bureautique ayant démarré en se basant sur le code source d'OpenOffice [2] et étant maintenant un projet autonome avec une équipe de développement différente. 16, 21, 22, 33, 39

1. https://www.libreoffice.org/
2. https://openoffice.apache.org/

FOSDEM

https://fosdem.org

Acronyme de « *Free and Open Source Developers' European Meeting* ». Il s'agit d'une conférence annuelle à l'accès gratuit hébergeant le temps d'un week-end de nombreuses présentations, ateliers et discussions autour du logiciel libre. Elle se déroule généralement à l'Université Libre de Bruxelles (ULB). 8

Git

https://git-scm.com/

Logiciel de gestion de versions (dit « système de versionnement »). Git est un logiciel permettant de sauvegarder l'historique de développement d'un projet (ses *commit*s) et d'organiser les contributions d'un petit ou grand nombre de personnes. Il est pensé pour mais pas limité aux logiciels libre, Git a été conçu à l'origine pour organiser le développement du noyau de système d'exploitation Linux. L'historique d'un projet tel que sauvegardé par Git est communément appelé un « dépôt Git ». 17, 20, 39, 40

GitHub

https://github.com/

Une plateforme d'hébergement de dépôts Git avec *bug tracker* et autres outils annexes, gérée par une entreprise privée. 14, 15, 18, 20, 33–35, 39

Linux

https://www.kernel.org/

Un système d'exploitation rentrant dans la catégorie du logiciel libre. 40

pull request

Une proposition de contribution à un projet. Une *pull request* prend la forme d'un ensemble de *commit*s que des personnes internes ou externes au projet soumettent à la validation des personnes ayant la permission de modifier son code source. Les *pull request* font typiquement l'objet de relectures, commentaires et demandes d'ajustement de la part des mainteneurs du projet et de ses autres contributeurs. 15, 17, 20, 27, 39

SYNVA

https://sfc.unistra.fr/formations/[...]

Master 2 ingénierie des SYstème Numériques Virtuels pour l'Apprentissage. Une formation dispensée par l'université de Strasbourg. 9

Résumé

Ce mémoire entreprend d'identifier quels indicateurs facilement observables par un acteur individuel peuvent servir à estimer l'accessibilité d'un projet de logiciel libre pour les personnes souhaitant y contribuer pour la première fois. L'ubiquité des technologies numériques dans nos vies pose de plus en plus la question de la place du logiciel libre et de son intérêt pour la transparence des logiciels et leur gestion démocratique, pour autant le simple fait de rendre le code source d'un logiciel publique ne le rend pas automatiquement accessible, et les nouveaux contributeurs de ces projets rencontrent de nombreuses barrières d'entrée les empêchant parfois de mener leurs contributions à leur terme. Au travers d'une analyse à grande échelle de l'archive de Software Heritage, nous testons la pertinence de trois indicateurs dans l'identification des projets de logiciel libre accessibles aux nouveaux contributeurs. Nos résultats montrent un lien de corrélation positif entre le nombre de contributions menées à leur terme au sein d'un projet par de nouveaux contributeurs et la présence d'instructions de contribution, ainsi qu'une corrélation positive entre ce même nombre et le nombre de contributeurs uniques récents du projet, ils ne permettent en revanche pas de mettre en évidence un quelconque lien avec le nombre de *commit*s récents. De tels indicateurs trouveraient leur utilité notamment dans l'enseignement des pratiques du logiciel libre, les enseignants de ce sujet ayant souvent du mal à sélectionner des projets accessibles sur lesquels faire travailler les étudiants.

Mots-clés : Logiciel libre, Analyse automatique de dépôts logiciels, Barrières d'entrée du logiciel libre, Nouveaux contributeurs.

Abstract

This dissertation tries to identify which signals easily observable by an individual can help estimate the accessibility of a FOSS project for people who want to contribute to it for the first time. The ubiquity of digital technology in our lives asks the question of the role of Free and Open Source Software in it and its benefits for software transparency and their democratic governance. However, simply making the source code of a software public doesn't automatically make it accessible, and new contributors approaching these projects often face many barriers to entry that prevent them from bringing their contribution to completion. Through a large-scale software mining of the Software Heritage archive, we test the pertinence of three signals in the identification of accessible FOSS projects for new contributors. Our results show a positive correlation between the number of new contributors of a project successfully bringing their contribution to completion and the presence of contributing guidelines, as well as a positive correlation between that same number and the number of recent unique contributors in the project; on the other hand, they show no evidence for any link regarding the number of recent commits. Such signals could find a use in the teaching of FOSS practices, as teachers of this subject often find it difficult to select an accessible project for their students to work on.

Keywords: FOSS, Mining Software Repositories, Open source barriers to entry, New contributors.