École Polytechnique de Montréal

Cours ING6900 Méthodes de recherche

Proposition de recherche

Analyse des *Self-Admitted Technical Debts* et conception d'un outil d'évaluation de la qualité des projets logiciels

Présenté par

Cédric NOISEUX

1588195

Candidat en M.Sc.A.

en génie informatique

sous la direction de Giuliano Antoniol

Département Génie Informatique

Remis à Cevdet Akyel

12 décembre 2016

1. Introduction

1.1. Description du contexte et revue de littérature

Les *Self-Admitted Technical Debts* (SATD) et les *Technical Debts* (TD) sont des solutions temporaires et peu optimales introduites consciemment dans un code par les programmeurs (A. Potdar, E. Shihab, 2014). Différentes motivations justifient ce comportement : le programmeur désire corriger rapidement des bugs dans l'exécution d'un programme, l'équipe se situe dans l'implémentation initiale des composantes, le développeur n'a pas de connaissance profonde du projet, n'est pas très compétent ou est peu expérimenté (G. Suryanarayana, G. Samarthyam, T. Sharma, 2014). La particularité des SATD est qu'elles sont introduites et documentées volontairement par le programmeur contrairement aux TD qui sont introduites mais pas documentées (G. Bavota, B. Russo, 2016). Ces pratiques sont courantes et elles existent car elles sont souvent plus faciles et plus rapides à réaliser qu'une composante introduite de manière réfléchie et logique. Ces dettes affectent les logiciels à différents niveaux et se présentent sous différentes formes, que ce soit en rapport à la conception, aux requis ou aux tests logiciels (E. D. S. Maldonado, E. Shihab, 2015, N. S. Alves, L. F. Ribeiro, V. Caires, T. S. Mendes, R. O. Spínola, 2014). La rapidité de résolution des problèmes, au détriment de la qualité d'un programme, est l'aspect justifiant l'introduction de ces dettes. Cette pratique n'est pas très encouragée car elle ne permet pas la conception d'un produit efficace au niveau rapidité de conception et d'exécution (E. Allman, 2012), maintenabilité et qualité (S. Wehaibi, E. Shihab, L. Guerrouj, 2016, N. Zazworka, M. A. Shaw, F. Shull, C. Seaman, 2011), et coûts (Y. Guo, 2011). De plus, tout comme les dettes économiques, les dettes techniques voient leur impact négatif grandir en importance dans le temps de conception. Il devient donc évident que l'introduction de dettes techniques dans un projet logiciel affecte la qualité globale de celui-ci de manière continue. Malheureusement, le temps requis pour concevoir et coder une solution réfléchie n'est parfois pas compatible avec le cycle de développement ou les attentes des usagers (N. Brown, 2010). Dans ce cas, les développeurs doivent rapidement résoudre les défauts et n'ont donc pas le temps de réaliser des ajustements efficaces. Ceci est inquiétant puisque les SATD peuvent être liés à des défauts sévères ou critiques.

1.2. Problématique

Plusieurs stratégies peuvent être employées pour réduire le nombre de SATD et TD dans un projet logiciel : réfléchir à la bonne solution avant de débuter l'implémentation, effectuer du *code refactoring*, mesurer de manière continue les dettes techniques ou prendre pour habitude de corriger rapidement les dettes, etc. (S. Ambler, 2013) Ces solutions sont intéressantes mais elles ne permettent pas un suivi efficace et la correction complète des dettes. Plusieurs études ont été réalisées dans le but d'analyser les SATD et TD au niveau de de leur diffusion, de leur impact sur la qualité, de leur criticité, de leur survie ou des acteurs. Les méthodes de détection employées sont souvent considérées comme sujet à amélioration de même que la méthodologie et les méthodes d'analyse qui présentent plusieurs menaces à la validité des résultats. Une approche n'ayant pas été testé encore serait de combiner les deux types de dettes de manière à présenter un portrait global d'avantage précis de la qualité d'un projet logiciel par rapport à la présence des dettes techniques. Pour se faire, la détection minutieuse des SATD serait accomplie avant de les utiliser comme données d'entrée à un système d'apprentissage (*machine learning*) qui permettrait de détecter la totalité des TD d'un projet. L'idée est donc d'abord de détecter automatiquement les SATD de manière à ce qu'ensuite le système d'apprentissage parvienne à détecter des relations entre ces SATD et les blocs de code correspondants. Le but final est de réussir à détecter les TD en totalité dans

un projet logiciel grâce au modèle généré par le système d'apprentissage pour ainsi évaluer la qualité de celui-ci. Cependant, cette approche n'a jamais été utilisé auparavant dans ce contexte et plusieurs types de système d'apprentissage existent. Il est d'abord essentiel de trouver un type convenable pour le travail à réaliser, mais encore, le *machine learning* est un domaine complexe et la réussite du projet n'est donc pas assuré. Il est ensuite important d'acquérir une compréhension et une expérience significative dans le domaine avant de penser mener le projet à sa fin. L'outil de détection des SATD peut aussi être problématique, les études passées ont souvent démontré des faiblesses face à l'activité de détection et rien ne prouve que le travail à réaliser à ce niveau sera dénué d'embuches dans ce projet.

2. Objectifs

2.1. Question de recherche

Comment identifier et repérer les *technical debts* dans un projet logiciel, soit les modifications qui ont les caractéristiques typiques des *self-admitted technical debts,* mais n'étant pas documentées explicitement?

2.2. Objectif général

L'objectif général du projet est de développer un outil permettant de repérer les SATD et les utiliser pour l'apprentissage d'un modèle (*machine learning*) pour classifier les changements dans un projet logiciel, en termes de TD ou non, et évaluer la qualité de celui-ci.

2.3. Objectifs spécifiques

Cet objectif général peut être divisé en cinq objectifs plus spécifiques qui seront à la base de la méthodologie détaillée :

1. Repérer les SATD à l'aide des lignes de commentaires
2. Évaluer la pertinence des SATD
3. Définir les métriques à utiliser pour le système d'apprentissage
4. Implémenter l'algorithme du système d'apprentissage pour détecter les TD
5. Valider et améliorer l'algorithme
6. Évaluer la qualité logiciel

2.4. Hypothèses scientifiques

2.4.1. Hypothèse #1

Le nouvel outil de détection des SATD développé à l'aide des lignes de commentaires permet de diminuer le nombre de faux positifs, de suivre minutieusement l'évolution des SATD et de mieux analyser leur comportement.

Justification de l'originalité : Les méthodes de détection utilisées dans les études précédentes sont loin d'être optimales et ne dressent pas un portrait significatif de la présence des SATD dans les projets. De nouvelles approches seront testées lors de la conception de l'outil de détection.

Réfutabilité : L'hypothèse sera réfutée si le taux de faux positif est égal ou plus grand à celui des études passées, si aucune amélioration n'est remarquée au niveau de la rapidité d'analyse et au niveau de la quantité et de la qualité des informations obtenus.

2.4.2. Hypothèse #2

L'utilisation d'un système d'apprentissage pour le repérage des TD et l'évaluation de projets logiciels permet d'augmenter l'efficacité de détection et d'améliorer l'analyse de qualité des logiciels.

Justification de l'originalité : L'utilisation d'un système d'apprentissage n'a pas été testé pour évaluer et repérer les TD. L'analyse des lignes de commentaires est la méthode la plus populaire actuellement utilisée.

Réfutabilité : L'hypothèse sera réfutée s'il est impossible d'utiliser un système d'apprentissage dans le contexte du sujet de recherche ou si l'efficacité de la méthode n'est pas supérieure à celles présentement employées.

3. Méthodologie

3.1. Revue littéraire

La revue de littérature est essentielle à la bonne préparation d'un projet de recherche. Elle permet de résumer et de définir un portrait global de l'ensemble du travail qui a été accompli sur un sujet précis. Dans le cas présent, elle est principalement utile pour connaitre les différentes méthodes de détection des SATD avec leurs bons et mauvais points. Elle permet aussi d'éviter la répétition de travaux déjà réalisés (ex. analyse de même projets logiciels, de mêmes types, de même langage, etc.). Certains aspects trouvés lors de la revue de littérature seront toutefois réutilisés (ex. liste de patrons de SATD, outils logiciels tel que srcML, etc.) en parti ou en totalité. La revue littéraire permet aussi de vérifier que la direction originale du projet, le système d'apprentissage, n'a pas été emprunté auparavant et que l'ensemble des projets à étudier est unique et nouveau.

3.2. Définition de la problématique

La définition de la problématique est une étape importante à la réalisation du projet de recherche puisqu'elle permet de se préparer adéquatement la rencontre de problèmes futurs. Cette affirmation est vraie particulièrement pour l'étape de l'implémentation du système d'apprentissage. L'outil de détection des SATD devrait causer peu de problèmes critiques compte tenu des nombreux antécédents connus. Les systèmes d'apprentissage sont peu utilisés dans le contexte de ce projet de recherche et de nombreux inconnus peuvent se présenter et causer problème.

3.3. Conception de l'outil de détection

Cette étape consiste à concevoir l'outil de détection des SATD dans les projets logiciels étudiés. Un ensemble de projets doit d'abord être déterminé en tenant compte : du langage utilisé, de la taille des projets, de l'architecture et de la base de développement. Il est nécessaire de travailler sur un ensemble unique et couvrant un large spectre de logiciels. L'outil de détection est ensuite construit en prenant compte de ces critères. L'algorithme est principalement divisé en quatre parties : clonage des répertoires Git, extraction des commentaires, identification des SATD et création du fichier de sortie des SATD. L'utilisation de GitHub est requise pour obtenir de l'information sur les dettes dans le temps grâce à l'utilisation des *commits*. Plusieurs études ont déjà conçu un outil semblable à celui-ci, de nombreuses ressources sont donc disponibles et fournissent un bon point de départ.

3.4. Analyse et définition des métriques

L'outil précédemment conçu peut posséder des lacunes et peut être sujet à amélioration. Comme mentionné dans l'hypothèse #1, un grand nombre de faux positifs peuvent être

détecté et plusieurs SATD peuvent être ignorés. Il est donc essentiel d'optimiser et de vérifier manuellement les résultats obtenus pour ensuite apporter les modifications nécessaires. Cette étape est importante puisque les entrées fournies au système d'apprentissage seront fortement basées sur le fichier de sortie des SATD. Les détails sur fichier d'entrée et les métriques restent à être spécifiés selon le modèle qui sera utilisé. Ils seront différents dépendamment du type de système employé.

3.5. Implémentation du système d'apprentissage

Pour implémenter un système d'apprentissage il faut d'abord décider lequel sera le plus pertinent dans le contexte du projet de recherche. Il existe trois types principaux : apprentissage supervisé, apprentissage sans supervision et apprentissage avec renforcement. Parmi ces types, plusieurs options sont disponibles : régression linéaire, régression logistique, arbre de décision, réseau bayésien, réseau de neurones récurrent, etc. Il est donc important de réfléchir à toutes les options possibles avant de débuter l'implémentation. Suite à cela, l'écriture du code peut commencer. Peu de détails peuvent être partagés pour l'instant puisque la méthode qui sera employée n'a pas encore été définie. Toutefois, un réseau de neurones récurrent semble être un choix valable à prime abord. Cette étape de la méthodologie est la plus importante et celle qui distinguera en grande partie ce projet de recherche des autres études faites dans le passé.

3.6. Validation et évaluation des données

Une fois l'algorithme terminé, tout comme pour l'outil de détection, la validation des résultats obtenus est requise. Un type de système d'apprentissage a été sélectionné précédemment, basé sur de la recherche d'information et des recommandations. Toutefois, ce choix peut s'avérer inutilisable ou peut présenter des défauts. Il est donc important de vérifier si les dettes techniques et le résultat de la qualité des projets logiciels sortis par le modèle sont corrects. Il est aussi possible qu'il soit simplement impraticable d'utiliser un certain type de système d'apprentissage. Bref, ce sont tous des actes de vérification et validation essentiels à exécuter pour l'évaluation critique précise de l'algorithme. Si les résultats s'avèrent bons, l'analyse des données peut être entreprise de manière à évaluer les projets logiciels choisis à l'étape 3.3. Des relations et des tendances entre les projets peuvent être découvertes de même que la proposition de travaux futurs.

3.7. Échéancier de réalisation

Les quatre premières sessions sont principalement consacrées au suivi des cours obligatoires et à un début de revue de littérature et de définition du projet. À l'automne 2016, le début des travaux concrets débutent officiellement, en commençant par l'outil de détection et l'analyse des métriques. Ces deux étapes devraient être terminées en grande partie pour le début de 2017. Les quatre premiers mois de cette année, et probablement le début de l'été, devraient être consacrés presque exclusivement à l'implémentation, la validation et l'évaluation de l'algorithme du système d'apprentissage. L'été 2017 devrait être consacré à la rédaction du mémoire et la préparation de la présentation du projet. Le tableau 1 permet de visualiser l'échéancier du projet de recherche.

Tableau 1 : Échéancier du projet de recherche

Échéancier		2015	2016			2017	
		A	H	E	A	H	E
Exigences académiques	Suivi des cours	■	■		■		
Activités	Revue de littérature			■	■		
	Définition du projet			■	■		
	Conception de l'outil de détection				■	■	

Activités	Analyse et définition des métriques						
	Implémentation du système d'apprentissage						
	Validation et évaluation des données						
Biens Livrables	Soumission de l'article						
	Écriture du mémoire						
	Présentation du projet						

3.8. Budget

Peu de matériel est requis pour la réalisation du projet de recherche, la majorité du travail s'effectue sur des ordinateurs qui sont déjà en quantité appréciable à l'intérieur du groupe de recherche. Toutefois, du matériel informatique, tel que des disques durs externes, des logiciels et des licences seront probablement nécessaire au courant de la réalisation du projet. Une part importante du budget se trouve au niveau de l'aide financière fourni à l'étudiant. Des frais de papier et encre sont également étalés tout au long du projet de même que des frais de publication à la fin de celui-ci. Il pourrait y avoir des frais de déplacement pour des conférences. D'autres dépenses inconnues pourraient s'ajouter sporadiquement dans le temps. Le tableau 2 présente le budget du projet de recherche par session.

Tableau 2 : Budget du projet de recherche

Dépenses	2015	2016			2017	
	A	H	E	A	H	E
Aide financière	0	0	0	5384	5384	5384
Ordinateur	0	0	0	1000	0	0
Logiciels	0	0	0	300	100	100
Matériel informatique	0	0	0	100	100	100
Papier et encre	50	50	50	50	50	100
Déplacements	0	0	0	100	100	100
Frais de publication	0	0	0	0	0	250
Autres	100	100	100	100	100	100
Total	**150**	**150**	**150**	**7034**	**5834**	**6134**

3.9. Contraintes du projet

La plus grande contrainte du projet se situe au niveau des projets étudiés. Certains projets sont énormes, un grand nombre de langages sont utilisés, des nombreuses architectures existent, etc. Cette grande diversité cause problème au niveau du temps d'analyse et de conception (détection de SATD, système d'apprentissage). Si une grande variété de projets sont étudiés, énormément de temps supplémentaire devra être ajouté pour la réalisation du projet car les outils devront être compatibles et fonctionner avec cette grande diversité.

4. Contributions attendues

4.1. Biens livrables

Une fois le projet terminé, un outil de détection des SATD et un modèle de détection des dettes techniques et d'évaluation de la qualité seront disponible pour utilisation. Des commentaires seront ajoutés en quantité importante pour assurer une grande maintenabilité. Une présentation du projet, un mémoire et un article scientifique seront aussi produit dans le cadre de ce projet de recherche.

4.2. Propriété intellectuelle

Le projet est réalisé au sein du groupe de recherche Soccerlab à l'École Polytechnique de Montréal. Les résultats et les produits fournis par ce projet appartiendront donc uniquement à l'École Polytechnique de Montréal au sein du groupe de recherche et au directeur de maitrise Giuliano Antoniol.

4.3. Impacts de la recherche

Le projet amènera principalement des avancés à propos de la compréhension des dettes techniques, de leur évolution et de la qualité logiciel. Avec l'utilisation d'un système d'apprentissage il sera possible de déterminer des relations et des tendances entre pratiques de codage et introduction de dettes techniques. L'utilisation de Github pour l'analyse des projets permettra d'analyser les dettes techniques dans le temps grâce à l'utilisation des *commits*. Le modèle généré par le système d'apprentissage permettra de fournir un portrait global de la qualité des logiciels analysé par celui-ci. L'automatisation que permettra le modèle du système d'apprentissage permettra une constance et une flexibilité dans l'évaluation des logiciels informatiques. Les biens livrables seront un atout pour la conception de code de qualité et pour l'efficacité dans le temps de conception.

5. Références

Potdar, A., & Shihab, E. (2014, September). An Exploratory Study on Self-Admitted Technical Debt. In *ICSME* (pp. 91-100).

Suryanarayana, G., Samarthyam, G., & Sharma, T. (2014). Chapter 1 – Technical Debt. Dans M. Kaufmann (édit.), *Refactoring for software design smells: Managing technical debt*. (p. 1-7). Tiré de http://site.ebrary.com/lib/polymtl/detail.action?docID=10985419.

Bavota, G., & Russo, B. (2016, May). A large-scale empirical study on self-admitted technical debt. In *Proceedings of the 13th International Workshop on Mining Software Repositories* (pp. 315-326). ACM.

Maldonado, E. D. S., & Shihab, E. (2015, October). Detecting and quantifying different types of self-admitted technical debt. In *Managing Technical Debt (MTD), 2015 IEEE 7th International Workshop on* (pp. 9-15). IEEE.

Alves, N. S., Ribeiro, L. F., Caires, V., Mendes, T. S., & Spínola, R. O. (2014, September). Towards an ontology of terms on technical debt. In *Managing Technical Debt (MTD), 2014 Sixth International Workshop on* (pp. 1-7). IEEE.

Allman, E. (2012). Managing technical debt. *Communications of the ACM*, *55*(5), 50-55.

Wehaibi, S., Shihab, E., & Guerrouj, L. (2016, March). Examining the impact of self-admitted technical debt on software quality. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* (Vol. 1, pp. 179-188). IEEE.

Zazworka, N., Shaw, M. A., Shull, F., & Seaman, C. (2011, May). Investigating the impact of design debt on software quality. In *Proceedings of the 2nd Workshop on Managing Technical Debt* (pp. 17-23). ACM.

Guo, Y., Seaman, C., Gomes, R., Cavalcanti, A., Tonin, G., Da Silva, F. Q., ... & Siebra, C. (2011, September). Tracking technical debt—An exploratory case study. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on* (pp. 528-531). IEEE.

Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., ... & Sangwan, R. (2010, November). Managing technical debt in software-reliant systems. In *Proceedings of the FSE/SDP workshop on Future of software engineering research* (pp. 47-52). ACM.

Disciplined Agile Delivery, S. Ambler. (2013). 11 Strategies For Dealing With Technical Debt. Tiré de http://www.disciplinedagiledelivery.com/technical-debt/.