Impact of Renaming on Software Change Metrics

Pierre Chanson

Mémoire de stage de Master2
Encadrants: Jean-Rémy Falleri et Matthieu Foucault

LaBRI, UMR 5800
F-33400, Talence, France

Email: pierre.chanson@etu.u-bordeaux.fr,
{falleri,mfoucault}@labri.fr

15 mai 2014

Table des matières

1 Introduction

L'accès aux dépôts logiciels a rendu possible de nombreux travaux de recherche sur l'évolution logicielle. Plus particulièrement, les dépôts de code source gérés par des outils de contrôle de versions (Version Control System, VCS, comme SVN, Mercurial ou encore Git) contiennent l'historique de construction d'un logiciel. Des études se basent sur l'analyse de ces historiques. Tels que la prédiction de bugs un des défis connus du Génie Logiciel, dont le but est de prédire le nombre de bugs et leur localisations dans la prochaine version d'un logiciel. Cette étude se base sur les métriques de procédés comme predicteurs de bugs. Les métriques de procédés se concentrent sur l'évolution d'un logiciel et mesures les modifications subies par les entités d'un code source durant leur cycle de vie. L'hypothèse principale étant que la manière dont les entités du code ont changés a un impact majeur sur la qualité de leur prédiction de bugs.
Or au cours de son histoire, un fichier peut être renommé ou déplacé dans un autre dossier du projet.
Théoriquement, si le renommage d'un fichier à un moment donné de son histoire n'est pas pris en compte, le calcul d'une métrique de procédé sur ce fichier sera faussé. En effet, si on identifie le fichier par son nom, on perdra les informations récoltés avant le renommage. Par ailleurs on peut penser que le refactoring, dont le renommage de fichiers, est très present dans le dévelopement des logiciels à succès d'aujourd'hui. En pratique, nous n'avons pas de chiffres pour le montrer.
Dans un premier temps nous effectuerons une étude de l'existant sur les méthodes utilisés pour détecter le refactoring, les logiciels qui ont étés étudiés, les métriques de procédés ainsi que les VCS. Puis nous choisirons un ensemble de projets cohérant pour faire nos propres expérimentations, nous définirons un niveau de granularité, nous ferons une analyse manuelle de ces projets pour récupérer les renommages réels. Ensuite nous définirons un modèle et nous utiliserons un outils pour récupérer les renommages. Enfin nous définirons comment calculer certaines métriques de procédés afin de mesurer l'impact réel. Plus tard, les résultats de nos expérimentations aménéront à une publication dans la conference ICSME 2014.

2 Etat de l'art

2.1 Evolution logiciel et refactoring

On peut régulièrement lire en introduction d'articles des propos sur l'importance du refactoring, ce qui inclue le renommage, dans les projets à succès (TODO) mais on obtient pas de chiffres précis.

2.2 Métriques de procédés et évolution logiciel

Les métriques de procédés (change metrics) permettent de calculer à quel point une entité de code source à été modifiée au cours d'une période donnée dans l'histoire d'un logiciel. On les utilises usuellement dans la dernière période avant la dernière

	Renaming handling		
		Automatic	
Tool	Manual	Standard	Optional
CVS			
Subversion	×		
Mercurial	×		×
Git			×

TABLE 1 – Handling of renaming of the main VCS tools.

version, l'objectif étant de prédire les bugs qui apparaitront lors de la prochaine release. Elles ne considère donc que les entités étant toujours présentes à la fin de la période et qui ont étés actives dans la période.
Radjenovic et al [3] identifient trois métriques de procédés les plus utilisés pour la prédiction de bugs : Le nombre de développeurs [4] (Number of Developers, NoD), Le nombre de modifications [1] (Number of Changes, NoC) et le Code Churn [2] (CC). Nous donnerons une définition et une méthode précise pour les calculer dans nos expérimentations.

2.3 Métriques et Renommage Existant

Nous nous somme donc intéressés aux études passées qui pouvaient traiter les métriques de procédés dans la prédiction de bugs, et vérifié si ces études avaient considérés le renommage de fichiers. L'article (TODO) référence 26 articles qui nous intéresse, 15 sur des projets industriels, 11 sur des logiciels open-source. TODO

2.4 Les outils

Comme nous l'avons vue, il existe un certains nombre de gestionnaires de versions pour effectuer notre détection de renommage, étant donné que nous avons simplement besoin de versions à comparer. Nous avons néhanmoins étudié les VCS potentiels, afin de voir quel VCS sera le plus apte à gérer les renommages. La table1 résume notre étude. TODO Git propose un algorithme de détection automatique de renommage.

2.5 "Origin Analysis"

L'algorithme utilisé par Git est connu sous sous le nom de "Origin Analysis" et est expliqué par Godfrey et al dans les articles, (TODO) (An integrated approach for studying architectural evolution, Tracking Structural Evolution Using Origin Analysis, Using origin analysis to detect merging and splitting of source code entities) Origin Analysis TODO

3 Problématique

Nous n'avons pas réellement trouvé d'études traitant le renommage. Ces études ont-elles volontairement ou non omis le renommage ? La problématique qui se pose est donc, quelle est la quantité de renommages ou déplacements de fichiers dans les projets ? Où interviennent-ils ? Ont-ils un réel impact sur les métriques de procédés ?

4 Un ensemble de projet

Nous avons commencés par selectionner un ensemble de projets sur lesquels effectués nos expérimentations. Des projets open-source et conséquents et connues de la communautés MSR. Voir table 2 (TODO)

5 Première analyse à grain fin

Le premier travail réalisé a été de faire une étude manuelle des renommages dans les VCS. Nous avons sélectionnés 100 commits de manière aléatoire et étudié le renommage d'entités dans ces commits. Nous avons définie le changement d'identité (TODO) et diférencié le renommage direct du renommage induit (TODO)

6 Analyse à gros grain

6.1 Première experience

6.2 deuxième expérience

6.2.1 Métriques et Renommage

Un gestionnaire de versions (VCS) offre plusieurs moyens de calculer ces métriques car il stocke les informations sur les entités modifiés à chaque nouvelle version, l'auteur de ces modifications, la date etc. De plus il permet la récupération du contenu de chaque entité et de l'ensemble d'un projet à une version donnée. Pour calculer ces métriques, il est donc possible d'analyser chaque entités modifiés lors d'une période puis de ne garder uniquement les entités toujours présente à la dernière version de notre période.
Par ailleurs, il faut noter qu'un VCS identifie une entité par son chemin + nom de fichier. On en déduit qu'un renommage du fichier ou d'un dossier, aura un impact sur le calcul des métriques. Pour expliquer cet impact, on présente un exemple d'historique d'un logiciel figure 1. Ce projet ne contient qu'une entité, Test.php, qui est renommé en Hello.php dans la dernière version. Dans cet exemple nous calculons NoD entre la version 1 et 3.
Le NoD d'un entité de code source au cours d'une période de son histoire correspond au nombre de développeurs ayant étés identifiés comme auteurs d'une modification sur l'entité pendant la période donnée.

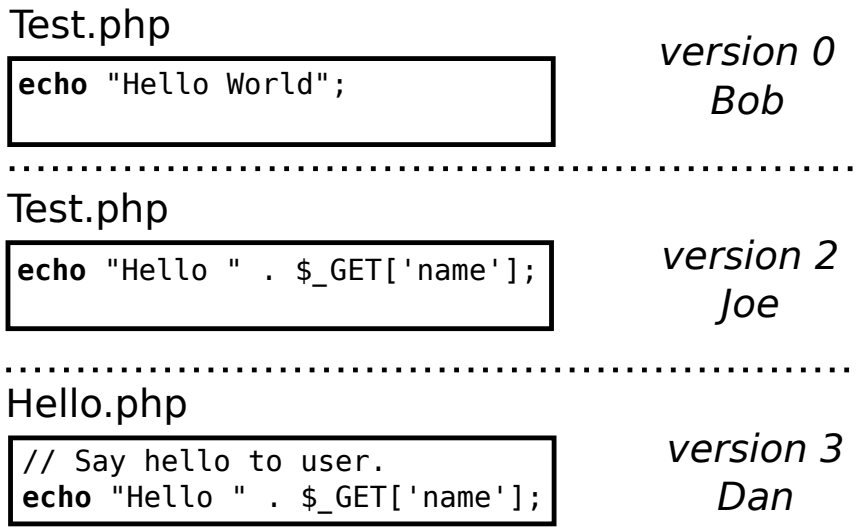

FIGURE 1 – Example of a project history. The project is composed of only one file `Test.php` which is renamed to `Hello.php` in the last version.

La dernière version ne contient qu'une entité. C'est donc cette entité uniquement qui sera considérée. De plus l'identité exacte de cette entité n'aparait que lors de la version 3. Le calcul des métriques est donc trivial, NoD = 1, NoC = 1, CC = 2.
Par ailleurs, si on prend en compte le fait que ce fichié a été renommé, il y a trois versions à regarder en ce qui concerne l'entité. etc...TODO

Nous avons choisi de nous concentrer sur les trois métriques de procédés identifiés plus tôt pour mesurer l'impact du renommage. A partir de script Ruby sur nos projets, voici plus précisement comment nous avons procédés pour les calculer :
(TOTO)
NOD :
NOC :
CC :

7 Resultats

7.1 resultats première expérience

7.2 resultats deuxième expérience

8 Conclusion

Références

[1] Todd L. Graves, Alan F. Karr, J. S. Marron, and Harvey Siy. Predicting fault incidence using software change history. *IEEE Trans. Softw. Eng.*, 26(7) :653–661, July 2000.

[2] John C. Munson and Sebastian G. Elbaum. Code churn : A measure for estimating the impact of code change. In *Software Maintenance, 1998. Proceedings. International Conference on*, page 24–31, 1998.

[3] Danijel Radjenović, Marjan Heričko, Richard Torkar, and Aleš Živkovič. Software fault prediction metrics : A systematic literature review. *Information and Software Technology*, 55(8) :1397–1418, August 2013.

[4] Elaine J. Weyuker, Thomas J. Ostrand, and Robert M. Bell. Do too many cooks spoil the broth ? using the number of developers to enhance defect prediction models. *Empirical Software Engineering*, 13(5) :539–559, October 2008.