Impact of Renaming on Software Change Metrics

Pierre Chanson, Matthieu Foucault, Jean-Rémy Falleri and Xavier Blanc
University of Bordeaux
LaBRI, UMR 5800
F-33400, Talence, France
Email: {pchanson,mfoucault, falleri,xblanc}@labri.fr

***Abstract*—Change metrics, such as Churn or Number of Developers, are good indicators for Software quality, as it has been observed by several studies. However, as these metrics are computed from the changes performed to a software artifact, their value may be skewed in case of artifact renaming, which can therefore be a important threat to validity. This issue is underestimated in most of the existing software study. We therefore propose to assess its impact in this paper. To that extent, we have performed an empirical study on five open-source programs with the intent to evaluate the amount of renaming and their effect on change metrics. We observed that the renaming can significantly alter the metrics for some projects. We also noticed that this effect can be minimizing by following some simple guidelines we present in this paper.**

I. Introduction

Process metrics such as Churn, which measures how much a Software artifact has been modified during its life cycle, or Number of developers, which measures how many developers did contribute to a software artifact, reveal the distribution of the workload of a Software project during its life cycle. They are defined to measure developer's activity with the main insight that developers habits have a high impact on the software quality [**?**]. For instance, Bird et al. have shown that the Ownership metrics, which is a variant of the number of developers, has an impact on the number of post release failures in industrial software projects [**?**].

Measuring process metrics requires to consider a history of a Software project, contrary to classic metrics, such as LOC (Line of Code) or DIT (Depth of Inheritance Tree), that require just one snapshot of the project (usually the last one). Moreover, as software projects are nowadays managed by VCSs (Version Configuration Systems), the history must be composed of successive versions stored by VCS, and must reflect the real evolution of software artifacts. Further, a particular attention should be paid to artifact renaming that have been performed between versions as they may have a deep impact on the measure, and therefore may be an important threat to validity. For example, if we imagine a Software project where all the artifacts have been renamed in the middle of its life cycle, then needless to say that not considering them will return skewed measure for Churn or Number of developers metrics.

In theory, artifact renaming does have an impact of process metrics. In practice, how many renaming occurs? When do they occur? And, what is their impact on process metrics? This paper aims to answer these questions. To that extent, it proposes a deep study of five real open source software projects with the objective to first propose descriptive statistics helping to understand the importance of artifact renaming in Software projects. The results we observed are highly important as renaming reach almost 50% of the software artifacts for some projects. We then have computed two measures for process metrics, one that considers artifact renaming and one that does not, with the intent to evaluate the impact of renaming. Finally based on our observations, we propose some guidelines that aim at minimizing the threat of artifact renaming. For instance, we observed that there are less renaming in maintenance branches stored in VCS. As a consequence, when such a branch exists, it is better to compute process metrics on it.

This paper therefore provides the following contributions:

- A study of the artifact renaming phenomena made on five large open source Software projects.
- An impact analysis for process metrics.
- A set of guidelines aiming at reducing the threat of artifact renaming.

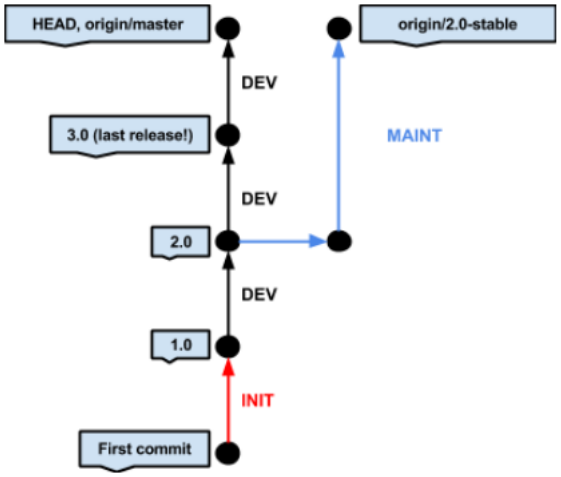
This paper is structured as follows: Section II explains how process metrics are computed and presents how the artifact renaming has an impact on them. Section III-C presents the detailed methodology of our study, including the construction of the dataset and the different analyses we realized. Section V presents the main results of our study which shows that the artifact renaming phenomena does occur and does have an impact on process metrics. This section also presents our guideline to limit the threat of process metrics. Finally, Section VI of our study and Section VII concludes this paper.

II. Related Work

III. Methodology

A. Where do we look ?

We want to compare the proportion and the quantity of renames detected by git in this 3 pieces of the software:
The Maintenance part, the Development part and the Initialisation part. Plus we want to divide the Development part by major releases.



Three types:
MAINT: Caintenance branch, a branch which has its head out of master branch. Only commits which are reachable from the branch and not from master.
INIT: Commits from the first one (included) to the first major release tag on branch master.
DEV: Commits between this major release tag to the next one on branch master.
(last release!): commits from the last major release tag to HEAD.

B. Measures

We will define here the measure we want to observe on the history of the projects creation.

Number of files (#F): number of files in the whole project at the head of branch , respectively end of release (=beginning to next release)
Number of active files (#AF): number of files active (=created or deleted or copied or modified or renamed) in this branch, resp. release.
Percentage of active files:

$$\%AF = \frac{\#AF}{\#F}$$

Number of modification operations (#MO): number of modifications (=create, delete, copy, rename, modif) on files recorded in the branch, resp. release.
Number of rename operations (#RO): number of renames recorded on the branch, resp. release
Percentage of rename operations :

$$\%RO = \frac{\#RO}{\#MO}$$

Percentage of files renamed:

$$\%F_R = \frac{\#AF_R}{\#F}$$

this give us the proportion of all files in the project renamed in this branch, resp. release and shows its impact on the whole project.
Percentage of active files renamed:

$$\%AF_R = \frac{\#AF_R}{\#AF}$$

this give us the proportion of active files in the branch, resp. release, renamed in the branch, resp. release. So

$$\%AFR \geq \%FR$$

C. Process

We will describe here the process to follow to obtain this numbers considering any git repository which follows our model. A Ruby script implementation of the following process is given in annexe.
First of all we need to list major releases tags in a chronological ascending order. This part can not be automated because of the different tag conventions between projetcs like:

- PHPunit: 3.5.0, 3.6.0 etc.
- Pyramid: 1.0, 1.1 etc.
- Jenkins: jenkins-1_400, jenkins-1_410 etc.
- Rails: v2.0.0, v2.1.0 etc.

Begin the process:
We list the project remotes branches. Browsing the branches, all of them we be considered as maintenance branch except origin/master, the initial and developement branch. Most of the time, specific maintenance branches are followed by stable. But if the branch is listed here, it means it has its head out of master branch and so it is a maintenance branch since its last merge with master.

The work is divided in two major step:

- The maintenance branches
- origin/master

If the branch is origin/master, the work will be divided again in:

- first commit(included) to first release tag
- fisrt tag to last tag (one release after another)
- last tag to HEAD

So in any of the four conditions, we will generate the git log between revision range. The major part of the work will be on this commits log. We choosed to work on a general log more than mining the history of every files one after the other for performance reasons. Especially on big projects (rails, jenkins). Moreover this technique only works for the files alive at the HEAD of the project and not from one previous revision. So we analyse all the log in one block, with the good command and options it contains enough informations.

Log lines stored in simple structures like arrays, we can easily count modifications or renames detected.
We will also need to get all the alive files at the end of the log, at the second revision range (branch head or the end of one release).

Then the principal algorithm will browse the log :
Browsing the log in a chronological order will allow us to follow the renames or modifications and rebuilt the history of files. The typical rename in the log looks like "rename bob/{henry ⇒ josef}/george.py (86%)" As exemple, if "bob/henry/george.py" is already recorded, we need to follow "bob/josef/george.py" instead in the renames and modifications record. There is some other cases to consider, for exemple "{henry ⇒ }" or without "{" or the "copy" etc. Each files recorded are unique and only the last name of renamed files is considered. We finally need exclude all the files deleted before the end of the commits log comparing with the list of alive files at the end of the revision range calculated earlier.

IV. Study

V. Results

VI. Threats to validity

VII. Conclusion