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*—Software change metrics, such as *code churn* or *number of developers*, has been shown good indicators for software quality in several studies. However, as these metrics are computed from the modifications performed to a software artifact, their values may be skewed in case of artifact renaming, which can thus be a important threat to validity of studies using such metrics. We therefore assess its impact in this article. To that extent, we perform an empirical study on five open-source popular and mature software projects with the intent to evaluate the amount of renaming and its effect on change metrics. We observed that renaming can significantly alter the metrics in some projects. We also noticed that this effect can be minimized by following some simple guidelines presented in this paper.**

I. Introduction

Software *change metrics* measure how the artifacts of a software project have been modified during their life-cycle. For instance, the *code churn* measures how much lines have been added or removed, and *number of developers* measures how many developers did contribute to the artifact. The main hypothesis behind these metrics is that the manner in which the artifacts have been developed has a huge impact on their quality. Such an hypothesis have been confirmed in several studies, such as [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**], [**?**]. For instance, Bird et al. have shown that the ownership metrics, which are variants of the number of developers metric, are strongly correlated to the number of post-release defects in industrial software projects [**?**].

Contrarily to classical product metrics (such as *number of methods*) which only require one version of a software project, change metrics require its history. Generally, software artifacts are seen as a sequence of versions, and change metrics are computed from measures performed on each version of the sequence. Generally, computing change metrics is straightforward when the projects are managed by a version control system (VCS), which is almost always the case.

However, retrieving the complete sequence of version of a software artifact is not always easy. Indeed, it is not rare that developers change some artifacts name during the project life-cycle. When such a situation occurs, relating the versions of the renamed artifacts with the versions before its renaming is not easy. If renaming is not taken into account, all its versions before renaming are lost and it can therefore have a huge effect on the change metrics values.

In theory, artifact renaming does have an impact on change metrics. In practice, how many renaming occurs? When do they occur? What is their impact on change metrics? This paper aims at answering these questions. To that extent, we perform an in-depth study of mature and popular open-source software projects. Our objective is first to provide descriptive statistics helping to understand the amount and location of artifact renaming in software projects. Then we assess the impact of taking into account renaming on the values of classical change metrics. Finally, based on our observations, we provide simple guidelines that will help researchers and practitioners to better compute change metrics.

Our results indicate that artifact renaming do occur in projects and can be intensive. We observed up to 99% of renamed files in one project. We also observed that renaming is most likely to happen in software development rather than in software maintenance, especially in the first release. Finally, we also observed that it can significantly alter the values of change metrics, and therefore can be a serious threat to the validity of studies using such metrics.

To sum-up, this article makes the following contributions:

- A study of the artifact renaming phenomenon on five mature and popular open-source software projects.
- Detailed information on the amount and location of renaming.
- An analysis of the impact of taking renaming into account on the values of change metrics.
- Several simple guidelines aiming at reducing the threat of artifact renaming when computing change metrics.

This paper is structured as follows: Section II explains how change metrics are computed and presents how artifact renaming has an impact on them. Section III presents the detailed methodology of our study, including the construction of our corpus and the different experiments we performed. Section IV presents the results of our study which shows that the artifact renaming phenomenon does occur and does have an impact on change metrics. This section also presents our guidelines to limit the threat of renamed artifacts when computing change metrics. Finally Section VI concludes this paper.

II. Change Metrics

In this section, we give some background information on change metrics, and how there are computed using version control systems. Then, we explain using a concrete example how renaming can affect the values of these metrics.

Fig. 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.

A. Computing Change Metrics

As explained in Section I, change metrics are computed from the sequence of versions of a software artifact. Since projects are almost stored in version control systems (VCSs), we described how the software artifacts are stored in such VCSs. A VCS generally considers a project as a set of files. It stores every version of the project since the beginning of its development. Several metadata are attached to each version: the identity of the developer that committed it, the files that have been added, modified or deleted since the previous version, and its date.

To recognize the files across the versions, VCSs use their absolute path in the repository. It means that as soon as a file is moved or renamed, the VCS will detect it as the deletion of the file with the old absolute path, and the addition of the file with the new absolute path. Although several VCSs provide commands or algorithms to record renamed and moved files, only Git handles this problem out of the box.

Computing change metrics is a straightforward process. Change metrics are computed in a period of time. Such period is represented as a sequence of versions in the VCS. Change metrics are usually computed for every file present at the last version of the period. Then the sequence of versions is browsed backwards until it reaches the first version or the creation of the file. When the version metadata indicates that the file of interest have been modified or created, this file version is included in the sequence of versions concerning the file. Then the change metric is computed using this sequence of file versions.

B. Renaming and Change Metrics

Figure 1 shows an example of a simple software project history. This project contains only one file, `Test.php`, which is renamed in the last version to `Hello.php`. As an example of change metrics we will use *code churn*, *number of developers* and *number of modifications*. The definition of number of developers and number of modifications is straightforward. Code churn correspond to the number of lines added and deleted in the file since its initial version.

If renaming is not taken into account, the last version of the project of Figure 1 contains only one file, that has only one version. Therefore its code churn is 2 (it has 2 lines of codes), and the number of developers and modification is 1. However, if renaming is taken into account, the last version of the project contains one file that has 3 versions. In this case the code churn is 4 (1 for version 1, 2 for version 2 and 1 for version 3). The number of developers and modifications are 3. Therefore this small example shows the importance of taking renaming into account.

III. Methodology

In this section we present the methodology of our experiment. We want to answer the following research questions:

1) When in the history of projects is renaming located, and what is the typical amount of renaming?
2) Is renaming able to skew significantly the values of change metrics?

To answer these questions we conduct two successive experiments. In these experiments, we will use a dataset of five popular and mature open-source projects. As explained before, VCSs do not handle renaming out of the box, except Git. Therefore, the five projects we selected are managed by Git. As change metrics are usually computed on source code files, we browsed the files of each project and discarded the non code files from the analysis.

A. First Experiment: Location and Amount

Software projects usually undergo distinct phases in their life-cycle. Usually the development is started, then a version is released to the public and maintained, while a new version is being prepared, and so on. In our study we want to assess the amount of renaming in these phases. To that extent, we split a project history in various *periods*. As described in Section II, a period is a sequence of commits. Each period has a kind, as described in Figure 2. The *init* kind is given to the period containing the commits from the beginning of the development to the first release. The *dev* kind is given to periods where a release is being developed. The *maint* kind is given to periods where a release is being maintained. In our project, we can distinguish dev and maint periods because the project we selected have separate branches for software maintenance and development. Our intuition concerning the amount of renaming it is the higher in the init period. Then it is from low to high in dev periods. Finally it is low in maint periods, because the maintenance phase is mostly dedicated to bug fixes.

To measure the location and amount of renaming in the projects, we manually extract the periods from the VCSs of our projects. For each period, we use the Git built-in algorithm to detect renamed files. Finally, for each period, we compute the following metrics:

- *Number of files (#F):* number of files in the project at the last version of the period.

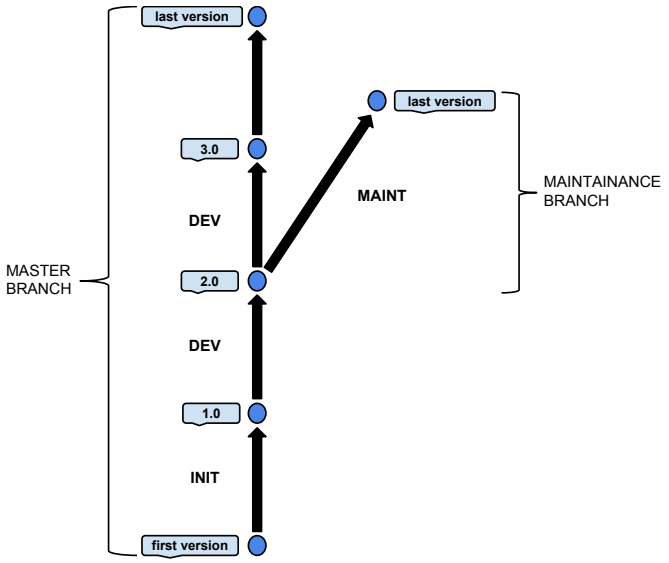

Fig. 2: Our period kinds, as follows. *Init:* versions from the beginning (included) to the first release. *Dev:* development versions between two successive releases. *Maint:* maintenance versions for a specific release. Note that all the versions after the last release are discarded.

- *Number of active files ($\#AF$):* number of files created, deleted, copied, modified or renamed during the period and present at the last version of the period.
- *Percentage of active files ($\%AF$):* $\%AF = \frac{\#AF}{\#F}$.
- *Number of renamed active files ($\#AF_r$):* number of active files that have been renamed.
- *Percentage of files renamed ($\%F_R$):* $\%F_R = \frac{\#AF_R}{\#F}$.
- *Percentage of active files renamed ($\%AF_R$):* $\%AF_R = \frac{\#AF_R}{\#AF}$.

B. Second Experiment: Effect on Metrics

By conducting the first experiment, we will obtain the amount of renaming of each period of each project. The goal of this second experiment is to assess if renaming can skew significantly the values of change metrics. Our objective is to provide a worst-case analysis. Therefore, we will select in the results of the first experiment the dev or maint period with the higher amount of renaming in each project. In these periods, we will compute three well-known change metrics (already described in Section II): *code churn*, *number of developers* and *number of modifications*. This metrics will be computed two times: once without renaming detection, and once with renaming detection. We will finally examine if the metrics computed without renaming detection correlate with the values computed using renaming detection.

IV. Results

A. First Experiment

The results of the first experiment are shown in Table I, Table II, Table III, Table IV and Table V. These tables show several interesting particularities. Firstly, the amount of renaming vary a lot between projects and release. For instance Jenkins has at most 10% of its files renamed in the worst period while PHPUnit has 98.51%. In general, there is a large amount of periods with 0% of renamed files.

Regarding the location, the init period seems to be the worst one. It generally has the greater percentage of renamed files (except in PHPUnit). The dev periods are more prone to renaming than the maint periods, as all five projects have close to 0% of renamed files in maint periods. Finally dev periods can contain a lot of renaming. The results seems to indicate that the major releases are the worse.

		Renaming metrics				
Period	Kind	$\#F$	$\#AF$	$\%AF$	$\%F_r$	$\%AF_r$
init ▷ 1.390	INIT	795	795	100.0	9.55	9.55
1.390 ▷ 1.400	DEV	825	236	28.6	0.84	2.96
1.400 ▷ 1.410	DEV	824	153	18.56	0.0	0.0
1.410 ▷ 1.420	DEV	843	354	41.99	0.23	0.56
1.420 ▷ 1.430	DEV	814	190	23.34	1.22	5.26
1.430 ▷ 1.440	DEV	818	126	15.4	0.0	0.0
1.440 ▷ 1.450	DEV	838	142	16.94	0.0	0.0
1.450 ▷ 1.460	DEV	861	140	16.26	0.0	0.0
1.460 ▷ 1.470	DEV	869	101	11.62	0.11	0.99
1.470 ▷ 1.480	DEV	886	114	12.86	0.45	3.5
1.480 ▷ 1.490	DEV	909	124	13.64	0.33	2.41
1.490 ▷ 1.500	DEV	922	130	14.09	0.86	6.15
1.500 ▷ 1.510	DEV	933	114	12.21	0.1	0.87
1.510 ▷ 1.520	DEV	948	174	18.35	0.0	0.0
1.520 ▷ 1.530	DEV	962	191	19.85	0.0	0.0
1.530 ▷ 1.540	DEV	895	129	14.41	0.0	0.0
1.540 ▷ 1.550	DEV	904	159	17.58	0.0	0.0
1.409	MAINT	823	28	3.4	0.0	0.0
1.424	MAINT	791	47	5.94	0.0	0.0
1.447	MAINT	828	57	6.88	0.0	0.0
1.466	MAINT	864	70	8.1	0.0	0.0
1.480	MAINT	898	113	12.58	0.0	0.0
1.509	MAINT	937	182	19.42	0.0	0.0
1.532	MAINT	901	182	20.19	0.0	0.0

TABLE I: Amount and location of renaming in Jenkins

		Renaming metrics				
Period	Kind	$\#F$	$\#AF$	$\%AF$	$\%F_r$	$\%AF_r$
init ▷ 1.0	INIT	4	4	100.0	100.0	100.0
1.0 ▷ 1.1	DEV	12	12	100.0	0.0	0.0
1.1 ▷ 1.2	DEV	8	8	100.0	62.5	62.5
1.2 ▷ 1.3	DEV	11	10	90.9	0.0	0.0
1.3 ▷ 1.4	DEV	16	16	100.0	6.25	6.25
1.4 ▷ 1.5	DEV	19	19	100.0	15.78	15.78
1.5 ▷ 1.6	DEV	21	19	90.47	0.0	0.0
1.6 ▷ 1.7	DEV	22	18	81.81	0.0	0.0
1.7 ▷ 1.8	DEV	26	26	100.0	0.0	0.0
1.8 ▷ 1.9	DEV	26	23	88.46	0.0	0.0
1.9 ▷ 2.0	DEV	29	28	96.55	0.0	0.0
2.0 ▷ 2.1	DEV	81	81	100.0	7.4	7.4
1.8	MAINT	26	3	11.53	0.0	0.0
1.9	MAINT	27	20	74.07	0.0	0.0

TABLE II: Amount and location of renaming in JQuery

B. Second Experiment

The results of the second experiment are shown in Table VI. It shows that the correlation between the metrics highly depend on the considered period and metric. Change metrics of the Jenkins, Pyramid and Rails projects are not affected by renaming as correlation values are close to 1 in every case. On the other hand, change metrics for the PHPUnit and JQuery projects can be severely impacted by renaming. On JQuery, the

Period	Kind	Renaming metrics				
		$\#F$	$\#AF$	$\%AF$	$\%F_r$	$\%AF_r$
init ▷ 3.5	INIT	130	130	100.0	60.76	60.76
3.5 ▷ 3.6	DEV	121	121	100.0	4.13	4.13
3.6 ▷ 3.7	DEV	124	124	100.0	0.8	0.8
3.7 ▷ 4.0	DEV	135	135	100.0	98.51	98.51
1.3	MAINT	13	13	100.0	0.0	0.0
2.3	MAINT	46	46	100.0	0.0	0.0
3.0	MAINT	206	206	100.0	0.0	0.0
3.1	MAINT	194	16	8.24	0.0	0.0
3.2	MAINT	285	170	59.64	0.0	0.0
3.3	MAINT	342	330	96.49	0.58	0.6
3.4	MAINT	395	0	0.0	0.0	0
3.5	MAINT	124	0	0.0	0.0	0
3.6	MAINT	126	0	0.0	0.0	0
3.7	MAINT	125	1	0.8	0.0	0.0
4.0	MAINT	135	1	0.74	0.0	0.0
4.1	MAINT	120	0	0.0	0.0	0

TABLE III: Amount and location of renaming in PHPUnit

Period	Kind	Renaming metrics				
		$\#F$	$\#AF$	$\%AF$	$\%F_r$	$\%AF_r$
init ▷ 1.0	INIT	45	45	100.0	77.77	77.77
1.0 ▷ 1.1	DEV	46	34	73.91	8.69	11.76
1.1 ▷ 1.2	DEV	58	41	70.68	0.0	0.0
1.2 ▷ 1.3	DEV	72	65	90.27	4.16	4.61
1.3 ▷ 1.4	DEV	72	52	72.22	0.0	0.0
1.4 ▷ 1.5	DEV	68	54	79.41	0.0	0.0
1.0	MAINT	45	12	26.66	0.0	0.0
1.1	MAINT	46	13	28.26	0.0	0.0
1.2	MAINT	58	15	25.86	0.0	0.0
1.3	MAINT	72	6	8.33	0.0	0.0
1.4	MAINT	72	11	15.27	0.0	0.0
1.5	MAINT	68	0	0.0	0.0	0

TABLE IV: Amount and location of renaming in Pyramid

Period	Kind	Renaming metrics				
		$\#F$	$\#AF$	$\%AF$	$\%F_r$	$\%AF_r$
init ▷ 0.10	INIT	170	170	100.0	26.47	26.47
0.10 ▷ 0.11	DEV	180	101	56.11	2.22	3.96
0.11 ▷ 0.12	DEV	192	93	48.43	0.0	0.0
0.12 ▷ 0.13	DEV	217	123	56.68	0.0	0.0
0.13 ▷ 1.1	DEV	312	253	81.08	8.33	10.27
1.1 ▷ 2.0	DEV	373	335	89.81	5.09	5.67
2.0 ▷ 2.1	DEV	436	298	68.34	0.45	0.67
2.1 ▷ 2.2	DEV	467	284	60.81	0.42	0.7
2.2 ▷ 2.3	DEV	482	257	53.31	0.2	0.38
2.3 ▷ 3.0	DEV	570	565	99.12	26.49	26.72
3.0 ▷ 3.1	DEV	630	503	79.84	1.11	1.39
3.1 ▷ 3.2	DEV	684	451	65.93	0.14	0.22
3.2 ▷ 4.0	DEV	734	689	93.86	4.9	5.22
1.2	MAINT	361	146	40.44	0.0	0.0
2.0	MAINT	381	68	17.84	0.0	0.0
2.1	MAINT	442	118	26.69	0.22	0.84
2.2	MAINT	470	67	14.25	0.0	0.0
2.3	MAINT	512	260	50.78	0.0	0.0
3.0	MAINT	575	307	53.39	0.0	0.0
3.1	MAINT	638	266	41.69	0.15	0.37
3.2	MAINT	686	297	43.29	0.0	0.0
4.0	MAINT	734	255	34.74	0.0	0.0
4.1	MAINT	756	117	15.47	0.0	0.0

TABLE V: Amount and location of renaming in Rails

churn metric is resilient to renaming, but number of developers and modifications are significantly impacted. On PHPUnit, all the metrics are significantly impacted by renaming. On these two projects, the metric more sensible to renaming is number of developers. In this case, it could seriously invalidate the result of a study using these metrics.

1) Guidelines: According to the results of our two experiments, we can deduce several simple guidelines to compute change metrics. Firstly, it is important to avoid init periods at all costs. Indeed, these periods usually undergo a significant amount of renaming. On the other hand, maint periods can be picked safely as they very seldomly contain renaming, in tiny amount. Computing change metrics on dev periods is more tricky. As we have seen, it can contain low to high amount of renaming. Dev periods corresponding to major revisions seems to contain much more renaming, so it is recommended to be extra careful with these periods. Anyway, we recommend to apply a renaming detection algorithm systematically to avoid picking up the wrong period.

V. Related Work

VI. Conclusion

In this article, we assessed the impact of renaming on the values of software change metrics. We conducted an empirical study on five popular and mature open-source software projects. We observed that the initial development period of the projects are more prone to renaming than the other periods of the history. We also observed that maintainance periods contain only a tiny amount of renaming. More importantly, we observed that development periods can contain a significant amount of renaming, especially those that correspond to major release. Finally, we observed that renaming can significantly skew the value of change metrics. Therefore, researchers and practitioners should be very careful when computing change metrics. We recommend to avoid computing change metrics on initial development periods. On the contrary, change metrics can be computed out of the box on maitainance periods. Finally, we highly recommend to use rename detection algorithm when computing change metrics on development periods as it can change significantly the values of change metrics.

As a future work, we plan to evaluate and compare the accuracy of existing renaming detection algorithms. We also plan to evaluate the impact of code merging on software change metrics.

	Change metric					
	Churn		Devs		Modifications	
Period	Active	All	Active	All	Active	All
jenkins 1.420 ▷ 1.430	0.89	1	1	1	1	1
jquery 1.1 ▷ 1.2	0.98	0.98	0.08*	0.08*	0.59*	0.59*
phpunit 3.7.0 ▷ 4.7.0	0.6	0.6	0.38	0.38	0.71	0.71
pyramid 1.0 ▷ 1.1	0.96	0.97	1	1	1	1
rails 2.3.0 ▷ 3.0.0	0.98	0.98	0.96	0.96	0.93	0.93

TABLE VI: Spearman correlation coefficients between values of change metrics with and without renaming. Values followed by a * are not significant under the 0.05 p-value.