Revisiting Turnover-Induced Knowledge Loss in Software Projects

Mathieu Nassif and Martin P. Robillard
School of Computer Science
McGill University
Montréal, QC, Canada
Email: {mnassif, martin}@cs.mcgill.ca

***Abstract*—In large software projects, tacit knowledge of the system is threatened by developer turnover. When a developer leaves the project, their knowledge may be lost if the other developers do not understand the design decisions made by the leaving developer. Understanding the source code written by leaving developers thus becomes a burden for their successors.**

In a previous paper, Rigby et al. reported on a case study of turnover-induced knowledge loss in two large projects, Chromium and a project at Avaya, using risk evaluation methods usually applied to financial systems. They found that the two projects were susceptible to large knowledge losses that are more than three times the average loss. We report on a replication of their study on the Chromium project, as well as seven other large and medium-sized open source projects. We also extended their work by studying two variations of the knowledge loss metric, as well as the location and persistence of abandoned files.

We found that all projects had a similar knowledge loss probability distribution, but extreme knowledge loss can be more severe than those originally discovered in Chromium and the project at Avaya. We also found that, in the systems under study, abandoned files often remained in the system for long periods.

I. Introduction

Each significant change a developer contributes to a software's source code captures some knowledge. This knowledge becomes important when a modification must be made to the code to avoid changes that violate the original design or introduce bugs.

Modern version control systems, such as Git, can attribute each line of code to the series of developers who modified it. Using tools such as `git-blame`, it is possible to identify the last developer who touched any given line of code. If this last developer left the project, the knowledge associated with the current form of this line can become inaccessible. In this sense, if no other developer still in the development team is familiar with a code fragment, the intrinsic knowledge related to this fragment can be lost, and further maintenance may require additional effort to regain this knowledge.

We are interested in understanding the evolution of such knowledge loss through time. In this paper, we refer to knowledge loss as the number of files that have been written mostly by developers who left the development team (see Section II for a precise definition). We focus on very high losses because of the threat they represent for software projects.

The idea of knowledge loss is not limited to the software domain. For example, Delta Air Lines saw many of its experienced mechanics leave in the 1990s [1]. In the short term, Delta was able to save on personnel cost, but the remaining mechanics were not as efficient, leading to an overall increase in costs and decrease in customer satisfaction. Modeling knowledge loss has the potential to assist decision makers in mitigating rare but damaging events caused by the sudden departure of important personnel.

An initial quantitative knowledge loss model was introduced by Rigby et al. [2]. They adapted two metrics, *value at risk* and *expected shortfall* [3], from the financial risk modeling domain to software development. To assess the metrics, they conducted a study on two large software projects: the open-source Chromium web browser project and a proprietary telecommunication system developed at Avaya. This initial study found that (1) the historical loss distribution was positively skewed, leading to larger unusual losses than one would naively expect, (2) the two projects were susceptible to losses more than 3.6 times the average loss, at least 5% of the time, and (3) the actual knowledge loss differed from simulations where the leaving developers are chosen at random, suggesting that developers with more contributions were less likely to leave the projects.

This paper reports on a study that is an independent replication of the original research by Rigby et al., in which we completely re-implemented, tested and expanded their original approach. This work aims at exploring more deeply the possible analyses that can be supported using the knowledge loss model, and understanding its limitations. We make the following contributions: first, we validated the three claims mentioned in the previous paragraph with eight medium and large projects, including Chromium. Second, we devised and evaluated two variations of knowledge loss computation: we modified the periods over which knowledge loss was computed, and we weighted each file in a project proportionally to their size (the original approach counted each file equally). Finally, we studied the persistence of abandoned files and their location in the structural organization of our target projects.

The rest of this paper is organized as follow. Section II defines the model of knowledge loss introduced by Rigby et al., and the modifications we made. Section III explains the methodology we followed. Section IV presents our results, and limitations and threats to validity are discussed in Section V.

We discuss related work in Section VI and conclude in Section VII.

II. Modeling Knowledge Loss

The main modeling objective is to quantify knowledge loss in an efficient, reliable and scalable way. To achieve this objective and consistently with the original approach, we rely on the assumption that only the last developer who modified a line understands the line. Although this is a strong assumption, it does not overly threaten the model because the measure is aggregated over entire files, and the threshold for a file to be abandoned is high (90% of the lines).

A. Original Knowledge Loss Model

Rigby et al. define a *leaver* as a developer who is no longer involved a project. To categorize leavers, they looked at the commit history, and considered each developer as a leaver after the last commit they made. They only looked at commits at least one year old, so that recently leaving developers would have been absent from the commit history for at least one year.

In each code file of a software system, for any fixed date, each line can be attributed to the last developer who modified this line using the `git-blame` function (or its equivalent in other version control systems). A *line* is considered *abandoned* at a certain date if it is attributed to a leaver. Rigby et al. consider a *file* as *abandoned* if at least 90% of its lines are abandoned. Once abandoned, a file has three possible futures: it can be recovered, if active developers make new modifications on the file after it has been abandoned; it can be deleted, thus reducing the burden of future maintenance; or it can stay abandoned.

For example, let us assume a developer D makes 200 commits to a project, at times $T_1, \dots T_{200}$, and never commits any code after T_{200}. We consider D to be a leaver starting at time T_{200}. If at time T_1, D created a new file, and no other developer changed it since, the file will be considered abandoned from time T_{200} until more modifications to the file bring the ratio of abandoned lines back under 90%.

For a given period of time, knowledge loss represents the number of files that have been abandoned during this period. Therefore, files are included in the knowledge loss calculation only at the time they are abandoned, unless they are recovered and abandoned again. Knowledge loss is reported as an absolute number of files, as opposed to a proportion, because in large projects, ratios would be overly impacted by the very large magnitude of the denominator.

Based on the definition of knowledge loss, we can compute its empirical probability distribution using the software history. In addition to the mean and median, we use two other statistics, adapted from financial risk management, to get a sense of how large unusual losses can be. The first additional statistic is the *knowledge at risk* (KaR), computed for a fixed confidence level $\alpha \in (0, 1)$. The original study used $\alpha = 95\%$, and we retained this value for our replication.

The knowledge at risk corresponds to the α percentile of the knowledge loss distribution. For example, in Figure 1 we see that the 95th percentile of the distribution corresponds to 685 files (vertical dotted line), which means that overall 685 files or more were abandoned during 5% of the time periods.

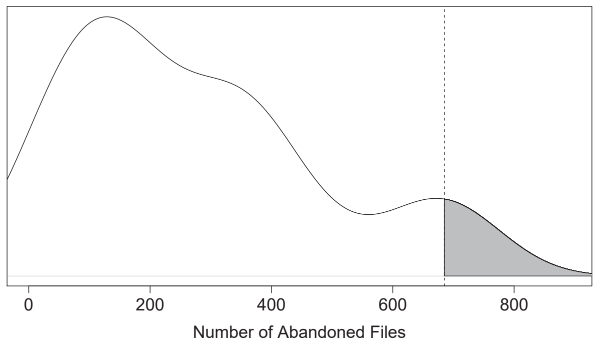


Fig. 1. Representation of the knowledge at risk and expected shortfall on the Chromium knowledge loss probability distribution

The knowledge at risk represents a lower bound of the value of unusually large losses. It does not capture the characteristics of the probability distribution for the higher losses. To get a better estimation of what happens at the right end of the distribution, the *expected shortfall* (ES) estimates the size of such losses. It is the expected value of the highest $1 - \alpha$ part of the distribution. For example, in Figure 1, considering only the position of the vertical dotted line would give us no information about the form of the shaded area. The expected shortfall provides this information by taking the expected value in this zone: we find that in the 5% worst cases, the average of the knowledge loss is 700 files.

Note that without modifying the definition, the knowledge at risk and expected shortfall can be computed both for continuous and discrete distributions. Although the knowledge loss is intrinsically a discrete variable (an absolute number of files), instead of using histograms, we extrapolate a continuous probability distribution from our empirical data, as in the original approach. We do so to make more realistic estimates, and to avoid over-fitting our observations.

B. Limitations and Modifications from the Original Model

The model originally described by Rigby et al. has some limitations that we addressed in this study.

a) Period length: The original experiment focused only on quarters, i.e., three-month periods.[1] One could be interested in computing knowledge loss for other time periods. We address this limitation by computing and comparing knowledge loss for three-month and two-week periods. We introduce this variation in the case study to assess the usability of the metrics with shorter time periods. We do not use larger periods because beyond one quarter the proportional decrease in number of data points limits the applicability of the approach.

[1]The original paper erroneously stated that a quarter is a four-month period. After communication with one of the original authors, we confirmed that they used proper quarters.

b) File weights: The original model does not consider the size of the abandoned files. Although the size of a file is not a direct indication of its complexity, it can easily be presumed to be related to the amount of knowledge captured by the file. Therefore, abandoning a large file should have more impact on the knowledge loss of the system than abandoning a series of small files.

We address this issue by computing a *weighted knowledge loss*, and comparing it to the the unweighted knowledge loss. To compute this variation, once we have determined which files are abandoned, instead of returning a count of those files, we attribute a weight to each file, including files that are not abandoned, and return the sum of the weights of the abandoned files. This weight is proportional to the length of the file, and all weights of all files add up to the number of files in the project. For example, let us assume a project has three files, A (10 lines), B (20 lines) and C (30 lines). The average length of the three files is 20 lines. If we determine that file C is abandoned, we would return a weighted knowledge loss of $\frac{30}{20} = 1.5$, instead of the original value of 1 abandoned file. Therefore, the number returned by the metric can be regarded as an "equivalent number of average files", in the sense that it can represent either many small files, or a lesser number of larger files. The length of the file is calculated as its number of lines, because the file abandonment threshold is based on lines, rather than characters or bytes.

c) Location and persistence of abandoned files: The original approach does not take into consideration what happens to abandoned files. It only considers how much knowledge is lost at each stage. It would be sensible to also investigate whether all abandoned files fall into a few modules, or are scattered throughout the project, and for how long they remain abandoned in the project. While all abandoned files can represent a technical debt that accumulates in the project, the strategies to mitigate the cost of abandoned files could be different if they are found in abandoned modules, or if they are scattered.

We address this issue by computing heat tree maps of abandoned files for each project in the state in which they were at the end of the last period (quarter) we analyzed. These maps show where the abandoned files are located in terms of the modular decomposition of the system.

We also compute the proportion of abandoned files that remain abandoned for every number of quarters, to get an idea of the impact of abandoned files. A file will remain abandoned until it is recovered (due to new modifications) or deleted.

III. Data Collection and Analysis

In this replication study, we re-implemented the empirical and simulated loss distribution computations from the original approach by Rigby et al., and evaluated them on a set of eight open source projects. We followed as closely as possible the original methodology, modifying it only slightly to make it easier to generalize it across the selected projects. We further expanded the approach in three new dimensions, described in Section II-B. The data generated for this study is publicly available for download [4].

A. Original Approach

Rigby et al. used two large software projects, the Chromium[2] web browser and a proprietary project from the telecommunication company Avaya. Their first step was to inspect the two projects to link the different usernames and emails used by a single developer. For Chromium, this step was done using the name aliasing tool by Canfora et al. [5]. They also removed from the projects any third party library used. For Chromium, the developers use a dedicated `third_party` folder to group all third party code.

They downloaded the whole commit history, and excluded the last year of both projects to discriminate leavers from developers who did not commit in the last year. They also excluded the first two years of the commit history because the project had been migrated.[3]

They partitioned the commit history of both projects into quarters, and evaluated knowledge loss for each quarter, obtaining 8 and 17 knowledge loss measures for the Avaya project and Chromium, respectively. They extrapolated a knowledge loss probability distribution from these measures, and computed the mean, median, knowledge at risk and expected shortfall for both projects.

In addition to studying the historical knowledge loss distribution, the authors of the original study performed Monte Carlo simulations for the impact of alternate developer departure scenarios. For each quarter, they discriminated core from non-core developers. The core developers are the most active developers who collectively contributed to 80% of the project. This separation is used to make sure high turnover of transient developers would not bias the simulations. They then kept the same knowledge distribution among developers, i.e., the attribution of lines to developer was kept constant, and computed knowledge loss that would have happened if a different group of developers left the project. For each simulation, they kept the number of core and non-core leavers equal to the historical numbers. Only the identity of the leavers changed. They performed 1000 simulations and reported the distribution of the simulated knowledge loss for each quarter.

B. Modifications to the Original Methodology

In addition to expanding the original model as mentioned in Section II-B, i.e., by varying the length of the intervals over which knowledge loss is computed, introducing a weighting function and evaluating the evolution of abandoned files in the project, we adapted the original approach to make the methodology more consistent across all projects.

Because the Chromium project uses the email address as username, Rigby et al. only needed to match the emails,

[2]The original paper uses the term *Google Chrome* to refer to this project. In this paper, we will use the name *Chromium* to avoid confusion with Google's proprietary version. The project analyzed is the same.

[3]A migration causes the `git-blame` function to erroneously attribute all lines to the developer who made the migration commit.

but this is not true in general for all projects. Therefore, we matched developers based on both emails and username. Furthermore, we found two instances where the name aliasing tool could not match names and emails, both in the same project. We manually corrected these instances to avoid inaccuracies in the distribution of knowledge loss.

To detect the third party code in our evaluation, we created a list of folder names likely to contain third party files, and we removed anything from the list from further consideration. This list is composed of `third_party`, `lib`, `library`, `include`, `plugin`, `plug-in`, `doc` and `documentation`. The last two elements are related to documentation rather than third party code, but we also excluded all documentation from our analysis. This list is case insensitive, but otherwise the folder name must match exactly one of the elements of the list. We used such a list, rather than manually inspecting all projects, to make our approach more generalizable.

Finally, we slightly modified the original approach when partitioning the history into three-month periods. We did not use the regular quarters, i.e., from January 1st to March 31st, etc. Instead, we aligned the end of the last time period with the date exactly one year before we cloned the repositories, and defined the time period sequence by working backward in time from that date. We made this decision for simplicity of our implementation, because we wanted to easily change the length of the periods, and there is no standard partitioning of the years into periods of two weeks. For the remainder of this paper, we will refer to any three-month period as a quarter.

As in the original study, we excluded the first two years from the history of each project, including the projects that were not migrated, to exclude the initial development phase.

The method used to extrapolate the loss distributions is not mentioned in the original paper, so we used the default probability density function extrapolation method from the R programming language. This method uses a Gaussian kernel, with Silverman's "rule of thumb" for the bandwidth. [6]

C. Data Collection

We also extended Rigby et al. study by considering more projects. We designed a randomized sampling strategy to select projects from GitHub,[4] a popular git repository hosting service.

We searched GitHub for projects meeting the following conditions.

1) Popularity score of at least 100 stars,
2) Modified by at least 50 contributors,
3) Larger than 100 Mb (medium) or 1 Gb (large),
4) At least 500 code files,
5) Created before January 1st, 2013.

These thresholds were selected to ensure we studied large, mature projects histories. We excluded small projects to avoid knowledge loss distributions only dependent on one or two developers, in which case sophisticated analysis such as the knowledge at risk would not be useful.

[4]https://www.github.com

We manually vetted the list and removed any repository which, while it passed the initial filters, did not correspond to an active software project. For example, our requirements would not filter out a repository that was migrated to another repository a long time ago, and was not kept up-to-date.

After applying those filters, we were left with around 1000 medium-sized projects, and fewer than 60 large ones. We randomly selected four projects of the first group, and three of the second group. We also added the Chromium project to compare with the original results, ending up with four projects in each group.

The final selection of projects for this study is provided in Table I. It includes projects from different domains and written in different languages. It ranges from Linux OS to lesser known projects. Although we do not make any claim that this sample is representative of any larger set of projects, it provides a diverse set of distinct cases for our repeated-case study.

IV. Results

In the first part of this section we report on the replication of the empirical and simulated knowledge loss distributions. We pursue with the two variations we proposed, biweekly and weighted knowledge loss, and finish with the location and persistence of abandoned files.

A. Loss Per Quarter

Figure 2 shows the empirical quarterly knowledge loss distribution of all eight projects, and Table II shows the value of the different metrics we computed, as well as the number of quarters used to compute these metrics.[7]

The loss distribution for Chromium (Figure 2f) is similar to the one shown in the original paper (Figure 2 of Rigby et al.'s paper). For Chromium we found a median, mean, knowledge at risk and expected shortfall of 194, 268, 682 and 700 files respectively, whereas the values for Rigby et al.'s study of the same statistics are 132, 194, 444 and 709 files. These differences are plausible given that we have different periods.

Most of the graphs are very similar. They have a high peak near small values, which decays quickly, and there are often a few values at the very far right. This shows that unusually high losses happened with a notable frequency.

Two of the graphs stand out from the others. First, the Chromium distribution (Figure 2f) shows a similar trend than the others, but without any extreme values. Second, the Gitlab CE distribution (Figure 2d) is almost symmetrical, only slightly positively skewed. We can see from Table I that Gitlab

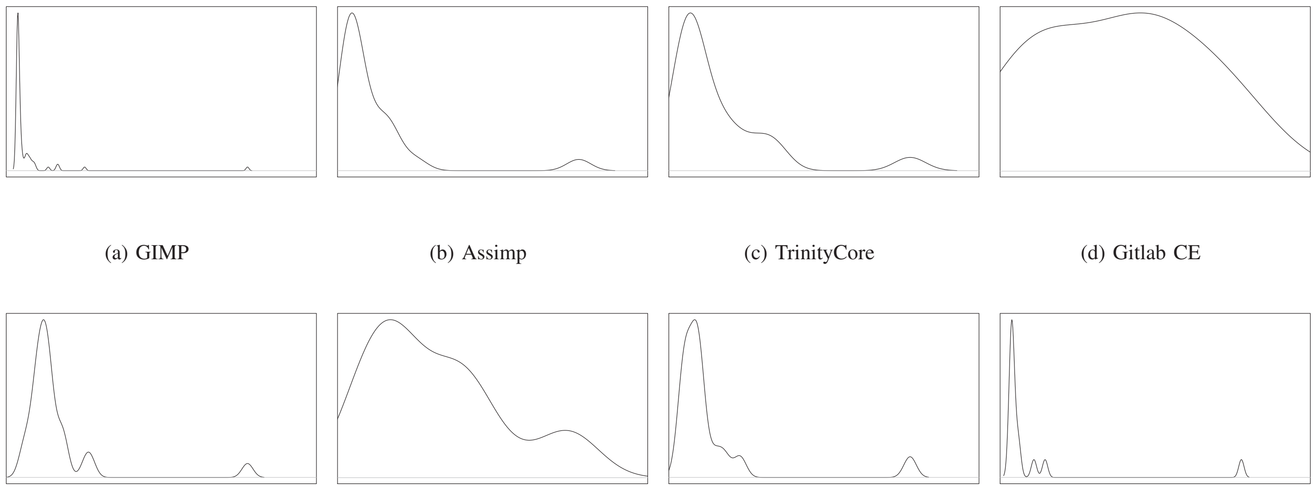
[5]We used the Raspberry Pi repository (https://github.com/raspberrypi/linux) for our study, because it was randomly selected first. This explains the late creation date.

[6]Although there is a repository for Chromium on GitHub, this repository does not contains the actual code, which is hosted on the Google source repository. Therefore, we did not include the star rating of the GitHub repository in this table.

[7]In these distributions and the following, the labels of the axes have been removed as we are mostly interested in the shape of the distributions rather than the exact values. For a better understanding of the magnitude of the losses, the reader can refer to the corresponding tables.

TABLE I
SUMMARY OF THE SELECTED PROJECTS

Project	Creation	Size	Files	Dev.	Stars	Language	Domain	Source Link
						Medium		
GIMP	Jan 1997	324 Mb	3200	210	645	C	Image editor	https://github.com/GNOME/gimp
Assimp	May 2008	133 Mb	600	152	1651	C++	3-D modeling library	https://github.com/assimp/assimp
TrinityCore	Oct 2008	823 Mb	12 000	334	3535	C++	Gaming framework	https://github.com/TrinityCore/TrinityCore
Gitlab CE	Oct 2011	232 Mb	1500	1124	18 696	Ruby	Git hosting platform	https://github.com/gitlabhq/gitlabhq
						Large		
Linux[5]	Jan 2007	1.6 Gb	35 000	6974	3540	C	Operating system	https://github.com/raspberrypi/linux
Chromium	Jul 2008	6 Gb	30 000	6511	–[6]	C++	Web browser	https://chromium.googlesource.com/chromium/src/
Kodi	Sep 2009	1.4 Gb	4000	525	5647	C++	Media player	https://github.com/xbmc/xbmc
Apereo CAS	Jul 2010	1.3 Gb	900	99	2332	Java	Authentificaion service	https://github.com/apereo/cas



(a) GIMP (b) Assimp (c) TrinityCore (d) Gitlab CE

(e) Linux (f) Chromium (g) Kodi (h) Apereo CAS

Fig. 2. Quarterly knowledge loss distribution. The number of files is shown on the x-axis, and the y-axis is the value of the probability distribution function. Note that all axis have different scales.

TABLE II
MEDIAN, MEAN, KAR, ES OF THE QUARTERLY HISTORICAL LOSS

Project	Median	Mean	KaR	ES	Periods
GIMP	0	6	25	65	68
Assimp	1	4	11	25	23
TrinityCore	23	66	177	298	21
Gitlab	9	9	16	17	9
Linux	233	297	577	1218	27
Chromium	194	268	682	700	22
Kodi	76	134	397	1069	18
Apereo CAS	0	9	41	103	15

CE is the youngest project of all. Therefore, a possible cause of the shape of the distribution could be that there has not been any extreme event yet.

The median loss, mean loss, knowledge at risk and expected shortfall of each project are shown in Table II. We can see that the knowledge at risk and the expected shortfall are between 1.8 and 4.6 (KaR) and between 1.9 and 11.4 (ES) times the mean loss, validating the magnitude of the original paper, where the ratios were 2.3 and 3.6, but showing that they can be even worse than the original approximations. There does not seem to be a difference between smaller and larger projects since in both groups there are projects where the knowledge at risk and expected shortfall are low (Gitlab CE and Chromium) or high (GIMP and Apereo CAS) relatively to the mean loss.

Findings: We validated the results from Rigby et al. regarding the shape of the distribution and the magnitude of the ratio between the expected shortfall and the mean. However, for most of the projects, this ratio is higher than what is observed with Chromium.

B. Simulated Knowledge Loss

The second set of replicated results are the Monte Carlo simulations. Figure 3 shows the results of the simulations for each quarter. For each graph, the vertical line shows the 90% confidence interval, ranging from the 5th quantile to the 95th quantile. Thus, the knowledge at risk is the top of the vertical line. On the vertical line, the dot represents the average value of the simulated knowledge loss. Over the line, a triangle

TABLE III
MEDIAN, MEAN, KAR, ES OF THE BIWEEKLY HISTORICAL LOSS

Project	Median	Mean	KaR	ES	Periods
GIMP	0	1	1	12	442
Assimp	0	1	3	9	146
TrinityCore	1	11	55	122	136
Gitlab	0	2	6	12	57
Linux	33	57	193	382	183
Chromium	29	46	144	219	144
Kodi	2	21	60	293	114
Apereo CAS	0	1	3	22	93

TABLE IV
MEDIAN, MEAN, KAR, ES OF THE QUARTERLY WEIGHTED HISTORICAL LOSS

Project	Median	Mean	KaR	ES	Periods
GIMP	0	5	27	56	68
Assimp	0	2	11	15	23
TrinityCore	9	27	86	92	21
Gitlab	3	4	11	13	9
Linux	209	257	505	1089	27
Chromium	197	250	655	741	22
Kodi	42	110	472	824	18
Apereo CAS	0	5	21	55	15

represents the expected shortfall. Finally, a cross indicates the historical knowledge loss for each quarter.

The graphs of GIMP, Gitlab CE and Kodi (Figures 3a, 3d and 3g) are similar in the sense that for most of the quarters, the simulated knowledge loss remains low, but for a few isolated quarters, the graphs show a high peak. In contrast, the other graphs show a smoothly varying simulated knowledge loss distribution. This difference shows that, at least for GIMP, Gitlab CE and Kodi, exceptionally high knowledge losses happened with notable frequency.

Comparing the historical knowledge losses with the distributions of the simulated knowledge losses, we can see that simulated losses generally overestimate the historical losses, except for Linux, where the simulations seem to be well-balanced. This goes with the original claim that simulations may overestimate the knowledge loss, possibly because highly invested developers are less likely to leave a project.

The Linux project stands out in the simulations. First, the bias that can be observed in the other simulations does not appear for this project. Also, the values of expected shortfall are much higher than the knowledge at risk, implying a particularly long tail (even more than the other projects) in the knowledge loss distribution. Further research would be necessary to understand the longer tail and absence of bias in Linux.

Findings: As was observed in the original study, simulations can overestimate knowledge loss. In addition, we were able to identify different patterns from the simulations.

C. Loss Per 2-Week Period

Next, we analyze the effect of using shorter time periods to compute knowledge loss. Figure 4 shows the empirical distribution of biweekly knowledge loss, and Table III shows the value of the different metrics we computed, as well as the number of periods used to compute these metrics.

Again, the graphs show a long tail distribution. This gives us more confidence in our results for the quarterly distributions obtained with fewer data points, because we can observe a similar behavior.

Comparing the probability distribution for 3-month and 2-week periods, we see that the extreme values in the biweekly distributions are more distant relative to the mean than in the quarterly distributions. This is expected as a consequence of the central limit theorem, as the quarterly distributions are roughly the sum of six biweekly distributions. The Chromium (Figure 4f) and Gitlab CE (Figure 4d) distributions are now more similar to the other distributions.

Comparing the knowledge at risk and expected shortfall to the mean leads to similar conclusions than with the quarterly distributions, but with a larger difference between the mean and the knowledge at risk. In this case, the knowledge at risk can be up to 15 times higher than the mean, and is never less than 4.8 times the mean. Therefore, when planning in terms of weeks rather than months, one should expect higher extremes.

Findings: Our results for the biweekly loss distribution validate that the insights obtained in the other experiments are reliable, despite the lower number of empirical data points. Also, it confirms that biweekly knowledge loss is susceptible to more extreme events, relative to the mean, than quarterly losses.

D. Weighted Knowledge Loss

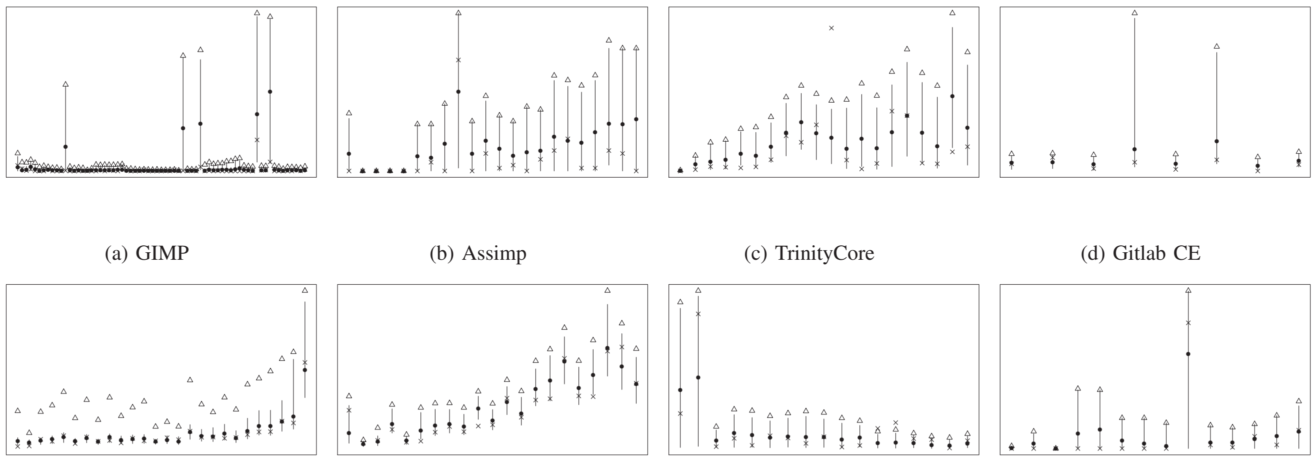
We continue with the weighted knowledge loss per quarter. Figure 5 shows the empirical distribution of weighted knowledge loss for all projects, and Table IV shows the metrics computed over these distributions.

We can see from our results that weighting the files according to their size has an impact on the knowledge loss distributions.

First, the knowledge losses are generally smaller than previously. This is shown by the magnitude of the different metrics. This may be due to the fact that small files are more easily abandoned than large ones, possibly because large files have more contributors, or because they mostly represent core files that need to be updated frequently.

Additionally, the TrinityCore, Gitlab CE and Chromium distributions (Figures 5c, 5d and 5f) have a different shape. The TrinityCore distribution is smoother. It does not have the typical high peak and a few extreme values on the far right. On the other hand, the Gitlab CE and Chromium distributions seem to be more skewed than previously. These differences show that taking into account the size of the file can have a notable impact on the knowledge loss metric. It also shows that the bias introduced by file size variability is not the same for every project.

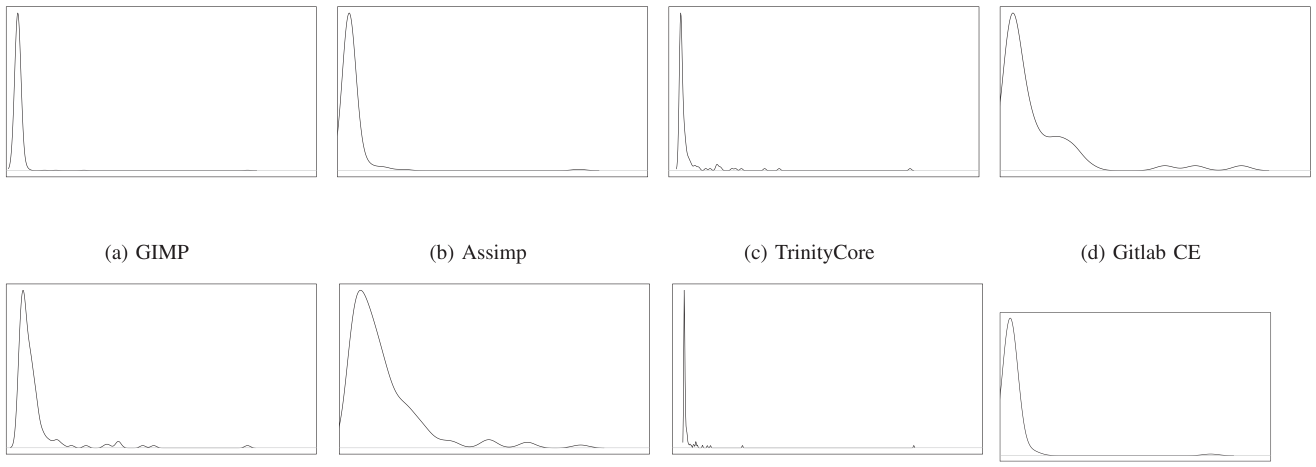
Findings: Unweighted file counts do not take into account the variance in the amount of knowledge contained in different files. Such considerations can create a noticeable difference in



(a) GIMP (b) Assimp (c) TrinityCore (d) Gitlab CE

(e) Linux (f) Chromium (g) Kodi (h) Apereo CAS

Fig. 3. Simulated quarterly knowledge loss for all projects. Each distribution is represented by a vertical bar. The x-axis shows the time, and the y-axis shows the number of files. In each distribution, the full line represents a 90% of the data, the triangle shows the expected shortfall, and the dot shows the mean. The cross shows the historical value.



(a) GIMP (b) Assimp (c) TrinityCore (d) Gitlab CE

(e) Linux (f) Chromium (g) Kodi (h) Apereo CAS

Fig. 4. Biweekly knowledge loss distribution for all projects. The number of files is shown on the x-axis, and the y-axis is the value of the probability distribution function. Note that all axis have different scales.

the results. This finding suggests that more effort should be put in defining a good weighting function to obtain a better quantification of knowledge loss.

E. Distribution and Persistence of Abandoned Files

The heat tree maps representing the proportion of abandoned files in each folder of the projects in the last quarter used in the previous analyses are shown in Figure 6. For each figure, a rectangle represents a folder of the project, and nested rectangles represent nested folders. The area of a rectangle is proportional to the number of files in the folder, and its color represents the proportion of files that are abandoned in the folder, from 0% (light shade) to 100% (dark shade). To improve the clarity of the figures, we considered only up to five nested folders (three for Linux and Chromium). All files in deeper sub-folders were included in the fifth (resp. third) sub-folder.

The aspect of the tree maps varies from a project to another. GIMP, Assimp and Apereo CAS (Figures 6a, 6b and 6h) show a low proportion of abandoned files for almost all folders, and only a few folders that are almost completely abandoned. On the other hand, TrinityCore, Gitlab CE and Chromium (Figures 6c, 6d and 6f) show a more uniform proportion of abandoned files, without any folder standing out

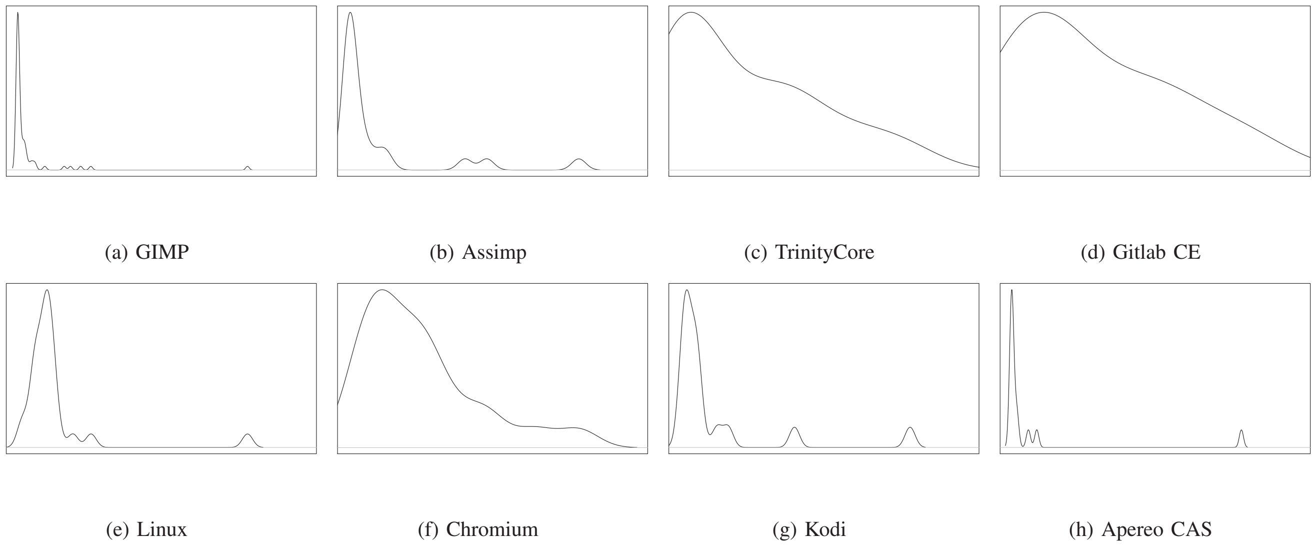


Fig. 5. Quarterly weighted knowledge loss distribution for all projects. The equivalent number of average files is shown on the x-axis, and the y-axis is the value of the probability distribution function. Note that all axis have different values.

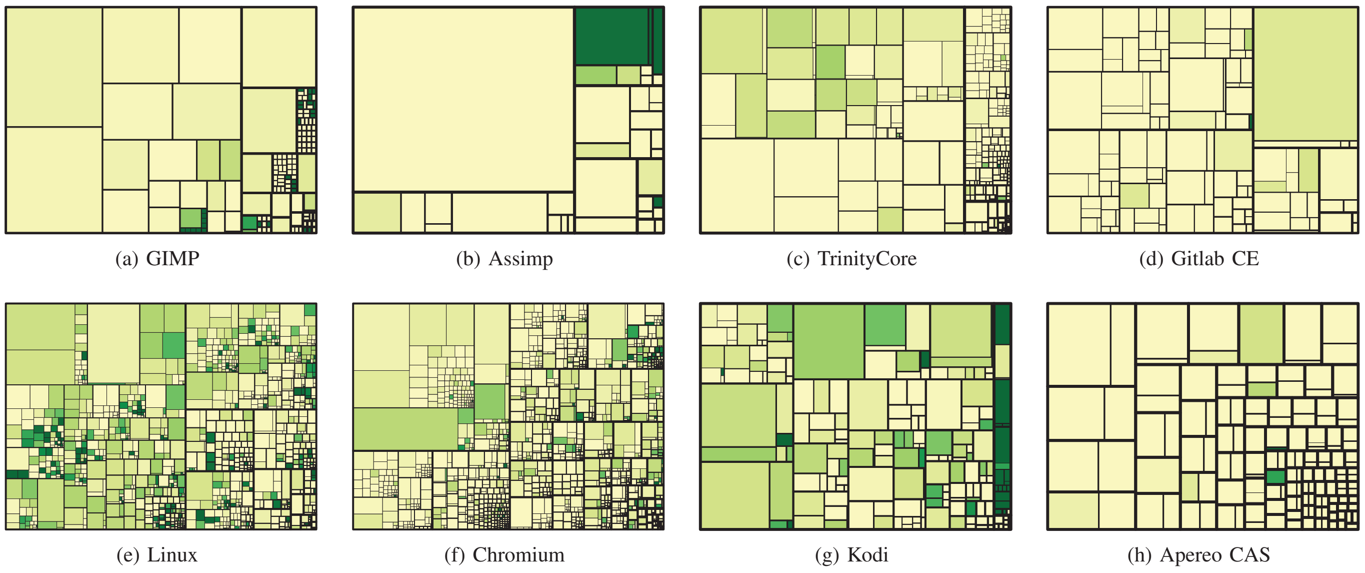


Fig. 6. Proportion of files abandoned in each folder of each project. Each rectangle represents a folder of the projects, with nested rectangles being subfolders of the outer rectangle. The size of each rectangle is proportional to the number of files it contains, and the color represents the proportion of abandoned files among them (darker shades correspond to higher proportions).

as completely abandoned. The two other projects, Linux and Kodi (Figures 6e and 6g) are between the two extremes, with abandoned files spread out in many folders, but with also some folders with very high proportions of abandoned files.

To evaluate the persistence of the abandoned files, we computed the ratio of abandoned files that persisted (i.e., were not deleted or recovered) for *at least* N quarters, for any N. The results for all projects are shown in Figure 7. Note that some lines are shorter than others. This is a consequence of shorter histories. The longest line, associated to GIMP, expands beyond the scale of the figure.

Looking at the persistence plot (Figure 7), we can see that only Apereo CAS seems to get rid of abandoned files quickly. For all the other projects, at least 25% of the abandoned files persist for two years or more. For five of them, more than one half of the abandoned files persisted at least two years.

Findings: A large proportion of abandoned files remained in the systems for many years. The knowledge loss model is fine-grained enough to detect different patterns in the location of file abandonment. Understanding such patterns can help mitigate the impact of knowledge loss on future development.

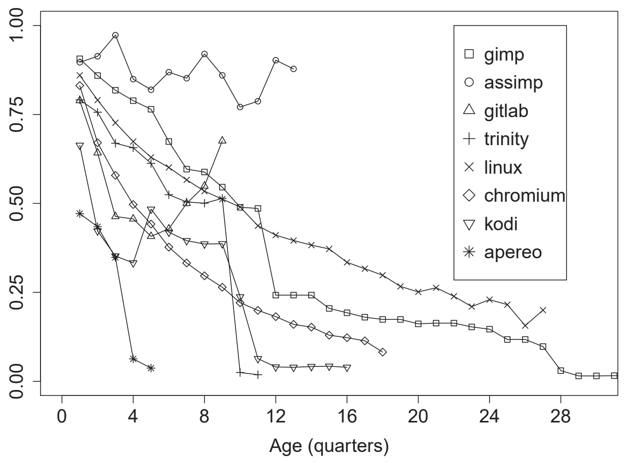


Fig. 7. Ratio of Abandoned Files that Persisted Through N Quarters. To obtain each point, the set of all files that have been abandoned at any point in the history of the project is taken, and we exclude from this set all files abandoned in the last $N-1$ quarters. We then compute the ratio of files among this trimmed set that persisted through at least N quarters.

V. Limitations and Threats to Validity

A. Limitations

The metrics presented in Rigby et al.'s work [2] and reused in this paper aim at creating a robust analytical turnover risk profile, but is only a first step in creating better structures to understand large unexpected turnover and ideally to inform recovery actions. In this sense, there are still limitations to the metrics presented.

First, knowledge loss is an abstract concept that may not be perfectly captured by any product or process metric. Using the last developer who changed a line of code as a proxy for knowledge is only a rough approximation of which files in a project are abandoned. There are people in a project that may not produce a lot of commits, but still have a very detailed knowledge of one or many modules, such as reviewers or documentation experts. However, obtaining precise estimations of knowledge for these roles is out of the scope of this paper.

Second, each line of a file is treated equally, which means that a line with a single bracket is treated as if it captured as much knowledge as a line with a complete statement, which is a strong assumption. While other code units can be presumed to be more accurate proxies for knowledge, previous work has shown that the number of lines of code is highly correlated with many other code metrics, including the number of operators or variables [7]. Additionally, lines can be counted reliably and unambiguously across languages, whereas statements would require a language-dependent code analysis to be used.

Third, we grouped the files by folder to evaluate whether projects tend to lose files in chunks or spread across all folders, under the assumption that a folder would represent a meaningful design unit of the project.

Fourth, the knowledge loss model only shows how many files are abandoned in a given period. It does not show what happen to those files once they are abandoned.

Fifth, the simulated loss distributions rely on the strong assumption that each developer (among the core or the non-core group) has the same probability of leaving a project. Such an assumption may generate a biased knowledge loss distribution. Despite the strength of this assumption, it is required to make a generalizable simulation.

Finally, computing the loss distribution of very large projects, like Chromium, over their whole history is computationally expensive. This is due to the need to blame every changed file of the project multiple times, so the larger and older a project is, the longer it will take. This limitation can become problematic if the knowledge loss model is to be used by real-world teams.

More generally, given the intractable number of factors impacting large software projects, knowledge loss models are not likely to be reliable for predicting rare events and their consequences. Rather, we see their value in supporting an understanding of the present dynamics of knowledge flow in a software project. While this paper focuses on large turnover events as an example of analysis that can be made from the model, the bulk of the distributions are also insightful, and will be the subject of future work.

B. Threats to Validity

We applied the method from Rigby et al. to eight open source projects to verify whether the original results could be applied to other projects. However, we cannot assert that the claims are generalizable. We can only offer insights about which ones are more reliable. We also limit the scope of our study to popular, open source, mature projects. We leave the study of other types of projects to future work.

We performed many of our analyses on small numbers of data points, and the object of our study was to understand rare events. Therefore, we could not compute statistics that would have high confidence values. For example, for all projects except GIMP, we could only use fewer than 30 quarters, after removing the first two years. To mitigate this threat, we computed additional analyses using two-weeks periods and simulations. The similar results across different projects also adds to the reliability of our findings.

We also followed the assumption that a developer is leaving a project after their last commit. This assumption may be incorrect in many situations, for example if the developer takes a new role in the project, such as a reviewer. Similarly, a developer who left the project, only to come back many years later, would be considered active by our analysis during their time outside the project. However, this assumption was necessary because open-source projects do not typically have publicly available rosters of active developers.

Finally, the username and email grouping phase can introduces errors if it is not done properly. In this study, we manually verified the output of our automatic grouping phase, but an ideal solution would be able to verify exactly the identity of each developer.

VI. Related Work

This study builds on previous efforts to quantify knowledge loss in software projects. We independently replicated and expanded the results by Rigby et al., [2] described in Section III. Similarly, Izquierdo-Cortazar et al. [8] studied abandoned code at the line-level, on four open-source projects. They show the potential of their model by profiling the evolution of the abandoned lines for each project.

Understanding the evolution of knowledge in a software project integrates the more general objective of understanding the social dimensions of software development, to complement traditional code analyses [9]. The study of knowledge loss in software projects intersects two complementary lines of inquiry in software engineering: research on the *impact of turnover in development teams* and research on *expertise modeling for software developers*.

A. Impact of Turnover

Robles and Gonzalez-Barahona [10] studied the patterns of developer turnover and code ownership in open-source software. They found several patterns, e.g., code gods, that can offer insights about how developers chose to leave a project. Such patterns could help improve the accuracy of the Monte Carlo simulations we performed. Foucault et al. [11] studied the impact of internal and external turnover. They found that external turnover led to a lower software quality, but they did not find a statistically significant correlation between internal turnover and software quality.

A common scenario used to evaluate the sensitivity of software projects to developer turnover is the "truck factor" (also, "bus factor"). This metric evaluates how likely the project is to become unsustainable after many developers leave the project at once, as if they were hit by a truck. Torchiano et al. [12] studied this metric and proposed a threshold for the number of developers that could leave the project before it becomes unsustainable. Cosentino et al. [13] developed a tool to address the difficulty of computing the truck factor for large projects.

Since developer turnover is inevitable in any long term project, studies have looked at ways to mitigate its negative effects. In addition to evaluating the impact of turnover on software success, Hall et al. [14] investigated the relation between the motivation of developers and turnover rates. They found that low motivation levels lead to high turnover, and thus suggest factors to increase motivation of developers in order to improve software success. Pee et al. [15] compared prior works on the mitigation of the negative effects of turnover, and found two of them, succession planning and the use of knowledge repositories, to be useful.

In contrast to this previous work, this paper presents a different approach to evaluate the impact of turnover, based on a quantification of the knowledge left behind leaving developers. It presents a new perspective that can be complementary to the previous ones.

B. Expertise Models of Software Developers

Several techniques have been developed to identify experts among the team of developers of a software project. McDonald and Ackerman developed Expertise Recommender [16], a technique using multiple heuristics, including looking at the last developer who modified a code fragment. Mockus and Herbsleb developed another technique, Expertise Browser [17], to find experts. Both of these techniques mine the histories of the software projects to compute their expertise model.

More recently, Fritz et al. [18] developed and evaluated a degree-of-knowledge model based on both change histories and interactions with source code elements. They showed that their degree-of-knowledge model can be used to improve existing expertise recommendation approaches. Bird et al. [19] studied the effect of the distribution of code ownership, and found that high levels of code ownership, which would translate to higher expertise, correlates with higher software quality, as measured by the number of defects.

The knowledge loss model parallels this work as it is another attempt to evaluate expertise in a project. However, the objective of the model is not to identify experts, but to evaluate the transfer of knowledge itself.

VII. Conclusion

As open source projects become larger and more popular, they become unsustainable by any single developer, and their maintenance relies on the common effort of tens, hundreds or even thousands of developers. However, every modification by a developer binds some of the developer's knowledge to the file. When the developer leaves the project, this knowledge is lost, as even proper documentation cannot fully capture the rationale of every single decision made by the developer.

We reported on the replication of a previous case study by Rigby et al. which profiled the knowledge loss induced by developers who leave a software project. We validated three of their original claims: (1) the knowledge loss probability distribution is positively skewed, (2) projects are susceptible to large unexpected losses, to an even higher degree than what was observed by Rigby et al., and (3) simulated loss distributions from random departures are greater than the historical losses. Additionally, we found that the model was robust to a change in the length of the interval used when computing losses, and that counting each file equally in the knowledge loss model can overestimate the losses. Finally, we found that the model is fine-grained enough to detect patterns in the location of abandoned files, and that over 25% of the abandoned files remain in the system for at least two years.

Acknowledgment

The authors are grateful to Peter Rigby for clarifying some of the technical aspects of the initial study. This work was funded by NSERC.

References

[1] S. Parise, R. Cross, and T. H. Davenport, "Strategies for preventing a knowledge-loss crisis," *MIT Sloan Management Review*, vol. 47, no. 4, p. 31, 2006.

[2] P. C. Rigby, Y. C. Zhu, S. M. Donadelli, and A. Mockus, "Quantifying and mitigating turnover-induced knowledge loss: Case studies of Chrome and a project at Avaya," in *Proceedings of the 38th ACM/IEEE International Conference on Software Engineering*, 2016, pp. 1006–1016.

[3] A. J. McNeil, R. Frey, and P. Embrechts, *Quantitative risk management: Concepts, techniques and tools*. Princeton University Press, 2015.

[4] M. Nassif and M. P. Robillard, "Replication package for 'Revisiting Turnover-Induced Knowledge Loss in Software Projects'," in *Proceedings of the 33rd IEEE International Conference on Software Maintenance and Evolution*, 2017, Artifact description. [Online]. Available: http://www.cs.mcgill.ca/~swevo/knowledgeloss/

[5] G. Canfora, L. Cerulo, M. Cimitile, and M. Di Penta, "Social interactions around cross-system bug fixings: The case of freebsd and openbsd," in *Proceedings of the ACM/IEEE 8th Working Conference on Mining Software Repositories*, 2011, pp. 143–152.

[6] B. W. Silverman, *Density estimation for statistics and data analysis*. CRC Press, 1986, vol. 26.

[7] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting fault incidence using software change history," *IEEE Transactions on Software Engineering*, vol. 26, no. 7, pp. 653–661, 2000.

[8] D. Izquierdo-Cortazar, G. Robles, F. Ortega, and J. M. Gonzalez-Barahona, "Using software archaeology to measure knowledge loss in software projects due to developer turnover," in *Proceedings of the IEEE 42nd Hawaii International Conference on System Sciences*, 2009, pp. 1–10.

[9] N. Bettenburg and A. E. Hassan, "Studying the impact of social interactions on software quality," *Empirical Software Engineering*, vol. 18, no. 2, pp. 375–431, 2013.

[10] G. Robles and J. M. Gonzalez-Barahona, "Contributor turnover in libre software projects," in *Proceedings of the IFIP International Conference on Open Source Systems*, 2006, pp. 273–286.

[11] M. Foucault, M. Palyart, X. Blanc, G. C. Murphy, and J.-R. Falleri, "Impact of developer turnover on quality in open-source software," in *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 829–841.

[12] M. Torchiano, F. Ricca, and A. Marchetto, "Is my project's truck factor low?: theoretical and empirical considerations about the truck factor threshold," in *Proceedings of the 2nd ACM International Workshop on Emerging Trends in Software Metrics*, 2011, pp. 12–18.

[13] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "Assessing the bus factor of git repositories," in *Proceedings of the 22nd IEEE International Conference on Software Analysis, Evolution and Reengineering*, 2015, pp. 499–503.

[14] T. Hall, S. Beecham, J. Verner, and D. Wilson, "The impact of staff turnover on software projects: the importance of understanding what makes software practitioners tick," in *Proceedings of the ACM SIGMIS CPR conference on Computer personnel doctoral consortium and research*, 2008, pp. 30–39.

[15] L. G. Pee, A. Kankanhalli, G. W. Tan, and G. Tham, "Mitigating the impact of member turnover in information systems development projects," *IEEE Transactions on Engineering Management*, vol. 61, no. 4, pp. 702–716, 2014.

[16] D. W. McDonald and M. S. Ackerman, "Expertise recommender: a flexible recommendation system and architecture," in *Proceedings of the ACM Conference on Computer-Supported Cooperative Work*, 2000, pp. 231–240.

[17] A. Mockus and J. D. Herbsleb, "Expertise browser: A quantitative approach to identifying expertise," in *Proceedings of the 24th International Conference on Software Engineering*, 2002, pp. 503–512.

[18] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill, "A degree-of-knowledge model to capture source code familiarity," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, vol. 1, 2010, pp. 385–394.

[19] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code!: examining the effects of ownership on software quality," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European conference on Foundations of Software Engineering*, 2011, pp. 4–14.

Replication Package for "Revisiting Turnover-Induced Knowledge Loss in Software Projects"

Mathieu Nassif and Martin P. Robillard
School of Computer Science
McGill University
Montréal, QC, Canada
Email: {mnassif, martin}@cs.mcgill.ca

I. Introduction

We provide a replication package containing all data generated during our study on turnover-induced knowledge loss. This package contains the relevant data to compute knowledge loss from eight open source projects. It can be downloaded from our website, http://www.cs.mcgill.ca/~swevo/knowledgeloss/.

The replication package consists mainly of a `Data` folder containing one subfolder for each project. The following eight projects have been studied (the name of the associated subfolder are indicated in parentheses).

- Assimp (`assimp`)
- Apereo CAS (`cas`)
- Chromium (`chromium`)
- GIMP (`gimp`)
- Gitlab CE (`gitlabhq`)
- Linux (`linux`)
- TrinityCore (`trinitycore`)
- KODI (`xbmc`)

Along with the `Data` folder are a copy of our paper, a `README` file explaining how to read the data, and a `Procedure` file explaining how to reproduce the data.

II. Content

Each subfolder contains the following files.

source.url: URL link to the GitHub repository hosting the project. For Chromium, which is not hosted on GitHub, this file links to the source repository, and the additional file **documentation.url** links to the documentation page.

authors.csv: List of all authors that participated to the project. Authors are identified by their email used to commit.

periods_2weeks.csv and **periods_3months.csv**: List of epoch timestamps respectively 2 weeks and 3 months apart. Each line corresponds to one author. Multiple emails will appear on the same line if they are likely to correspond to the same person.

commits_authors.csv: List of all commits to the project, with its author.

commits_files.csv: List of all commits to the project, with the list of files modified in the commit.

leavers_2weeks.csv and **leavers_3months.csv**: List of all developers who left the project between the corresponding timestamp and the previous.

newcomers_2weeks.csv and **newcomers_3months.csv**: List of all new developers for each period.

ownership_2weeks and **ownership_3months** folders: These folders contain the `git-blame` information. Each folder contains a list of csv files identified by the period timestamp. Each of these files contains the condensed `git-blame` information; it shows only the number of lines attributed to each author.

III. Reproducing the Results

The replication package contains a file `Procedure.txt` describing how to reproduce our results. This file contains, when possible, the Git commands we used, and a detailed explanation on how to parse their output.

Generating the ownership information is computationally expensive. Depending on the number of files and commits in the targeted project, it can take from hours to days to compute. Fortunately, this task is highly parallelizable, so it can be completed more quickly on a network of computers. The other files can be generated more quickly. It should take a modern computer no more than a few minutes for each file.