GAP: Forecasting Commit Activity in git Projects

Alexandre Decan[a], Eleni Constantinou[a,b], Tom Mens[a], Henrique Rocha[c]

[a]*Software Engineering Lab, University of Mons*
Avenue Maistriau 15, B-7000 Mons, Belgium – first.last@umons.ac.be
[b]*Software Engineering and Technology group, Eindhoven University of Technology*
Groene Loper 5, 5612 AE, Eindhoven, Netherlands – e.constantinou@tue.nl
[c]*Lab On Reengineering (LORE - Ansymo), University of Antwerp*
Middelheimlaan 1, B-2020 Antwerpen, Belgium – henrique.rocha@uantwerp.be

Abstract

Abandonment of active developers poses a significant risk for many open source software projects. This risk can be reduced by forecasting the future activity of contributors involved in such projects. Focusing on the commit activity of individuals involved in git repositories, this paper proposes a practicable probabilistic forecasting model based on the statistical technique of survival analysis. The model is empirically validated on a wide variety of projects accounting for 7,528 git repositories and 5,947 active contributors. We found that a model based on the last 20 observed days of commit activity per contributor provides the best concordance. We also found that the predictions provided by the model are generally close to actual observations, with slight underestimations for low probability predictions and slight overestimations for higher probability predictions. This model is implemented as part of an open source tool, called GAP, that predicts future commit activity.

Keywords: git, commit activity, developer abandonment, distributed software development, prediction model

1. Introduction

Software development is inherently a social activity, especially in open source software (OSS) development, as OSS projects are developed and maintained by communities of interacting contributors [1]. The evolution of OSS projects depends on the presence of volunteers or paid contributors, i.e., company employees supporting the project, that will take on the maintenance effort [2]. However, company support in OSS is rather scarce [3], requiring

the help of volunteers to maintain OSS projects. The risk of contributors becoming inactive can be detrimental for the project as less than 50% of such abandoned projects find new contributors to take on their maintenance [4]. It is therefore important to retain existing contributors to minimize such risks, as well as to prevent the knowledge loss that occurs when important contributors abandon the projects they are involved in [5].

The topic of contributor retention has been extensively studied by the research community. Most studies performed an *a posteriori* analysis and investigated the effects of contributor abandonment on the project sustainability over time, revealing that contributor retention is one of the many factors that plays a role in a project's sustainability over time [6, 7, 8, 9, 10]. However, OSS contributors and community managers can benefit from an *a priori* approach that will identify contributors at risk of abandoning the project early. Such an approach would consist of monitoring the activity of contributors to detect probable future abandonments, and to mitigate the associated risks as early as possible, e.g., by engaging with the contributor, or by identifying other qualified contributors to take over the work.

Existing project monitoring solutions provide useful information about a project's contributor community, e.g., Bitergia's analytics service[1] analyses project community activity, performance, diversity, size and demography. Although these analytics are highly relevant to get insight in the past and current project activity, they cannot be utilized to assess the risk of losing existing contributors in the near future. Currently, there is no methodology in the literature to predict when the next contributor activity will take place, neither automated tool support to perform such an analysis. This paper fills that gap by providing a method to forecast the future development (commit) activity of OSS contributors on git repositories. By doing so, our work enables community managers to properly assess abandonment risks and tackle such issues before they become detrimental to the project continuation.

We propose a simple probabilistic forecasting model that relies on the previous and recent contributions of individual contributors to estimate the elapsed amount of time until their next contribution. We validate the forecasting model on a wide range of diverse projects. To this end, we use a large collection of git repositories involving 5,947 distinct software developers contributing to 7,528 software repositories, covering a wide range of projects

[1] `bitergia.com/bitergia-analytics/`

in terms of longevity (short/long-lived), technical size (number of commits) and community size (small/large groups of contributors). The model is operationalised through the Git Activity Predictor (GAP), an open source tool that supports different output formats and visualisations of the predictions.

The remainder of this paper is structured as follows. Section 2 presents the related work. Section 3 introduces the probabilistic activity forecasting model. Section 4 validates the model through an empirical case study on Cargo. Section 5 presents the GAP tool implementing this model. Section 6 discusses the benefits that GAP can bring to OSS project communities, possible limitations of the underlying model, and presents potential directions of future work. Section 7 discusses the threats to validity. Finally, Section 8 concludes.

2. Related work

The evolution of OSS projects depends on the presence of volunteers or paid contributors, i.e., company employees supporting the project, that will take on the maintenance effort [2]. Voluntary work is an important asset playing a big role in OSS project long term survival and continuity. It is essential both to attract newcomers and retain developers to maintain a critical mass of core developers in a project [11]. Coelho and Valente reported that 41% of failed open source projects cited a reason related to a lack of time or interest of the developer team [12].

Many studies have focused on developer attraction and retention. Steinmacher et al. conducted a literature review identifying barriers encountered by newcomers to OSS projects and proposed a classification of these barriers based on socio-technical attributes, e.g., technical knowledge, experience, and social interactions [13]. In a follow-up work, they presented FLOSScoach, a dedicated portal created to support newcomers to OSS projects [7]. Fronchetti et al. found through a study of 450 OSS projects that the most influencing factors to attract contributors are the number of stars, the time to merge pull requests, and the number of programming languages used [9].

Lin et al. analysed the evolution of industrial OSS projects to identify how the duration and type of contributions affect developer turnover [14]. They found that developers that remain active for longer periods of time mainly focus on contributing to the source code and on modifying existing files instead of creating new ones. Iaffaldano et al. conducted interviews and analysed the reasons for contributors taking temporary breaks [15]. Their

analysis suggests that longer temporary breaks can lead a contributor to a permanent abandonment state. Constantinou and Mens investigated socio-technical factors in relation to developer retention in the RubyGems and npm ecosystems [8]. They found that both social and technical activities influence developer abandonment, and that socially active developers are less likely to abandon.

Developer retention has been shown to influence project sustainability. The tight relationship between an author and its contributed source code makes software development susceptible to knowledge loss when authors leave the project and abandon their code [5]. Gamalielsson and Lundell showed that the long-term involvement of developers is favourable for a project's sustainability [6]. Constantinou and Mens studied the Ruby ecosystem on GitHub and found a high impact of contributor abandonment on the sustainability of Ruby projects [16]. Avelino et al. studied the abandonment of open source projects by their core developers and the motivation and difficulties faced by new core developers taking over these projects [4]. They report that only 41% of the studied projects recovered after all their core developers abandoned them. Miller et al. looked at the reasons why established open source contributors disengage. They suggest that data-driven systems could be developed to help identify groups at risk [10].

These studies reveal the need for project community monitoring tools to predict which individuals are more likely to abandon soon, and to mitigate such risks early. Our work aims to address this need, by proposing an activity prediction model that can be used as a basis to assess the likelihood of contributor abandonment.

Although there are studies on the analysis of developer contributions in open-source projects, these studies focus mainly on measuring or characterising developer contributions, rather than on *when* these contributions will occur. For example, Amor et al. proposed to characterize the complete development activity by combining data and metrics from different data sources (versioning system, mailing list, issue tracker, etc.) to measure the total effort invested in a project [17]. This idea was further extended by Gousios et al. who proposed a model delivering accurate developer contribution measurements based on a combination of traditional contribution metrics and data mined from software repositories [18]. Mockus et al. studied how much effort remains to be spent on a specific software project and how that effort will be distributed over time. They proposed a model to predict software project effort, schedule and defects so as to plan resource allocation [19]. Weicheng

et al. observed that developers often do not commit for a long time. They explored developer commit habits and their relationship with file version evolution on eight GitHub projects using four metrics [20], including the time interval between two commits. They showed that the average time between commits ranges from 2 to 5 days, depending on the number of changed files. They concluded that knowing the habits of developers is helpful to make a better work plan.

Our work is inspired by the above studies, but focuses on forecasting the future activity of developers in terms of commit activity. The model we propose is based on the technique of survival analysis [21]. This technique analyses "time to event" data with the aim to estimate the survival rate of a given population. It has been used in other empirical software engineering studies, e.g., to estimate the survival of open source projects over time [22], to analyse the use and removal of functions in PHP code [23], to analyse the survival of database access libraries in Java code [24], to compare the release cycle of reusable components in software ecosystems [25], to analyse retention of developers in software ecosystems [8, 14], or to understand how contributors join the core team of a FLOSS project [26].

3. Probabilistic Activity Forecasting Model

The goal of this paper is to provide a practicable forecasting model that estimates the elapsed amount of time until the next contribution of a git author. Such a model relies on recent information about previous commits of individual contributors to estimate the expected amount of time until their next activity will take place. In Section 3.1 we evaluate the statistical modelling technique that we deem most appropriate for this purpose. Section 3.2 provides more details on the selected technique.

3.1. Selected technique

We propose a model based on the technique of survival analysis, also known as event history analysis [21]. This technique analyses "time to event" data with the aim to estimate the survival rate of a given population, i.e., the expected time duration until a specific "event of interest" happens. Survival analysis models take into account the fact that some observed subjects may be "censored" when the event of interest was not (yet) observed for them. Survival analysis is a natural choice when it comes to predict the likelihood

of an event such as death of an biological organism, failure of mechanical component, or the occurrence of a commit activity in a project.

While other techniques could have been considered to predict future activity of git contributors, they were excluded for practical reasons. *Auto-Regressive Integrated Moving Average (ARIMA)* is a well-known statistical technique for predicting future points in a time series [27], but it requires the time series to be stationary. In our data, we found that the number of nonseasonal differences (i.e., the degree of differencing) needed for stationarity varies in time and depends on the considered contributor and project. As such, ARIMA cannot be applied to project contributors without prior individual validation. Doing so would require a lot more historical data that is not necessarily available for smaller projects or for less active contributors.

More advanced models, such as semi-parametric *proportional hazards* models or fully parametric *accelerated failure time* models, have the advantage that they can take into account the effect of covariates on events [28]. Such models often lead to more accurate predictions, but come at the extra cost of more demanding assumptions: they require that all individuals have the same hazard function, up to a scaling factor. Moreover, these models need to be trained with appropriate covariates. Identifying such covariates is not an easy task and collecting the data for them is challenging in practice, even for simple covariates (e.g., activity of the contributor in other projects, other kinds of activities carried out by the contributor, etc.). While these data can be collected once to feed and train the model, it is still required to collect new data each time the model is used for predictions on a new contributor, a new project or a new date. This makes such models difficult to use in practice. In comparison, a survival analysis model only requires to collect, for each individual, a limited number of previous contributions within the same project.

3.2. Survival analysis model

The prediction model we propose relies on the Kaplan-Meier estimator [29], a common non-parametric statistic to estimate survival functions. A survival function defines the probability of surviving past time t or, equivalently, the probability that the event has not occurred yet at time t.

In our case, it provides probabilities for individual git contributors to remain active in function of the time elapsed since their previous recorded day of activity. We decided to focus on daily commit activity for pragmatic reasons: it is unlikely that a project community manager would need to predict

the activity of a contributor up to the nearest second or even hour. While an even coarser granularity of time (e.g., weeks instead of days) might lead to more accurate models, it would be too coarse to provide useful insights.

Concretely, given an individual contributor and a project, for each day when a git activity is recorded, we compute the duration until the next day of commit activity of this contributor. The last n computed durations are then fed to the model and used to estimate the survival rate of the considered event.

As an example, Figure 1 shows Kaplan-Meier survival functions at four different time points in 2018 and in 2019, obtained by extracting and analysing the commit activity of an anonymized contributor of project serde-rs/serde on GitHub. For each considered time point, the model is built based on the last $n = 20$ time deltas (the elapsed time between the previous commit day and the next one) for the commit activity of the contributor.[2]

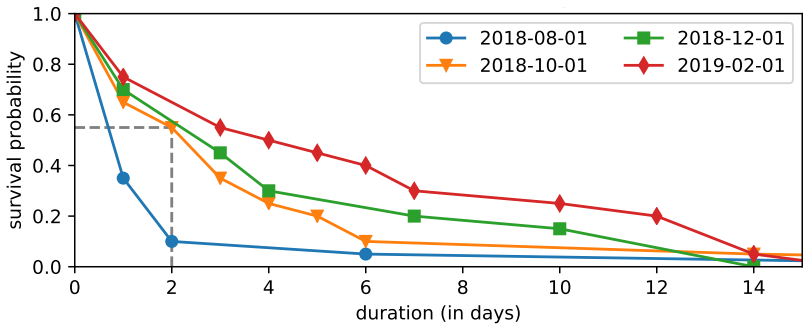

Figure 1: Survival probability for event "the contributor commits to the git repository" of project serde-rs/serde on GitHub, computed at four different time points.

The figure shows the survival probability for the event "the contributor commits to the git repository" as a function of the time elapsed since the previous day of activity of this contributor. Each coloured curve shows the probability (on the y-axis) that the next commit *will not* take place in a specific number of days (value on x-axis). Based on this probability, the complement probability that a commit *will* take place in a specific number of days can be easily computed. For example, consider the orange curve corresponding to October 1st 2018. There is a 45% probability ($= 1.0 - 0.55$)

[2]The motivation of the choice of $n = 20$ will be discussed in detail in Section 4.2.

that the next commit will take place within 2 days, as illustrated by the dashed grey lines in Figure 1.

To have a first indication of the expected accuracy of the proposed forecasting model, we applied it on all days of activity of an anonymized git contributor on project serde-rs/serde on GitHub. Concretely, for each date of recorded activity, a prediction model was built based on the last $n = 20$ time deltas. These models were used to predict the duration until next contribution with probabilities $0.5, 0.6, 0.7, 0.8$ and 0.9. The predicted durations are compared with the actual duration between the currently considered day and the day of the next recorded activity.

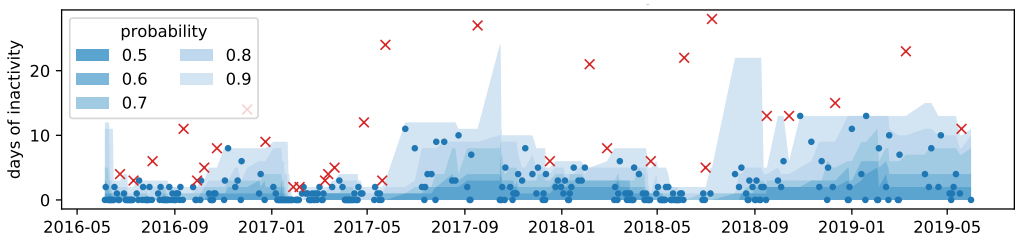

Figure 2: Rolling application of the probabilistic model for an anonymized contributor on git project serde-rs/serde on GitHub. Blue dots • correspond to *correctly predicted* days of activity while red crosses × correspond to *incorrectly predicted* days of activity.

Figure 2 shows the *actual* days of commit activity of the contributor, and the *predicted* probabilities as areas with different shades of blue. Blue dots • correspond to *correctly predicted* days of activity (they belong to a blue shared area), while red crosses × correspond to those days of activity for which the model *failed to correctly predict* the time until the next commit. The figure provides initial insight (for a single contributor on a single project only) in the accuracy of the model, in that there are considerably more correct predictions than there are incorrect ones.

4. Validation of the Probabilistic Model

This section carries out a full-fledged empirical study to evaluate the accuracy of the model on a large and varied dataset containing thousands of projects and contributors. A replication package of the model and its evaluation is provided at `doi.org/10.5281/zenodo.3666048`.

4.1. Selected Dataset

To validate the probabilistic forecasting model, we need a large dataset containing thousands of git repositories involving thousands of contributors. The dataset should include a wide range of software projects serving different purposes, and exhibiting a wide variation in terms of longevity and size. The dataset should exclude git repositories that have been created merely for experimental or personal reasons, or that only show sporadic traces of commit activity [30].

Registries of reusable software packages (e.g., npm for JavaScript, Cargo for Rust, Maven for Java, or PyPI for Python) are good candidates to find such large datasets, as they typically host thousands of language-specific software packages at different levels of maturity and popularity and serving different purposes. However, not all packages belonging to such registries necessarily have an associated git repository.

To choose an appropriate dataset, we considered all 36 package registries available in the libraries.io 1.2.0 dataset of package managers [31]. We retained the Cargo package registry as an adequate case study since it includes over ten thousand distinct packages covering 47 different application domains[3], and a very high proportion of these packages (84.5%) have an associated git repository. For comparison, only 68.4% of all npm packages report an associated git repository, and even less for PyPI (61.0%) and Maven (47.5%).

Over 98% of all git repositories associated to Cargo projects were hosted on GitHub. We cloned and locally analysed all those GitHub repositories that were available at the time of our analysis. This corresponded to 75.2% of the initial dataset, accounting for 9,208 projects. For each project, the commits of all branches were extracted. Commits found in multiple branches (i.e., commits that were merged in other branches) were deduplicated and considered only once. This left us with over 1.2M commits.

To safeguard the validity of our results, we addressed the issue of multiple identities for distinct contributors within and across all git repositories. Following the approach of Avelino et al. [4], the GitHub usernames were retrieved using the GitHub API and mapped to contributors' name and email pairs. An alternative way to resolve multiple identities would have been to resort to an identity merging algorithm [32, 33, 34, 35]. However, such algorithms require manual inspection and validation of the results, resulting in a

[3] `crates.io/categories`

very labour-intensive and error-prone process.

We also removed 6,975 contributor/project pairs with a single day of activity as it is obviously not possible to make predictions based on a single day of activity.

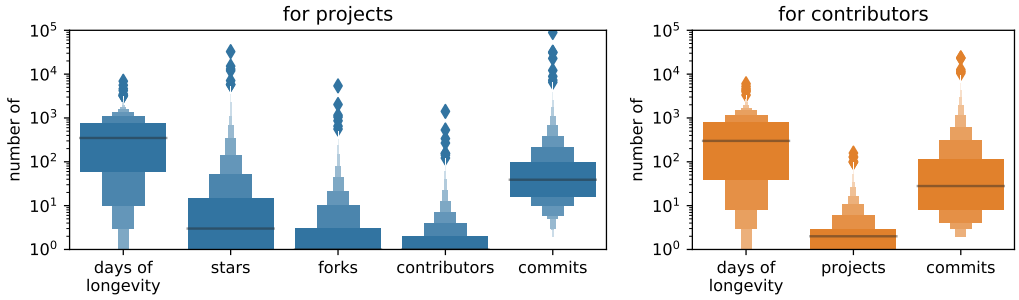

Figure 3: Distributions of several characteristics for the considered projects and contributors.

Our final dataset includes 7,528 projects, 5,947 developers, and accounts for 374,622 distinct days of commit activity when considering all 17,340 contributor/project pairs. Figure 3 shows the distributions of several characteristics of the considered projects and contributors by means of boxen plots [36]. We observe that the dataset covers a wide range of projects and contributors with a diversity in terms of longevity, technical size (number of commits), social size (number of contributors per project and number of projects per contributor) and popularity (number of stars and forks).

It is important to note that not all contributor/project pairs can be used to train the forecasting model. For a given n, the model requires at least n durations (corresponding to at least $n+1$ days of git commit activity) to be trained, and an extra one for its validation. Some contributor/project pairs will not have a commit-day history length of at least $n+2$ samples and therefore have to be excluded for validating the model.

Figure 4 shows the number of contributor/project pairs (left y-axis) and the number of samples in the dataset (right y-axis) with respect to the value of n. We observe a rapid decrease in the number of pairs for low values of n, a consequence of the presence of many short-lived contributions. For instance, going from $n = 1$ to $n = 2$ results in 3,231 fewer pairs (-18.6%) accounting for 17K samples (-4.9%). However, the set of samples on which the model is evaluated remains large even for higher values of n. For instance, the dataset

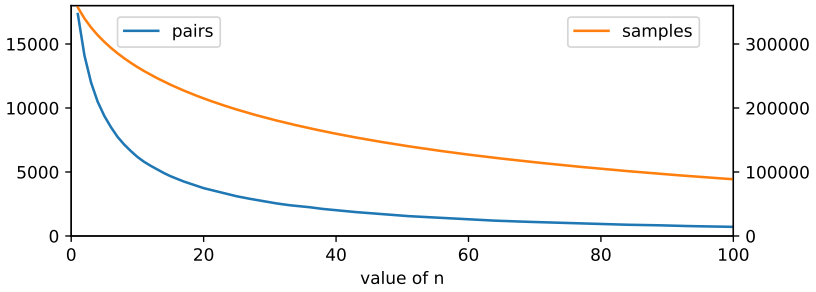

Figure 4: Number of contributor/project pairs (left y-axis) and number of samples in the dataset (right y-axis) in function of n.

still contains 214,993 samples for $n = 20$, 141,679 samples for $n = 50$, and 88,644 samples for $n = 100$.

4.2. Evaluation of the Probabilistic Model

Using the selected dataset, for each recorded day of activity of each contributor in each project, $n = 20$ durations corresponding to the time intervals between the 21 previously recorded days of activity were fed individually into the model, resulting in one model per day of activity, per contributor and per project. Next, the values predicted by the model were compared with the actual duration between the considered day of activity and the next one, following a rolling-origin/walk-forward validation method [37].

Because of the probabilistic nature of the activity prediction model, it cannot be evaluated in a straightforward way [38]. This is because the model does not return a prediction in the form of "Alice will contribute again in 5 days". Instead, it provides a probability distribution for the next contribution of Alice, e.g., there is a 70% chance that Alice will contribute in the next 10 days and a 90% chance that she will contribute in the next 20 days. Therefore, we first check whether the returned probability distributions correspond to the distribution of actual observations. In other words, we verify if the values predicted with a probability p actually correspond to $100 * p$ percent of the actual observations.

To this end, our model is evaluated using a reliability plot [38], shown in Figure 5. It is a variant of a Q−Q (quantile-quantile) plot showing for each probability the proportion of predictions that are actually observed (i.e., that are less than or equal to the prediction). A point (p, y) in the figure indicates that $100 * y$ percent of the values predicted for probability p are correct upper

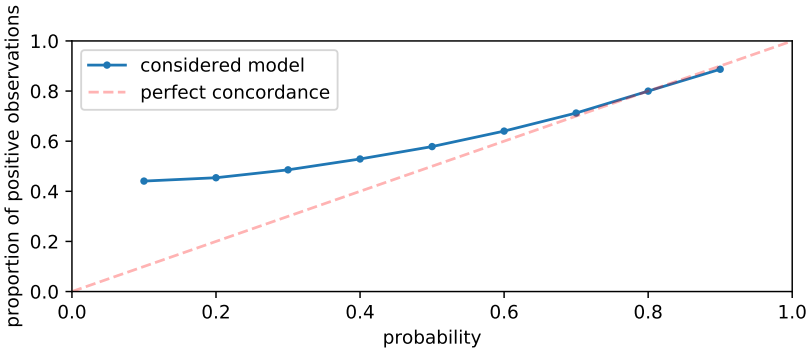

Figure 5: Reliability plot relating the estimated probability distribution to the distribution of actual observations.

bounds when they are compared with the actual observations. A perfectly concordant model is obtained when $y = p$, i.e., when $100 * p$ percent of the observations are less than or equal to the predicted values associated with probability p. As can be observed in Figure 5, high probability values have a very small distance between the blue line and the ideal situation represented by the red dotted line. This means that the higher the probability is, the better the concordance of the model.

High probability forecasts ($p \geq 0.8$) are very slightly underconfident (they are predicted less often than they are observed), and low probability forecasts (especially below 0.5) are strongly overconfident (they are predicted far more often than they are actually observed). The observed overconfidence for low probability predictions can be explained by the low dispersion of values that are predicted with lower probabilities p: from 94.2% (for $p = 0.1$) to 36.9% (for $p = 0.5$) of the predictions are of 0 days, a duration that was actually observed in around 43.4% of the cases considered. This is a consequence of many contributors having many periods of time with a (mostly) daily activity, and explains why at least 44.1% of the predictions are actually observed even when the probability is low ($p = 0.1$).

While the evaluation results presented above rely on a model based on the last $n = 20$ past observed durations, we experimentally evaluated the model on a range of increasing values for n, respectively 10, 20, 30, 50, 75 and 100. Larger values of n imply that we require and consider a longer commit history, incidentally excluding spontaneous contributors with short-lived contributions. Smaller values of n indicate that we only rely on the most

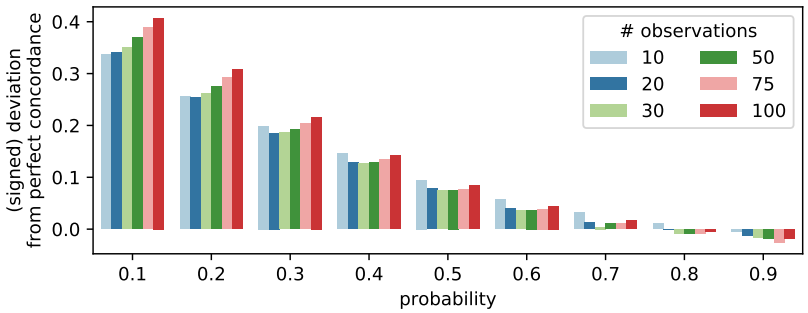

Figure 6: Deviation from perfect concordance w.r.t. the number of past observations used to train the model.

recent days of activity to forecast when future contributions will take place. Figure 6 shows the deviation of the model with respect to the perfect concordance for different values of n. Smaller deviation indicates a closer match to the perfect concordance and consequently a better model. We computed the sum of squared deviations from perfect concordance for each considered value of n to compare the resulting models. We found that considering the last 20 observed durations produces the lowest score (0.24) compared to the other values that have a score comprised between 0.25 (+4%, for $n = 30$) and 0.34 (+41%, for $n = 100$).

4.3. Accuracy of the Forecasting Model

The accuracy of the forecasting model is evaluated by analysing the error between the predicted and actual (observed) date of the next commit. Since such errors can be either overestimations or underestimations, we computed the number of observations for which the predicted value is above (overestimation), below (underestimation), or equal to the observed value. Figure 7 shows the proportion of overestimations, underestimations and correct estimations for probabilities ranging from 0.1 to 0.9.

We observe that predictions made with lower probabilities are mostly underestimations while predictions made with higher probabilities are mostly overestimations. This should not come as a surprise since the estimations provided by the prediction model increase with the probabilities. On average, the model predicts lower values for lower probabilities, and higher values for higher probabilities.

Figure 8 presents the distributions of the differences between predicted

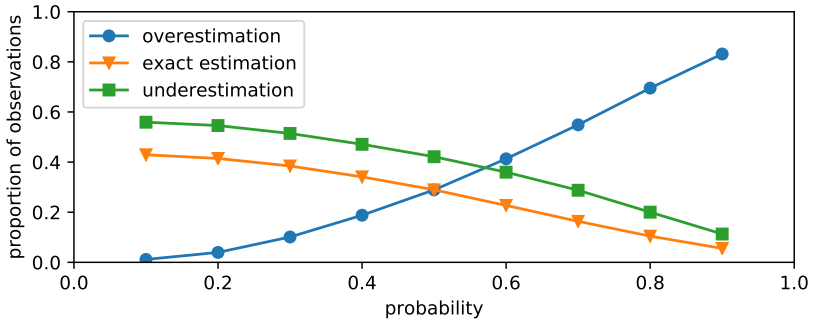

Figure 7: Proportion of overestimations, underestimations and exact estimations made by the forecasting model, for probability values ranging from 0.1 to 0.9.

and observed durations of inactivity for probabilities ranging from 0.1 to 0.9. Values closer to zero correspond to more accurate predictions. The dashed gray line corresponds to a "perfect" match between the predicted and observed time until the next commit. Values above the dashed line indicate overestimations, while values below indicate underestimations of the time until the next commit.

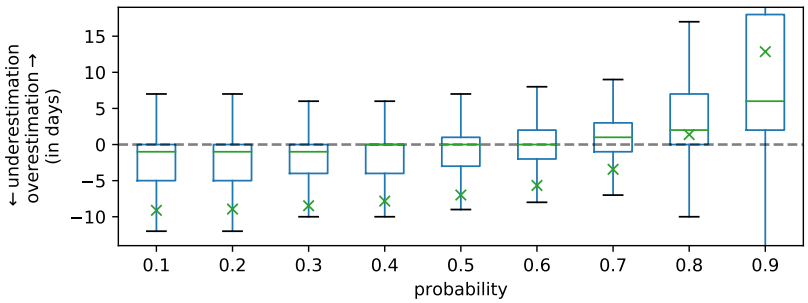

Figure 8: Distribution of the differences between predicted and observed durations of inactivity.

We observe that most differences are quite close to 0. For probabilities $p < 0.5$, half of the differences between the estimated value and the observed value are comprised between 0 (25^{th} percentile) and -5 days (75^{th} percentile). For probabilities $p > 0.5$, positive differences gradually replace negative differences as the probability increases, confirming what was observed for Figure 7. For instance, for $p = 0.8$ half of the differences range between 0 and 7 days with an average of 1.4 days, close to the median value. For $p = 0.9$, the

distribution of the positive differences is much more skewed, with a median value of 6 days, an average of 12.9 days and a 75th percentile of 18 days.

5. The GAP tool

Based on the probabilistic forecasting model of Section 3, the Git Activity Predictor (GAP) was implemented in Python 3.5+. It is packaged and installable through pip, the official package manager for Python.[4] GAP makes it possible to forecast developer activity in the form of git commits. Version 0.11.0 of GAP was used for this paper.

The tool has also been integrated by the Bitergia company[5] in their software development analytics and project monitoring dashboard GrimoireLab. This will facilitate its accessibility and take up by open source project communities.

Figure 9 presents the general architecture and process followed by GAP. Given a list of git repositories and an optional identity mapping file, GAP analyses the repositories to generate per contributor probabilistic forecasting models for the next commit activity of the contributor at a specific point in time. The forecasts can be reported in four different formats: (i) simple text, (ii) comma-separated values (csv), (iii) JSON, and (iv) bar activity charts.

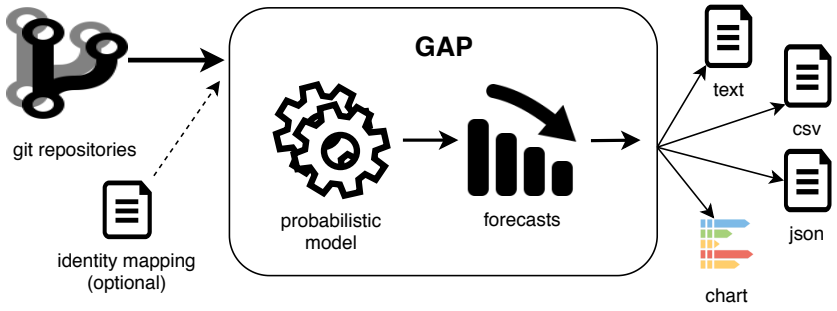

Figure 9: Presentation of the tool architecture.

GAP comes with a command-line interface, summarised in Figure 10. In its simplest form, it only requires a path to the git repository of the software project that needs to be analysed. One or more such repositories can be

[4] `pip install git+https://github.com/alexandredecan/gap`
[5] `bitergia.com`

```
$ gap -h
usage: gap [-h] [--date DATE] [--obs OBS]
           [--probs [PROB [PROB ...]]]
           [--limit LIMIT] [--mapping [MAPPING]]
           [--branches [BRANCH [BRANCH ...]]]
           [--as-dates]
           [--text | --csv | --json | --plot [PLOT]]
           [PATH [PATH ...]]
```

Figure 10: List of GAP command-line arguments.

provided as input, depending on the scope of the analysis. For example, a company may desire to analyse the commit activity of all its paid contributors in the open source projects the company is involved in; package maintainers may desire to analyse the global activity of all the packages they are maintaining; a package developer may wish to analyse the activity in a project and all its dependents; and so on. By default, GAP analyses all branches of each repository but the list of branches to analyse can be specified.

A file mapping multiple authors into specific identities can be provided to group contributors into teams, to merge multiple identities of the same contributor into one, to anonymise author identities, or to exclude specific authors from the analysis. Parameters can be provided to set the date of the analysis (current day by default), the number of observations used by the model (20 by default), the list of probability values ($0.5, 0.6, 0.7, 0.8$ and 0.9 by default), and the minimal recency of the last activity (30 days by default). Predictions can be expressed either as dates or as relative time differences (by default).

GAP analyzes the commit history per contributor and predicts the future activity according to the forecasting model of Section 3, producing activity forecasts for each contributor. Figure 11 shows the output of GAP for a randomly chosen Rust project on git, analysed on 2019-05-16. The authors listed in the output are auto-generated names that we have provided to the identity mapping file anon.csv to guarantee author anonymity.

The output shows, for each recently active contributor (first column), the time difference in days since the last recorded commit activity (second column), and the expected number of days until the next predicted day of activity according to a certain probability threshold ranging between 0.5 and

```
$ gap ./repo --date 2019-05-16 --mapping anon.csv
author          last  0.5  0.6 0.7 0.8  0.9
M. Johnston        0    3    3   4   6    7
D. Young           0    1    1   2   3    4
B. Rodriguez       0    1    1   3   3    6
D. Johnson         0    2    3   4   4    8
Z. Andrews         0    4    5   8  10   23
J. Berry           0    3    6   7  29  131
bots (grouped)     0    1    1   1   1    2
L. Owen           -1    3    7   7  17   22
S. Allen          -2    3    7  12  25   38
J. Smith          -4    0    3   6   7   21
J. Schultz        -4    5   10  13  23   38
M. Fry            -9   -6   -3   0   8   21
J. Lopez         -17  -11  -11  -9  -6   -1
S. Lewis         -20  -12  -10  -3  50   96
```

Figure 11: Example of running GAP on a Rust project's git repository. (Author names are anonymised and replaced by autogenerated ones.) Reported values represent the predicted number of days until the next commit activity will take place. Negative values indicate that no activity has been recorded within the predicted time interval. For example, J. Lopez was predicted with 80% probability to have committed 6 days before 2019-05-16.

0.9 (last five columns). Predicted values greater than zero indicate that the next day of activity is expected in the future, while negative values indicate that the predicted day of activity should have taken place before the specified day of analysis.

The output of GAP can also be used as a basis for a project-level dashboard that visualises the project's past and estimated future commit activities, as in the example provided in Figure 12, generated by GAP by appending `--plot` to the command-line argument. In Figure 12, the orange squares represent the dates of recorded commit activity of each contributor. The blue rectangles indicate the predicted duration until the next activity for different probabilities. For instance, the figure shows that *M. Fry* (the third contributor in Figure 12) has been inactive for 8 days (counting backwards from the date of the analysis) but there is a 70% probability that this contributor will contribute within the day, and 80% within the next 8 days. On the other hand, the inactivity of *J. Lopez* (the second contributor in Figure 12) already exceeds the duration predicted with a probability of 90%, indicating an unexpected irregularity in the activity frequency.

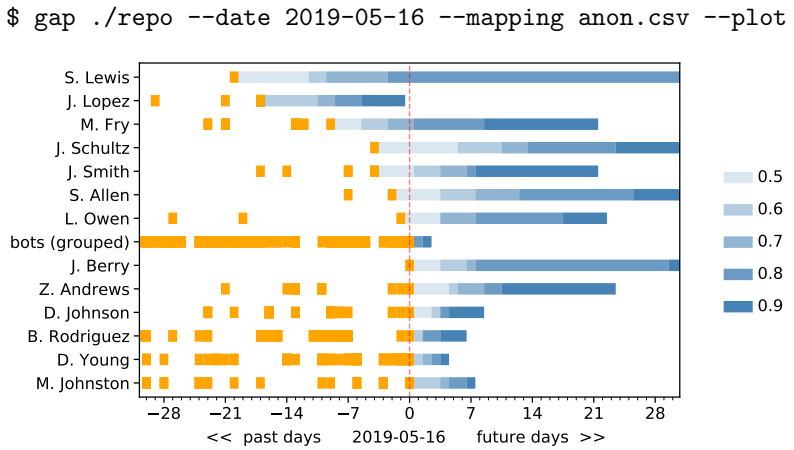

Figure 12: Proposed activity forecasting visualisation for a selected git project. GAP was applied on 2019-05-16 for this example. (Autogenerated names are used for the project contributors.)

The line with *"bots (grouped)"* in the figure shows the activity of *bots*. Two bots were manually identified in the dataset for this particular project repository and we used the identity mapping file to merge them into a single identity. As expected, bots reveal a much more frequent activity pattern. With few exceptions, they commit on a daily basis. Hence, there is 80% probability that either bot will commit again the next day.

6. Discussion and future work

Keeping track of contributors' activity or lack of activity can be very important for the project's sustainability [6, 8]. Longer periods of inactivity of a contributor may lead to a permanent disengagement and can have a negative effect on a project. For projects involving a large number of contributors, it is not feasible to contact them directly and ask when their next activity is going to take place. Since each contributor has different activity rhythms, it is difficult to assess manually when someone is falling behind their regular schedule. Consequently, a forecasting model needs to take into account different activity behaviours for each person [15]. The value of GAP relies on such occasions. GAP identifies the likelihood of a new contribution based on the past activity profile of each individual contributor. Therefore, the forecasts produced by GAP are specific to each individual. The tool can

therefore be used to verify which contributors are close to becoming inactive and plan accordingly, possibly mitigating abandonment issues.

As discussed in Section 4.3, estimations provided by GAP are in general close to the observed durations. Forecasts with a higher probability are mostly overestimations, i.e., the predicted time for the next activity is usually higher than the actual time a contributor usually takes. For lower probabilities, most of the observed differences between predictions and actual observations are slight underestimations. When we consider the scenario of identifying people getting close to becoming inactive, overestimation or underestimation is not a real issue. Indeed, since underestimations mostly occur for lower probabilities, they mainly affect short-term predictions (e.g., predicting two days of inactivity instead of three). Overestimations are a priori of no consequence since the activity will be observed before the predicted date.

Project managers can decide on the most adequate abandonment risk policy for dealing with inactive contributors. For instance, some managers might prefer overestimating in order to have some leeway before contacting (and possibly bothering) an inactive contributor. Others might prefer to underestimate the predictions to mitigate the loss of certain contributors as early as possible. Either way, GAP can be configured to give a better estimation based on each project manager's specific policy.

Although a project manager could use the forecasting model on git commit activity for the aforementioned reasons, git commits constitute only a subset of all activity that could be made by project contributors. The forecasting model can be adapted to take into account activities at another level of granularity (e.g., pull requests instead of commits), other technical activities (e.g., code reviews, issue triaging), or even social activities (e.g., end-user support, discussions on mailing list). The model can be easily adapted to consider individually each of these types of activities. However, it would fail to deal with multiple simultaneous activities. The reason is that the used statistical Kaplan-Meier estimator is not able to handle competing events and competing risks: "*this method can handle only one single event at a time: all other events are treated as censored observations and the complement of the Kaplan-Meier estimate (1-KM) is interpreted as the probability of the event of interest in a hypothetical world in which the competing event does not exist*" [39]. Consequently, a promising future work is to explore different techniques (such as the Cumulative Incidence Function [40]) that are

able to deal with competing events and, by extension, deal with simultaneous activities of different types.

Not only can the model be used for real time monitoring of a project or an ecosystem community, but it could also lead to interesting insights in the context of an *a posteriori* analysis. Indeed, even if the proposed model is a predictive model, the underlying statistical technique of survival analysis is essentially descriptive. The study of the evolution of the survival function of contributors through time constitutes an interesting research track, which can be useful to understand the evolution of contributors' behaviour (e.g., contribution phases or cycles), or the evolution of contribution profiles (e.g., an occasional contributor becoming regular, or vice-versa). At ecosystem scale, such a study could provide useful insights, e.g., on inter-project migrations or abandonment, as "*a dead or dying project can also be indicated [...] through lost energy from formerly involved contributors, who may have moved on to other projects that better capture their interest*".[6]

This paper mainly focused on "when" the next commit activity will take place, and not on the reasons for such an activity. While from a practical point of view, more advanced models based on Accelerated Failure Time and Proportion Hazard Models have been discarded for practical reasons (because of the amount of data needed to train and use them), such models may be particularly interesting in the context of an *a posteriori* analysis. Since they can take into account the effect of covariates on the probability of occurrence of an event, they could lead to interesting insights to identify and understand the factors that could influence the commit activity of contributors.

7. Threats to Validity

Following the structure recommended by Wohlin et al. [41] we discuss the threats that might affect the validity of our findings, and how we have tried to mitigate them.

Construct validity concerns the relation between the theory behind the experiment and the observed findings. In our work, the main sources of such threats are related to distorted git commit histories due to the presence of *micro-commits* and *commit squashing* [30].

[6] `github.com/todogroup/todogroup.github.io/blob/master/content/en/guides/shutting-down.md`

The practice of micro-commits tends to result in sequences of many small commits that should be considered together as single atomic changes. Micro-commits can lead to overestimating the activity and productivity of a contributor. Since micro-commit sequences tend to take place in short time spans (e.g., the commits are spread over not more than a couple of hours), our findings should not be affected by their presence, as our model relies on the duration between *days* of commit activity. As such, even a huge number of commits contributed during the same day will still be counted as a single event (one day of commit activity by the contributor).
The practice of commit squashing corresponds to the *a posteriori* grouping of several commits. Depending on a project's policy, such commit squashes are mainly observed in pull requests. The rationale is to deliberately group all commits related to a contribution into a single commit in order to ease the reading of the project's git history. Commit squashing inevitably leads to an underestimation of the actual contributions, as it implies the removal of potentially many commits. Unfortunately, there is no reliable technique to identify and/or split these commits to retrieve the original ones. However, it should have little influence on the accuracy of the predictions made by our model as long as squashing is consistently performed on the git history of a given contributor within a given project.
Another threat to construct validity stems from the presence of contributors with multiple identities (different git "author" field values), since the commit activity of a given individual could be spread over these different identities. To mitigate this threat, we followed an approach similar to Avelino et al. [4] by merging the multiple identities using GitHub usernames in our dataset, as explained in Section 4. It cannot be excluded that some GitHub accounts are shared by multiple contributors (false positives). In that case, GAP reports on the combined activity of this account instead of the individual activity of each contributor using it. It cannot be excluded either that some contributors use multiple GitHub accounts at the same time (false negatives). In that case, GAP reports on the activity of each account individually, and not on the combined activity.

Internal validity concerns choices and factors internal to the study that could influence the observations we made. The main threat comes from the range of observations selected to feed to the model. We decided to rely on a variable time period containing exactly n observations to ensure that the model is trained with a fixed amount of data regardless of a contributor's

activity frequency. Choosing a fixed time period (e.g., n days or weeks) might lead to either a large or a small number of observations, depending on the contributor's activity frequency. The choice of $n = 20$ is discussed and supported in Section 4.2, where we showed the impact of this parameter on the concordance of the model.
While the technique of survival analysis makes no assumption on the unit of time required to measure time to event, the choice of that unit has a direct consequence of how commits are aggregated and fed to the model. We selected *days* as unit of time because it is a good compromise between practicability and accuracy. A larger granularity (e.g., *weeks*) would lead to more accurate predictions but provide less interesting insights for project managers. A smaller granularity (e.g., *hours*) would lead to less accurate results, without making them substantially more informative from a practical point of view.

Conclusion validity concerns the degree to which the conclusions we derived from our data analysis are reasonable. Our conclusions are mostly based on the results of the validation of the model. Since we carefully validated the model following a rolling-origin/walk-forward validation method [37] based on observed historical data, the conclusions we have drawn are unlikely to be affected by such threats.

External validity concerns whether the results and conclusions can be generalized outside the scope of this study. The main threat to external validity relates to the dataset we used to validate the model. To alleviate a potential selection bias related to the profile of contributors and projects, we selected a large dataset containing a wide variety of projects and contributors. The fact that the selected projects are all distributed through a package registry constitutes a possible bias. Indeed, one could expect these projects to be more mature, exhibiting more stable and less impulsive activity patterns than projects being in an early development phase. We are confident that the selected dataset is sufficiently heterogeneous in order not to affect the conclusions obtained, but we do not claim that it is representative of all possible git projects and contributors.
Although the technique used in this paper can be easily transposed to any type of activity and any type of version control system, we cannot guarantee that it will result in accurate predictions. For example, git repositories associated with non-code-based, personal or experimental projects may exhibit totally different activity patterns that cannot be captured by the model. The

same argument also holds for other types of activity, such as pull requests, code reviews, social communication, etc. Other version control systems may give rise to a different notion of contributor activity. Due to its decentralized nature, git encourages the creation of commits regardless of the degree of completion of the work performed in the local branch. In contrast, centralised systems such as Subversion encourage contributors to commit changes to a repository only when these changes reflect a single purpose (fixing of a specific bug, addition of a new feature, etc.). This difference in how work is committed to the repository may influence the observed frequency of contributions, and can therefore affect the generalization of our findings.

8. Conclusion

Keeping track of the presence or lack of activity of project contributors can be very important for the project's sustainability. Existing methods and solutions follow an *a posteriori* approach, since they provide information and insights on the past project activity. In practice, community managers also need indicators of potential risks stemming from contributors abandoning their projects. Focusing on the specific activity of contributing to git software repositories, there are currently no techniques or automated tools for predicting the future commit activity of project contributors.

This paper proposes a technique and associated tool to address these issues. In doing so, we hope to enable project managers and practitioners to monitor the future development activity in their git repositories and deal with contributor abandonment risks early, before their departure becomes potentially detrimental to the project. We presented a probabilistic forecasting model to estimate when the next commit activity of a given git contributor will occur. This model relies on survival analysis, a statistical technique that we used to estimate a contributor's probability of committing again, based on the past durations between days of observed commit activity for that contributor. The model relies on the last 20 observed durations between days of activity per contributor, which presents several benefits: (i) it does not require to collect or process a large amount of data; (ii) little prior knowledge is required, making it applicable to short-lived as well as long-lived projects; and (iii) since the analysis requires not more than a few seconds per project repository, it can be used for real-time monitoring, even for large communities of project contributors.

We evaluated the model on more than 7.5K git repositories corresponding to a wide variety of projects. We showed that estimations provided by our approach are generally close to the observed durations. By analysing the accuracy of the prediction model, we found that forecasts with a higher probability are mostly overestimations, while most of the observed differences between predictions for lower probabilities and actual observations are slight underestimations. We found that relying on the last $n = 20$ observed durations between days of activity provides the results with the lowest deviation.

To showcase the practicality of the model, we operationalised it through the Git Activity Predictor (GAP), an open source command-line utility. This tool enables practitioners and researchers to benefit from the proposed forecasting model and to apply it on sets of git repositories. GAP has built-in support for identity mapping to anonymise contributors and to merge identities (e.g., to deal with multiple identities, or to combine bot activity), thus both preserving privacy and providing accurate results. It supports different output formats, including a bar activity chart visualisation of the recent and predicted commit activities of contributors. GAP has also been integrated as part of the open source GrimoireLab software development analytics platform.[7]

We plan to evaluate and extend the probabilistic model and tool for other types of project forecasting activities (e.g., issue report interaction, code review or mailing list activity), and to combine different types of activity to better understand and predict the activity dynamics of individual project contributors. In addition, the model could be extended to the team level, to understand and forecast activity for groups of collaborating contributors within the same project.

Acknowledgments

This research was supported by the Fonds de la Recherche Scientifique – FNRS under Grants number T.0017.18, R.60.04.18.F (FRQ-FNRS collaborative research project "SECOHealth") and O.0157.18F-RG43 (Excellence of Science project "SECO-Assist").

[7] `chaoss.github.io/grimoirelab/`

References

[1] K. Crowston, J. Howison, The social structure of free and open source software development, First Monday 10 (2) (2005). doi:10.5210/fm.v10i2.1207.

[2] D. Riehle, P. Riemer, C. Kolassa, M. Schmidt, Paid vs. volunteer work in open source, in: 2014 47th Hawaii International Conference on System Sciences, 2014, pp. 3286–3295. doi:10.1109/HICSS.2014.407.

[3] L. F. Dias, I. Steinmacher, G. Pinto, Who drives company-owned OSS projects: internal or external members?, Journal of the Brazilian Computer Society 24 (1) (2018). doi:10.1186/s13173-018-0079-x.

[4] G. Avelino, E. Constantinou, M. T. Valente, A. Serebrenik, On the abandonment and survival of open source projects: An empirical investigation, in: Int'l Symp. Empirical Software Engineering and Measurement (ESEM), IEEE, 2019. doi:10.1109/ESEM.2019.8870181.

[5] P. C. Rigby, Y. C. Zhu, S. M. Donadelli, A. Mockus, Quantifying and mitigating turnover-induced knowledge loss: case studies of Chrome and a project at Avaya, in: Int'l Conf. Software Engineering, ACM , 2016, pp. 1006–1016. doi:10.1145/2884781.2884851.

[6] J. Gamalielsson, B. Lundell, Sustainability of open source software communities beyond a fork: How and why has the LibreOffice project evolved?, J. Systems and Software 89 (2014) 128 – 145. doi:10.1016/j.jss.2013.11.1077.

[7] I. Steinmacher, T. U. Conte, C. Treude, M. A. Gerosa, Overcoming open source project entry barriers with a portal for newcomers, in: International Conference on Software Engineering, ACM, 2016, pp. 273–284. doi:10.1145/2884781.2884806.

[8] E. Constantinou, T. Mens, An empirical comparison of developer retention in the RubyGems and npm software ecosystems, Innovations in Systems and Software Engineering 13 (2) (2017) 101–115. doi:10.1007/s11334-017-0303-4.

[9] F. Fronchetti, I. Wiese, G. Pinto, I. Steinmacher, What attracts newcomers to onboard on OSS projects? tl;dr: Popularity, in: Open Source Systems, Springer, 2019, pp. 91–103.

[10] C. Miller, D. G. Widder, C. Kästner, B. Vasilescu, Why do people give up FLOSSing? A study of contributor disengagement in open source, in: Open Source Systems, Springer, 2019, pp. 116–129.

[11] I. Qureshi, Y. Fang, Socialization in open source software projects: A growth mixture modeling approach, Organizational Research Methods 14 (1) (2011) 208–238. doi:10.1177/1094428110375002.

[12] J. Coelho, M. T. Valente, Why modern open source projects fail, in: Joint Meeting on Foundations of Software Engineering (FSE), ACM, 2017, pp. 186–196. doi:10.1145/3106237.3106246.

[13] I. Steinmacher, M. A. Graciotto Silva, M. A. Gerosa, D. F. Redmiles, A systematic literature review on the barriers faced by newcomers to open source software projects, Information and Software Technology 59 (C) (2015) 67–85. doi:10.1016/j.infsof.2014.11.001.

[14] B. Lin, G. Robles, A. Serebrenik, Developer turnover in global, industrial open source projects: Insights from applying survival analysis, in: Int'l Conf. Global Software Engineering (ICGSE), 2017. doi:https://doi.org/10.1109/ICGSE.2017.11.

[15] G. Iaffaldano, I. Steinmacher, F. Calefato, M. Gerosa, F. Lanubile, Why do developers take breaks from contributing to OSS projects? a preliminary analysis, in: Int'l Workshop on Software Health (SoHeal), IEEE, 2019, pp. 9–16. doi:10.1109/SoHeal.2019.00009.

[16] E. Constantinou, T. Mens, Socio-technical evolution of the Ruby ecosystem in GitHub, in: Int'l Conf. Software Analysis, Evolution and Reengineering (SANER), 2017, pp. 34–44. doi:10.1109/SANER.2017.7884607.

[17] J. J. Amor, G. Robles, J. M. Gonzalez-Barahona, Effort estimation by characterizing developer activity, in: Int'l Workshop on Economics driven software engineering research, ACM, 2006, pp. 3–6. doi:10.1145/1139113.1139116.

[18] G. Gousios, E. Kalliamvakou, D. Spinellis, Measuring developer contribution from software repository data, in: Int'l Working Conf. Mining Software Repositories (MSR), ACM, 2008, pp. 129–132. doi:10.1145/1370750.1370781.

[19] A. Mockus, D. M. Weiss, P. Zhang, Understanding and predicting effort in software projects, in: Int'l Conf. Software Engineering, 2003, pp. 274–284. doi:10.1109/ICSE.2003.1201207.

[20] Y. Weicheng, S. Beijun, X. Ben, Mining github: Why commit stops – exploring the relationship between developer's commit pattern and file version evolution, in: Asia-Pacific Software Engineering Conf. (APSEC), Vol. 2, 2013, pp. 165–169. doi:10.1109/APSEC.2013.133.

[21] O. Aalen, O. Borgan, H. Gjessing, Survival and Event History Analysis: A Process Point of View, Springer, 2008. doi:10.1007/978-0-387-68560-1.

[22] I. Samoladas, L. Angelis, I. Stamelos, Survival analysis on the duration of open source projects, Information and Software Technology 52 (9) (2010) 902 – 922. doi:10.1016/j.infsof.2010.05.001.

[23] P. Kyriakakis, A. Chatzigeorgiou, Maintenance patterns of large-scale PHP web applications, in: Int'l Conf. Software Maintenance and Evolution, 2014, pp. 381–390. doi:10.1109/ICSME.2014.60.

[24] A. Decan, M. Goeminne, T. Mens, On the interaction of relational database access technologies in open source Java projects, in: A. Bagge, T. Mens, H. Osman (Eds.), Post-proceedings of the 8th Seminar on Advanced Techniques and Tools for Software Evolution, Vol. 1820, CEUR-WS.org, 2017, pp. 26–35.

[25] A. Decan, T. Mens, P. Grosjean, An empirical comparison of dependency network evolution in seven software packaging ecosystems, Empirical Software Engineering 24 (2019) 381—416. doi:10.1007/s10664-017-9589-y.

[26] C. Bird, A. Gourley, P. Devanbu, A. Swaminathan, G. Hsu, Open borders? Immigration in open source projects, in: Int'l Workshop on Mining Software Repositories (MSR), 2007. doi:10.1109/MSR.2007.23.

[27] R. Shumway, D. Stoffer, Time Series Analysis and Its Applications With R Examples, 3rd Edition, Springer, 2011. doi:10.1007/978-1-4419-7865-3.

[28] D. Collett, Modelling Survival Data in Medical Research, Second Edition, Chapman & Hall/CRC Texts in Statistical Science, Taylor & Francis, 2003.
URL https://books.google.be/books?id=Hm_FfYRHCuAC

[29] E. L. Kaplan, P. Meier, Nonparametric estimation from incomplete observations, J. American Statistical Association 53 (282) (1958) 457–481. doi:10.1080/01621459.1958.10501452.

[30] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, D. Damian, The promises and perils of mining GitHub, in: Int'l Conf. Mining Software Repositories, ACM, 2014, pp. 92–101. doi:10.1145/2597073.2597074.

[31] A. Nesbitt, B. Nickolls, Libraries.io open source repository and dependency metadata (2018). doi:10.5281/zenodo.1196312.

[32] G. Robles, J. M. González-Barahona, Developer identification methods for integrated data from various sources, in: Int'l Conf. Mining Software Repositories, ACM, 2005. doi:10.1145/1082983.1083162.

[33] C. Bird, A. Gourley, P. Devanbu, M. Gertz, A. Swaminathan, Mining email social networks, in: Int'l Conf. Mining Software Repositories, ACM Press, 2006, pp. 137–143. doi:10.1145/1137983.1138016.

[34] M. Goeminne, T. Mens, A comparison of identity merge algorithms for software repositories, Science of Computer Programming 78 (8) (2013) 971–986. doi:10.1016/j.scico.2011.11.004.

[35] E. Kouters, B. Vasilescu, A. Serebrenik, M. G. J. van den Brand, Who's who in Gnome: using LSA to merge software repository identities, in: Int'l Conf. Software Maintenance, IEEE, 2012, pp. 592–595. doi:10.1109/ICSM.2012.6405329.

[36] H. Hofmann, K. Kafadar, H. Wickham, Letter-value plots: Boxplots for large data, Tech. rep., had.co.nz (2011).

[37] R. M. Stein, Benchmarking default prediction models: Pitfalls and remedies in model validation, Journal of Risk Model Validation 1 (1) (2007) 77–113. doi:10.21314/JRMV.2007.002.

[38] D. Wilks, Forecast verification, in: Statistical Methods in the Atmospheric Sciences, Vol. 100 of International Geophysics, Academic Press, 2011, Ch. 8, pp. 301 – 394. doi:10.1016/B978-0-12-385022-5.00008-7.

[39] M. Noordzij, K. Leffondré, K. J. van Stralen, C. Zoccali, F. W. Dekker, K. J. Jager, When do we need competing risks methods for survival analysis in nephrology?, Nephrology Dialysis Transplantation 28 (11) (2013) 2670–2677. doi:10.1093/ndt/gft355.

[40] P. C. Austin, D. S. Lee, J. P. Fine, Introduction to the analysis of survival data in the presence of competing risks, Circulation 133 (6) (2016) 601–609. doi:10.1161/CIRCULATIONAHA.115.017719.

[41] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, A. Wesslen, Experimentation in Software Engineering - An Introduction, Kluwer, 2000.