# On the Influence of Maintenance Activity Types on the Issue Resolution Time

Alessandro Murgia
University of Antwerp
Antwerp, Belgium
alessandro.murgia@uantwerpen.be

Giulio Concas
University of Cagliari, Italy
Cagliari, Italy
concas@diee.unica.it

Roberto Tonelli
University of Cagliari, Italy
Cagliari, Italy
roberto.tonelli@diee.unica.it

Marco Ortu
University of Cagliari, Italy
Cagliari, Italy
marco.ortu@diee.unica.it

Serge Demeyer
University of Antwerp
Antwerp, Belgium
serge.demeyer@uantwerpen.be

Michele Marchesi
University of Cagliari, Italy
Cagliari, Italy
michele@diee.unica.it

## ABSTRACT

The ISO/IEC 14764 standard specifies four types of software maintenance activities spanning the different motivations that software engineers have while performing changes to an existing software system. Undoubtedly, this classification has helped in organizing the workflow within software projects, however for planning purposes the relative time differences for the respective tasks remains largely unexplored.

In this empirical study, we investigate the influence of the maintenance type on issue resolution time. From GitHub's issue repository, we analyze more than 14000 issue reports taken from 34 open source projects and classify them as corrective, adaptive, perfective or preventive maintenance. Based on this data, we show that the issue resolution time depends on the maintenance type. Moreover, we propose a statistical model to describe the distribution of the issue resolution time for each type of maintenance activity. Finally, we demonstrate the usefulness of this model for scheduling the maintenance workload.

## Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*Process metrics*; K.6.3 [**Management of Computing and Information System**]: Software Management—*Software maintenance*

## General Terms

Measurement

## Keywords

Software Maintenance, Issue repository, Issue resolution-time, Empirical software engineering

## 1. INTRODUCTION

Software maintenance is a key ingredient of any successful software project, certainly in modern development processes with their emphasis on iterative and incremental development. Already in 1976, Swanson introduced the first classification of software maintenance types to aid researchers and practitioners in describing the activities they are performing [15]. Swanson's classification has later been extended and today is incorporated into the ISO/IEC 14764 standard which defines four types of activities: (a) corrective maintenance is devoted to removing bugs; (b) adaptive maintenance is related to adding new features; (c) perfective maintenance deals with activities to enhance performance; (d) and preventive maintenance takes into account changes on software finalized to avoid future bugs.

In this paper, we investigate the relationship between the type of maintenance activity (as defined by the ISO/IEC 14764) and the time required for finishing work items. For our empirical study, we rely on GitHub's issue repository that hosts issues that are ascribable as requiring corrective, adaptive, perfective or preventive maintenance. We try out several statistical distributions to see which ones are suitable for describing the distribution of the issue resolution time for each type of maintenance activity.

This work follows the Goal-Question-Metric paradigm [16]. The *goal* of this study is to evaluate the impact of the type of maintenance activity in the issue resolution process. The *focus* is to evaluate how the metric issue resolution time changes for the corrective, adaptive, perfective and preventive maintenance. For this purpose, we investigate which statistical model is suitable for describing the resolution time distribution. The *viewpoint* is that of issue triager and researchers. The first one, scheduling maintenance workload, is interested in evaluating the resolution time per maintenance activity. The latter, studying software maintenance based on data mined from software repositories, is interested on how to exploit the maintenance type for building better predictive models. The *environment* of this paper regards the issue tracking repository with 14000 issues taken from 34 open source projects. To achieve our purpose, we pursue the following research questions:

- *RQ1: Is the issue resolution time dependent on maintenance type?*

- *RQ2: Is it possible to model the distribution of the issue resolution times with respect to the type of maintenance considered?*

This paper is organized as follow. In section 2, we show how maintenance types and issue resolution time are treated in literature. In section 3, we provide the required background related to distribution functions. In section 4 we describe the data used for the empirical study. In section 5 we present our results and in section 6 we provide an operative example on how to use these findings. The threats to validity are reported in section 7. Finally, in section 8 we wrap up the work.

## 2. RELATED WORK

Software maintenance types and issue resolution time are topics commonly analyzed in software engineering. However, with rare exceptions [10, 17], these topics are never analyzed together. This section presents how the maintenance activities have been considered in context of issue resolution time.

The bug-oriented repository Bugzilla played a key-role for the analysis of the time spent on the maintenance activity. Demeyer et al. showed that this repository is the most common used in mining software conferences [6]. Bugzilla has been widely studied because it stores bugs of Eclipse and Mozilla, two of the most common case studies [12, 2, 7, 20]. Other bug-oriented repositories employed for these type of study have been the FreeBSD's bug repository [3] and the Google Code's bug tracker [2].

Panjer predicts the bug fixing time analyzing the bug reports of the project Eclipse [12]. He points out that the resolution time of bugs with severity blocker, critical and trivial is lower than the resolution time of enhancements. Similar analysis was performed by Zhang et al. to explore developer's delays during bug fixing for three projects of Eclipse [20]. They use Bugzilla's severity to distinguish between bug and enhancement. Investigating the factors that are relevant for the delay of the triaging, they discover that developers who fix a bug are faster in updating the bug status on the repository.

Bhattachary and Neamtiu investigate which factors influence the bug fixing time using the bug reports of Chrome, Eclipse and three products from the Mozilla project [2]. They build a bug-fix time prediction model and demonstrated that several models proposed in literature cannot be replicated when adopted on large projects used in bug studies. Their results indicate that the predictive power of such models is between 30% and 49% and they conclude that there is a need for more independent variables to construct a prediction model. Similar analysis is performed by Giger et al. [7] and Bougie [3]. In Giger et al. [7] they use the projects Eclipse, Mozilla, and Gnome, in Bougie et al. [3] they use the FreeBSD's bug repository. Unfortunately, in these cases the authors do not make any distinction related to the type of issue involved (e.g.; enhancement, bug), so it is not possible to make any conjecture related to the type of maintenance they analyzed.

Due to the bug-oriented nature of the repository, the previous works are mainly focused on the issue resolution time related to the corrective maintenance.

One step toward the analysis of the relationship between maintenance type and issue resolution time is done by Weiss et al. and Mockus and Votta [17, 10].

Weiss et al. take into account issues labeled as bug and feature, namely issues that can be assumed to be related to corrective and adaptive maintenance [17]. This study does not have any analysis on perfective and preventive maintenance. Moreover, the number of issues ascribable as corrective and maintenance activity are only 273 and all of them belong to just one project. Finally, they only use mean and standard deviation to characterize the difference between the issue resolution time of bugs and features. These statistics are not reliable to characterize right-skewed or fat-tail distributions such as the issue resolution time distributions. From the comparison of the mean values, they show that corrective maintenance is faster than adaptive maintenance.

Mockus and Votta analyze the issue resolution time associated to the corrective, adaptive, and perfective maintenance activities [10]. In the study, the type of maintenance activity is only inferred by the authors reading the developer commit message. Indeed, the issue tracking system used in the study did not have any field to specify the type of issue. Moreover, the analysis is limited to only one commercial switching software. The study uses only plots to show how the issue resolution time changes according to the type of maintenance. Comparing the plots, the authors highlights that corrective changes have the shortest resolution time, followed by perfective changes. Unfortunately, the analysis does not provide many details related to the distribution of the issue resolution time.

The last two studies, even if they consider more than one type of maintenance, are not focused on how to model the distribution of the issue resolution time.

There are two main differences between our study and the previous ones.

- We take into account all categories of maintenance reported in ISO/IEC 14764, namely corrective, preventive, perfective and adaptive maintenance [1]. Our investigation, which involves 34 projects, uses the GitHub's repository where developers keep track of any maintenance activity performed in the system.
- We model the issue resolution time using statistical distribution. The benefits of this analysis is not only the possibility to distinguish among the different maintenance activities, but also to provide reliable estimates of the issue resolution times.

## 3. DISTRIBUTION FUNCTIONS

In this section, we present the lognormal and Weibull distribution functions. Such distributions are suited to model sample data presenting leptokurtic behavior (a "fat tail" distribution) and were already used for modeling software metric distributions [4, 5].

Equations 1 and 2 describe respectively the mathematical function of lognormal and Weibull distributions. Here, the variable $x$ represents the issue resolution time of a particular maintenance activity. Using these functions we can model the distribution of the issue resolution time for any maintenance type independently.

In the rest of paragraph we briefly discuss how these distributions might be linked to the issue resolution time generation. Finally, we describe how to represent such distributions.

$$F(x) = \frac{1}{\sqrt{2\pi}\sigma x} \cdot e^{-\frac{lnx-\mu}{2\sigma^2}} \quad (1)$$

$$F(x) = 1 - e^{(\frac{x}{\lambda})^k} \quad (2)$$

## 3.1 Lognormal

The lognormal distribution has been used to analyze the distribution of lines of code in software systems which presents a fat tail [20]. The associated generative process considers *units* with a *property* characterized by a value, like classes with a certain number of lines of code. The units are randomly selected to increment their property value, and such increment is directly proportional to the actual property value. It can be demonstrated that such process produces a statistical distribution with a fat tail, which can appear as a power law with a cut-off at large values.

In our context, the process leading to a lognormal distribution of the issue resolution time may be obtained considering issues as units and the property as the fixing time. Developers start working for addressing an issue when it enters the issue tracking system, in the meanwhile new issues are introduced. Issues are handled independently and by different developers, and it is likely that difficult issues are managed in multiple working sessions, so that when an issues is tackled, developers work on it for a certain time, and then stop, leaving the remaining work to another session. The time needed to work on an issue in the following working session can be considered roughly proportional to the time accumulated in the previous sessions, since it is proportional to the overall complexity and difficulty associated to the issue. Furthermore issues can be considered as randomly selected by developers, since they are randomly introduced in the issue tracking system as they are discovered. This rough hypothesis could explain the good fitting provided by the lognormal distribution.

## 3.2 Weibull

The Weibull distribution models a system with an initially fixed number of *components* with a certain *failing rate*. Eventually, the fraction of failed components saturates to one. Also this process can generate a statistical distribution providing a power law fat tail.

For what concerns the process leading to a Weibull distribution in our dataset, the issues represent the components and the fixing time represent the average failing time. Since this may vary randomly, there will be issues quickly solved, corresponding to components failing rapidly, and there will be long lasting issues, corresponding to more robust components, whose number decreases as the duration increases, satisfying the hypotheses of the Weibull model.

## 3.3 Cumulative Complementary Distribution Function

In probability theory and statistics, the cumulative distribution function (CDF) describes the probability that a real-valued random variable X with a given probability distribution will be found at a value less than or equal to x. In the case of a continuous distribution, cumulative complementary distribution function (CCDF) is its complementary defined as 1-CDF. The plot of CCDF is useful whenever we are dealing with distributions right-skewed and with fat tail. Indeed, such distributions cannot be characterized by statistics like mean and standard deviation [11].

## 4. EXPERIMENTAL SETUP

This section presents in 4.1 the dataset we use, in 4.2 how we map issues to maintenance activity and finally in 4.3 how we measure the issue resolution time.

## 4.1 Dataset

For our empirical study we use the issue reports recorded in the GitHub[1] repository. We use this repository since it host many open source projects, it is documented[2] and it is publicly available for replication studies [8]. In Figure 1, we report the portion of interest of dataset schema. Issue reports are characterized by two elements:

- Events. Events have *action* and *action specific* fields used to describe respectively the type of event (e.g. the time the issue was closed or the time a pull request pointed by the issue was merged).

- Label. Users may add a label to an issue report *to signify priority, category, or any other information that you and your fellow maintainers find useful.* From labels like "bug", "feature" it is possible to infer the type of maintenance performed (e.g.; bug fixing, feature introduction). We assume that these labels are accurate since they are *freely* provided by the developer who was aware of the activity performed. This labels are not attached with default values (like priority normal in Bugzilla) but are spontaneously added by developers whenever they want to add a relevant information for the issue handling.

Since we are interested in maintenance activity that are complete, we select only closed issues, namely we removed every issue report without a closed event.

Table 1 shows the dataset statistics: almost 50% of issues are labeled, within this 50%, almost 30% are labeled with keyword we can relate to the maintenance activity that was performed. For the empirical study, we use 34 (out of 90) projects since they have at least one issue with a label related to a maintenance activity. Our sample take into account projects developed with different programming languages (e.g., C, Java, ...) and with few or lot of developers involved (up to 95). From this projects we collect more than 14,000 issues.

## 4.2 Mapping issue to the maintenance type

Issue reports in GitHub may refer to any type of maintenance. Classification of these issues is not easy since none of the authors contributed to any of these projects. For this reason, we relied on our developer experience to classify the issue according to the label they have. We believe that labels used by developers in GitHub are reliable since assigned

[1]GitHub is a web-based service providing a collaborative software development environment and a social network for developers

[2]http://ghtorrent.org/relational.html

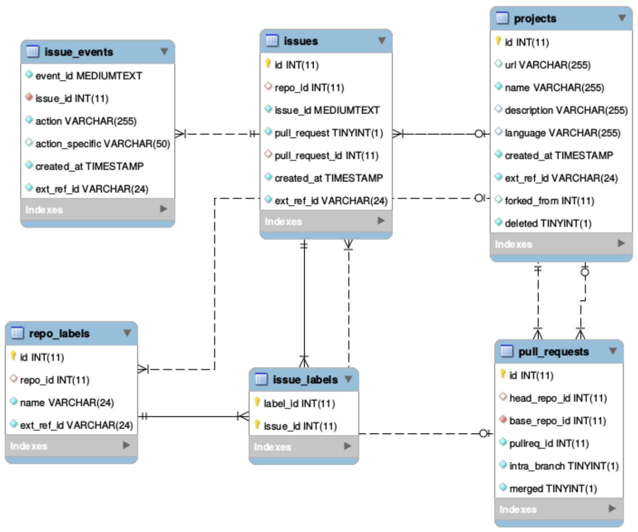


Figure 1: Github dataset schema.

Table 1: *GitHub dataset statistics.*

| | |
|---|---|
| Number of projects | 90 |
| Number of projects with labeled issues | 34 |
| Number of different languages per projects with labeled issues | 12 |
| All Issues | ≃ 100000 |
| Issues with at least one label | ≃ 50000 |
| Issues related to *maintenance* | ≃ 39000 |
| Issues related to *maintenance* with status *closed* | 14298 |
| Mean of labeled issues per projects | 44 |
| Standard Deviation of labeled issues in projects | 72 |
| Min labeled issues per projects | 1 |
| Max labeled issues per projects | 345 |
| Min developers per projects with labeled issues | 1 |
| Max developers per projects with labeled issues | 95 |

by the person who actually performed the maintenance activity. The first two authors manually inspected all issue labels in the dataset and, where feasible, they mapped the label to a particular maintenance activity. As guideline for the labeling we refer to several papers presented in literature [1, 10, 15]. We used this approach since it was already successfully adopted by Mockus and Votta and Purushothaman and Perry [10, 13]. In these cases the authors determine the type of maintenance analyzing the text message submitted by the developer.

We did not map any label that was not clear or was not ascribable to a maintenance type. For example, labels like *GUI* and *Mobile* are generic an do not give hints on the type of maintenance performed, hence were classified in a separate category. For the same reason, we did not consider issues that use labels ascribable to different type of maintenance. Mapping of these labels as either a corrective, perfective, adaptive and preventive would have introduced a threats to validity in our analysis.

Table 2 shows the mapping with the percentages of maintenance types.

Table 2: *Table of mapping issue-labels/maintenance-type*

| Maintenance type | Labels | Samples |
|---|---|---|
| Corrective | Bug, Bugs, Bug Report, non critical bug, Type-Bug,Critical,bug, Defect, Framework bug, Rails bug, browser bug, Type-Defect,Crashes - 500 errors | 6754 (47.9%) |
| Perfective | Enhancement, Type-Enhancement, Improvement,Type-Improvement,Cleanup refactoring, code refactoring | 2397 (17%) |
| Adaptive | Feature, Feature Request, New feature,Approved feature,Type-New Feature | 4723 (33.5%) |
| Preventive | Test, Testing Framework | 226 (1.6%) |
| Not Mapped (due to doubtful label) | Task, Suggestion, ... | 3149 (8% of all labeled issues) |
| Not Mapped (due to multi type of maintenance performed) | Third party issue, Status-Started, Cleaning, ... | 467 (1 % of all labeled issues) |

## 4.3 Issue Resolution Time

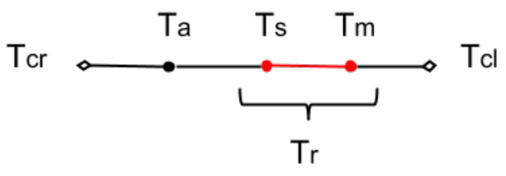
Figure 2 shows the typical issue timeline in GitHub:

- $T_{\mathrm{cr}}$ represents the time an issue is created.
- $T_{\mathrm{cl}}$ represents the time an issue is closed.
- $T_{\mathrm{a}}$ represents the time an issue is assigned to a developer.
- $T_{\mathrm{s}}$ is the time a developer subscribe that an issue as been assigned to him.
- $T_{\mathrm{m}}$ represent the time when an issue is merged in the repository, namely the local commit is merged in the remote repository.

In our analysis we are interested in analyzing the *working* time spent by the developer to resolve the issue. For this reason, we do not consider the triaging steps needed to assign the issue $T_{\mathrm{a}}$ - $T_{\mathrm{cr}}$, nor the steps between an issue is merged into the code base and confirmed as being closed $T_{\mathrm{cl}}$ - $T_{\mathrm{m}}$. We compute the issue resolution time as the difference between $T_{\mathrm{m}}$ and $T_{\mathrm{s}}$, namely we compute the difference between the first time the issue is subscribed and the last time the issue is merged in the repository. In this way we take into account also supplementary fixes due to re-opened bugs. Our assumption is that the developer subscribes for issue resolution when she is ready for the maintenance activity; whereas she merges the code change only when the maintenance activity is complete. We assume that such time-checkpoints are representative of the time spent to resolve the issue [3, 12].

## 5. RESULTS AND DISCUSSION

For each research question, we first discuss its motivation, followed by the approach we used and finally we present our findings.



$$Tr = Tm - Ts \tag{3}$$

Figure 2: Example of timeline for GitHub issue.

*RQ1. Is the issue resolution time dependent on maintenance type?*

**Motivation**. Weiss et al. reveal that JBoss's bugs are on average addressed quicker than new features [17]. Assuming that bugs and features are addressed with different type of maintenances, we may hypothesize that issue resolution time is influenced by the maintenance type.

A similar hint is provided by the relationship between code change and maintenance type. Hindle et al. show that large commits — commits that involve more than thirty files — are related more likely to perfective than to corrective maintenance [9]. Purushothaman et al. show that even in each file the percentage of lines of code modified is not equally distributed among the different types of maintenance activities [13]. Having evidence that type of maintenance "correlates" with the amount of files and lines of code handled, we hypothesize that the type of maintenance "correlates" with issue resolution time as well.

**Approach**. In order to detect differences among the resolution times belonging to different maintenance activities we adopted the Wilcoxon test [18, 14], which is non-parametric and thus can be used with no restrictions nor hypotheses on the statistical distribution of the sample populations. The test is suitable for comparing differences among the averages or the medians of two populations when their distributions are not gaussian. For the analysis, we use the one sided Wilcoxon rank sum test using the 5% significance level (i.e., p-value $< 0.05$) and we compare each resolution time dataset with all others datasets.

Table 3: *Wilcoxon test for groups split by maintenance, test column indicates if the median of the first group is greater or less than the second type(n.s stands for not statistically significant.)*

| Pair of Groups Compared | Test | p-value | Effect Size |
|---|---|---|---|
| Corrective vs Perfective | greater | 2.3E-5 | 0.024 |
| Corrective vs Adaptive | less | 1.9E-5 | 0.096 |
| Corrective vs Preventive | less | 2.4E-5 | 0.076 |
| Adaptive vs Preventive | less | 4.3E-3 | 0.049 |
| Adaptive vs Perfective | greater | n.s | 0.124 |
| Perfective vs Preventive | less | n.s | 0.144 |

**Findings. The issue resolution time depends on maintenance type**. Figure 3 reports resolution times boxplots in a logarithmic scale (time is expressed in days). The time distributions are right-skewed with values ranging across different orders of magnitudes, suggesting a non gaussian distribution of the data.

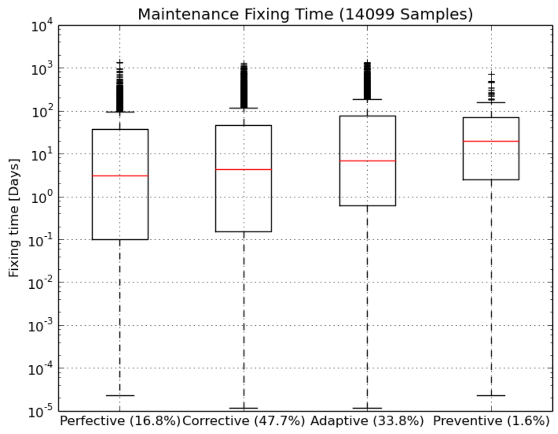


Figure 3: Resolution time in days grouped by maintenance type

Table 3 reports the result of the Wilcoxon test. The p-values obtained for all the couples, which are all below 0.005, indicating that issue fixing times are significantly different for different maintenance types. In particular the test performed for each couple shows to a high significance level that: **perfective maintenance is on average faster than corrective maintenance; corrective is on average faster than adaptive; adaptive is on average faster than preventive**. We conclude that different maintenance activities do have different issue resolution times.

*RQ2. Is it possible to model the distribution of the issue resolution times with respect to the type of maintenance considered?*

**Motivation**. RQ1 exhibits that the issue resolution time depends on the type of maintenance handled. Here, we investigate how to describe the distribution of the issue resolution time to highlight the differences among the maintenance types. A correct modeling is indeed crucial for scheduling the development activities and estimate them.

**Approach**. We analyze the lognormal and Weibull statistical distributions models. These distributions have been proved to be suitable to model software metric distributions [4, 5]. Moreover, in section 3 we explained which process may lead to such distributions in the context of the issue resolution time.

**Findings**.

In tables 4, 5 and 6 we report the lognormal and Weibull best fitting parameters with the significance levels, represented by the fitting coefficient $R^2$, along with the size of dataset used for the fitting. We do not report a similar table for preventive maintenance because the distribution consists of only 200 sample, thus it is not possible to repeat the same analysis with different sample size as for the other maintenance activities. We compute these parameters taking into account issues handled with corrective maintenance, but the same behavior is exhibited by the best fitting parameters associated to other maintenance types; here skipped for the sake of simplicity. These tables show that lognormal and Weibull have good fitting for the raw data in all cases, with a level of significance higher than 90%. The best fitting parameters are in agreement with the results of the Wilcoxon rank sum test (table 3). Moreover, such parameters are quite stable and do not vary with the considered sample size. This is a relevant point since it enables these models to be used also in the early stages of the project development when few hundreds of resolved issues are available. In our empirical study, just using a random sample of 3% (200 out of 6754 - number of issues for corrective maintenance) of the issues, we can predict the issue resolution time. In figures 4c, 4b, 4d and 4a we plot the Cumulative Complementary Distribution Function (CCDF) of issue resolution time. As we can see, the first 200 points are practically overlapping all other points as well as the Weibull curve.

For all these reasons, the **statistical distribution models are suitable to quantify how much the issue resolution times is influenced by the type of maintenance.**

From the comparison of tables 4, 5 and 6, we can highlight some differences between Weibull and lognormal models. The Weibull distribution shows less stable best fitting parameters compared to lognormal. However, it has a better fitting as confirmed by the coefficient $R^2$. This fact is highlighted also in figures 4c, 4b, 4d and 4a where the Weibull distribution exhibits a better overlapping with raw data. We must underline that our data is limited to few years. This introduces a upper cut off on resolution times, which cannot be longer than the time span analyzed. This may be the reason why the lognormal fails to fit well in the tail of the distribution, even if the fitting parameters are stable. We believe that in the case of data without such upper cut off the lognormal would provide a better fit and more accurate estimates. In fact the lognormal curve well overlaps the bulk of the distribution and mainly fails in the tail, for larger values, where it remains much higher than the data curve. But in this region data are sparse and rare, because of the finiteness of the dataset. In principle, with an infinite amount of data, the fat tail would present arbitrarily large values. Namely, if one could consider arbitrarily large datasets, there will be maintenance operations lasting an arbitrarily large amount of time. In case of finite dataset, data in the tail become sparse and the related complementary cumulative distribution will drop to zero faster than in the case if theoretically infinite dataset. This explain why the lognormal distribution best fitting, which suits for a theoretically infinite dataset with no cut off in the tail, fails in fitting properly the data in the tail while is very good for data in the bulk.

## 6. HOW TO USE STATISTICAL DISTRIBUTION MODELS

We present two practical examples to show the effectiveness of our model. In the first example we tackle the problem of estimating the time effort needed to resolve accumulated issues over time. Suppose a company started working on solving issues and collected issue resolution times. After (say) six months of work it accumulated a queue of unsolved issues during the last two weeks or the last month. We simulated this scenario using our data, considering the six months commits divided in two groups: "solved issues" (the set SI), and "accumulated issues" during last two weeks (the set AI), still to be solved. Using the set SI we estimated the Weibull parameters for the various kinds of maintenance. The set AI is composed by:

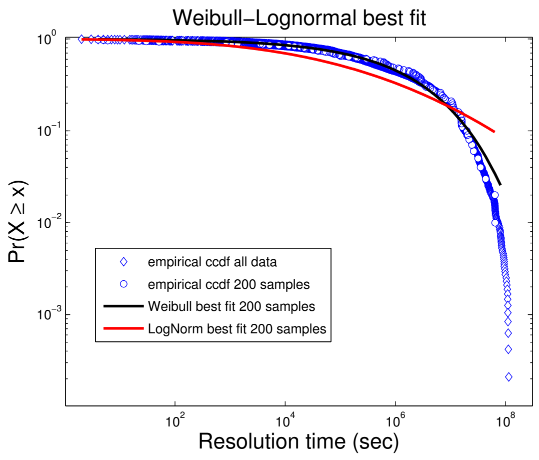
- Last two weeks: 8 corrective, 7 adaptive, 3 perfective, 0 preventive
- Last month: 15 corrective, 15 adaptive, 7 perfective, 0 preventive

From the knowledge of the Weibull CCDF we partitioned the set AI into bins, corresponding to percentage of issues multiple of ten, namely 10%, 20%, and so on. For each percentage we determined the issue resolution times for each kind of maintenance. For example, from the Weibull parameters for corrective maintenance we have that 10% of issues are solved in less than 540 seconds, 20% in less than 6400 seconds, and so on. This procedure corresponds to solve the inverse transform of the CCDF for issue resolution times at discrete value, which can be very practical. Next we used these data to statistically infer times needed for solving the issues. For example, 10% of the 8 corrective maintenance operations, will be solved in 540 seconds, so that this amount can be estimated multiplying 0.8 by 540 seconds. Then another 0.8, which together with the first makes the 20% of data, takes less than 6400 seconds. Thus we multiply 0.8 by 6400 and sum it up with the estimate obtained for the first 10%, and so on. We stop the count when we reach the 90%, since the Weibull CCDF goes to 100% when time is infinite.

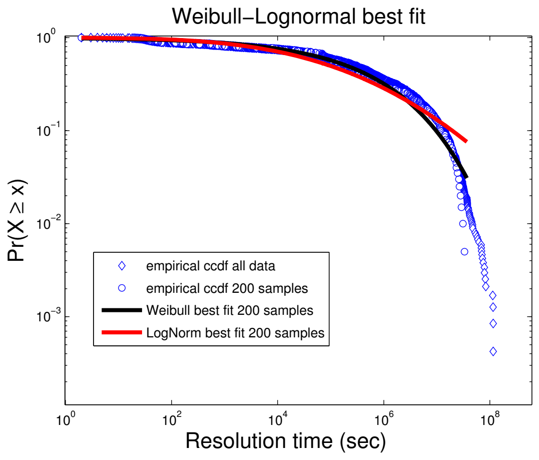
For the last two week data, the lower estimate provides 207 days for corrective maintenance, 251 days for the adaptive, 62 days for the perfective. Since we stop the count when we reach the 90%, these can be considered lower estimates. Next we randomly selected the issue resolution time for 8 corrective maintenance operations, for 7 adaptive maintenance operations, and 3 perfective maintenance operations from our dataset and summed up all the resolution times, and repeated the procedure three times. The results of the trials are in table 7.

In the second example the company wants to commit with customers requiring a certain amount of improvements, and needs to estimate the fraction of improvements required by customers that can be solved in a given amount of time, before committing itself. We simulated the number of possible improvements required by customers using our data, estimating the perfective maintenance operations requested in a month. We averaged this number over 10 months, and obtained 20 requests, which is suitable for a medium size company. Again we used the Weibull CCDF with parameters estimated by the set SI, and partitioned the number of perfective maintenance operation requested into bins corresponding to percentage of issues multiple of ten. We suppose the company wants to estimate the time for solving 70% of requests. From the Weibull CCDF best fitting we obtain, using the same procedure as above, an estimate of 43 days for solving 14 issues. Next we randomly selected the issue resolution time for 14 perfective maintenance operations from our dataset and summed up all the resolution times, and repeated the procedure three times. The results of the trials are reported in table 8.

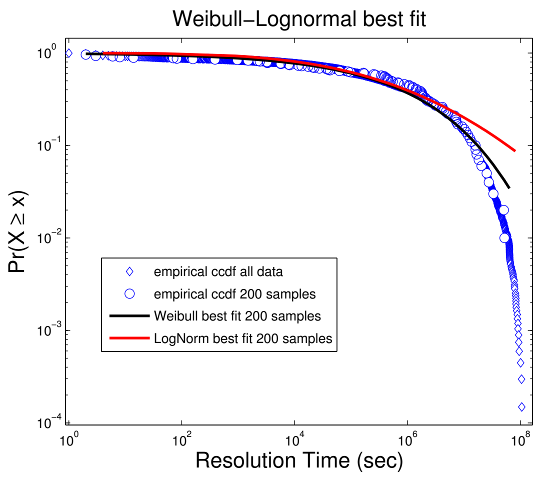
These three trials display a large variability which is intrinsic of data distributed according to a power-law in the tail. In fact, there are many issues with relatively low fixing times, but there are also issues with a very large fixing time, even if these latter are much less. Thus, the result of summing up all resolution times for 14 issues randomly selected



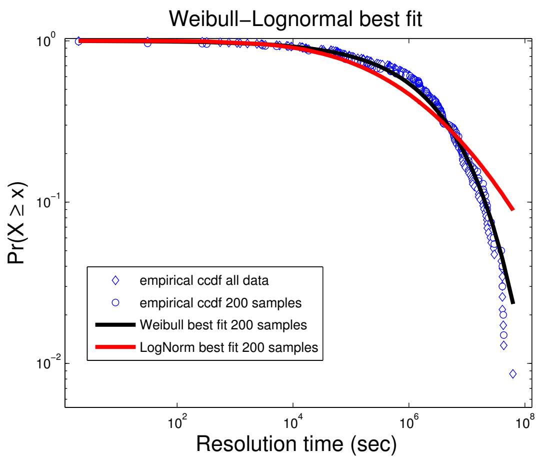
(a) Lognormal and Weibull CCDF fitting for corrective maintenance.



(b) Lognormal and Weibull CCDF fitting for perfective maintenance.



(c) Lognormal and Weibull CCDF fitting for adaptive maintenance.



(d) Lognormal and Weibull CCDF fitting for preventive maintenance.

Table 4: *Lognormal and Weibull fitting parameters for different sample size for corrective maintenance type.*

| | Lognormal | | | Weibull | | |
|---|---|---|---|---|---|---|
| Sample Size | $\mu$ | $\sigma$ | $R^2$ | $\lambda$ | k | $R^2$ |
| 100 | 11.186 | 4.9773 | 0.95285 | 1.0561e06 | 0.3502 | 0.9902 |
| 100 | 11.349 | 5.0921 | 0.93528 | 0.8970e06 | 0.2852 | 0.9863 |
| 200 | 11.77 | 4.5228 | 0.9271 | 0.8843e06 | 0.3128 | 0.9858 |
| 200 | 11.267 | 5.023 | 0.94094 | 1.1848e06 | 0.3479 | 0.9915 |
| 500 | 11.877 | 4.4152 | 0.92547 | 1.0019e06 | 0.3192 | 0.9891 |
| 500 | 11.554 | 4.7862 | 0.92842 | 0.8924e06 | 0.2883 | 0.9767 |
| 1000 | 11.716 | 4.605 | 0.9249 | 0.9475e06 | 0.3030 | 0.9839 |
| 1000 | 11.71 | 4.4949 | 0.93989 | 0.9095e06 | 0.3033 | 0.9921 |
| All | 11.743 | 4.5949 | 0.93899 | 0.9279e06 | 0.3008 | 0.9859 |

Table 5: *Lognormal and Weibull fitting parameters for different sample size for perfective maintenance type.*

| | Lognormal | | | Weibull | | |
|---|---|---|---|---|---|---|
| Sample Size | $\mu$ | $\sigma$ | $R^2$ | $\lambda$ | k | $R^2$ |
| 100 | 11.313 | 4.0977 | 0.9495 | 5.364e05 | 0.3151 | 0.984 |
| 200 | 11.751 | 4.4427 | 0.9449 | 9.7704e05 | 0.2958 | 0.9839 |
| 500 | 11.447 | 4.3648 | 0.9554 | 7.1057e05 | 0.294 | 0.9875 |
| 1000 | 11.562 | 4.2992 | 0.9533 | 7.6992e05 | 0.2990 | 0.9891 |
| All | 11.587 | 4.2370 | 0.9476 | 7.6707e05 | 0.3038 | 0.9886 |

Table 6: *Lognormal and Weibull fitting parameters for different sample size for adaptive maintenance type.*

| | Lognormal | | | Weibull | | |
|---|---|---|---|---|---|---|
| Sample Size | $\mu$ | $\sigma$ | $R^2$ | $\lambda$ | k | $R^2$ |
| 100 | 13.073 | 3.6821 | 0.9574 | 2.2962e06 | 0.3917 | 0.9875 |
| 200 | 13.148 | 3.2827 | 0.9688 | 2.1859e06 | 0.4101 | 0.9963 |
| 500 | 12.611 | 4.0179 | 0.9592 | 1.7755e06 | 0.4308 | 0.9895 |
| 1000 | 12.747 | 3.8036 | 0.9623 | 1.8473e06 | 0.3602 | 0.9904 |
| All | 12.776 | 3.6721 | 0.9700 | 1.8448e06 | 0.3615 | 0.9945 |

Table 7: *Issue resolution time estimations.*

| Maintenance type | total resolution time in days 1st trial | total resolution time in days 2nd trial | total resolution time in days 3rd trial |
|---|---|---|---|
| 8 corrective maintenance | 523 | 313 | 316 |
| 7 adaptive maintenance | 437 | 307 | 683 |
| 3 preventive maintenance | 47 | 119 | 120 |

Table 8: *Issue resolution time estimations for preventive maintenance.*

| # issues | total resolution time in days 1st trial | total resolution time in days 2nd trial | total resolution time in days 3rd trial |
|---|---|---|---|
| 14 | 532 | 304 | 167 |

can largely vary according to the possibilities of selecting 14 issues all with low resolution time, or 14 issues containing even one single issue with a very large resolution time.

## 7. THREATS TO VALIDITY

In this section we present the threats to validity of our study according to the guidelines reported in [19].

**Threats to internal validity** concern confounding factors that can influence the obtained results. We assume a causal relationship between the issue resolution time and the type of maintenance performed [9, 10, 13]. However, several factors can influence the issue resolution time like the complexity of the issue, type of project, project deadline etc.. Due to the large number of samples, projects and time frame analyzed, we assume that these factors may compensate each other. We hypothesize that our dataset is big enough to be avoid the bias related to specific factors.

**Threats to construct validity** focus on how accurately the observations describe the phenomena of interest. In this study, the elements of interest are the issue resolution time and the type of maintenance. The first element represents the *working* time spent to address a maintenance activity (stripped by the triaging times not devoted to code maintenance). We compute this value as the difference between (1) the time when the issue is subscribed by the developer and (2) the time when the issue is merged in the repository. We use these time-checkpoints assuming that the developers subscribe an issue when they ready to change the maintain the code; whereas they merge the code change only when the maintenance activity is completed. The adoption of issue fields to determine the resolution time has been already used in literature [3, 12].

The second element represents the type of activity performed by the developer to address the issue. We use the labeling provided by developer to classify the issue and then the type of maintenance performed. Since the labeling was done by people aware of the activity performed, we consider accurate and reliable such labels. To limit the impact of a subjective interpretation of the labels, the first and the second author referred to definition of maintenance types provided in literature [1, 10, 15]. We rely on this approach since it was already successfully adopted in literature [10, 13]. We decided to do not consider the issue for the analysis whenever the classification of its labels was not possible or doubtful. We assume that these labels (e.g.; *GUI*, *Mobile*) do not belong to specific maintenance category, namely they do not introduce a bias (due to a maintenance type under-represented).

**Threats to external validity** correspond to the generalizability of our experimental results. In this study, we use more than 30 projects that are representative of open source domain. To generalize our findings we should extend the analysis to industrial projects.

**Threats to reliability validity** correspond to the degree to which the same data would lead to the same results when repeated. We address this threat describing all steps of our experiments and using a dataset freely available [8]. Repeating our analysis may easily lead other researchers to the same results.

## 8. CONCLUSION

Software maintenance is a process difficult to understand and manage and yet crucial to organize the company's resource. In literature there is little information on how the maintenance activity influences the issue resolution time. Moreover, in the few cases where these two topics are both considered, the analysis is oversimplified since it adopts few statistic measures that cannot properly describe the specificities (e.g; right-skewness) of the issue resolution time distribution. Our empirical study analyzes the data stored in the GitHub's issue tracking system. From this repository we analyze 34 open source projects and more than 14000 issues that were ascribable as requiring corrective, adaptive or perfective maintenance.

This paper shows that the issue resolution time depends on the type of maintenance performed. We discovered that corrective and perfective maintenance are generally shorter than the other maintenance activities, whereas adaptive and perfective maintenance requires generally the highest resolution-time. This result arises questions on why perfective maintenance appears to be faster than corrective, adaptive and preventive maintenance. Moreover, it points out that models for effort estimation of maintenance activities based mainly on data extracted from bug oriented repositories (e.g.; Bugzilla) may provide biased estimation (towards corrective maintenance).

We use the lognormal and Weibull distribution models to quantify the contribute of the maintenance type on the issue resolution time. For both models, we demonstrate their suitability to analyze samples with different size. We demonstrate that statistical distribution models can be exploited for project's scheduling. For this purpose, we provide an operative example to describe how project managers can balance the maintenance activities according to project's constrains.

## Acknowledgments

This work has been sponsored by (i) Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen) under project number 120028 entitled Change-centric Quality Assurance (CHAQ) and (ii) Regione Autonoma della Sardegna (RAS), Regional Law No. 7-2007, project CRP-17938 "LEAN 2.0"

## 9. REFERENCES

[1] International standard - iso/iec 14764 ieee std 14764-2006. *ISO/IEC 14764:2006 (E) IEEE Std 14764-2006 Revision of IEEE Std 1219-1998)*, 2006.

[2] P. Bhattacharya and I. Neamtiu. Bug-fix time prediction models: can we do better? In *MSR*, pages 207–210, 2011.

[3] G. Bougie, C. Treude, D. M. Germán, and M. D. Storey. A comparative exploration of freebsd bug lifetimes. In *MSR*, pages 106–109, 2010.

[4] G. Concas, M. Marchesi, A. Murgia, R. Tonelli, and I. Turnu. On the distribution of bugs in the eclipse system. *IEEE Trans. Software Eng.*, 37(6):872–877, 2011.

[5] G. Concas, M. Marchesi, S. Pinna, and N. Serra. Power-laws in a large object-oriented software system. *Software Engineering, IEEE Transactions on*, 33(10):687–708, 2007.

[6] S. Demeyer, A. Murgia, K. Wyckmans, and A. Lamkanfi. Happy birthday! a trend analysis on past msr papers. MSR '13, pages 353–362, 2013.

[7] E. Giger, M. Pinzger, and H. Gall. Predicting the fix time of bugs. RSSE '10, pages 52–56. ACM, 2010.

[8] G. Gousios. The ghtorrent dataset and tool suite. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR'13, pages 233–236, 2013.

[9] A. Hindle, D. M. German, and R. Holt. What do large commits tell us?: A taxonomical study of large commits. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories*, MSR '08, pages 99–108, New York, NY, USA, 2008. ACM.

[10] A. Mockus and L. G. Votta. Identifying reasons for software changes using historic databases. In *Proceedings of the International Conference on Software Maintenance (ICSM'00)*, ICSM '00, pages 120–, Washington, DC, USA, 2000. IEEE Computer Society.

[11] M. E. Newman. Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351, 2005.

[12] L. D. Panjer. Predicting eclipse bug lifetimes. In *MSR*, page 29, 2007.

[13] R. Purushothaman and D. E. Perry. Toward understanding the rhetoric of small source code changes. *IEEE Trans. Softw. Eng.*, 31(6):511–526, June 2005.

[14] S. Siegel. Nonparametric statistics for the behavioral sciences. 1956.

[15] E. B. Swanson. The dimensions of maintenance. In *Proceedings of the 2nd international conference on Software engineering*, ICSE '76, pages 492–497. IEEE Computer Society Press, 1976.

[16] R. van Solingen, V. Basili, G. Caldiera, and H. D. Rombach. *Goal Question Metric (GQM) Approach.* John Wiley Sons, Inc., 2002.

[17] C. Weiß, R. Premraj, T. Zimmermann, and A. Zeller. How long will it take to fix this bug? In *MSR*, page 1, 2007.

[18] F. Wilcoxon and R. A. Wilcox. *Some rapid approximate statistical procedures.* Lederle Laboratories, 1964.

[19] R. K. Yin. *Case study research: Design and methods*, volume 5. sage, 2009.

[20] F. Zhang, F. Khomh, Y. Zou, and A. E. Hassan. An empirical study on factors impacting bug fixing time. In *Proceedings of the 2012 19th Working Conference on Reverse Engineering*, WCRE '12, pages 225–234, Washington, DC, USA, 2012. IEEE Computer Society.