Bayesian Analysis of Bug-Fixing Time using Report Data

Renan Vieira
Federal University of Ceará
Fortaleza, Ceará, Brazil
renan.vieira@alu.ufc.br

Diego Mesquita
Getulio Vargas Foundation
Rio de Janeiro, Rio de Janeiro, Brazil
diego.mesquita@fgv.br

César Lincoln Mattos
Federal University of Ceará
Fortaleza, Ceará, Brazil
cesarlincoln@dc.ufc.br

Ricardo Britto
Ericsson / Blekinge Institute of Technology
Karlskrona, Sweden
ricardo.britto@ericsson.com

Lincoln Rocha
Federal University of Ceará
Fortaleza, Ceará, Brazil
lincoln@dc.ufc.br

João Gomes
Federal University of Ceará
Fortaleza, Ceará, Brazil
jpaulo@dc.ufc.br

ABSTRACT

Background: Bug-fixing is the crux of software maintenance. It entails tending to heaps of bug reports using limited resources. Using historical data, we can ask questions that contribute to better-informed allocation heuristics. The caveat here is that often there is not enough data to provide a sound response. This issue is especially prominent for young projects. Also, answers may vary from project to project. Consequently, it is impossible to generalize results without assuming a notion of relatedness between projects.
Aims: Evaluate the independent impact of three report features in the bug-fixing time (BFT), generalizing results from many projects: bug priority, code-churn size in bug fixing commits, and existence of links to other reports (*e.g.*, depends on or blocks other bug reports).
Method: We analyze 55 projects from the Apache ecosystem using Bayesian statistics. Similar to standard *random effects* methodology, we assume each project's average BFT is a dispersed version of a global average BFT that we want to assess. We split the data based on feature values/range (e.g., with or without links). For each split, we compute a posterior distribution over its respective global BFT. Finally, we compare the posteriors to establish the feature's effect on the BFT. We run independent analyses for each feature.
Results: Our results show that the existence of links and higher code-churn values lead to BFTs that are at least twice as long. On the other hand, considering three levels of priority (low, medium, and high), we observe no difference in the BFT.
Conclusion: To the best of our knowledge, this is the first study using hierarchical Bayes to extrapolate results from multiple projects and assess the global effect of different attributes on the BFT. We use this methodology to gain insight on how links, priority, and code-churn size impact the BFT. On top of that, our posteriors can be used as a prior to analyze novel projects, potentially young and scarce on data. We also believe our methodology can be reused for other generalization studies in empirical software engineering.

CCS CONCEPTS

• **Software and its engineering → Maintaining software**; • **General and reference → Empirical studies**; • **Mathematics of computing → Bayesian computation**.

KEYWORDS

Bug Reports, Bayesian Modeling, Open-Source, Bug Fixing Time

ACM Reference Format:
Renan Vieira, Diego Mesquita, César Lincoln Mattos, Ricardo Britto, Lincoln Rocha, and João Gomes. 2022. Bayesian Analysis of Bug-Fixing Time using Report Data. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Catolino et al. [7] highlights that back in 2017, there was a perception that bugs were "eating the world". Once the software systems grow in size and complexity, and developers need to work under frequent and tight deadlines, the introduction of bugs cannot be seen as an unexpected phenomenon. Therefore, bug fixing becomes a very resource-consuming activity, leading to costs by order of billions per year. Finding, reporting, and fixing bugs consumes 50% of the software developers' time (on average) [6].

Nowadays, software development teams frequently rely on Issue Tracking Systems (ITS) to control changes in a project and manage backlogs of issues[1] to be addressed [4, 25, 28]. There are several available ITS, such as Bugzilla, YouTrack, and Jira, which are widely adopted in both commercial and open-source projects.

A bug report must provide a textual description showing the steps needed to reproduce the failure being reported. The information available in an ITS varies from one product to another. The Jira tool, for instance, provides information in different dimensions, such as: *importance-related* as `Priority`; the *text-related* as `Summary` and `Description`; *person-related* as `Assignee` and `Reporter`; and *link-related* as `Outward` and `Inward` that describes how a bug report affects or is affected by another bug report inside Jira. Additionally, an ITS also records the bug report opening and closing times, making it possible to compute the bug-fixing time (BTF).

It is worth noticing that information available in a bug report is essential to support the following tasks: (i) the bug triage (who will be assigned to fix a bug); (ii) the bug fixing scheduling (prioritization

[1]An issue could represent a feature, an enhancement, a bug, a doubt, a user story, a task, or another concern in the project.

of which bugs will be fixed in a given period of time); and (iii) the bug correction itself. Understanding how the attributes of a bug report impact the overall bug fixing process is essential to build reliable estimation models to optimize resource allocation.

Many studies explored information on bug reports available on different ITS, focusing on aspects such as: "was this bug already registered?" [9, 22], "who is the best person to fix this bug?" [15, 30], "is this a real bug?" [18], "is this report good, and does it have enough information?" [46], "what is its priority?" [33], and "how much time is necessary to fix this bug?" [2, 16, 44]. These (and other) existing studies provide helpful evidence to improve the bug-fixing process.

The typical approach used to understand how the information of bug reports impacts other independent variables (e.g., BFT) relies on some statistical framework. de Oliveira Neto et al. [8] reports that frequentist statistical approaches became the standard tool to provide this kind of insight in Empirical Software Engineering. However, the Bayesian framework is another option that sometimes is overlooked at first because it does not offer out-of-the-box solutions like the statistical tests of the frequentist framework. Recent studies [10, 12] have been advocating for the use of the Bayesian framework as an alternative to the more traditional statistical test. These studies highlight advantages of the Bayesian framework [11, 35]: a fine-grained data model's building control, better visual appeal of the results, and the use of additional information as prior. Additionally, Bayes statistics and posterior distributions can be a starting point for analyzing younger projects. The model can be updated to best represents the newly acquired evidence as new data emerges from the project lifetime.

In this paper, we show the process of using Bayesian statistics to analyze bug report data. Specifically, we use Bayesian statistics to empirically evaluate the relation between the BFT and (i) bug report priority, (ii) links between reports, and (iii) code-churn size of bug-fix commits. Our results show that the existence of links and higher code-churn values lead to BFTs that are at least twice as long. On the other hand, considering three priority levels (low, medium, and high), we observe no difference in the BFT.

Our main contributions are two fold. First, our results may support practitioners during bug triage and scheduling by providing helpful information concern the BFT based on exiting bug report data. Second, the methodology can be reused by researchers in different software projects data, as the presented posterior distributions can be used as priors for analysis in a similar context.

2 BAYESIAN DATA ANALYSIS OVERVIEW

This section briefly introduces core Bayesian data analysis (BDA) concepts. While we cover the basics of BDA, we refer the reader to, *e.g.*, Gelman et al. [14] or Kruschke [20] for a thorough presentation of the subject.

re-evaluate our opinion on θ. We refer to our updated belief as the *posterior* distribution and compute it using the Bayes' rule:

$$p(\theta|y) = \frac{p(\theta)p(y|\theta)}{\int_{\Theta} p(\theta, y)p(y)\,\mathrm{d}\theta}. \tag{1}$$

The first step in Bayesian modeling is to choose *likelihood* function $p(y|\theta)$, *i.e.*, an observation model. Choosing an appropriate likelihood requires analyzing the nature of the data y. For instance, if our observations are numbers of system failures in a given time interval, $p(\cdot|\theta)$ should have non-negative integer support — the most common choice for count being Poisson distribution $Pois(\lambda)$, governed by the parameter λ. Then, we must choose a prior . In the case of a *likelihood* $Pois(\lambda)$, we must choose a prior $p(\lambda)$, which means the plausibility of the values that λ can assume before the data is observed.

Ideally, we can use previous analyzes and observations or specific domain knowledge to define *informative* priors. In cases where it is impossible to provide *informative* priors, we must at least ensure that the priors cover a reasonable value range or, conversely, rule out unusual value ranges as highly unlikely [10]. These types of priors are called *weak* or *weakly informative*. We can test several *priors* and perform a sensitivity analysis to check which one best fits our data [14].

Computational methods. Once we have a model, the next step is making the inference. We must fit the model, which means that we have to compute the posterior distribution $p(\theta|y)$. However, doing so analytically is often challenging since it requires computing the denominator of Bayes' rule, which is usually intractable. For simple cases with a small number of parameters, one can use grid or quadratic approximation to calculate the posterior [23]. Nonetheless, the weapon of choice for most Bayesians are Markov chain Monte Carlo (MCMC) sampling methods, such as sequential Monte Carlo and Hamiltonian Monte Carlo [14]. In this paper, we evaluate models that are computed using MCMC.

Hypothesis testing. The posterior distribution encapsulates all information we have gathered on θ, subjective or not, and we can use it to probe any hypothesis. For instance, we can evaluate the probability that $\theta > a$ taking the expected value of the indicator function $\mathbb{1}[\cdot > a]$, i.e.:

$$p(\theta > a) = \int_{\theta\in\Theta} \mathbb{1}[\theta > a] p(\theta|y)\,\mathrm{d}\theta.$$

Unfortunately, computing exact integrals over the posterior are often intractable. However, given a set of MCMC samples $\mathcal{S} = \{\theta^{(k=1)}\}^{K}$, we can approximate the expected value of any function g of θ as:

$$\mathbb{E}_{p(\theta|y)}[g(\theta)] = \int_{\theta\in\Theta} g(\theta)p(\theta|y)\,\mathrm{d}\theta \approx \frac{1}{K}\sum_{k=1}^{K} g\big(\theta^{(k)}\big).$$

Simulating novel data. We can easily simulate novel data with the posterior samples in our hands. We do so by sampling from:

$$\int_{\theta\in\Theta} p(y_\star, \theta|y) = p(y^\star|\theta)p(\theta|y)\,\mathrm{d}\theta,$$

which using MCMC samples resumes to repeating the following process: i) pick a sample $\theta^{(k)}$ from $\mathcal{S}$; ii) sample from our observation model conditioned on $\theta^{(k)}$, i.e., $p(y_\star|\theta^{(k)})$. Once the simulated data is drawn for a model that describes well our data's generative process, it should look similar to the observed data [13], as any discrepancy between the sample data and the observed data indicates potential failings in the proposed model.

2.2 Hierarchical Models

The structure of data in a specific domain can indicate some relation or connection between the parameters of the model [13]. Consider the example of our selected dataset composed of 55 open source projects, all coming from the Apache Ecosystem, with all reports mined from JIRA ITS. For instance, it might be reasonable to expect that these projects, coming from the same source, may present similar behavior in terms of types of bugs or the time to fix them, among other similarities. In this case, we can define in our model that the estimates of the parameter θ_i, representing the average time to fix a bug in a project i, are drawn from a prior distribution (conditioned by a parameter θ_0), also representing the average time to fix a bug, but considering *all* projects behavior.

The advantage of this kind of approach, called Hierarchical or Multi-level models, is that they are less inclined to underfit or overfit the data when compared to single-level models, dealing better with the imbalance in sampling and better modeling between variance among groups and individuals [23]. From an intuitive point of view, this mechanism allows for transferring information between different projects. This also will enable projects with fewer data to borrow strength from inferences in more mature projects. Besides allowing us to estimate the parameters θ of each project, hierarchical modeling also provides us with a distribution over the global parameter θ_0 — in our case, the global average BFT —, which is more suitable for generalization results.

2.3 Motivation to use Bayesian Data Analysis

The workflow to understand how specific bug report characteristics are related to others that significantly impact the bug triage process (e.g., bug-fixing time, priority, or report quality) generally would use some statistical framework. For example, frequentist statistical approaches have been the standard tool to provide this kind of insight in empirical software engineering studies [35]. However, the Bayesian framework is another option that is sometimes over-viewed because it does not offer out-of-the-box solutions as the statistical tests of the frequentist framework. In addition, some works have advocated using the Bayesian framework as an alternative to the more traditional statistical test use. In general, the BDA workflow is more informative about the data and the outcome of the analysis when compared to frequentist approaches [14].

The Bayesian Data Analysis characteristics come with a cost: while frequentist statistical approaches provide a group of tests covering several data scenarios, the Bayesian framework requires more detailed attention as we have to build our models from the bottom up. There are a few steps and attention to details to cover, and there are some literary works that describe the process at length Gelman et al. [14], McElreath [23]. In contrast, others have been more active in highlighting the advantages of using the BDA in empirical software engineering [11, 35].

Besides all the highlights provided by some researchers, one more specific objective point was crucial to adopting the Bayesian approach in this paper: we have multiple data sources in the dataset. As we intended to provide conclusions about all projects, hierarchical models are a potent tool provided by BDA that helps test hypotheses about the data more generalistic. The counterpoint in the frequentist statistical approaches is to combine p-values from different statistical test results from different data sources [17]. As covered by [11], there is not a uniform view about when and how the adjustment of p-values from different statistical tests. In the same work, the authors show the impact of using different techniques to adjust p-values and how the different techniques impact the final conclusions regarding the analysis.

It is important to notice that both frequentist and bayesian analysis provides similar conclusions when correctly applied [39], but BDA provides a few particularities that we consider more appealing than the frequentist framework. We summarize the motivation to apply BDA in the following paragraphs.

Flexibility to Create Models. Due to BDA framework flexibility, we have total control of the assumptions regarding the model and data. As we describe every aspect of the model, this provides a better comprehension of the whole modeling process, allowing a more detailed review and criticism from peers abroad.

Hierarchical Models. We have data mined from several projects, each one with its particularities. Hierarchical models provide ways to summarize data from different sources to give us a more general picture of a similar behavior underlying their idiosyncratic. The use of hierarchical models serves us as an alternative to possible pitfalls of selection p-values adjustments and combinations.

Posterior Distributions as Results. The outcome of every BDA are posterior and predictive distributions. These posterior distributions describe our models' parameters based on our assumptions and data. However, the use of prior is an inherent characteristic of BDA. For future works that perform similar analysis on other bug reports data, the obtained posterior distributions in this research can be used as priors in their models, creating a chain of knowledge of the same domain [11, 23].

3 STUDY DESIGN

3.1 Goal and Research Questions

Our study explores the usage of Bayesian statistics to analyze the impact of bug report priority, links, and code-churn size on bug fixing-time. We analyzed a dataset with 10 years of bug-fixing records from 55 projects from the Apache ecosystem (see Section 3.2).

Our investigation answers the following research questions:

RQ1: *How does the existence of links in bug reports impacts the BFT*?

It is not rare to find relationship between bug reports. For example, a relationship can indicate if a bug report `A` is `blocked` by another bug report `B`. Answering RQ1 will allow us understand if the existence of such relations impacts the bug-fixing time (BFT).

RQ2: *How does the priority level of a bug report impacts the BFT*?

Every bug report has an associated priority value, which may indicate urgency or severity. Existing studies [3, 33, 38] have investigated the role of priority in the bug fixing process. In this paper, we focus on the impact of priority on the BFT.

RQ3: *How does the code-churn size of fixing commits relates to the BFT?*

Bug-fix commits are those bringing the changes that fix a reported bug. Thus, it is possible to compute the code-churn size related to a fix commit. To answer this research question, we first group the bug reports into two categories: (i) reports with low code-churn values and (ii) reports with high code-churn values (the threshold is the project's code-churn median value). Next, we evaluate if the average BFT of both groups is significantly different.

3.2 Dataset Characterization

We use the Vieira et al. [40] dataset as the source to answer our research questions. The dataset comprises bug report information regarding 55 open-source projects from the Apache ecosystem with different levels of maturity and categories (e.g., big-data, web-framework, database, and cloud). All bugs reported in the dataset were identified and fixed between 2009 and 2018.

We use seven features available in the dataset: (i) the *link-related fields InwardIssueLinks* and *OutwardIssueLinks*; (ii) bug report *Priority*; (iii) the *time-related fields CreationDate* and *ResolutionDate*, to estipulate the BFT - the bug report lifespan, calculated as the difference between the report data resolution and creation; `AddLines` and `DelLines`, representing, respectively, the number of added and deleted lines in the reported bug source code file, which was fixed by a commit defined in the dataset.

In the following paragraphs, we detail the fields mentioned above, showing how they are represented in the dataset, and why we choose them to use in our investigation.

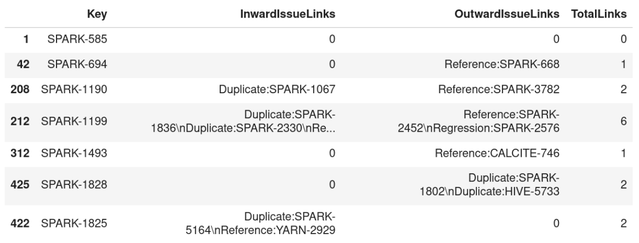
Bug Report Links. The relations between bug reports are recorded in *InwardIssueLinks* and *OutwardIssueLinks* fields. As the names suggest, given an issue report r_i with an inward link that refers to another issue report r_j, this indicates that somehow, r_j relates to r_i. Similarly, given an issue report r_i with an outward link that refers to another issue report r_j, this indicates that somehow, r_i relates to r_j. In short, the bug reports can be seen as vertices of a graph and the links between them as oriented edges of this graph.

The *link-related* fields are string fields in the dataset. If it is empty (NaN or 0), it indicates no link of the specific field (inward or outward) type. If not empty, the field contains a string of unique Keys separated by a line break (if there is more than one). The keys in Jira follow the format {project}-{number}. Hence, given the project Spark, examples of keys would be SPARK-213 and SPARK-481.

Figure 1 shows the possible scenarios around links between bug reports. It shows seven bug reports of the Spark project in five columns, in order: (i) the Pandas Dataframe[2] index; (ii) the bug report unique key; (iii) the inward links references; (iv) the outward links references; (v) the total report number of links.

Bug Report Priority. The bug report priority has been the subject of several studies with different purposes. The majority of

[2]https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html



	Key	InwardIssueLinks	OutwardIssueLinks	TotalLinks
1	SPARK-585	0	0	0
42	SPARK-694	0	Reference:SPARK-668	1
208	SPARK-1190	Duplicate:SPARK-1067	Reference:SPARK-3782	2
212	SPARK-1199	Duplicate:SPARK-1836\nDuplicate:SPARK-2330\nRe...	Reference:SPARK-2452\nRegression:SPARK-2576	6
312	SPARK-1493	0	Reference:CALCITE-746	1
425	SPARK-1828	0	Duplicate:SPARK-1802\nDuplicate:HIVE-5733	2
422	SPARK-1825	Duplicate:SPARK-5164\nReference:YARN-2929	0	2

Figure 1: Example of links in Spark's bug reports. All links are associated with a type (*e.g.*, duplicate or reference). While some reports may not present any link (SPARK-585), others only present one type of reference (SPARK-684 and SPARK-1825), and others may present several links, even with references to a different project issue report (SPARK-1493).

them deals with proposals to automatically assign priority to bug reports [36, 37, 43]. Other studies try to characterize why a bug report priority would change [3]. A few studies use the priority information, along side other bug reports features, as criteria to understand software quality and bug report quality [19, 21].

The priority brings an idea of bug urgency and defines the category of how important it is to fix a specific bug when compared to others. This concept is strengthened when we list, in crescent order of importance, the field default values in Jira: *trivial*, *minor*, *major*, *critical*, and *blocker*. Out of the 55 projects present in the dataset, 54 use the default Jira priority values. The exception is the Cassandra project, that has only 3 priority levels: *low*, *normal*, and *urgent*.

Figure 2 uses all projects (except Cassandra) to present some priority information. The visualization on the left shows the absolute number of the report with each priority. The projects have a different number of bug reports and distinct lifespans. For example, some of them have more than 16 years of development (*e.g.*, Hadoop and HBase), while others have less than six years of development (*e.g.*, SystemML and MADlibr). Thus, looking only at the absolute values could present a biased behavior from the older projects. We create another visualization displayed on the right. The visualization shows the average proportion for each priority of all projects. The vertical line presents the standard deviation.

The majority of the reports present the *minor* or *major* priority. It is important to notice that *major* is the default value priority when a report is created, which may explain the high number of reports with this priority. The visualization on the right presents a similar behavior when compared to the one on the left. Still, it shows that the *trivial*, *critical*, and *blocker* can present smaller values, close to 0% in some projects.

Bug Report Code-Churn Size. Code churn measures the changes made to a component over a period of time [24]. In our study, we compute the code-churn size of the files changed by a bug-fix commit. There are different ways to compute code churn [24, 29], and the choice of how to compute it depends on the available data and the objective of the analysis. In this work, we define the code-churn as *the sum of added and removed lines*. In general, the reports have an associated bug-fix commit, which has a given number of added and removed lines.

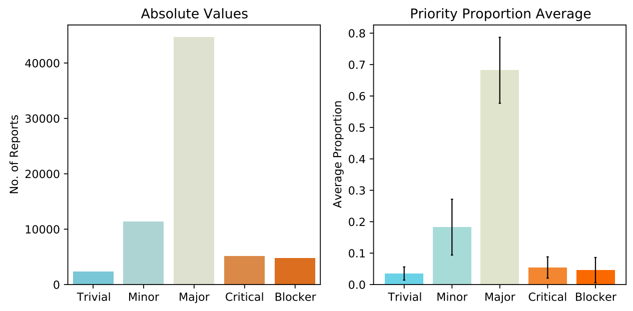


Figure 2: Bug Reports Priority Distribution. On the left, it is presented the absolute number of bug reports with a given priority for all projects. On the right, it is presented the average proportion of a given priority in a project. Both visualizations consider 54 projects from the dataset. Most bug reports present priority values as *minor* or *major*.

There are cases where no such commit exists in our dataset. The reasons may vary: the report represents an already previously fixed bug, document a duplicated bug, or the bug is fixed in a commit of a different but. In these cases, we discard these reports from the code-churn analysis, only using the reports with associated commits.

3.3 Data Pre-processing Details

Given a project, for each feature's i) links, ii) priority and iii) code-churn, we create groups of reports based on the specific criteria of the selected feature values:

- For 'links', we split the data into two groups: the group of reports with links (R_{wl}) and the group with no links (R_{nl}). As presented in Subsection 3.2, there are several scenarios of links in the reports. However, for this first round of analysis, we choose only to consider the existence or not of *some* link.
- For 'priority', we split the data into three groups: the group of reports with trivial-minor priority (low priority, R_{lp}), the group with major priority (medium priority, R_{mp}) and the group with critical-blocker priority (high priority, R_{hp}). This grouping of the lower and higher priority is justified for two major reasons: the Cassandra project uses three levels of priority (low, normal, and urgent), and some smaller projects do not contain examples of reports with all priority. With this approach, we can deal with all the projects simultaneously.
- For 'code-churn', we split the data into two groups: reports with higher code-churn values (R_{hcc}) and reports with lower code-churn size (R_{lcc}). Given a project, the threshold to split both groups is the median code-churn of its bug reports.

We justify the interest in studying the relation between bug fix time with each one of the features as follows. The relation (links) between issue reports seems to be overlooked by papers that study bug reports. For instance, with a quick search for papers with keywords as 'bug reports', 'links', 'relationship', 'Jira reports', we only could find three papers that deal explicitly with links in bug reports [5, 32, 34]. Also, we have not found any proposal that uses machine learning techniques to estimate the BFT and consider the relationship between reports as features. Links can represent several types of relation, as presented in Figure 1. It is reasonable to believe that a blocked report one will only be fixed after the blocker report is resolved, implying some interference of a report over another. Other types of relationships, as duplicated, are also an indication that fixing one can impact considerably other ones.

On the other hand, the priority is an objective of the study in several papers that deal with similar data and are almost used in all predictive models as features. However, it is never clear how a bug report priority is directly related to a bug fixing time. An argument that the priority is a measure of importance or urgency, hence asks for more attention and rapid responses. However, priority carries no information about the complexity of the bug (i.e., a minor bug may be more complex than a simple but blocker bug). With this analysis, we intend to bring some light to the matter.

Code-churn is a widespread metric in software engineering research. However, it is post-bug fix information: it is only known after the bug resolution. However, we argue that once the bug is located in a class, function, or file, one could use prior information about the code-churn values in this specific bug location to estimate (along with the report information) the time to fix the bug. For instance, if a bug is located in a class that, based on historical data, demands higher patch code-churn values, this could indicate that this bug will also take a higher BFT.

3.4 Modeling Process and Models Description

The analysis starts with two proposed models as hypothesis to explain the generative data process. Given a feature, we fit both models for each one of its groups. After that, we compare the adverse groups' μ posterior distribution to draw our conclusions. For instance, for the 'links' analysis, we first fit a model using R_{nl} data and then another model using the R_{wl} data. Then we use each model's μ posterior distribution to verify the difference between both groups.

We first define some sets, distributions, and variables that we use to describe the models:

- *days* / d: The time to fix the bugs in days, as non-negative real numbers. For all models, we considerer $log(days) \sim \mathcal{N}(\mu, \sigma^2)$, as *days* can not assume negative numbers.
- $\mathcal{G}$: the groups of bug fixing time in *days*. The groups of data are $\mathcal{G} = \{R_{nl}, R_{wl}, R_{lp}, R_{mp}, R_{hp}, R_{lcc}, R_{hcc}\}$, as presented in subsection 3.3.
- $\mathcal{P}$: The set of all projects. $\mathcal{P} = \{p_1, ..., p_{55}\}$, each p_i being one of the projects presented in the dataset presented in [40].
- $\mathcal{N} \sim (\mu, \sigma^2)$: The Normal distribution, defined the parameters mean μ and variance σ^2.
- Inv-Gamma(α, β) / Γ^{-1}: The Inverse Gamma distribution, defined by parameters α and β. Usually, the Inverse Gamma is used as prior for the variance in BDA.

The first model is a single-level model, where we fit using only one project data at a time. We use a weakly informative prior for all parameters. The following model is presented in equation (2), and we call it 'specific-model'.

$$\mu \sim \mathcal{N}(0,2), \quad \sigma^2 \sim \text{Inv-Gamma}(3,3), \quad \log(\text{days}) \sim \mathcal{N}\left(\mu,\sigma^2\right). \tag{2}$$

Using this representation, we fit a total of 385 models: 110 for links (for each of the 55 projects, we fit a model using R_{wl} data and another using R_{nl} data), 110 for code-churn (for each of the 55 projects, we fit a model using R_{lcc} data and another using R_{hcc} data) and 165 for priority (for each of the 55 projects, we fit a model using R_{lp} data, another using R_{mp} and another using R_{hp}).

The second model is a Hierarchical Model (HM), where we fit all projects data at once. We also use a weakly informative prior for all proposed models. We have a μ_0 representing the parameter to estimate for all projects population, while we have one μ_i to describe each project p_i. The following model is called 'HM-AP' (Hierarchical Model-All Projects) and it is described in equation (3).

$$\begin{aligned}&\mu_0 \sim \mathcal{N}(0,2), \ \sigma_0^2 \sim \text{Inv-Gamma}(3,3),\\ &\sigma_i^2 \sim \text{Inv-Gamma}(3,3), \forall p_i \in \mathcal{P},\\ &\mu_i \sim \mathcal{N}\left(\mu_0,\sigma_0^2\right), \forall p_i \in \mathcal{P},\\ &\log(\text{days}_i) \sim \mathcal{N}\left(\mu_i,\sigma_i^2\right), \forall i \in \mathcal{P}.\end{aligned} \tag{3}$$

Using the HM-AP, we fit a total of seven models: two for links (one using R_{wl} data of all projects and another using R_{nl} data of all projects), two for code-churn (one using R_{gcc} and another using R_{scc} data of all projects) and three for priority (same logic as previous, R_{lp}, R_{mp}, and R_{hp} data of all projects). The proposed hierarchical model intends to capture the global bug report behavior based on the particular data of each project. 'HM-AP' assumes that there is no other similarity aspect between the projects besides they all have bug reports.

4 RESULTS

With all models defined, we use Stan, specifically PyStan[3] which is a Python interface to Stan, a package for Bayesian inference. Stan is a state-of-the-art platform for statistical modeling and high-performance statistical computation. The computation goes as presented in Section 2: we compute the posterior distribution $p(\mu,\sigma^2)$ using MCMC for each model, given the data presented in $\mathcal{G}$. After that, we compare the opposite μ posterior distributions for each feature group using five summarizations: the Maximum a Posteriori $\hat{\mu}_{MAP}$ estimator (the most plausible value for the estimator μ); the Lower (CI_L) and Upper (CI_U) value of the 95% Confidence Interval (CI, also known as Uncertainty, Credible, or Compatibility Interval); the expected value of the difference between both BFT groups (4); and the probability of a group having a greater average fixing-time than the other (5), as presented in the following equations

$$\begin{aligned}&f_1(a,b) : a-b,\\ &E[f_1(a,b)] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_1(a,b)p(a)p(b)\,\mathrm{d}a\,\mathrm{d}b,\end{aligned} \tag{4}$$

$$\begin{aligned}&f_2(a,b) : 1 \text{ if } a > b, \ 0 \text{ otherwise},\\ &E[f_2(a,b)] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f_2(a,b)p(a)p(b)\,\mathrm{d}a\,\mathrm{d}b.\end{aligned} \tag{5}$$

We present the results visually through the μ marginal posterior distributions for each data group, the μ_{MAP} represented by a dotted line, and the CI by the filled area under the curve. We also present the numeric values with an associated table for each proposed model, the summarization tables, along with the values obtained using the equations 4 and 5. The results are grouped by RQ and presented in the following subsections in a similar manner. First, we show the results using the 'specific-model' of four projects to show some divergent scenarios regarding the possible conclusions about the difference between the groups of features. The complete posterior distribution visualizations and summarization tables for all 55 projects can be found in the replication package[4].

4.1 RQ1. How does the existence of links in bug reports impacts the BFT?

Answer to RQ1: Considering the marginal posterior distribution of μ for groups of bug reports with and without links, along with their summarization, we found that bugs with links tend to need 2.4 more times, on average, to be fixed than bugs with no links.

Figure 3 shows four projects 'specific-models' marginal μ posterior distributions, while Table 1 the distribution summarization.

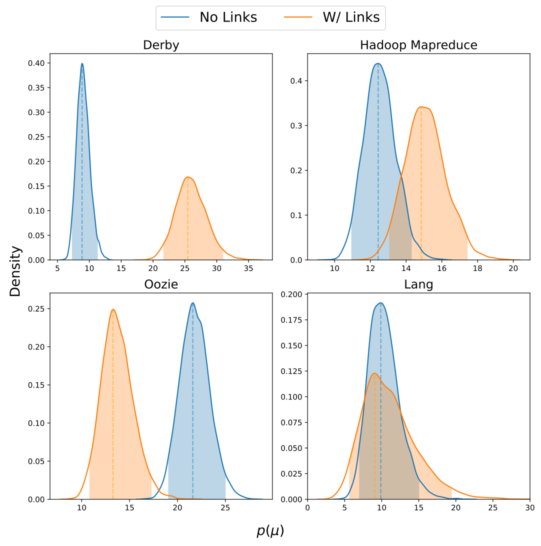


Figure 3: μ Marginal Posterior Distributions, specific-models. The average bug fixing time of reports with no links vs. the ones with links. The conclusions diverge depending on the selected project.

[3]https://pystan.readthedocs.io/en/latest/

[4]https://doi.org/10.6084/m9.figshare.20315076

Table 1: μ Posterior Distribution Summary, using 'specific-models' and 'links' data groups.

No Links (a)

Project	CI_L	CI_U	μ_{MAP}	E[F1(a,b)]	E[F2(a,b)]
Derby	7.24	11.35	8.86	-16.93	0.00
Lang	6.87	15.10	9.88	-0.68	0.46
Mapreduce	10.92	14.35	12.44	-2.55	0.04
Oozie	18.97	25.04	21.60	8.07	1.00

W/ Links (b)

Project	CI_L	CI_U	μ_{MAP}	E[F1(b,a)]	E[F2(b,a)]
Derby	21.55	31.08	25.46	16.93	1.00
Lang	5.52	19.45	9.08	0.68	0.54
Mapreduce	13.01	17.45	14.85	2.55	0.96
Oozie	10.79	17.29	13.27	-8.07	0.00

The conclusions differ based on the project we analyze. For Derby, is evident the difference between the both groups, with the reports with links taking more time to be fixed than those with no links. In Hadoop MapReduce, we also see a similar behavior but with some superposition of both distributions. In Oozie, we see an inverse behavior: reports with no links present higher BFT than those with links. Finally, the Lang project presents no difference between both groups. While most of the projects show a behavior similar to Derby and Hadoop Mapreduce, it is hard to conclude the real impact of links in the report BFT, taking each project individually. The results give us a general picture of each project's behavior but do not help us to verify a bug reports global behavior.

As the project's individual analysis does not help to answer our research question, this justifies using hierarchical models to summarize the population's behavior of bug reports. We present the marginal $\mu - 0$ posterior distributions obtained using the 'HM-AP' in Figure 4 and the summarization in Table 2.

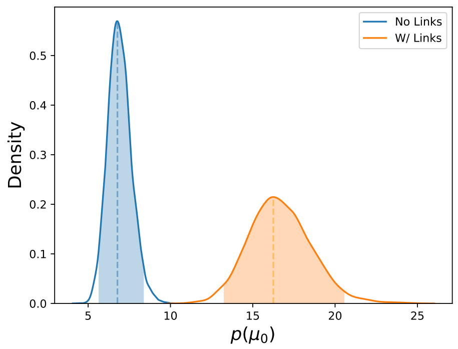


Figure 4: μ_0 Marginal Posterior distributions, 'HM-AP', 'links' data groups.

The results show a significant difference between both average BFTs, where reports with links (group 'a') need more time to be fixed than reports with no link (group 'b'). The expected difference between both groups is 9.76 days, suggesting that bugs with links tend to take 2.4 more times to be fixed than those with no links. The probability of group 'b' is more significant than group 'a' is 1.

Table 2: μ_0 Posterior Distribution Summary, 'HM-AP', 'links' data groups

No Links (a)

CI_L	CI_U	μ_{0MAP}	E[F1(a,b)]	E[F2(a,b)]
5.63	8.41	6.78	-9.76	0.00

W/ Links (b)

CI_L	CI_U	μ_{0MAP}	E[F1(a,b)]	E[F2(a,b)]
13.20	20.59	16.25	9.76	1.00

4.2 RQ2. How does the priority level of a bug report impacts the BFT?

Answer to RQ2: Considering the marginal posterior distribution of μ_0 for groups of reports with low, medium, and high priority, along with their summarization, we found that bug priority does not have a significant impact on the bug-fixing time.

The Figure 5 shows four projects 'specific-models' marginal μ posterior distributions, while Table 3 the distributions summarization. Once again, as presented in the results for links, we selected four different scenarious of possible conclusions.

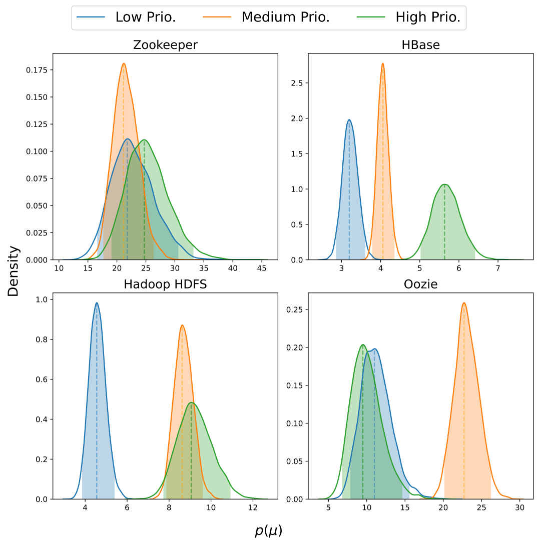


Figure 5: μ Marginal Posterior Distributions, 'specific-models'. The average bug fixing time of reports difference levels of priority. The conclusions diverge depending on the selected project.

Depending on the selected project, the conclusions diverge. For Zookeeper, the distributions are majority overlapped, an indication of a small significant difference between the three groups. The data

Table 3: μ Posterior Distribution summary, 'specific-models', using priority data groups

Low Priority (a)

Project	CI_L	CI_U	μ_{MAP}	E[F1(a,b)]	E[F1(a,c)]	E[F2(a,b)]	E[F2(a,c)]
HBase	2.85	3.61	3.20	-0.84	-2.46	0.00	0.00
HDFS	3.85	5.42	4.55	-4.11	-4.65	0.00	0.00
Oozie	7.78	15.64	11.01	-11.84	1.18	0.00	0.67
Zookeeper	16.45	30.69	21.79	1.21	-2.51	0.60	0.31

Medium Priority (b)

Project	CI_L	CI_U	μ_{MAP}	E[F1(b,a)]	E[F1(b,c)]	E[F2(b,a)]	E[F2(b,c)]
HBase	3.78	4.36	4.06	0.84	-1.62	1.00	0.00
HDFS	7.87	9.62	8.62	4.11	-0.54	1.00	0.29
Oozie	20.11	26.27	22.72	11.84	13.01	1.00	1.00
Zookeeper	17.64	26.38	21.16	-1.21	-3.73	0.40	0.19

High Priority (c)

Project	CI_L	CI_U	μ_{MAP}	E[F1(c,a)]	E[F1(c,b)]	E[F2(c,a)]	E[F2(c,b)]
Hbase	5.01	6.43	5.63	2.46	1.62	1.00	1.00
HDFS	7.72	10.93	9.06	4.65	0.54	1.00	0.71
Oozier	6.71	14.64	9.49	-1.18	-13.01	0.33	0.00
Zookeeper	18.97	33.31	24.73	2.51	3.73	0.69	0.80

from HBase presents a very distinct behavior for each group, with the order of bug fixing average time being low, medium, and high priority. For both HDFS and Oozie, only one group presents a more distinct behavior when compared with the other two. In Hadoop HDFS, bugs with low priority take less time than bugs with medium and high priority, both presenting very similar estimators values for μ. With Ozzie, we also notice a similarity between bugs with low and high priority, while reports with medium priority take more bug-fixing time. As presented in the results with links, it is hard to conclude the real impact of priority in the report bug fixing time, taking each project individually.

We use the hierarchical 'HM-AP' to draw conclusions for the priority groups. The μ_0 bug reports population posterior distributions is presented in Figure 6 and its summarization in Table 4.

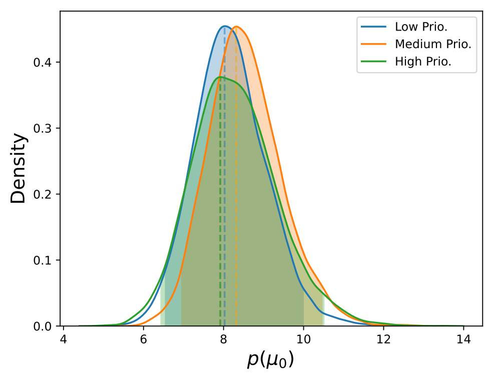


Figure 6: μ_0 Marginal Posterior distributions, 'HM-AP', 'priority' data groups.

The results suggest that the difference between the groups of reports with distinct priority is unclear. However, there are different levels of uncertainty interval across the μ_0 distributions, but the $\mu_{0_{MAP}}$ values are very similar. Finally, we highlight the average difference time between the groups (E[F1] values) of all groups: all of them are unrepresentative.

Table 4: μ_0 Posterior Distribution Summary, 'HM-AP', 'priority' data groups

Low Priority (a)

CI_L	CI_U	$\mu_{0_{MAP}}$	E[F1(a,b)]	E[F1(a,c)]	E[F2(a,b)]	E[F2(a,c)]
6.53	10.04	8.03	-0.37	-0.12	0.39	0.47

Medium Priority (b)

CI_L	CI_U	$\mu_{0_{MAP}}$	E[F1(b,a)]	E[F1(b,c)]	E[F2(b,a)]	E[F2(b,c)]
6.93	10.50	8.31	0.37	0.25	0.61	0.58

High Priority (c)

CI_L	CI_U	$\mu_{0_{MAP}}$	E[F1(c,a)]	E[F1(c,b)]	E[F2(c,a)]	E[F2(c,b)]
6.41	10.57	7.92	0.12	-0.25	0.53	0.42

4.3 RQ3. How does the code-churn size of fixing commits relates to the BFT?

Answer to RQ3: Considering the marginal posterior distribution of μ for groups of reports with higher and lower co/de-churn values, along with their summarization, we found that bug patches with greater code-churn values tend to need 5 more days (2 times more) to be fixed than bug patches with smaller code-churn values.

The Figure 7 shows four projects 'specific-models' marginal μ posterior distributions, using the code-churn data groups, while Table 5 shows the distributions summarization. Once again, as presented in the results for previous features, we selected four different scenario of possible conclusions.

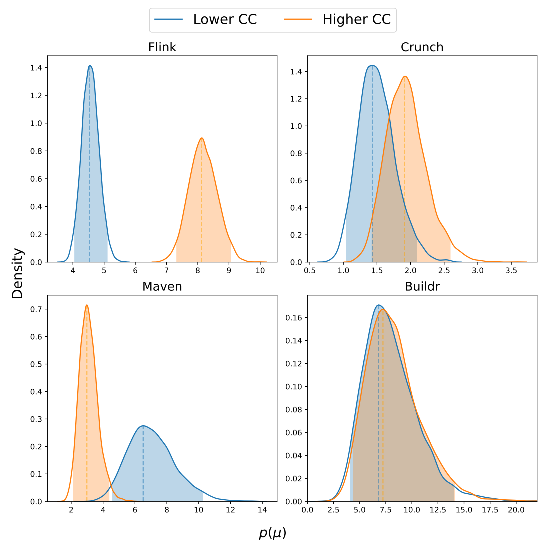


Figure 7: μ Marginal Posterior Distributions, 'specific-models'. The average bug fixing time of reports difference two groups of code-churn values. The conclusions diverge depending on the selected project.

The code-churn results present similar behavior as presented in the links results. Flink and Crunch results show that patches with higher code churn take more time to fix than those with lower

Table 5: μ_0 Marginal Posterior Distribution Summary, 'specific-models', 'code-churn' data groups.

Lower Code Churn (a)

Project	CI_L	CI_U	μ_{MAP}	E[F1(a,b)]	E[F2(a,b)]
Buildr	4.10	14.08	6.82	-0.26	0.47
Crunch	1.03	2.11	1.43	-0.44	0.14
Flink	4.04	5.12	4.54	-3.60	0.00
Maven	4.56	10.30	6.52	3.95	1.00

Higher Code Churn (b)

Project	CI_L	CI_U	μ_{MAP}	E[F1(a,b)]	E[F2(a,b)]
Buildr	4.24	14.28	7.24	0.26	0.53
Crunch	1.42	2.60	1.91	0.44	0.86
Flink	7.31	9.07	8.13	3.60	1.00
Maven	2.10	4.40	2.98	-3.95	0.00

code-churn sizes. However, in Flink, the difference is evident, while Crunch presents a significant overlap. Maven project presents an inverse behavior, with patches with higher values of code-churn taking less time to be fixed than the lower ones. In Buildr, the values for estimator μ are almost identical. Once again, the particular nature of each project shows dissonant conclusions.

With code-churn data groups, we fit another 'HM-AP' to draw our conclusions for code-churn group. The marginal μ_0 bug reports population posterior distributions of is presented in Figure 8 and the summarization in Table 6.

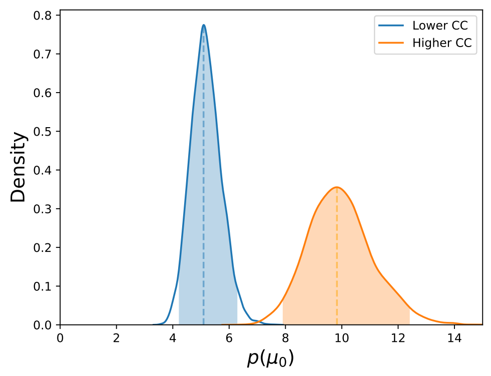


Figure 8: μ_0 Marginal Posterior Distributions, 'HM-AP', 'code-churn' data groups.

The results show a significant difference between both average bug fixing time, where reports patches with higher code-churn values (group 'a') need more time to be fixed than reports patches with lower code-churn values (group 'b'). The expected difference between both groups are 4.78 days, suggesting that bugs of group 'b' take, approximately, double the time be fixed than those of group 'a'. The probability of group 'b' being greater than group 'a' is 1.

Table 6: μ_0 Posterior Distribution Summary, 'HM-AP', 'code-churn' data groups.

Lower Code Churn

CI_L	CI_U	$\mu_{0_{MAP}}$	E[F1(a,b)]	E[F2(a,b)]
4.20	6.31	5.09	-4.78	0.00

Higher Code Churn

CI_L	CI_U	$\mu_{0_{MAP}}$	E[F1(a,b)]	E[F1(a,c)]
7.89	12.42	9.83	4.78	1.00

5 DISCUSSION

This paper presents an analysis of the interplay between the three bug reports features - links, priority, and bug-fixing code churn size - and the bug-fixing time. To the best of our knowledge, this is the first study using hierarchical Bayes to extrapolate results from multiple projects and assess the global effect of different attributes on the BFT.

Regarding the results, we highlight a few points. First, the relationship between bug reports seems to be overlooked, which appears to be wasted potential for deeper analysis and predictive models. We look to use Graph Neural Network [45] to verify how these relations can improve the state-of-art estimation results in future works. The results regarding code-churn can also be an exciting addition to BFT estimation when used with bug localization strategies. To conclude the analysis of the results, the priority not having much evidence of being impactful on the BFT may not discard it entirely from being used in predictive models, as it can be related with other features. The priority seems to provide some contextual information about the situation of an open bug compared to other ones. Priority may play a role when the context of the specific report is known, which is rarely the case in several papers. For instance, a newly reported high-priority bug (blocker, critical) can take more time to be fixed if it competes for resources with several other high-priority bugs. The same report could be fixed early in a scenario of several low-priority bugs. Also, it can be related to other responses in the platform as response-time from other developers, the number of comments and watches, or the time to review proposed patches. This also seems to be an interesting path to understand the priority role in the bug fixing process.

6 THREATS TO VALIDITY

The threats to the validity of our investigation are discussed using the four threats classification (conclusion, construct, internal, and external validity) presented by Wohlin et al. [42].

Conclusion Validity the main threats to this validity concern the choices in the modeling process: the weakly prior, the likelihood function. Regarding the first choice, Bayesian statistics results benefit from using more representative prior, ideally from a different data source (i.e., bug reports from other projects or posterior from previous analysis). However, even if we don't use data from other projects, we perform a sensitivity analysis to provide a reasonable prior based on previous studies. Some studies [1, 41] suggest that bugs are generally fixed in a few days, while other studies show that some bugs can take months to be considered fixed [26]. We provide a broad enough prior distribution to consider these cases. The expected BFT value is close to 2~3 days (most common cases), but also allows the model to contemplate instances with hundreds of days, even they are less plausible. The replication package[5] provides two sensitivity analyses to cover both threats. We show that the selected parameters and likelihood functions are appropriate: a predictive analysis of the prior and an analysis of how well a log-normal fit the log-BFT of most projects.

Internal Validity The existence of other features that can be highly correlated with the analyzed features and that can be the *actual* causal effect of the bug-fixing time. For instance, we show that reports with links present higher BFT. However, our analysis does not consider other features (*e.g.*, the number of comments and the existence of attached patches) that can be highly correlated with the presence of links and are the actual cause of higher BFT. We argue that the number of analyzed projects and bug reports — more than 70.000 reports from 55 projects — mitigate the chances of these correlations propagating thought all projects.

Construct Validity In the 'links' analysis, we had to ignore the types of relations between reports, only considering the *existence* of a link. This simplifies the analysis as we are not able to indicate which type of links really and how much it impacts the BFT. However, we had to perform this simplification due to the size of a few projects, as some of them do not have enough data to perform this level of type-of-links groups granularity.

External Validity All projects are open-source from the Apache ecosystem, indicating some source of low generalization capability. However, we argue that the sample contains 55 projects from nine categories (big-data, database, machine learning, library, to cite some), with different maturity levels, some of them dated from 2002 and others from 2018. We selected this dataset because this diversity allows us to generalize the results with more certainty.

7 RELATED WORK

Hooimeijer and Weimer [19] discuss the process of modeling bug reports quality. They present a bug report quality descriptive model based on 27,000 Mozilla Firefox reports. The analysis shows that the presence of an attachment tends to lead to higher values of bug-fixing time, while the comment count suggests that bugs that receive more attention get fixed faster. The self-reported severity at the report creation also plays a role in bug fixing.

Zimmermann et al. [46] investigates the quality of bug reports from the perspective of developers. To find out which features and elements matter the most, they asked several developers from Apache, Eclipse, and Mozilla projects to perform two tasks: i) a survey on bug reports important information and ii) rate the quality of bug reports on a five-point Likert scale (from very poor to very good). The analysis of the 466 responses revealed that most developers consider steps to reproduce, stack traces, and test cases as helpful. The authors also show that bug reports containing stack traces get fixed sooner, and those easier to read have lower lifetimes.

The study of Soltani et al. [31] aims to establish the significance of bug report elements. The authors interviewed 35 developers to gain insights into the importance of various contents in bug reports, followed by a survey applied to 305 developers. Based on the acquired data from these moments, the authors conclude that the essential elements are crash description, reproducing steps or test cases, and stack traces. They also evaluate the quality of bug reports of the 250 most popular projects on Github. Their analysis shows that crash reproducing steps, stack traces, fix suggestions, and user contents, have a statistically significant impact on bug resolution times between 76% to 33% of the projects.

Sasso et al. [27] describes what makes a satisficing (a neologism combining the verbs to satisfy and to suffice) a bug report. Based on a proposed a questionnaire to an open-source community, the authors gather the perception of how difficult it is to provide distinct kinds of information during the bug report record. They also mined content from Bugzilla and Jira to understand what users and developers collect and provide during the bug reporting. Based on more than 650,000 bug reports and the results from the questionnaire, the authors evaluate how the completeness of standard and project-specific attributes in a bug report related to its lifetime, similar to the BFT concept in our study. Finally, they highlight that number of words in the description and the summary are the features that impact the prediction the most.

None of these studies evaluate the relationship between report links, priority and code-churn size, and the BFT. Also, none of them use Bayesian statistics or incorporate previous studies results into their analysis. For instance, the studies [27, 31, 46] applies a similar methodology when using survey and questionnaire, and [31, 46] conclude similar things. This is a good example where studies using data of the same subject, trying to answer similar RQ can be a benefit of using previous results as priors for their study. For example, if the results were modeled using Bayes statistics, providing a conclusion based on posterior distributions, one can continually use previous results to gather more evidence of previous findings. Also, none of the them present numerically the impact of the analyzed features (i.e. the impact in days of the existence or not of an specific feature).

8 CONCLUSION AND FUTURE WORK

We presented the use of Bayesian workflow to analyze bug report data and assess the influence of three features w.r.t. BFT: links/relation between reports, priority, and bug patch code churn size. A first proposed model consider only one project at a time, capturing the individual behavior of each software; and second one, a hierarchical model that allows generalizing the results for the bug reports population, abstracting the specificities of each project. Based on inference results, we showed evidence that priority plays no role in BFT. In contrast, bug reports with higher values of code churn or bugs reports related to other bugs (with links) need at least double the time to be fixed compared to their counterparts.

We highlight our two-fold contributions. First, the results may support practitioners during bug triage, applying the same methodology in their projects to help understand the features that impact the BFT the most in their projects. Second, software researchers can reuse the presented methodology in different data projects, using the presented posterior distributions as priors for other future analysis in a similar context.

As a future work, a regression analysis can provide a more thoughtful view of all features presented in the used dataset, not only the ones selected in this paper, and how they relate to the BFT.

[5]https://doi.org/10.6084/m9.figshare.20315076

REFERENCES

[1] Shirin Akbarinasaji, Bora Caglayan, and Ayse Bener. 2018. Predicting bug-fixing time: A replication study using an open source software project. *Journal of Systems and Software* 136 (2018), 173 – 186. https://doi.org/10.1016/j.jss.2017.02.021

[2] Wisam Haitham Abbood Al-Zubaidi, Hoa Khanh Dam, Aditya Ghose, and Xiaodong Li. 2017. Multi-Objective Search-Based Approach to Estimate Issue Resolution Time. In *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering* (Toronto, Canada) *(PROMISE)*. Association for Computing Machinery, New York, NY, USA, 53–62. https://doi.org/10.1145/3127005.3127011

[3] Rafi Almhana, Thiago Ferreira, Marouane Kessentini, and Tushar Sharma. 2020. Understanding and Characterizing Changes in Bugs Priority: The Practitioners' Perceptive. In *2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. 87–97. https://doi.org/10.1109/SCAM51674.2020.00015

[4] Olga Baysal, Reid Holmes, and Michael W. Godfrey. 2013. Situational Awareness: Personalizing Issue Tracking Systems. In *Proceedings of the 2013 International Conference on Software Engineering* (San Francisco, CA, USA) *(ICSE '13)*. IEEE Press, 1185–1188.

[5] M. Borg, D. Pfahl, and P. Runeson. 2013. Analyzing Networks of Issue Reports. In *2013 17th European Conference on Software Maintenance and Reengineering*. 79–88. https://doi.org/10.1109/CSMR.2013.18

[6] Fiorenza Brady. 2013. Cambridge University report on cost of software faults, Press release, 2013. http://www.prweb.com/releases/2013/1/prweb10298185.htm). Accessed: 2020-01-02.

[7] Gemma Catolino, Fabio Palomba, Andy Zaidman, and Filomena Ferrucci. 2019. Not all bugs are the same: Understanding, Characterizing, and Classifying Bug Types. *Journal of Systems and Software* 152 (2019), 165–181. https://doi.org/10.1016/j.jss.2019.03.002

[8] Francisco Gomes de Oliveira Neto, Richard Torkar, Robert Feldt, Lucas Gren, Carlo A. Furia, and Ziwei Huang. 2019. Evolution of statistical analysis in empirical software engineering research: Current state and steps forward. *Journal of Systems and Software* 156 (2019), 246–267. https://doi.org/10.1016/j.jss.2019.07.002

[9] Neda Ebrahimi, Abdelaziz Trabelsi, Md. Shariful Islam, Abdelwahab Hamou-Lhadj, and Kobra Khanmohammadi. 2019. An HMM-based approach for automatic detection and classification of duplicate bug reports. *Information and Software Technology* 113 (2019), 98 – 109. https://doi.org/10.1016/j.infsof.2019.05.007

[10] Carlo Alberto Furia, Robert Feldt, and Richard Torkar. 2019. Bayesian Data Analysis in Empirical Software Engineering Research. *IEEE Transactions on Software Engineering* (2019), 1–1. https://doi.org/10.1109/TSE.2019.2935974

[11] Carlo A. Furia, Robert Feldt, and Richard Torkar. 2021. Bayesian Data Analysis in Empirical Software Engineering Research. *IEEE Transactions on Software Engineering* 47, 9 (2021), 1786–1810. https://doi.org/10.1109/TSE.2019.2935974

[12] Carlo A. Furia, Richard Torkar, and Robert Feldt. 2022. Applying Bayesian Analysis Guidelines to Empirical Software Engineering Data: The Case of Programming Languages and Code Quality. *ACM Trans. Softw. Eng. Methodol.* 31, 3, Article 40 (mar 2022), 38 pages. https://doi.org/10.1145/3490953

[13] A. Gelman, J.B. Carlin, H.S. Stern, D.B. Dunson, A. Vehtari, and D.B. Rubin. 2013. *Bayesian Data Analysis, Third Edition.* Taylor & Francis. https://books.google.com.br/books?id=ZXL6AQAAQBAJ

[14] Andrew Gelman, Aki Vehtari, Daniel Simpson, Charles C. Margossian, Bob Carpenter, Yuling Yao, Lauren Kennedy, Jonah Gabry, Paul-Christian Bürkner, and Martin Modrák. 2020. Bayesian Workflow. arXiv:2011.01808 [stat.ME]

[15] Philip J. Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. 2011. "Not My Bug!" And Other Reasons for Software Bug Report Reassignments. In *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work* (Hangzhou, China) *(CSCW '11)*. Association for Computing Machinery, New York, NY, USA, 395–404. https://doi.org/10.1145/1958824.1958887

[16] M. Habayeb, S. S. Murtaza, A. Miranskyy, and A. B. Bener. 2018. On the Use of Hidden Markov Model to Predict the Time to Fix Bugs. *IEEE Transactions on Software Engineering* 44, 12 (Dec 2018), 1224–1244. https://doi.org/10.1109/TSE.2017.2757480

[17] N A Heard and P Rubin-Delanchy. 2018. Choosing between methods of combining p-values. *Biometrika* 105, 1 (01 2018), 239–246. https://doi.org/10.1093/biomet/asx076 arXiv:https://academic.oup.com/biomet/article-pdf/105/1/239/23884003/asx076.pdf

[18] Kim Herzig, Sascha Just, and Andreas Zeller. 2013. It's Not a Bug, It's a Feature: How Misclassification Impacts Bug Prediction. In *Proceedings of the 2013 International Conference on Software Engineering* (San Francisco, CA, USA) *(ICSE '13)*. IEEE Press, 392–401.

[19] Pieter Hooimeijer and Westley Weimer. 2007. Modeling Bug Report Quality. In *Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering* (Atlanta, Georgia, USA) *(ASE '07)*. Association for Computing Machinery, New York, NY, USA, 34–43. https://doi.org/10.1145/1321631.1321639

[20] John Kruschke. 2015. *Doing Bayesian Data Analysis (Second Edition).* Academic Press, Boston.

[21] Madhu Kumari, Ananya Misra, Sanjay Misra, Luis Fernandez Sanz, Robertas Damasevicius, and V.B. Singh. 2019. Quantitative Quality Evaluation of Software Products by Considering Summary and Comments Entropy of a Reported Bug. *Entropy* 21, 1 (2019). https://doi.org/10.3390/e21010091

[22] Alina Lazar, Sarah Ritchey, and Bonita Sharif. 2014. Improving the Accuracy of Duplicate Bug Report Detection Using Textual Similarity Measures. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (Hyderabad, India) *(MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 308–311. https://doi.org/10.1145/2597073.2597088

[23] R. McElreath. 2020. *Statistical Rethinking: A Bayesian Course with Examples in R and Stan* (2nd ed.). Chapman and Hall/CRC.

[24] Nachiappan Nagappan and Thomas Ball. 2005. Use of Relative Code Churn Measures to Predict System Defect Density. In *Proceedings of the 27th International Conference on Software Engineering* (St. Louis, MO, USA) *(ICSE '05)*. Association for Computing Machinery, New York, NY, USA, 284–292. https://doi.org/10.1145/1062455.1062514

[25] Quentin Perez, Pierre-Antoine Jean, Christelle Urtado, and Sylvain Vauttier. 2021. Bug or not bug? That is the question. In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. 47–58. https://doi.org/10.1109/ICPC52881.2021.00014

[26] Ripon K. Saha, Sarfraz Khurshid, and Dewayne E. Perry. 2014. An empirical study of long lived bugs. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. 144–153. https://doi.org/10.1109/CSMR-WCRE.2014.6747164

[27] Tommaso Dal Sasso, Andrea Mocci, and Michele Lanza. 2016. What Makes a Satisficing Bug Report?. In *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. 164–174. https://doi.org/10.1109/QRS.2016.28

[28] N. Serrano and I. Ciordia. 2005. Bugzilla, ITracker, and other bug trackers. *IEEE Software* 22, 2 (2005), 11–13. https://doi.org/10.1109/MS.2005.32

[29] Yonghee Shin, Andrew Meneely, Laurie Williams, and Jason A. Osborne. 2011. Evaluating Complexity, Code Churn, and Developer Activity Metrics as Indicators of Software Vulnerabilities. *IEEE Transactions on Software Engineering* 37, 6 (2011), 772–787. https://doi.org/10.1109/TSE.2010.81

[30] Ramin Shokripour, John Anvik, Zarinah M. Kasirun, and Sima Zamani. 2015. A Time-Based Approach to Automatic Bug Report Assignment. *J. Syst. Softw.* 102, C (April 2015), 109–122. https://doi.org/10.1016/j.jss.2014.12.049

[31] Mozhan Soltani, Felienne Hermans, and Thomas Bäck. 2020. The significance of bug report elements. *Empirical Software Engineering* 25, 6 (01 Nov 2020), 5255–5294. https://doi.org/10.1007/s10664-020-09882-z

[32] C. Albert Thompson, Gail C. Murphy, Marc Palyart, and Marko Gašparic. 2016. How Software Developers Use Work Breakdown Relationships in Issue Repositories. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. 281–285.

[33] Yuan Tian, David Lo, Xin Xia, and Chengnian Sun. 2015. Automated Prediction of Bug Report Priority Using Multi-Factor Analysis. *Empirical Software Engineering* 20, 5 (Oct. 2015), 1354–1383. https://doi.org/10.1007/s10664-014-9331-y

[34] M. T. Tomova, M. Rath, and P. Mäder. 2018. Poster: Use of Trace Link Types in Issue Tracking Systems. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*. 181–182.

[35] Richard Torkar, Carlo Alberto Furia, Robert Feldt, Francisco Gomes de Oliveira Neto, Lucas Gren, Per Lenberg, and Neil A. Ernst. 2021. A Method to Assess and Argue for Practical Significance in Software Engineering. *IEEE Transactions on Software Engineering* (2021), 1–1. https://doi.org/10.1109/TSE.2020.3048991

[36] Jamal Uddin, Rozaida Ghazali, Mustafa Mat Deris, Rashid Naseem, and Habib Shah. 2017. A survey on bug prioritization. *Artificial Intelligence Review* 47, 2 (01 Feb 2017), 145–180. https://doi.org/10.1007/s10462-016-9478-6

[37] Qasim Umer, Hui Liu, and Inam Illahi. 2020. CNN-Based Automatic Prioritization of Bug Reports. *IEEE Transactions on Reliability* 69, 4 (2020), 1341–1354. https://doi.org/10.1109/TR.2019.2959624

[38] Q. Umer, H. Liu, and Y. Sultan. 2018. Emotion Based Automated Priority Prediction for Bug Reports. *IEEE Access* 6 (2018), 35743–35752. https://doi.org/10.1109/ACCESS.2018.2850910

[39] Noah N. N. van Dongen, Johnny B. van Doorn, Quentin F. Gronau, Don van Ravenzwaaij, Rink Hoekstra, Matthias N. Haucke, Daniel Lakens, Christian Hennig, Richard D. Morey, Saskia Homer, Andrew Gelman, Jan Sprenger, and Eric-Jan Wagenmakers. 2019. Multiple Perspectives on Inference for Two Simple Statistical Scenarios. *The American Statistician* 73, sup1 (2019), 328–339. https://doi.org/10.1080/00031305.2019.1565553 arXiv:https://doi.org/10.1080/00031305.2019.1565553

[40] Renan Vieira, Antônio da Silva, Lincoln Rocha, and João Paulo Gomes. 2019. From Reports to Bug-Fix Commits: A 10 Years Dataset of Bug-Fixing Activity from 55 Apache's Open Source Projects. In *Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering* (Recife, Brazil) *(PROMISE'19)*. ACM, New York, NY, USA, 80–89. https://doi.org/10.1145/3345629.3345639

[41] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller. 2007. How Long Will It Take to Fix This Bug?. In *Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007)*. 1–1. https://doi.org/10.1109/MSR.2007.13

[42] Claes Wohlin, Per Runeson, Martin Hst, Magnus C. Ohlsson, Bjrn Regnell, and Anders Wessln. 2012. *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated.

[43] Xin Xia, David Lo, Ying Ding, Jafar M. Al-Kofahi, Tien N. Nguyen, and Xinyu Wang. 2017. Improving Automated Bug Triaging with Specialized Topic Model. *IEEE Transactions on Software Engineering* 43, 3 (2017), 272–297. https://doi.org/10.1109/TSE.2016.2576454

[44] H. Zhang, L. Gong, and S. Versteeg. 2013. Predicting bug-fixing time: An empirical study of commercial software projects. In *2013 35th International Conference on Software Engineering (ICSE)*. 1042–1051. https://doi.org/10.1109/ICSE.2013.6606654

[45] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. 2018. Graph Neural Networks: A Review of Methods and Applications. *CoRR* abs/1812.08434 (2018). arXiv:1812.08434 http://arxiv.org/abs/1812.08434

[46] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss. 2010. What Makes a Good Bug Report? *IEEE Transactions on Software Engineering* 36, 5 (Sep. 2010), 618–643. https://doi.org/10.1109/TSE.2010.63