This document sums up the work of the programming assignment along with the source code on GitHub.

Prepared by **Ali ARABAT**

Project link: aliarabat/rails-issues-analysis (github.com)

Introduction

Ruby on Rails (RoR) or basically Rails is a web framework, that was primarily written in the Ruby Programming language. It is designed to create production-ready web applications in an efficient manner. Specially, when it comes to large-scale applications, it significantly helps the developers to implement such solutions that might require a considerable amount of effort. Therefore, it adopts the well-known pattern **MVC**, in which many web-based frameworks make use of it**. MVC** stands for Model-View-Controller, a three-layered principle, each of which is specified to perform a single job that has nothing to do with other layers. As a result, applications can envision a smooth transition whenever there are unanticipated changes in the future. Thus, rather than the greater performance it has achieved, it eventually simplifies the development and the maintaining process of a given project lifecycle, particularly for those who will join the related company in the meantime. On top of that, this will clearly teach the currently-working developers to follow software engineering standards that often impact in a way the cost to be paid by the majority of companies.

In line with that fact that, enterprises thrive to achieve strategic goals, such as (improving the reliability of their projects as well as guaranteeing the availability of their services in the long run) stand to be an essential step ahead to prevent unusual or unwanted breakdowns. In this direction, we have observed the importance of conducting an empirical study of the issues being reported in the **Rails** repository to get a better understanding of how developers report such issues also how much interest they showed toward them. We have noticed a significant increase in the number of reported issues in the beginning of the year 2022 by 146% compared with the ones of the recent five years, this is only limited to the moment of writing this paper, still on month March. Therefore, one can expect a historical year in terms of reported issues, the number of contributions involved to overcome such obstacles also the influences on the projects being served in the industry.

On the other hand, we have also observed that there are major contributions from developers who usually showed too much interest in identifying issues, sharing them with the community on different forums such as GitHub, Slack, in a purpose to resolve each issue at hand. For instance, Jonathan Hefner was ranked the first on the list being the most issue-reporting developer with an estimated value of 20 issues. At the same time, investigating the most frequently issued rails' component plays a key role in determining their importance degrees, in which we can easily put our efforts, that might require urgent interventions. For example, **actionrecord** has gained a lot of interest over other categories with a distinguished issues number of approximately 200. Further studies are still needed to investigate whether the projects relying on Rails abide the best practices and conventions issued by the Rails' core team, this approach can also be extended to include even more components, in other words, it is more interesting to see how often their maintainers use the features leveraged by Rails where other ones could be blinded.

The rest of this document highlights the following sections: Section **1** discusses the mechanisms used to both collect and visualize data. Section **2** tends to respond the main questions of this work. Finally, Section **3** sums up the work.

Data Collection & Data Visualization

This section spots the light on the main steps followed to clearly collect the data used for analyzing this work, whereas the second section investigates the different ways through which we successfully plot the data into meaningful charts along with the used technologies to fulfill this need.

Data Collection

The data used during this study were obtained from GitHub, especially the Rails web-based framework repository. The study was mainly focused on issues being reported by developers. Thus, we follow the steps declared below:

- First, we queried the GitHub Database using a JavaScript API consumer, which is fetch. Since the number of items per request allowed is 100, we had to make 5 requests to finally get to 500, which is out target.
- The data, then, has been processed for the needed tasks of this study.
- Each given task stores its data in a dedicated CSV file tor further analysis.

Data Visualization

Data visualization is one of most important steps of the entire workflow. Thus, we were careful about choosing the appropriate programming language to accomplish this mission. Thus, we opted for python which the widely used language for exploring and visualization the data. This step primarily consists of two chart types (Horizontal lollipop and bar charts). The process of visualization includes the steps listed below:

- Uploading the above generated CSV files to python Jupyther notebook.
- The header should be removed for the following files issues_per_year, issues_per_reporter and issues_per_category.
- Finally, execution of the python script.

Results

The aim of this study is to gain an insightful understanding of how developers report issues and how they behave toward them. For this reason, we try to investigate the following four questions:

Q1: How do the number of issues evolve across time?

Q2: Are there any periods in which we get more issues?

Q3: Is there anyone who reports more issues than others?

Q4: What is the most popular category (label)?

Q1: How do the number of issues evolve across time?

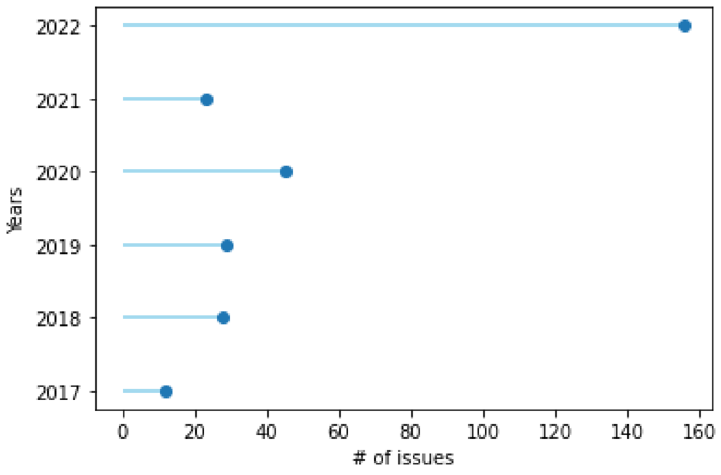

Figure 1 The number of reported issues for a given year

The percentage of issues for 2022 represents approximately 31.4% (157 samples) of the total studied issues (as shown in Fig1), by registering a significant increase of 146% compared with the ones of the last five years. Among the 31.4% studied issues, 60.5% have at least one comment, which clearly shows that more than half of the contributors share their ideas to reach out to an optimal solution.

Based on our manual analysis, we observe that **activerecord** is set to be the most investigated category with 46 out of 157, in other words, it has a great potential in terms of reported issues. However, developers might have inspected that many issues are related to either transaction with databases and associations between Rails' models. For instance, one issue message said that *Fix `remove_foreign_key` with `:if_exists`* whereas the other one was about to *Make ActiveRecord::Rollback bubble to outer transactions option when foreign key actually exists,* where both represent respectively models association and database transaction. We notice that **docs** have also received an important number of reported issues up to 19, which means that there was a lack in terms of information in the Rails-based documentation, thing that led developers to start contributing in a purpose to make the life of others much easier. Another category that attracted many eyes, **activesupport** (14 out of 157) which basically a library that comes with built-in utility classes and extensions that can be used within any rails' project to avoid reinventing the wheel. Other categories have received normal attentions regarding number of reported issues as less as 2 and as much as 14.

In terms of the issues' creator, we inspect that (65 ou 157) are being the contributors to that Rails' project. Besides, 22 out of 157 are just normal members. Whereas the remaining developers, which represent 71 out of 157 have no prior relation with Rails. Although, there is a huge number of contributions, it is of much interest to know whether the non-related 71 developers can actually have the sufficient potential for such a contribution in the future.

Q2: Are there any periods in which we get more issues?

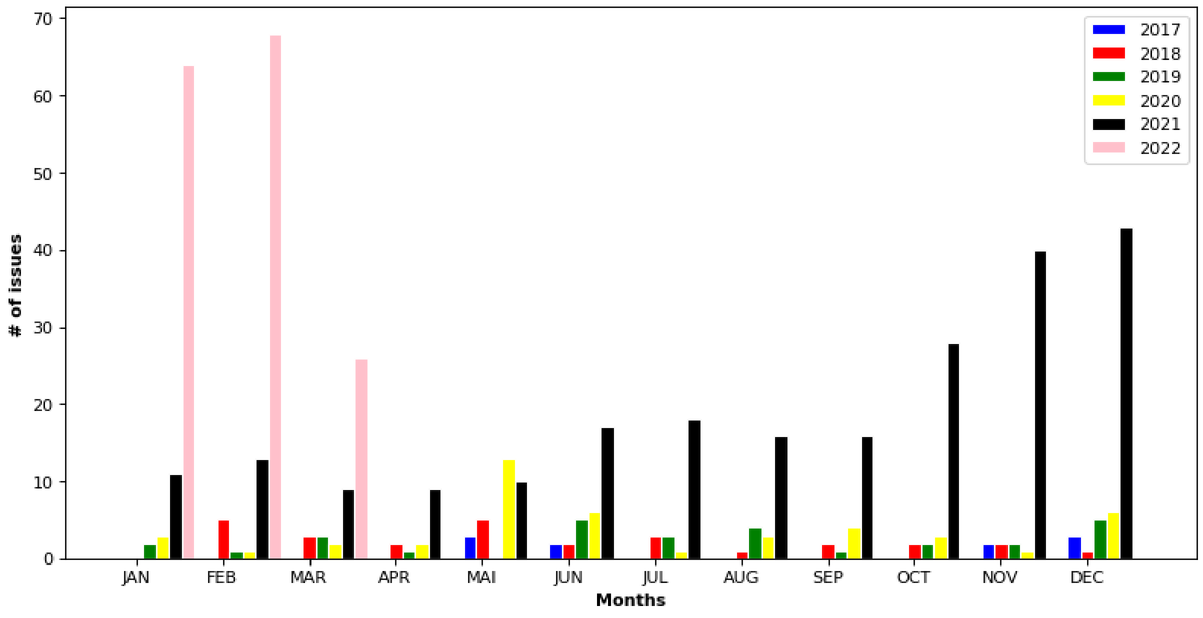

Figure 2 Number of monthly reported issues for a given year

Throughout the above figure, we observe that the number of reported issues between 2017 and 2020 is slightly small, and a bit identical between months, except for May 2020 which has known the highest value of around 13 compared to others. In 2021, we notice that number of reported issues increased sharply, in which the value gets higher from one month to another, especially over the last three months, where one can see substantial movement. During the first quarter of 2022, the number of reported issues is unique. We can observe a historical sharp rise by 146% in comparison with the last five years. That being said, people felt the greater importance of opening issues with the community in order to find suitable solutions to the encountered problem. Specially, the ones that are more critical to the main components of Rails project.

Q3: Is there anyone who reports more issues than others?

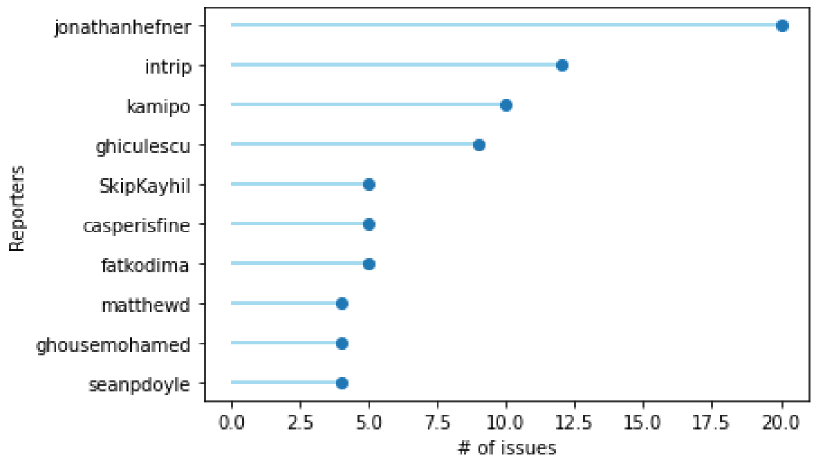

Figure 3 The top 10 most issue-reporting developers

Since the goal of this study is to gain a deep understating of how developers behave toward such reported issues. We observe a vital contribution among Rails project developers and maintainers during the last six years. As a result, such encouragements are always needed for the sake of attaining meaningful results, more precisely, regular improvements of such a project have to be always taken into account. Therefore, one can seek a small help that would someday solve many problems and make a major footprint toward progressive future. A true example of a subtle contribution, is that of **jonathanhefner** who was the top-rated issue-reporting developer to the Rails' project, he solely opened 20 issues, whereas **intrip** reported 12 issues. Kamipo was also one of the issuers with 10. The remaining has achieved contributions ranging from 4 to 9 issues.

Q4: What is the most popular category (label)?

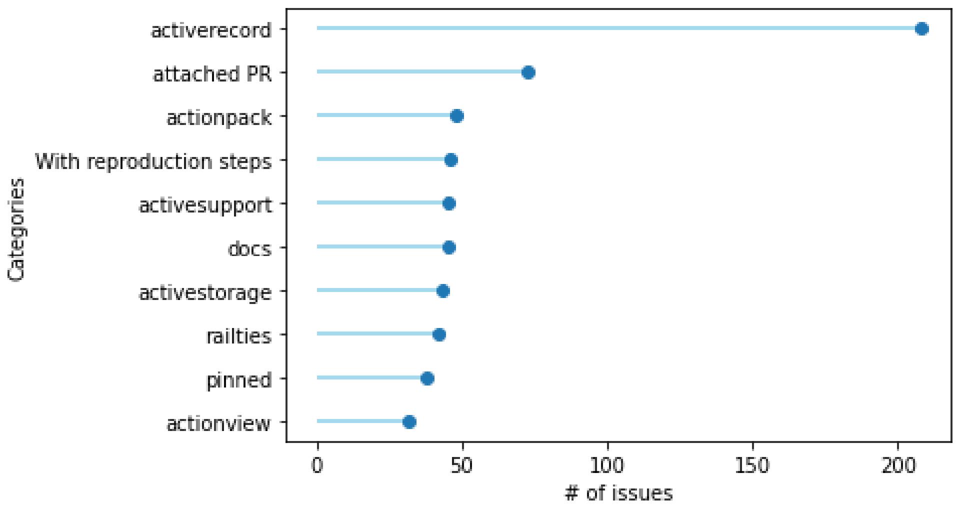

Figure 4 The top 10 most issued categories

Based on the figure above, we can see that **activerecord** is the most popular category used when creating such issue with more than 200 reporting statements. It acts like an ORM (Object Relational Mapping) in most of the other programming languages, its main job is guaranteeing a flexible relationship between Rails model classes and their corresponding in database tables as well as manipulating all the conventional methods when querying a database. This is one of the most featured integrations of an ORM. Besides, we observe that most of opened issues are often related to an association definition like (belongs to, has one or has many…). On top of that, there are major issues which primarily suffers from the lack of types compatibility between a Ruby Rails class and a database table. The other categories are similar in terms of the reported issues number as less as 35 and as much as 75.

Each of these categories is designed to do a specific task. For instance, **actionpack** groups a bunch of actions responsible for handling the incoming http request through ActionDisptach to an appropriate controller deriving from the ActionController::Base. **activesupport** offer a comprehensive library that helps saving more extra time during the development process.

Q5: Suggesting an open question:

During this study we have included so much metrics such as number of comments, labels also the association of issuers, to properly address the four main questions. Can we still envision better results and insights when including more advanced metrics (those we still did not use or basically look for a way to get them such a dedicated survey;) in the future?

To deepen our understanding on the current analysis. Can we consider extending this analysis on projects relying on Ruby Rails project? It might be a good idea to investigate the rationale uptake of basic or more advanced aspects of the framework.

Given the giant number of contributors, specially those who do not declare no prior relation, since we have no background about their proficiency. How far could they influence the performance of our analysis?

Conclusion

In this work, we performed an empirical study of how well developers report issues on the Rails project on GitHub based on several metrics. Thus, the studied issues were obtained using the js API consumer, for the visualization part, we opted for python, particularly the matplotlib library. Therefore, we have successfully addressed four main questions as follows with more in-depth interpretation, **first**, we observe that more than 31.4% of the studied issues have been exclusively reported over the first quarter of 2022, **second**, we notice that 2021-2022 was the most issues-received period, **third**, we investigate the most ranked issue-reporting developer, where jonathanhefner achieved the highest value with 20 issues, **fourth**, we notice that **activerecord** is the popular category being associated with such issues. Further analysis shall be made in the future to include more advanced metrics as well as transfer this learning on projects relying on Rails framework would be an outstanding approach.