GiLA: GitHub Label Analyzer

Javier Luis Cánovas Izquierdo, Valerio Cosentino, Belén Rolandi, Alexandre Bergel, Jordi Cabot

▶ **To cite this version:**

Javier Luis Cánovas Izquierdo, Valerio Cosentino, Belén Rolandi, Alexandre Bergel, Jordi Cabot. GiLA: GitHub Label Analyzer. International Conference on Software Analysis, Evolution and Reenginering (SANER), Mar 2015, Montreal, Canada. hal-01119396

HAL Id: hal-01119396
https://inria.hal.science/hal-01119396

Submitted on 23 Feb 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

GiLA: GitHub Label Analyzer

Javier Luis Cánovas Izquierdo*, Valerio Cosentino*, Belén Rolandi*, Alexandre Bergel†, Jordi Cabot*
*AtlanMod team (Inria, Mines Nantes, LINA), Nantes, France
†Department of Computer Science (DCC), University of Chile, Chile
*{javier.canovas,valerio.cosentino,maria.rolandi,jordi.cabot}@inria.fr †alexandre.bergel@me.com

***Abstract*—Reporting bugs, asking for new features and in general giving any kind of feedback is a common way to contribute to an Open-Source Software (OSS) project. In GitHub, the largest code hosting service for OSS, this feedback is typically expressed as new issues for the project managed by an issue-tracking system available in each new project repository.**

Among other features, the issue tracker allows creating and assigning labels to issues with the goal of helping the project community to better classify and manage those issues (e.g., facilitating the identification of issues for top priority components or candidate developers that could solve them). Nevertheless, as the project grows a manual browsing of the project issues is no longer feasible. In this paper we present GiLA, a tool which generates a set of visualizations to facilitate the analysis of issues in a project depending on their label-based categorization. We believe our visualizations are useful to see the most popular labels (and their relationships) in a project, identify the most active community members for those labels and compare the typical issue evolution for each label category.

I. Introduction

The development of Open-Source Software (OSS) relies on a community of users who freely contribute to their advance [1]. Among the different ways to contribute to OSS projects (e.g., testing, code contributions, etc.), probably the most common one (at least one of the easiest to start with) consists in providing feedback by reporting bugs or requesting improvements and new features. This feedback is typically expressed as new issues for the project managed by an issue-tracking system.

OSS projects are hosted in web-based code hosting platforms providing a source code repository for the project plus other project management tools. Among those platforms, GitHub[1] has become, with more than 10 millions of repositories, one of the most popular ones. Similar to the other platforms, GitHub provides a light-weight and flexible issue-tracker which also includes labeling support.

Labels (sometimes also known as tags) are a simple yet effective way to attach additional information (e.g., metadata) to project issues [2]. A label can give any user an immediate clue about what sort of topic the issue is about, what development task the issue is related to, or what priority the issue has. Furthermore, labels are also useful for project administrators, since they can serve both as classification and filtering mechanism, thus facilitating the managing of the project. The GitHub issue-tracking system comes with a few generic labels (i.e., *bug*, *duplicate*, etc.), but more interestingly, it provides the capability to create new custom labels that are adapted to the specificities of the project[2].

Labeling is regarded as a useful technique to organize and manage projects (see some anecdotical evidence[3]) but as the project grows, the number of labels and issues also increases and their analysis and management becomes more and more difficult. As a consequence, it may be no longer feasible to infer relevant information by manually browsing them, risking losing useful information for the optimal evolution of the project.

We believe that visualization techniques could be applied here to overcome this challenge. In particular, this paper proposes a tool to better understand how labels are being used in GitHub projects, with the aim of providing more insights into how such projects are being managed. Our tool provides three visualizations addressing three different viewpoints, specifically:

- *V1* Label usage, which helps to identify the most used labels and which ones are commonly used together.
- *V2* User involvement, which allows discovering the most active and knowledgeable users around each label.
- *V3* Typical Label timeline, which provides some insights about how issues under that label evolve over time (e.g., time to be treated).

The tool, called GiLA, is available online [3] (together with a demonstration video [4]) and can be used to explore these viewpoints on all the original projects (i.e., projects that are not a fork of a previous project) in GitHub. We believe that the results favour not only a better comprehension of the project but also help in its advancement, e.g., by helping to quickly identify experts on a particular topic/label.

The remainder of the paper is organized as follows. Section II reports on the existing related work. Section III describes and illustrates the three visualizations while Section IV presents the tool functionalities, its architecture and inner workings. Finally, Section V concludes the paper and presents some future work.

[1] http://github.com

[2] Browsing issues for any project quickly uncovers the usage of custom labels, some of them such as *documentation* or *feature* even more popular than some of the default ones, see: http://modeling-languages.com/use-of-issue-labels-in-github (accessed 11-26-14)

[3] http://www.stateofcode.com/2013/06/using-github-issues-effectively (accessed 11-26-14)
http://www.ianbicking.org/blog/2014/03/use-github-issues-to-organize-a-project.html (accessed 11-26-14)
https://twitter.com/briantford/status/492431188578275328 (accessed 11-26-14)

II. Related Work

In the last years, we have witnessed the development of many tools and services focused on the exploration and analysis of software repositories, mainly GitHub hosted ones. Even GitHub itself promotes it by means of organizing the so-called *GitHub Data Challenge*. Some examples would be: (1) *Octoboard*[4], which provides a dashboard showing overall activity metrics in GitHub; (2) *Popular Coding Convention*[5], which analyzes the main programming conventions used in GitHub; or (3) *The Open Source Report Card*[6], which provides an enriched version of developer profiles. Commercial tools such as *KeyLines* are also starting to incorporate GitHub analysis as part of their offering or at least as part of their marketing efforts[7].

Despite all these existing tools, to the best of our knowledge our tool is the only one focusing on the analysis and visualization of how issue labels are used in a project. Other works have been focused on recommending the most suitable labels for software artefacts (e.g., [5], [6], [7]) or have studied the role of issue-tracking systems in software development (e.g., [8], [9]), however, little attention has been paid to study how issue labels are used, in particular, with the aim of proposing useful visualization techniques for them.

III. Visualizing How Labels are Used

This section motivates and explains the three visualizations calculated and rendered by our tool. In particular, such visualizations give insights about (1) the general use of labels in the project, (2) the user involvement on each label category, and (3) the evolution of issues classified under a given label. We illustrate all of them using the *Netty* project[8], an event-driven asynchronous network application framework developed in GitHub, as a running example.

A. Label Usage Visualization

Motivation. Having an overview of the labels used in the project and their distribution among issues provides some insights about the relevance of specific topics in the project community and, since an issue in GitHub can be tagged with one or more labels, the relationships between them. Therefore, with this visualization, you can quickly identify the most popular labels and which ones are commonly used together.

Visualization. Label usage information is displayed using a graph-based visualization. Nodes represent labels and edges represent relations between two labels. We say that labels L_1 and L_2 are related if there is at least one issue annotated with both L_1 and L_2 in the project. The strength of a relation between L_1 and L_2 is measured in terms of the number of issues using both labels. The thickness of the edge is used to visualize such strength factor (the greater the number the thicker the edge). Finally, the size of each node indicates the number of issues classified with that label (the greater the number the larger the node).

[4]http://octoboard.com (accessed 11-26-14)

[5]http://sideeffect.kr/popularconvention (accessed 11-26-14)

[6]https://osrc.dfm.io (accessed 11-26-14)

[7]http://keylines.com/network-visualization/graphing-github (accessed 11-26-14)

[8]https://github.com/netty/netty (accessed 11-26-14)

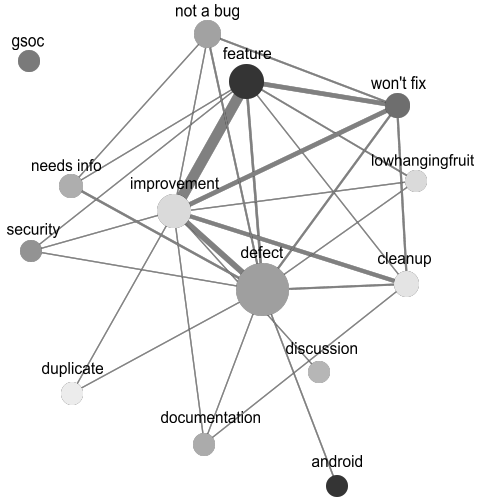

Fig. 1: Label usage visualization for the *Netty* project.

Example. Figure 1 shows this visualization on the example project. According to the node size, we can see that the most used label is *defect* followed by *feature* and *improvement*. The visualization also shows that *improvement* and *feature* are the labels most strongly connected, that is, the labels most frequently appearing together. *Improvement* can also be commonly found together with *defect*, *won't fix* and *cleanup* while *feature* is often used with *won't fix* issues. On the other hand, the label *gsoc* is always used alone. These results may help project developers to identify that the development effort is normally spent in improving and fixing issues of the project while little effort seems to be required to other activites such as documentation or security. It also suggests that quite a number of improvements and feature requests are discarded (relation with *won't fix* label).

B. User Involvement Visualization

Motivation. Visualizing the people actively participating (i.e., opening, closing, commenting) on issues in a given label category can quickly unveil the community leaders for the different aspects of the project. In particular, it can help identifying the collaborators that most frequently take decisions (e.g., accepting or rejecting issues) with regards to issues affecting a given topic. Similarly, we could use it to uncover potential experts on the topic and therefore people we could assign new issues on that same category in the future.

Visualization. User involvement is shown in a graph-based visualization that highlights the contribution of users on issues tagged with a given label. Users are represented as boxes while the label is represented as a circle in the center. The box size and edge thickness indicate the number of times such user has contributed to issues with that label. In particular, the box width and height are proportional to the number of created and closed issues, respectively. Additionally, box colors allow distinguishing between collaborators (i.e., administrators or members with high-level privileges, as white boxes) and users (i.e., black boxes). Besides, the edge between a user and the label indicates the number of comments such user has made on issues for that label. As before, the thickness indicates the degree of contribution.

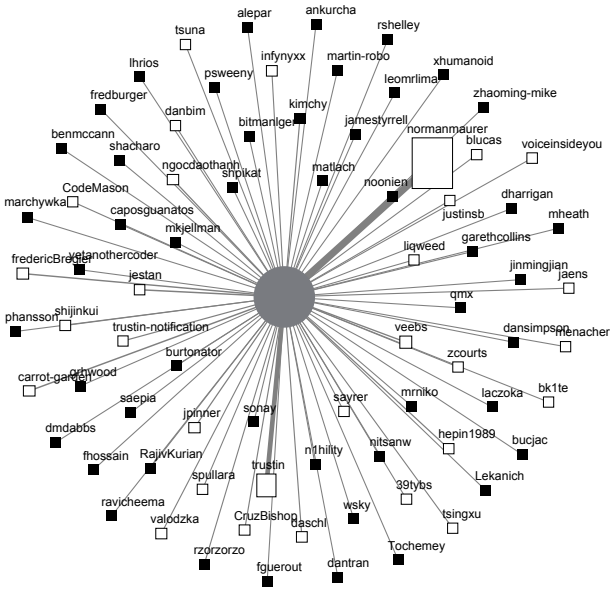

Fig. 2: User involvement visualization for label *improvement* in the *Netty* project.

Example. Figure 2 shows this visualization for the label *improvement* in the example project. From the size and color of the user rectangles we can see that *normanmaurer* and *trustin* are the collaborators that most often participate on *improvement* issues, specially *normanmaurer*. The fact that, for this user, the rectangle height is larger than its width, shows that such user has closed issues more times than opened them. The edge connecting the user rectangle with the label also reveals that *normanmaurer* and *trustin* are the users more actively commenting this kind of issues.

C. Label Timeline Visualization

Motivation. Analyzing how issues in each label typically evolve over time provides some insights about the main priorities and interests in a project (e.g., what type of issues receives the fastest initial response, which are usually solved quickly and what is the expected end resolution for issues classified under a certain label).

Visualization. Issue evolution is represented as a tree-like visualization that shows the average time for some important events in the lifecycle of issues tagged with a given label. The tree has a main path which includes two events: (1) average time for the first comment and (2) average time for the first comment from a collaborator of the project. This path then forks into three subpaths to represent (1) the percentage of issues closed (and the average time to be closed), (2) the percentage of issues merged (and the average time to be merged) and (3) the percentage of issues still open (and their average age).

Example. Figure 3 shows the resulting visualization for the label *feature* of the Netty project. The white and black stopping points tell that, in average, it takes around 7 days until an issue with this label receives the first comment (both from any user and a collaborator). As for the resolution, we can see that 5.73% of the issues are merged, 79.74% are closed and 14.54% are still open. The ordering of the branches reflects

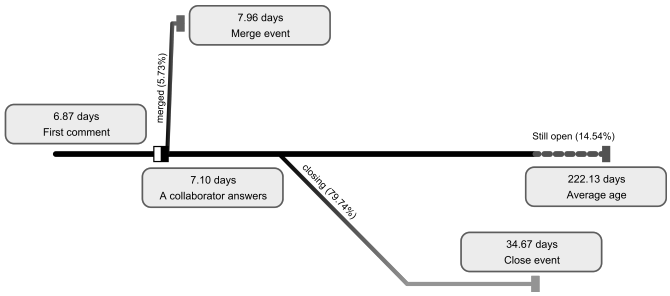

Fig. 3: Label timeline visualization for label *feature* in the *Netty* project.

that issues are merged faster (7.96 days) than they are closed (34.67 days in average) whereas the average age for pending issues is 222.13 days.

IV. Tool Support

These visualiations have been implemented as part of the a web application, called GiLA [3]. This section describes its main functionalities and architecture.

A. Functionalities

To use the tool, users have to access to the GiLA website [3]. Figure 6 shows the landing page of GiLA, shortly describing the available visualizations and together with a combobox to select the project to analyze. The combobox is initialized with a set of popular projects but users can just type the name of the project they would like to see. Once one project is selected, the user can access to the results page by clicking on the *Show me!* button.

The results page includes first three basic metrics concerning the labels usage in the project and then the three visualizations described before. Figure 7 shows an screenshot of the results page for the *Netty* project. While the first visualization is shown as soon as the user loads the page, the second and third visualizations (i.e., user involvement and label timeline visualizations) are initially empty since the user has to first choose the specific label to use in the visualization. Once done, the corresponding visualization is rendered and shown. Additionally, visualizations include tooltips to help the user to understand the results. For instance, in the last visualization the user can pass the mouse over the elements to visualize extra information.

B. Tool Architecture

Figure 5 depicts the main components of the GiLA architecture. As can be seen, GiLA is decomposed into a server side, which includes the dataset and the web services; and the client side with the website.

As dataset to perform the analysis, GiLA uses GHTorrent, an scalable and offline mirror of data offered through the GitHub REST API [10]. The dataset is provided as both relational and NoSQL databases. The former is the one used as source for the analysis. This database is loaded into our server and preprocessed to speed up the queries later on. In short, once a visitor selects a project, the tool does not trigger the computation of the data required for the visualization but

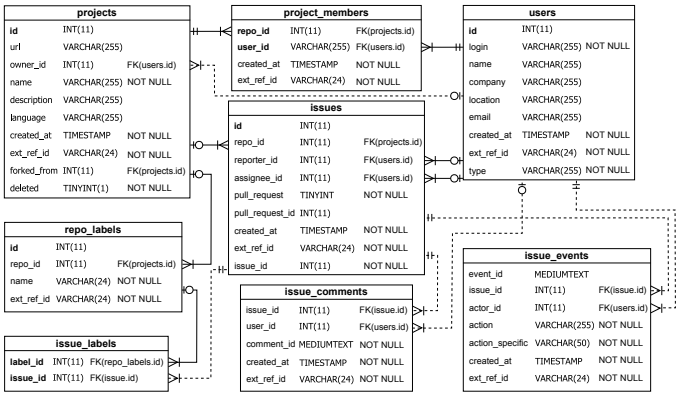

Fig. 4: Excerpt of the database schema considered.

directly queries the calculated results stored in auxiliary tables. As a trade-off, the results show the label usage in the project until the latest snapshot of GHTorrent uploaded to the tool. New snapshots can be imported by means of a set of developed scripts which update the database and create the auxiliary tables needed. Projects that are a fork of other projects are filtered out.

An excerpt of the core tables in the database schema is shown in Figure 4. The database stores information about projects, members, issues and labels. A project (see `projects` table) is managed by two kinds of *collaborators*: the owner[9] and the project members (see `project_members` table), the former has full control over the repository, whereas the latter are granted full management permissions. Each project keeps track of the submitted issues (see `issues` table), along with their labels (see `issue_labels` table), events (see `issue_events` table) and comments (see `issue_comments` table). Any user (see `users` table) can submit and comment issues, however, only collaborators can close (merge or reject) or reopen them. In addition, they are the only ones that can define labels (see `repo_labels` table) and assign them to issues.

On the server side, a set of web services (one per visualization plus one returning the list of GitHub projects) implemented as Java servlets are in charge of querying the database data and give it back to the client side for its rendering. For this rendering phase, and after evaluating several alternatives (among them D3[10], Google Charts[11], pygal[12] or Processing[13]) we opted for D3, a JavaScript library for manipulating documents based on data which facilitates the creation of SVG-based graphics. We believe this was the most suitable approach in our case given its efficiency and broad support.

V. Conclusion

In this paper we presented a tool to help managers and developers to better understand how issue labels are employed in their OSS projects. The tool comes with three visualizations

[9]Note that the owner can be either a single user or an organization (see `organization_members table`)

[10]http://d3js.org (accessed 11-26-14)

[11]https://google-developers.appspot.com/chart (accessed 11-26-14)

[12]http://pygal.org (accessed 11-26-14)

[13]https://www.processing.org (accessed 11-26-14)

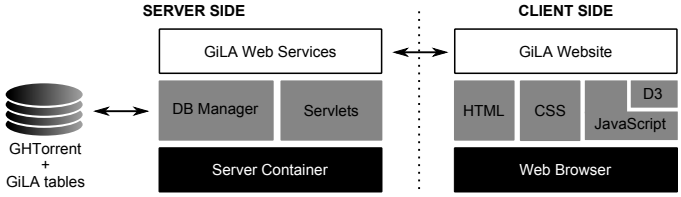

Fig. 5: Tool Architecture.

to highlight the frequency and relationships between labels, which users are involved in the creation/resolution of issues categorized with such labels and their typical evolution along the development process. The tool is provided as a web service available at [3] for the analysis of any GitHub project. A demonstration video has also been made available [4].

Our future plans for the tool include adding information about the evolution of a set of labels associated to an issue (when they are added/removed and by who) and adding some label-based metrics that help compare your project with regard to other GitHub projects using similar sets of labels. We are also interested in evaluating the scalability of the approach (e.g., how the database update performs when new GHTorrent snapshots are released) as well as conducting a user study to collect developer comments that may improve our visualizations. Finally, we would like to extend our approach to other web-based code hosting services (e.g., BitBucket, SourceForge, etc.) to identify whether there exist similarities in the way of managing projects across different social coding platforms.

References

[1] C. Bird, A. Gourley, P. Devanbu, U. C. Davis, A. Swaminathan, and G. Hsu, "Open Borders? Immigration in Open Source Projects," in *MSR conf.*, 2007, pp. 6–13.

[2] M. Storey, J. Ryall, J. Singer, D. Myers, and M. Muller, "How Software Developers Use Tagging to Support Reminding and Refinding," *IEEE Trans. Softw. Eng.*, vol. 35, no. 4, pp. 470–483, 2009.

[3] GiLA website. `http://atlanmod.github.io/gila` (accessed 11-26-14).

[4] GiLA demo video. `http://youtu.be/qZ1PFBTcs2A` (accessed 11-26-14).

[5] J. M. Al-Kofahi, A. Tamrawi, and T. N. Nguyen, "Fuzzy Set Approach for Automatic Tagging in Evolving Software," in *ICSM conf.*, 2010, pp. 1–10.

[6] X. Xia, D. Lo, X. Wang, and B. Zhou, "Tag Recommendation in Software Information Sites," in *MSR conf.*, 2013, pp. 287–296.

[7] S. Wang, D. Lo, B. Vasilescu, and A. Serebrenik, "EnTagRec: An Enhanced Tag Recommendation System for Software Information Sites," in *ICSM conf.*, 2014, pp. 291–300.

[8] T. F. Bissyande, D. Lo, L. Jiang, L. Reveillere, J. Klein, and Y. L. Traon, "Got issues? Who Cares About it? A Large Scale Investigation of Issue Trackers from GitHub," in *ISSRE symp.*, 2013, pp. 188–197.

[9] D. Posnett, A. Hindle, and P. Devanbu, "Got Issues? Do New Features and Code Improvements Affect Defects?" in *CSMR conf*, 2011, pp. 211–215.

[10] G. Gousios, "The ghtorrent dataset and tool suite," in *MSR conf.*, 2013, pp. 233–236.

Fig. 6: Landing page of GiLA.

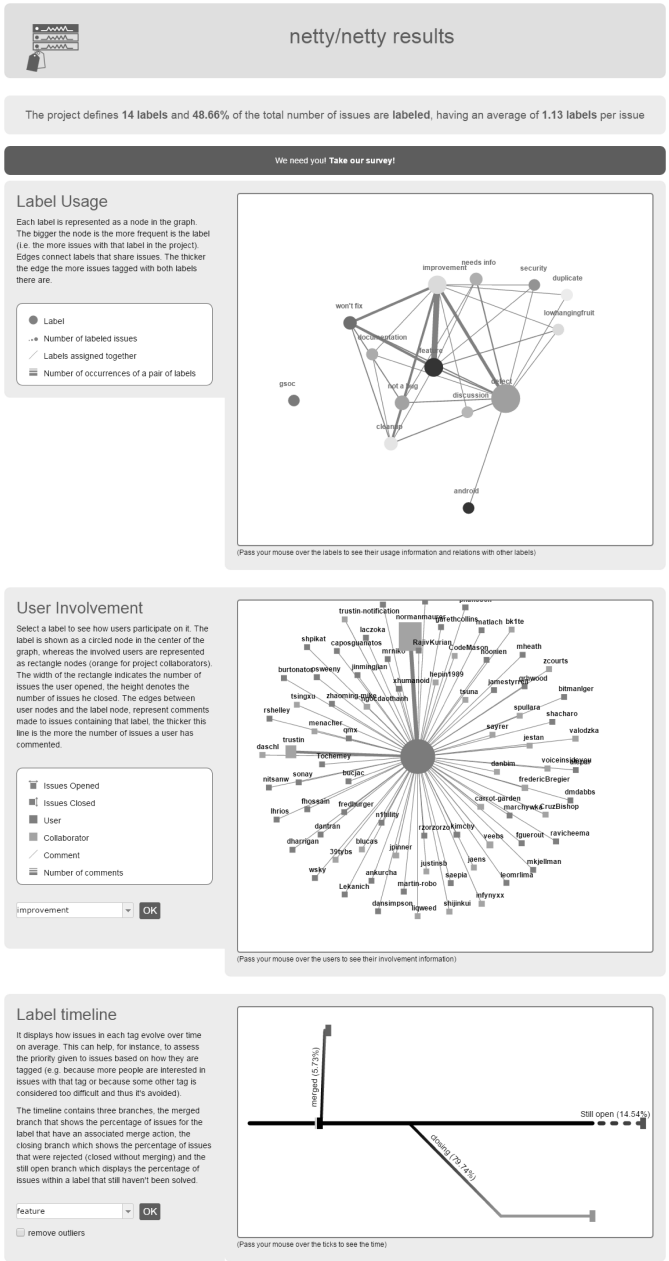

Fig. 7: GiLA results page for the *Netty* project.