Using PageRank to Reveal Relevant Issues to Support Decision-Making on Open Source Projects

Alessandro Caetano, Leonardo Leite, Paulo Meirelles, Hilmer Neri, Fabio Kon, Guilherme Travassos

▶ **To cite this version:**

Alessandro Caetano, Leonardo Leite, Paulo Meirelles, Hilmer Neri, Fabio Kon, et al.. Using PageRank to Reveal Relevant Issues to Support Decision-Making on Open Source Projects. 14th IFIP International Conference on Open Source Systems (OSS), Jun 2018, Athens, Greece. pp.102-113, 10.1007/978-3-319-92375-8_9 . hal-01875490

HAL Id: hal-01875490

https://hal.inria.fr/hal-01875490

Submitted on 17 Sep 2018

Using PageRank to reveal relevant issues to support decision-making on open source projects

Alessandro Caetano[1], Leonardo Leite[2], Paulo Meirelles[2,3],
Hilmer Neri[1,4], Fabio Kon[2], and Guilherme Horta Travassos[1]

[1] COPPE – Federal University of Rio de Janeiro
{alessandrocb,gth}@cos.ufrj.br
[2] FLOSS Competence Center – University of São Paulo
{leofl,kon}@ime.usp.br
[3] Department of Health Informatics – Federal University of São Paulo
paulo@softwarelivre.org
[4] UnB Faculty in Gama – University of Brasilia
hilmer@unb.br

Abstract. Software release planning is crucial to software projects that adopt incremental development. Open source projects depend on their globally distributed maintainers' communities who share project information, usually described in the software project repository as *issues*, to plan the contents and timing of the next releases. This paper introduces an approach based on software *issues* to support decision-making regarding open source software development activities such as release planning and retrospectives. It uses the PageRank algorithm to suggest an importance ranking of the software *issues* based on the *issues* dependencies topology. When based on a highly connected topology, project leaders can use this rank as an input to planning activities. The observation of two open source projects indicates the feasibility of our approach.

Keywords: Open Source Software, Free Software, Issue Management, Decision-Making, PageRank, Empirical Software Engineering.

1 Introduction

The developers' community is responsible for managing the next evolutionary actions in Open Source Software (OSS) projects. Such steps can identify different types of software *issues*, such as new features, artifacts, improvements, comments, bug fixes, among others. Collaborative Development Environments (CDE) [3], such as GitHub and GitLab, provide essential collaboration tools, such as *issue* trackers, in addition to their code repositories. An *issue* tracker supports the community in registering new *issues* and discussing them, keeping track of future work as well as recording the achieved results. Although the concept of "issue" has a broader meaning than the concept of "bug" (a.k.a. software defect), in some contexts, an *issue* tracker can be called a "bug tracker" and its produced report, a "bug report". An *issue* tracker can provide essential data about the software project history and status. Data-driven decisions can contribute to the software development process improvement, helping to the timely delivery of high-quality software [6].

Researchers have conducted investigations on mining software artifacts to provide useful insights for the decision-making process on software projects [1]. Codemine [6], for example, collects and analyzes engineering process data from across a diverse set of Microsoft product teams. Baysal *et al.* inquire Mozilla developers about how qualitative dashboards can support real-time developer decision-making for daily tasks [2]. Robles *et al.* mine commit history to estimate effort spent on a project [14]. Borges *et al.* mine open source GitHub repositories to predict popularity based on repository properties [4]. In our work, *issues* are the software artifacts mined to support decision-making.

An essential input data to planning activities are the history of the project (past *issues*) and the set of opened *issues*. One could consider a set of related *issues* as indicating some relevant theme within an OSS community. Open *issues* can point to essential topics so far neglected and deserving attention. It is also possible to use information regarding closed *issues* to observe where a project community concentrated its efforts. Thus, looking at the history of the *issues* is similar to performing a retrospective study, in which the software development team can learn from the experiences and plan for future improvements [15]. It is possible to use the retrospective based on the history of the *issues* to support decision-making and the prioritization of activities, and also to help identify most energy spent by the team during releases. The relevance of observing previously invested efforts is evident in the research of Robles *et al.*, in which a model estimates the effort on the OpenStack system based on its commit history [14].

Goyal and Sardana explore *issues* mining, comparing techniques for assigning the developer with maximum expertise related to a given bug to resolve it [8]. The match considers various meta-fields of the target bug and the meta-fields of bugs already solved by the developer. The authors also explore the effect of knowledge decay over time, a different meta-field weighting strategy, and developer commits history. Although our analysis target is the same, the bug report, our perspective is different, since we analyze the relevance of *issues* for the community as a whole, and not for specific developers.

Some works handle the summarization of bug reports containing lengthy conversations, so the reader can quickly grasp what matters in a given bug report. He *et al.* focus on summarization improvement based on duplicated bug reports analysis [10]. They apply the PageRank algorithm in a network in which nodes are the sentences of a bug report, and the similarity among them defines the edges, so the rank of sentences defines which ones belong to the summary. Although we also use PageRank to bug reports, our approach is different since we apply PageRank to reveal relevant *issues* from an *issue* set, and not to summarize a single bug report.

From another point of view, Steinmacher *et al.* identified 50 entry barriers faced by new developers in OSS projects [16]. Among them were i) the lack of a list of project needs and *issues*, ii) the organization of the backlog in the repository, and iii) the access to the tasks. Thus, besides supporting planning and retrospective activities, the history of the *issues* is input to newcomers to know the project better and decide where to focus efforts.

A well-organized project may have hundreds or even thousands of *issues*. Considering such significant set of *issues* is hard for decision-making, one should analyze every single software *issue* to produce consistent high-level patterns. For this reason, OSS communities could benefit from automated approaches to extract relevant information from the *issue* tracker to support decision-making activities.

In the *issue* tracker facilities provided by the GitHub and GitLab, users can reference an *issue* from comments or titles on other software *issues*, creating a network of linked *issues* (see Section 2). Considering such *issues* network, we propose in this paper an approach to define the top-ranked software *issues* from a given repository, by using the PageRank algorithm [13] to determine the relevance of *issues* based on their network centrality. *Issues* with a large number of references (dependencies) are more likely to be highly relevant, and consequently, those referenced by relevant *issues* are more likely to be also highly relevant. This way, the outcome of the PageRank algorithm (a list of software *issues* ordered by relevance) can support retrospective studies, planning activities, and look for relevant opportunities for newcomers to work out. Therefore, this paper intends to answer the following **research question**: *Can the PageRank algorithm identify relevant* issues *on OSS projects to support decision-making?*

We used two observational studies with two OSS projects to support the answering of this question. The observed results indicate that the use of PageRank could be somewhat feasible since some ranked relevant *issues* presented terms aligned to the planning documentation, and others were related to effort-consuming activities.

We organized the remaining of this paper as follows. Section 2 describes some basic concepts on the model of software *issues* and the PageRank algorithm. Section 3 presents our proposal to determine the relevance order of software *issues* according to the PageRank algorithm. In Section 4 we discuss the evaluation of our approach. Section 5 presents some threats to the validity of this study. Finally, Section 6 draws our conclusions and future work.

2 Background

In this section, we present how to build a graph representation of software *issues* extracted from Github and Gitlab platforms. We also discuss the PageRank, a link analysis algorithm that we applied to find the top-ranked software project *issues*.

2.1 The software *issue* model

Software *issues* are used in CDEs such as GitHub and GitLab to organize community activities. Software *issues* can have many purposes, such as discussing new ideas, asking for help, registering desired new features, artifacts improvements, fixes and so forth.

Each software *issue* has a number, a title, a textual description, and commentaries by its contributors. An *issue* title or commentary may refer to another software *issue* through a short link by using the "#" sign followed by the *issue* number. Commit messages can also refer to software *issues* in this style.

A software *issue* starts in the "open" state and finishes in the "closed" state. An *issue* is closed when it is addressed, rejected, or incorporated by another software issue. It may be assigned to a specific member of the software community and linked to a *milestone*. In this context, a *milestone* is a cohesive collection of software *issues*, possibly associated with a due date. A *milestone*, therefore, represents a project goal in a higher level of abstraction than just one issue.

2.2 The PageRank algorithm

Larry Page and Sergey Brin created the PageRank algorithm in 1999 [13]. It calculates the relative importance of web pages, and it has applications in search engines, traffic estimates, and web browsing. The premise of the PageRank algorithm is that each web page has some outbound and inbound links and that pages with a large number of links are more relevant than pages with fewer links. Besides, the algorithm takes into account the relevance of incoming links to a page: if the web page has an inbound link that has high relevance, that page tends to be more important than another one having several links coming from less relevant pages. The equation used to calculate the rank in the PageRank algorithm is:

$$PR(x) = \alpha \left(\frac{1}{N} \right) + (1 - \alpha) \sum_{y \in L(x)} \frac{PR(y)}{C(y)}, \tag{1}$$

in which x a web page, $PR(x)$ is the page rank of x. $L(x)$ represents the pages with links to x, $C(y)$ is the out-degree of y, α represents the probability of a random jump from one of the links, that is, in a random-surfer model it represents the probability of the surfer restart the algorithm at any given page, preventing the algorithm to be stuck in a node with zero out-degree. N represents the total number of pages analyzed.

3 Using PageRank to reveal relevant software *issues*

Our approach to analyzing the relevance of *issues* consists in using the PageRank algorithm, i.e., applying Equation 1 to software *issues* rather than to web pages. So, the first step of our procedure is generating a directed graph based on data retrieved from the software repository. We retrieve such data using the APIs provided by GitHub and GitLab. In the generated graph, a vertex represents a software issue, and a directed edge represents a link from a software *issue* to another one, only issues linked using the "#" sign are considered.

Before applying the PageRank algorithm, two transformations are performed in the graph. The first is to connect all software *issues* within the same milestone. We consider this connection because software *issues* within the same milestone have a semantic bind that means that all of them must be fulfilled so that a higher-level objective is achieved. The second transformation eliminates software *issues* with no links. A software *issue* is kept whether it has at least one inbound link or one outbound link. This transformation is made because software *issues* with no links are irrelevant to the analysis and affect the rank scale, so pruning these software *issues* generates a result that is easier to interpret.

After the graph is prepared, Equation 1 is applied to the graph with $\alpha = 0.85$, which is the default value used in NetworkX [9], a library for network manipulation we used, and the recommended value from the original Pagerank proposition [5]. The result is the assignment of a real number called "rank" to each software *issue* and a list of software *issues* ordered from the highest to the lowest rank.

Our implementation source code (including data retrieving, graph preparation, and PageRank execution) is available at GitLab[5]. We built the automated solution in Python. The top-ranked libraries we used are Matplotlib for data visualization [11]; Scipy[6]for numerical computing; Pandas[7] for data frame manipulation; NetworkX[9] to generate a digraph and compute the PageRank; and NLTK[8] to find the patterns we were looking for in the *issue* text.

4 Evaluation

We automated an approach to be used by any project that adopts the GitHub or GitLab *issue* trackers to organize the software community activities. We evaluated it by observing two OSS projects: Brazilian Public Software portal[9], with its project repository in its own GitLab instance, and Parliamentary Radar[10], which uses Github.com to host its project repositories. We choose these Brazilian software projects because they documented their roadmaps and backlogs, which enabled us to use this documentation to evaluate the algorithm results. Moreover, the proximity of our research group to the developers of these two software projects helped us to perform qualitative assessments regarding our findings with the project members.

The Brazilian Public Software (SPB) portal is an integrated platform for collaborative software development of OSS projects used by the Brazilian public administration [12]. It includes facilities for social networking, mailing lists, version control, and monitoring of source code quality, making it a system-of-systems.

[5] `https://gitlab.com/AlessandroCaetano/PageRanking`

[6] `https://scipy.org`

[7] `http://pandas.pydata.org`

[8] `http://nltk.org`

[9] `https://softwarepublico.gov.br`

[10] `http://radarparlamentar.polignu.org`

The Parliamentary Radar project uses open government data to perform cluster analysis on bill votes of legislative houses of Brazil.

The input to PageRank algorithm is the software *issues* graph. Figures 1 and 2 present the generation of the software *issues* graphs, which is the first step of our approach, for the SPB Portal and Parliamentary Radar projects. In these figures, the tiny red circles (altogether forming an ellipse) represent the software *issues*, whereas the straight lines represent links between them. In short, the graphs use a circular layout, as proposed by Doğrusöz et al. [7].

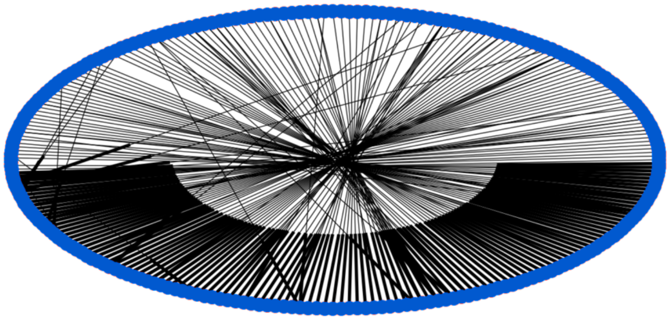

Fig. 1. Software *issues* circular graph of the SPB Portal project.

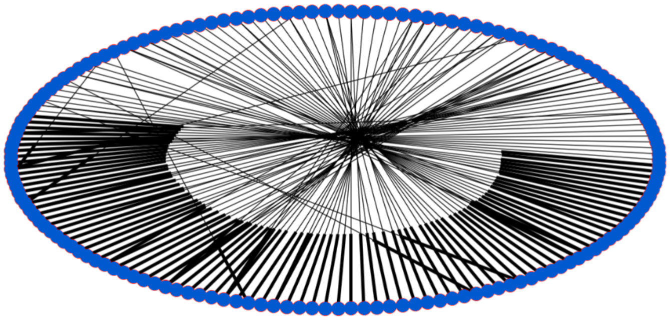

Fig. 2. Software *issues* circular graph of the Parliamentary Radar project.

The SPB graph already contains the transformations to linking the *issues* of the same milestone and pruning those with no links. The Radar Parliamentary graph includes the transformation of pruning software *issues* with no links. However, since the Parliamentary Radar project does not systematically use milestones, we did not perform the linkage of software *issues* of the same milestones to it.

For the SPB portal project, the removal of *issues* with no links decreased from 800 to 127 the number of software *issues* in the graph. After the cut, about

Table 1. The top ten ranked *issues* of the SPB Portal.

Issue	*Rank*
Moderation of saved resource values of usage report	0.00593
Configure NGINX to serve syslog data	0.00345
Show error message close to the institution field on usage report	0.00345
Run Gitlab 8.5 with the built package	0.00345
Global search improvement	0.00345
White screen on community lateral block edition	0.00345
Portal wiki news import	0.00304
Broken user registration (username: boscojr)	0.00263
Add user e-mail on join community request processing screen	0.00263
Remove SISP question from new institution creation	0.00263

62.5% of them have only one link. Some *issues* appear with two and three links, and there are also three *issues* with four, five and six links each. Using the *issues* graph of Figure 1, we performed the PageRank algorithm. Table 1 shows the top ten ranked *issues*.

Table 2. The top ten ranked *issues* of the Radar Parlamentar.

Issue	*Rank*
Import legislative house	0.01242
Controversial polls	0.01242
Highlight party on the chart	0.01242
Duplicated parties and parliamentarians	0.01242
Advanced analysis	0.01242
Filtered polling list refactoring	0.01112
Automatic creation of dumps	0.01047
Solving *issues* 248, 241 e 250	0.01047
Without party voters on cdep in 1997	0.01047
Creation of Executive Chief Importer	0.01047

The removal of Parliamentary Radar project *issues* with no links decreased from 300 to 71 the number of software *issues* in the graph. After the cut, about 50% of them had just one link. Some *issues* appear with two, three and four links, and there is one *issue* with eleven links. Using the *issues* graph of Figure 2, we performed the PageRank algorithm. Table 2 shows the top-ranked *issues*.

The rank values of SPB project *issues* varied from 0.002 to 0.0172. The median is 0.0028 and the mean 0.0040 with a standard deviation of 0.0018. About half of the *issues* presented the same rank of 0.002. The histogram in Figure 3 represents the distribution of the generated ranking. For the Parliamentary Radar project *issues*, the ranks values varied in a range from 0.0046 to 0.0124. The median is 0.0059 and the mean 0.0063 with a standard deviation of 0.0020. Figure 4 presents the histogram plot for the Parliamentary Radar.

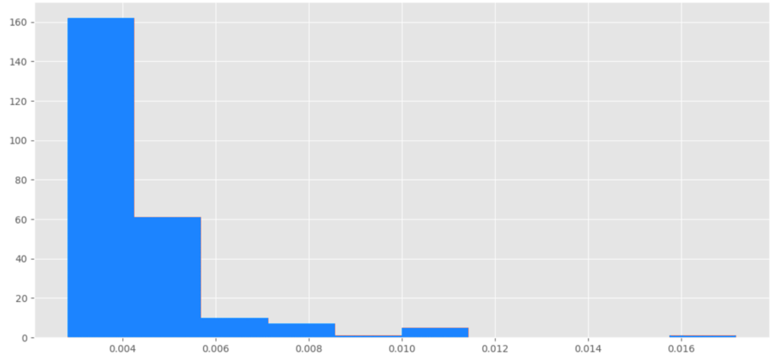

Fig. 3. Histogram of ranks for the SPB Portal *issues*.

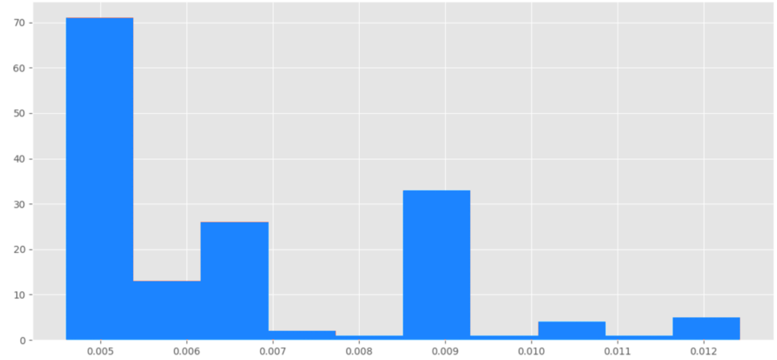

Fig. 4. Histogram of ranks for the Parliamentary Radar *issues*.

We compared the top-ranked software *issues* of the SPB portal project to the planning of the two last releases documented on the SPB wiki. We performed the comparison looking for the terms present in *issue* titles, descriptions, and wiki pages. In the SPB releases documentation, some priority features were "software usage report", "global search improvement", and "general improvements". These terms align with terms found in the top-ranked software *issues*. We also conducted an open interview with the SPB Portal coordinator presenting the ranked list of *issues* to him, who provided feedback that the ranked list showed the *issues* that consumed the most effort from the team at the project end. It suggests the feasibility of **our approach in capturing some of the relevant *issues* from the management point of view**.

The Parliamentary Radar project maintains its roadmap and backlog organized in the Wiki at its repository. Its backlog[11] describes features desired for the project. A set of software *issues* may represent different tasks to implement a feature. We compare the ranked list of software *issues* of Parliamentary Radar project and its roadmap to evaluate the result of our PageRank execution. In this way, it was possible to align the top-ranked *issues* with the features presented in the backlog to indicate if the proposed approach to determining the software *issue* relevance gives satisfactory results.

When comparing the Parliamentary Radar ranked software *issues* list with the project backlog, we observed that our solution identified software *issues* that were prioritized by developers. With this, we also conducted an open interview with three core developers of Parliamentary Radar and presented the ranked list of *issues* to them. We got the feedback that the highly relevant *issues* list presented the three *issues* that consumed the most effort from the team at that moment, which indicates that **our approach was able to identify highly relevant software *issues* from the developer point of view**.

Moreover, the Parliamentary Radar developers stated that if they had this suggestion when planning the next release, they could have prioritized other software *issues* included in the list instead of some that they preferred to prioritize. They also stated that the list of ranked software *issues* could also be useful during the team retrospective since it can help developers to identify the most discussed ones. The Parliamentary Radar developers also proposed that newcomers can use the list of ranked software *issues* to have an insight on what the software community is working on, creating a better way for their engagement in the software under development.

5 Threats to Validity

An internal validity threat occurs because the Parliamentary Radar roadmap had not been updated at the time we executed our scripts. So, the roadmap could not be related to software issues created after the last wiki update. However, the Parliamentary Radar wiki content had enough information to correlate software issues terms and planning documentation. Moreover, in the SPB Portal project, the wiki content was more reliable, so the sound results for both projects support the applicability of such comparison.

About external validity, we acknowledge that executing our approach with only two OSS projects imposes barriers to a broader generalization regarding our approach applicability. However, at least the positive observations of these two projects are encouraging and can motivate other communities to try out our proposal. Our approach is limited by how the developers organize the issue tracker of their projects. Since the algorithm depends on the links of the *issues*, it requires a cultural practice for the community to create these links.

In the evaluation process, the main reproducibility problem is the manual comparison made between terms found on top-ranked software issues and terms

[11] `https://github.com/radar-parlamentar/radar/wiki/Roadmap`

present in the planning documentation. An automated process for such comparison would be more suitable. On the other hand, the planning documentation for both SPB Portal and Parliamentary Radar projects are relatively small, so we hope someone trying to reproduce our study might get very similar results.

6 Conclusion

We presented the application of the PageRank algorithm to a network of software issues retrieved from OSS projects. Its use results in a ranking of issues, which can support a software community in decision-making activities, such as the planning of releases, retrospective studies, and helping newcomers to know the project better.

The presented results showed the feasibility of our approach for the Brazilian Public Software portal and Parliamentary Radar projects because some of the top-ranked software issues were also present in their planning documentation. Furthermore, in open interviews with the coordinators and the core developers of the projects, they found the software issues rank insightful in both management and development views. Therefore, we consider that the PageRank algorithm may be used to extract a small set of relevant issues from OSS repositories to support decision-making activities regarding retrospective studies, to support new developers to engage on activities that are currently being worked by the software community, and to track the effort of the team. However, before generalizing its use for all OSS projects, it is vital to understand peculiarities of the software community and project that could affect the results.

Answering the question *"Can the PageRank algorithm identify relevant issues on OSS projects to support decision-making activities?"*, the obtained results indicate that our approach can adequately work for those software projects using the Github or Gitlab software issue trackers and in which **the contributors create a substantial number of links among issues**. The issues should represent activities for the developers, the links between them should represent the relations between two or more activities, that way the algorithm can rank the issues helping the developers prioritize them. Given OSS projects following the specific structures presented in this study, we can answer our research question positively, since the algorithm was able to reveal a set of relevant issues regarding both the software projects. However, since we applied our approach to two OSS projects, more studies are necessary to strengthen this claim.

The adoption of our approach by OSS communities requires a way to integrate the execution and the presentation of results without the need to manually run scripts. To facilitate more studies based on our approach, we shared a package with the scripts used in this study on Gitlab, as described in Section 3. Future studies can evaluate the impact of our graph transformations, i.e., linking software issues within the same milestone and pruning those with no links. Finally, structured interviews with adopters of the approach would allow a better evaluation triangulation and broaden its applicability assessment for an extensive range of OSS projects.

References

1. Abdellatif, T.M., Capretz, L.F., Ho, D.: Software analytics to software practice: A systematic literature review. In: Proceedings of the First International Workshop on BIG Data Software Engineering. pp. 30–36. BIGDSE '15, IEEE (2015)
2. Baysal, O., Holmes, R., Godfrey, M.W.: Developer dashboards: The need for qualitative analytics. IEEE Software **30**(4), 46–52 (2013)
3. Booch, G., Brown, A.W.: Collaborative development environments. Advances in Computers, vol. 59, pp. 1 – 27. Elsevier (2003)
4. Borges, H., Hora, A., Valente, M.T.: Predicting the popularity of github repositories. In: Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering. pp. 9:1–9:10. PROMISE 2016, ACM (2016)
5. Brin, S., Page, L.: Reprint of: The anatomy of a large-scale hypertextual web search engine. Computer Networks **56**(18), 3825–3833 (2012)
6. Czerwonka, J., Nagappan, N., Schulte, W., Murphy, B.: CODEMINE: Building a software development data analytics platform at Microsoft. IEEE Software **30**(4), 64–71 (2013)
7. Doğrusöz, U., Madden, B., Madden, P.: Circular layout in the Graph Layout toolkit, pp. 92–100. Springer Berlin Heidelberg, Berlin, Heidelberg (1997)
8. Goyal, A., Sardana, N.: Efficient bug triage in issue tracking systems. In: Proceedings of the Doctoral Consortium at the 13th International Conference on Open Source Systems. pp. 15–24 (2017)
9. Hagberg, A.A., Schult, D.A., Swart, P.J.: Exploring network structure, dynamics, and function using NetworkX. In: Proceedings of the 7th Python in Science Conference. pp. 11–15. SciPy 2008 (2008)
10. He, J., Nazar, N., Zhang, J., Zhang, T., Ren, Z.: PRST: A pagerank-based summarization technique for summarizing bug reports with duplicates. International Journal of Software Engineering and Knowledge Engineering **27**(6), 869–896 (2017)
11. Hunter, J.D.: Matplotlib: A 2D graphics environment. Computing In Science & Engineering **9**(3), 90–95 (2007)
12. Meirelles, P., Wen, M., Terceiro, A., Siqueira, R., Kanashiro, L., Neri, H.: Brazilian Public Software Portal: An integrated platform for collaborative development. In: Proceedings of the 13th International Symposium on Open Collaboration. pp. 16:1–16:10. OpenSym '17, ACM (2017)
13. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab (1999)
14. Robles, G., González-Barahona, J.M., Cervigón, C., Capiluppi, A., Izquierdo-Cortázar, D.: Estimating development effort in free/open source software projects by mining software repositories: A case study of OpenStack. In: Proceedings of the 11th Working Conference on Mining Software Repositories. pp. 222–231. MSR 2014, ACM (2014)
15. Schwaber, K., Sutherland, J.: Sprint retrospective. In: The Definitive Guide to Scrum: The Rules of the Game. Scrum.Org and ScrumInc (2016), `www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf`
16. Steinmacher, I., Chaves, A.P., Conte, T.U., Gerosa, M.A.: Preliminary empirical identification of barriers faced by newcomers to open source software projects. In: 28th Brazilian Symposium on Software Engineering. pp. 51–60. SBES 2014, IEEE (2014)