

IFIP AICT 556



Francis Bordeleau
Alberto Sillitti
Paulo Meirelles
Valentina Lenarduzzi
(Eds.)

Open Source Systems

15th IFIP WG 2.13 International Conference, OSS 2019
Montreal, QC, Canada, May 26–27, 2019
Proceedings



Springer

Editor-in-Chief

Kai Rannenberg, Goethe University Frankfurt, Germany

Editorial Board Members

TC 1 – Foundations of Computer Science

Jacques Sakarovitch, Télécom ParisTech, France

TC 2 – Software: Theory and Practice

Michael Goedicke, University of Duisburg-Essen, Germany

TC 3 – Education

Arthur Tatnall, Victoria University, Melbourne, Australia

TC 5 – Information Technology Applications

Erich J. Neuhold, University of Vienna, Austria

TC 6 – Communication Systems

Aiko Pras, University of Twente, Enschede, The Netherlands

TC 7 – System Modeling and Optimization

Fredi Tröltzsch, TU Berlin, Germany

TC 8 – Information Systems

Jan Pries-Heje, Roskilde University, Denmark

TC 9 – ICT and Society

David Kreps, University of Salford, Greater Manchester, UK

TC 10 – Computer Systems Technology

Ricardo Reis, Federal University of Rio Grande do Sul, Porto Alegre, Brazil

TC 11 – Security and Privacy Protection in Information Processing Systems

Steven Furnell, Plymouth University, UK

TC 12 – Artificial Intelligence

Ulrich Furbach, University of Koblenz-Landau, Germany

TC 13 – Human-Computer Interaction

Marco Winckler, University of Nice Sophia Antipolis, France

TC 14 – Entertainment Computing

Rainer Malaka, University of Bremen, Germany

IFIP – The International Federation for Information Processing

IFIP was founded in 1960 under the auspices of UNESCO, following the first World Computer Congress held in Paris the previous year. A federation for societies working in information processing, IFIP's aim is two-fold: to support information processing in the countries of its members and to encourage technology transfer to developing nations. As its mission statement clearly states:

IFIP is the global non-profit federation of societies of ICT professionals that aims at achieving a worldwide professional and socially responsible development and application of information and communication technologies.

IFIP is a non-profit-making organization, run almost solely by 2500 volunteers. It operates through a number of technical committees and working groups, which organize events and publications. IFIP's events range from large international open conferences to working conferences and local seminars.

The flagship event is the IFIP World Computer Congress, at which both invited and contributed papers are presented. Contributed papers are rigorously refereed and the rejection rate is high.

As with the Congress, participation in the open conferences is open to all and papers may be invited or submitted. Again, submitted papers are stringently refereed.

The working conferences are structured differently. They are usually run by a working group and attendance is generally smaller and occasionally by invitation only. Their purpose is to create an atmosphere conducive to innovation and development. Refereeing is also rigorous and papers are subjected to extensive group discussion.

Publications arising from IFIP events vary. The papers presented at the IFIP World Computer Congress and at open conferences are published as conference proceedings, while the results of the working conferences are often published as collections of selected and edited papers.

IFIP distinguishes three types of institutional membership: Country Representative Members, Members at Large, and Associate Members. The type of organization that can apply for membership is a wide variety and includes national or international societies of individual computer scientists/ICT professionals, associations or federations of such societies, government institutions/government related organizations, national or international research institutes or consortia, universities, academies of sciences, companies, national or international associations or federations of companies.

More information about this series at <http://www.springer.com/series/6102>

Francis Bordeleau · Alberto Sillitti ·
Paulo Meirelles · Valentina Lenarduzzi (Eds.)

Open Source Systems


15th IFIP WG 2.13 International Conference, OSS 2019
Montreal, QC, Canada, May 26–27, 2019
Proceedings

Editors

Francis Bordeleau
École de Technologie Supérieure (ÉTS)
Montreal, QC, Canada

Paulo Meirelles
Federal University of São Paulo
São Paulo, Brazil

Alberto Sillitti
Innopolis University
Innopolis, Russia

Valentina Lenarduzzi 
Tampere University
Tampere, Finland

ISSN 1868-4238 ISSN 1868-422X (electronic)
IFIP Advances in Information and Communication Technology
ISBN 978-3-030-20882-0 ISBN 978-3-030-20883-7 (eBook)
<https://doi.org/10.1007/978-3-030-20883-7>

© IFIP International Federation for Information Processing 2019

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

Welcome to the 15th edition of the International Conference on Open Source Systems (OSS 2019) held in Montreal, Quebec, Canada, during May 26–27, 2019. The conference has been a reference point for the research community in Open Source Systems for 15 years, promoting research activities in the area and anticipating trends.

All of the submitted research papers went through a rigorous peer-review process. Each paper was reviewed by at least three members of the Program Committee. Of the 30 papers submitted, only ten were accepted as full papers (33%). We also accepted five experience reports, with a committee of experts evaluating each submission for new experiences that would be both interesting and beneficial to our community. We accepted papers dealing with several different aspects of OSS including: Mining OSS Data, Organizational Aspects of FLOSS Projects, FLOSS Adoption, FLOSS Cost and Licenses, FLOSS Education and Training.

We hope that you will find the OSS 2019 proceedings useful for your professional and academic activities.

Finally, we would like to thank all the people who contributed to OSS 2019 including the authors, the sponsors, the reviewers, the volunteers, and the chairs.

April 2019

Francis Bordeleau
Alberto Sillitti
Paulo Meirelles
Valentina Lenarduzzi

Organization

General Chair

Francis Bordeleau École de technologie supérieure (ÉTS), Canada

Program Chairs

Paulo Meirelles Federal University of São Paulo, Brazil
Alberto Sillitti Innopolis University, Russia

Program Committee

Alexandre Bergel	University of Chile, Chile
Abby Cabunoc Mayes	Mozilla, Canada
Andrea Capiluppi	Brunel University, UK
Kevin Crowston	Syracuse University, USA
Matt Germonprez	University of Nebraska, USA
Jesus M. Gonzalez-Barahona	Universidad Rey Juan Carlos, Spain
Imed Hammouda	Chalmers and University of Gothenburg, Sweden
Daniel S. Katz	University of Illinois at Urbana-Champaign, USA
Fabio Kon	University of São Paulo, Brazil
Akinori Ihara	Wakayama University, Japan
Luigi Lavazza	University of Insubria, Italy
Valentina Lenarduzzi	Tampere University, Finland
Manuel Mazzara	Innopolis University, Russia
Sandro Morasca	University of Insubria, Italy
Masood Mortazavi	Huawei, USA
John Noll	University of East London, and Lero, Ireland
Sarah Novotny	Google, USA
Peter Rigby	Concordia University, Canada
Chris Riley	Mozilla, USA
Gregorio Robles	Universidad Rey Juan Carlos, Spain
Walter Scacchi	University of California, USA
Alolita Sharma	Amazon Web Services, USA
Diomidis Spinellis	Athens University of Economics and Business, Greece
Kate Stewart	Linux Foundation, USA
Davide Taibi	Tampere University, Finland
Jens Weber	University of Victoria, Canada

Contents

Mining OSS Data

Building an Open-Source Cross-Cloud DevOps Stack for a CRM Enterprise Application: A Case Study	3
<i>Sebastian Schork, Feroz Zahid, Dipesh Pradhan, Sébastien Kicin, and Antonia Schwichtenberg</i>	

Open Source Vulnerability Notification	12
<i>Brandon Carlson, Kevin Leach, Darko Marinov, Meiyappan Nagappan, and Atul Prakash</i>	

Organizational Aspects of FLOSS Projects

EJ: A Free Software Platform for Social Participation	27
<i>Fábio Macêdo Mendes, Ricardo Poppi, Henrique Parra, and Bruna Moreira</i>	

Introducing Agile Product Owners in a FLOSS Project	38
<i>Matthias Müller, Christian Schindler, and Wolfgang Slany</i>	

What Are the Perception Gaps Between FLOSS Developers and SE Researchers? A Case of Bug Finding Research	44
<i>Yutaro Kashiwa, Akinori Ihara, and Masao Ohira</i>	

FLOSS Adoption

Fifteen Years of Open Source Software Evolution	61
<i>Francis Bordeleau, Paulo Meirelles, and Alberto Sillitti</i>	

Open Source Software Community Inclusion Initiatives to Support Women Participation	68
<i>Vandana Singh and William Brandon</i>	

Predicting Popularity of Open Source Projects Using Recurrent Neural Networks	80
<i>Sefa Eren Sahin, Kubilay Karpaz, and Ayse Tosun</i>	

What Attracts Newcomers to Onboard on OSS Projects? TL;DR: Popularity	91
<i>Felipe Fronchetti, Igor Wiese, Gustavo Pinto, and Igor Steinmacher</i>	

Why Do Developers Adopt Open Source Software?
Past, Present and Future. 104
*Valentina Lenarduzzi, Davide Tosi, Luigi Lavazza,
and Sandro Morasca*

Why Do People Give Up FLOSSing? A Study of Contributor
Disengagement in Open Source 116
*Courtney Miller, David Gray Widder, Christian Kästner,
and Bogdan Vasilescu*

FLOSS Cost and Licences

Open Source for Open Source License Compliance 133
Oliver Fendt and Michael C. Jaeger

Opportunity Costs in Free Open-Source Software 139
Siim Karus

FLOSS Education and Training

Does FLOSS in Software Engineering Education Narrow
the Theory-Practice Gap? A Study Grounded on Students' Perception 153
*Deborá Maria Coelho Nascimento, Christina von Flach Garcia Chavez,
and Roberto Almeida Bittencourt*

Faculty Development for FLOSS Education 165
Becka Morgan, Gregory W. Hislop, and Heidi J. C. Ellis

Author Index 173

Mining OSS Data



Building an Open-Source Cross-Cloud DevOps Stack for a CRM Enterprise Application: A Case Study

Sebastian Schork¹(✉), Feroz Zahid², Dipesh Pradhan², Sébastien Kicin¹,
and Antonia Schwichtenberg¹

¹ CAS Software AG, Karlsruhe, Germany

{sebastian.schork,sebastien.kicin,antonia.schwichtenberg}@cas.de

² Simula Research Laboratory, Fornebu, Norway

{feroz,dipesh}@simula.no

Abstract. Open Source software solutions play a critical role for the SMEs by enabling easy access to reusable software. Also, with the rapid growth in the popularity of the cloud technologies, computational demands of SMEs are cost-efficiently met by the public clouds as users can dynamically acquire resources on demand according to their needs. However, non-standardized cloud interfaces, lack of inter-cloud transparency, and complex cost models, often result in *vendor lock-in*. Once in vendor lock-in, cloud users have to live with a single cloud provider and accept whatever pricing schemes and SLAs are imposed. Moreover, new regulations covered by the General Data Protection Regulation (GDPR) in Europe require companies to enforce policies regarding secure storage of data in the cloud, as well as restrict moving confidential datasets outside Europe. This situation requires a more transparent use of cloud resources from multiple cloud providers, that conform with user's data privacy needs, service requirements, and budget.

In this paper, we discuss challenges and pitfalls of designing a Cross-Cloud DevOps stack for an *app*-based extension platform of a Customer Relationship Management (CRM) system. The fully-automated DevOps stack, based on open source software tools and technologies, has been developed in close coordination with an open source integration project, *Melodic*. With the help of our DevOps stack, third-party apps in our CRM software are now Multi-Cloud ready, and the data storage in the cloud by the users conforms to potential GDPR requirements. In addition, the deployment time of apps has been reduced to minutes, while the platform is able to scale up and scale down apps efficiently based on the current workload requirements, saving substantial cloud costs.

Keywords: Open source · Cross-Cloud · DevOps

This work has received funding from the European Union's H2020 research and innovation programme under grant agreement no. 731664 (MELODIC).

© IFIP International Federation for Information Processing 2019

Published by Springer Nature Switzerland AG 2019

F. Bordeleau et al. (Eds.): OSS 2019, IFIP AICT 556, pp. 3–11, 2019.

https://doi.org/10.1007/978-3-030-20883-7_1

1 Introduction

The major growth of the ICT industry over the last decade can be attributed towards enabling technologies based on open standards and protocols, making it possible to develop software solutions that can be delivered and integrated on any infrastructure, independent of the vendor. However, with the emergence of the cloud computing paradigm, a step is taken backward [1]. Cloud users are often forced into *vendor lock-in* due to the use of incompatible protocols and standards by the cloud service providers (CSPs). Vendor lock-in or propriety lock-in is an economic condition in which a customer is made dependent on the vendor-specific technology, products, or services by making it fairly difficult and costly to migrate to a competition [2].

The cloud-based customer relationship (CRM) software, *SmartWe*, is designed by CAS Software¹ to support users in an optimal way depending on their role, the business sector they are working in and the respective workflows. With the help of SmartWe’s software development kit (SDK) and UI tools, users and CAS development partners can adapt existing apps or develop new apps for the SmartWe platform. The apps can be offered and installed via an App Store. Initially, the SmartWe supports apps to be deployed on a single cloud platform. As we moved on with the development, we found that it was not easy to migrate third-party apps to another cloud platform because SmartWe and its deployment scripts became heavily dependent on the specific CSP’s APIs and management tools. As the new General Data Protection Regulation (GDPR) started being enforced in Europe, the user apps were obliged to conform to the data storage regulations laid down. This was not an issue in the first place for the product SmartWe due to its deployment in secure partner data centers but needs to be considered for the third-party apps that are developed by partners and users. For instance, some datasets used by extension apps, which are confidential to the users according to their data privacy needs, may not be allowed to be migrated to locations outside Europe. Furthermore, the requirements of the apps were dynamic and often updated, requiring manual updates to several scripts and resulting in slower cloud deployments for the third-party apps.

In general, no single CSP is able to provide all the features a user may need in a cost-efficient way, while satisfying the user’s security and performance requirements. As mentioned above, even the most dominant CSPs have limited geographical presence. If local legislation requires confidential data to be stored within a country’s geographical boundaries, the cloud user will need to rely on cloud providers owning infrastructure locally – or maybe even use a private cloud. Still, the same cloud user would ideally want to take advantage of cheap global cloud offerings for computation and storage when the strict data security requirements do not apply. With these challenges in place, we joined forces together with several other academic and industrial partners to tackle the need of an automated Cross-Cloud DevOps solution under the umbrella of an

¹ CAS Software AG - <https://www.cas.de/en/homepage.html>.

open source research and innovation project, Melodic². In this paper, we discuss challenges and pitfalls of designing such a Cross-Cloud DevOps stack for SmartWe based on open source tools and technologies. With the help of our DevOps stack, third-party apps management in SmartWe is now fully Multi-Cloud ready bringing our users out of the potential vendor lock-in. In addition, the data storage related to the third-party apps in the cloud now conforms to the GDPR requirements as users can specify their requirements and constraints through an innovate modelling interface. Furthermore, the deployment time for SmartWe and apps has been reduced to minutes, while the platform is able to scale up and down efficiently based on the current workload requirements.

The rest of this paper is structured as follows. In Sect. 2, we provide details about our SmartWe CRM software. In Sect. 3, we define the requirements for our automated Cross-Cloud DevOps system, as well as present the process of selecting and integrating the available Open Source Software (OSS), together with outlining the new development under Melodic. The resultant DevOps stack is presented and evaluated in, Sects. 4 and 5, respectively. We conclude in Sect. 6.

2 The SmartWe CRM

The cloud-based CRM software, SmartWe, is designed to support customers in their daily work by providing a tailored software tool with respect to the particular role of each user. Tailored business solutions support work flows according to the needs of the users and are much more efficient and usable than generic software systems [3]. SmartWe supports different ways to tailor the basic *anything* relationship management (xRM) solution to user role specific tasks. For instance, the user interface can be personalized by adding only a subset of the available apps. Moreover, using the provided SDK, an existing app can be tailored and extended, or a new apps can be developed for fulfilling user-specific requirements. The idea of an app-based xRM cloud software that can be adapted and extended to diverse use-cases and customer needs is highly promising from a marketing and business perspective. However, it is also challenging from the conceptual point of view, and even more so from the developer perspective.

The DevOps stack we present in this paper comes into play both for the transparent deployments of the new or customized apps (such as compute- and data-intensive extensions), as well as to make sure that the third-party apps do not affect the SmartWe system's performance and availability. For example, a CAS partner may develop a new app for either integrating and analyzing existing data from a third-party system or for analytics based on existing data from the primary CRM system, requiring secure data management and on-demand resource availability in the cloud. In these cases, there is a need for a dynamic, customized, and scalable deployment solution for the apps. The aforementioned two cases directly refer to SmartWe's two main pillars:

² The Melodic Cloud Project - <https://melodic.cloud/>.

- **Third-party apps that extend the SmartWe solution:** Both the data storage and the apps deployments in the cloud need to conform to the specified user-requirements related to the privacy and confidentiality.
- **Scalable SmartWe base deployment:** The dynamic increase in load to the SmartWe base deployment, because of the data-intensive or compute-intensive third-party apps, needs to be tackled automatically.

Both cases and their evaluation with the proposed DevOps stack is described in detail in Sect. 5. The deployment of SmartWe with and without our DevOps stack, based on an example two application instances with a load balancer and an external third-party app, is shown in Fig. 1.

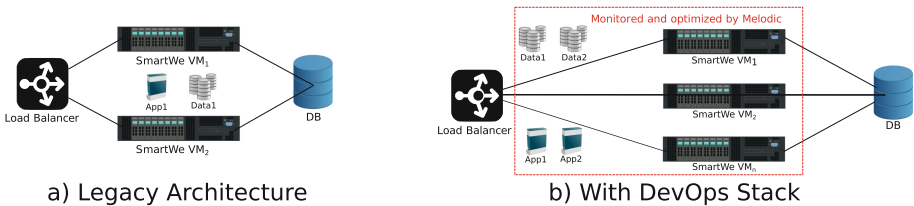


Fig. 1. SmartWe deployment without and with the DevOps stack based on Melodic

3 OSS Selection and Integration

In order to design a Cross-Cloud DevOps stack for the third-party apps in SmartWe, we first lay down a set of requirements. Next, in the context of our requirements, we surveyed the available OSS, and investigated related integration, compatibility, and licensing issues. Following requirements are specified. **Transparent deployment and execution** is necessary to enable automated app deployment across multiple CSPs. In addition, as both cloud by definition are unpredictable [4], as well as load on SmartWe platform from third-party apps is dynamic, we need a mechanism for **runtime adaptation** for SmartWe and deployed apps. Furthermore, as discussed in Sect. 1, with the GDPR requirements in place, our DevOps stacks needs to support defining data and component placement requirements and restrictions to cater for the user-specific need of **data privacy and confidentiality**.

3.1 Related Work

The challenges we face for our DevOps stack fall broadly into three areas: Cross-Cloud application deployments, resource management, and modelling/optimization of data-aware applications on heterogeneous infrastructures. In general, cloud federation [5] enables end users to integrate segregated resources from different cloud systems. Popular open-source cloud orchestration solutions,

like OpenStack [6], provide mechanisms to complement private cloud infrastructure with dynamically acquired resources from public clouds. Nevertheless, resource management is not well integrated with state-of-the-art federated cloud solutions. Further, none of the current cloud orchestration platforms supports context-awareness needed to optimize application deployments in Cross-Cloud environments, as needed by the SmartWe platform. Furthermore, Cross-Cloud application deployments are subjected to various resource abstraction models offered by different CSPs and a unified approach needed for interoperability is lacking [7, 8].

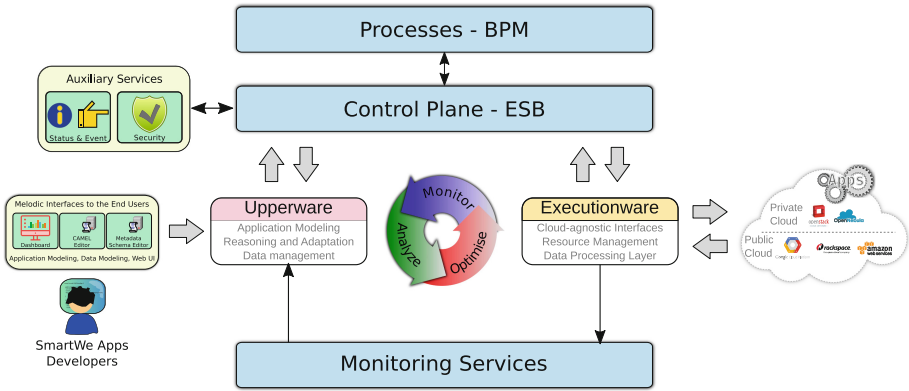


Fig. 2. Overview of the DevOps stack based on Melodic

Recent efforts, such as those in the PaaSage project [9], have targeted model-based approaches for the design, development, deployment, and self-adaptation of Cross-Cloud applications. In particular, cloud modelling frameworks, such as CloudMF [10], are in active development, to equip application developers with capabilities to define a rich set of design-time and runtime attributes like application requirements, Quality-of-Service (QoS) constraints, and security considerations for Cross-Cloud deployments. However, a large number of challenges still remain unaddressed. In particular, support of data-aware deployments in Cross-Cloud environments is still very restricted. Recently, various cluster management solutions have also gain popularity. But most of these solutions work only on statically available cluster and cloud resources, such as Mesos [11], Kubernetes [12] and Docker/Swarm [13]. Some Multi-Cloud deployment solutions, such as CYCLONE [14], are available but lack sufficient advanced reasoning support, as needed for the SmartWe app platform. DC/OS [15] integrates a range of software, like Mesos and Marathon, to provide an integrated platform for running applications and data services on heterogeneous platforms. However, DC/OS lacks native support for Cross-Cloud deployments and adaptation of applications. Moreover, advanced capabilities for reasoning for efficient resource management, according to the user-defined requirements and constraints, are

also missing. Furthermore, many advanced DC/OS features are only available in their closed-source enterprise version.

3.2 Available OSS and Integration

With the related work survey described in Sect. 3.1, it is quite evident that we need to both integrate the available OSS as well as develop additional capabilities to fulfil the needs of our DevOps stacks for the SmartWe app platform. In the Melodic project, we selected PaaSage OSS as the base Multi-Cloud platform for our stack development. Advanced capabilities related to the data-awareness are implemented as part of the new development in the Melodic OSS on top of the PaaSage code-base.

3.3 Licensing Compatibility

License compatibility is a crucial issue for an OSS integration project built upon existing software. The new components created in the scope of the Melodic itself are released under Mozilla Public License (MPL) v2. The choice of this particular license has resulted from deliberation over its compatibility with the GNU GPL and the licenses used by the Apache Software Foundation. MPL is a *weak copyleft* license designed to address the needs of both proprietary and open source developers [16, 17].

4 A Cross-Cloud DevOps Stack

An overview of the our DevOps stack architecture is given in Fig. 2. As shown in the figure, the DevOps stack is conceptually divided into three main component groups, the Melodic interfaces to the end users, the *Upperware*, and the *Executionware*. The Melodic interfaces to the end users include tools and interfaces used by the Melodic users to model their applications and datasets and interact with the Melodic platform. These interfaces are exposed to the SmartWe app developers using a modelling language called CAMEL [18]. Applications and data models created in CAMEL are given as input to the Melodic Upperware. The job of the Upperware is to calculate the optimal data placements and application deployments on dynamically acquired Cross-Cloud resources in accordance with the specified application and data models in CAMEL as well as in consideration of the current cloud performance, workload situation, and costs. The actual cloud deployments for the SmartWe apps are carried out through the Executionware. The Executionware is capable of managing and orchestrating diverse cloud resources, and it also enables support of Cross-Cloud monitoring of both deployed apps and the SmartWe base platform.

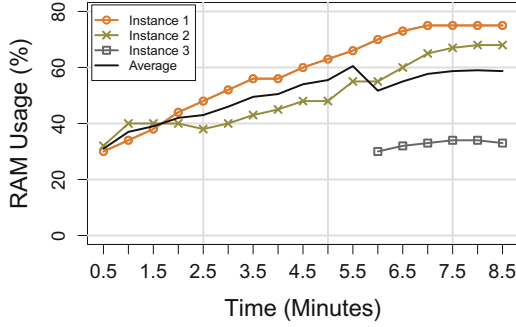


Fig. 3. Autonomic optimization of RAM by Melodic-enabled SmartWe platform

5 Evaluation

We evaluate the DevOps stack designed for apps in SmartWe platform with two different perspectives. First, we assess the usefulness, quality, and maintainability of the open source software developed, in relation to Melodic, to gauge its suitability for the long-term use by the SmartWe partners. Second, we evaluate the SmartWe platform and app scalability features offered through autonomic adaptation and optimization capabilities of the Melodic middleware platform.

CAS found that the initiative of an open source Cross-Cloud DevOps stack is very useful for many SMEs. Besides addressing vendor lock-in, the DevOps stack we developed also enables SMEs to adhere to the security and privacy requirements related to the data storage and processing in the cloud. The platform quickly became a catalyst to federate other companies dealing with similar business scenarios and thus building a developer community around it, bound by a common drive to support and improve open solutions for the cloud development and deployments. The first stages of the Melodic development quickly demonstrated that integrating existing software, with the help of an open source community, quickly realizes the software which requires a large amount of development time otherwise. Moreover, the quality of the software produced is satisfactory promising long-term maintainability. Finally, although Melodic should not be thought just as a *free software*, the fact that the providers require no licensing fees remains a decisive advantage when looking at the potential total cost of deploying the solution across the IT infrastructures.

From the technical perspective, SmartWe platform benefits from various features Melodic offers. Our DevOps stack enables dynamic adaptations to the deployed SmartWe platforms and apps. In the case of SmartWe platform, RAM usage is a critical metric for our live deployments. Increasing numbers of user sessions as well as app’s computational complexity leads to higher RAM usage and therefore represent a bottleneck. Based on sensor values and scalability rules, the mechanism autonomously decides on how to best optimize the current deployment. Figure 3 depicts an initial deployment of two instances of the SmartWe system being handled by a central load balancer. A scalability rule was written

requiring the average RAM load to be lower than 60% of the total available RAM. The moment when this limit is succeeded, a third application instance is added automatically by the Melodic, and the average RAM load stabilizes, as shown in the figure. A point to note here, however, is that the optimization offered by Melodic uses the concept of *utility function*. Each potential deployment solution determined by Melodic is evaluated regarding its utility before a final selection is made. Utility functions are application and deployment specific and were carefully designed for the SmartWe platform to meet our requirements.

6 Conclusion

In this paper, we discussed the need of an open source Cross-Cloud DevOps stack for our SmartWe CRM solution. With the help of an open source integration project, Melodic, our automated DevOps stack has enabled third-part apps, installable in SmartWe, to counter vendor lock-in. In addition, the user apps in SmartWe are now able to transparently take advantage of distinct characteristics of available private and public clouds, dynamically optimize resource utilization, and conform to the user's privacy needs and service requirements.

References

1. McKendrick, J.: Cloud computing's vendor lock-in problem: why the industry is taking a step backward. *Forbes*, November 2011
2. Opara-Martins, J., Sahandi, R., Tian, F.: Critical review of vendor lock-in and its impact on adoption of cloud computing (2014)
3. Weinhardt, C., Anandasivam, A., Blau, B., Stößer, J.: Business models in the service world. *IT Prof.* **2**, 28–33 (2009)
4. Schad, J., Dittrich, J., Quiané-Ruiz, J.A.: Runtime measurements in the cloud: observing, analyzing, and reducing variance. *Proc. VLDB Endow.* **3**(1–2), 460–471 (2010)
5. Kurze, T., Klems, M., Bermbach, D., Lenk, A., Tai, S., Kunze, M.: Cloud federation. *Cloud Comput.* **2011**, 32–38 (2011)
6. Sefraoui, O., Aissaoui, M., Eleuldj, M.: OpenStack: toward an open-source solution for cloud computing. *Int. J. Comput. Appl.* **55**(3), 38–42 (2012)
7. Petcu, D.: Portability and interoperability between clouds: challenges and case study. In: Abramowicz, W., Llorente, I.M., Surr ridge, M., Zisman, A., Vayssière, J. (eds.) *ServiceWave 2011*. LNCS, vol. 6994, pp. 62–74. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24755-2_6
8. Taherkordi, A., Zahid, F., Verginadis, Y., Horn, G.: Future cloud systems design: challenges and research directions. *IEEE Access* **6**, 74120–74150 (2018)
9. Bubak, M., Bališ, B., Kitowski, J., Król, D., Kryza, B., Malawski, M.: *PaaSage: model-based cloud platform upperware* (2011)
10. Ferry, N., Song, H., Rossini, A., Chauvel, F., Solberg, A.: CloudMF: applying MDE to tame the complexity of managing multi-cloud applications. In: *IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC)*, pp. 269–277. IEEE (2014)

11. Hindman, B., et al.: Mesos: a platform for fine-grained resource sharing in the data center. In: NSDI, vol. 11, p. 22 (2011)
12. Mcluckie, C.: Containers, VMs, Kubernetes and VMware. <https://cloudplatform.googleblog.com/2014/08/containers-vms-kubernetes-and-vmware.html>. Accessed 13 Jan 2019
13. Swarm: a Docker-native clustering system. <https://github.com/docker/swarm/>. Accessed 13 Jan 2019
14. Slawik, M., et al.: CYCLONE unified deployment and management of federated, multi-cloud applications. In: 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC), pp. 453–457. IEEE (2015)
15. What is DC/OS? <https://docs.mesosphere.com/1.7/overview/what-is-dcos/>. Accessed 13 Jan 2019
16. Rosen, L.: Which Open Source license should I use for my software. Open Source Initiative (2001)
17. Rosen, L.: Open Source Licensing: Software Freedom and Intellectual Property Law. Prentice Hall PTR, Upper Saddle River (2004)
18. Rossini, A.: Cloud application modelling and execution language (CAMEL) and the PaaS workflow. In: Advances in Service-Oriented and Cloud Computing-Workshops of ESOC, vol. 567, pp. 437–439 (2015)



Open Source Vulnerability Notification

Brandon Carlson¹, Kevin Leach²(✉), Darko Marinov¹, Meiyappan Nagappan³,
and Atul Prakash²

¹ University of Illinois at Urbana-Champaign, Urbana, USA
{blcr1sn2,marinov}@illinois.edu

² University of Michigan, Ann Arbor, USA
{kgleach,aprakash}@umich.edu

³ University of Waterloo, Waterloo, Canada
mei.nagappan@uwaterloo.ca

Abstract. The use of third-party libraries to manage software complexity can expose open source software projects to vulnerabilities. However, project owners do not currently have a standard way to enable private disclosure of potential security vulnerabilities. This neglect may be caused in part by having no template to follow for disclosing such vulnerabilities. We analyzed 600 GitHub projects to determine how many projects contained a vulnerable dependency and whether the projects had a process in place to privately communicate security issues. We found that 385 out of 600 open source Java projects contained at least one vulnerable dependency, and only 13 of those 385 projects had a security vulnerability reporting process. That is, 96.6% of the projects with a vulnerability did *not* have a security notification process in place to allow for private disclosure. In determining whether the projects even had contact information publicly available, we found that 19.8% had no contact information publicly available, let alone a security vulnerability reporting process. We suggest two methods to allow for community members to privately disclose potential security vulnerabilities.

Keywords: Vulnerable dependency · Security disclosure · Open source

1 Introduction

Open source project maintainers often ignore or overlook important preventative maintenance tasks, including security scans for vulnerabilities in libraries [16], even with automated upgrades of libraries [19]. Neglecting such scanning tasks can impact both the end user and the software maintainer. “Using Components with Known Vulnerabilities” [23] was listed among the top ten application security risks in 2017.

Bug bounty programs can incentivize security disclosures [15] but have not taken hold as a standard policy. The National Institute of Standards and Technology is constructing a new process for receiving, analyzing, and responding

to vulnerability disclosures [25], providing at least a guideline for open source projects to establish their own policies. Such vulnerability disclosure policies are growing in importance due to the continued increase of information breaches [2]. For example, Equifax recently fell victim to a high-profile breach, resulting from an unpatched open source dependency (Apache Struts) [24]. In this paper, we suggest approaches to address this shortcoming in the open source community.

This paper makes the following contributions:

- An empirical study on how many open source projects are using vulnerable dependencies, contain security policies, and ways for individuals to contact the open source project.
- A quantitative analysis of the findings from the data collection process using open source Java projects from GitHub.
- Recommendations for improving security of open source projects through improved communication among the open source community, security researchers, and the open source repository providers.

2 Motivating Experience

We were originally motivated by the Equifax breach [24], which was related to a vulnerable dependency (Apache Struts). GitHub already provides a dependency scanning tool that will alert project owners if vulnerabilities are discovered in those dependencies [7]. However, project owners are not required to address such reports. Indeed, project owners could decide the vulnerability is unexploitable through their project or that it is not worth the effort to address. In such a scenario, project owners that ignore reports from dependency scanning tools pose a risk to the community at large. Thus, we sought to investigate the prevalence of projects requiring vulnerable versions of Apache Struts.

We used Snyk [26], a dependency scanning tool, to analyze various open source projects on GitHub for vulnerable dependencies. Indeed, we found many projects depended on a vulnerable version of Apache Struts. We visited each such repository to determine if a bug bounty program or disclosure process was documented. In the majority of cases, projects contained no guidelines for reporting vulnerabilities. We attempted to communicate this vulnerable dependency through a combination of emails to project owners, opening issues, and submitting pull requests.

Project owners offered myriad responses. While some did not respond and some thanked us for the report, one owner requested private communications over public pull requests or open issues. Unfortunately, attempting personal contact for other projects resulted in our GitHub account being flagged for spam. These diverse and sometimes discouraging responses suggest the open source community could benefit from some standardized practice among open source projects for reporting such vulnerabilities without facing retribution. Motivated by this need, we chose to systematically analyze a corpus of open source projects for vulnerable dependencies and corresponding policies for reporting them.

3 Project Selection and Data Collection

To learn whether open source projects contain both easy to access contact information and a security vulnerability reporting policy we selected open source Java projects from GitHub using seven different sources: (1) Libraries.io’s public dataset [21]; (2) Legunsen et al. [17]; (3) Munaiah et al. [20]; (4) GitHub trending Java projects [11]; (5) GitHub and Government list of government-sponsored open source projects [9]; (6) GitHub and Government list of government-funded, research-related open source projects [10]; and (7) GitHub and Government list of civic hackers open source projects [8]. These sources entail a combination of public repositories (1 and 4), previously published work (2 and 3), and government-funded and whitehat hacker projects (5, 6, and 7). We focused on Java projects because it is a widely used language [12] amenable to our software scanning infrastructure [26]. Table 1 summarizes the statistics of the projects from each of these sources.

Table 1. Projects selected by resource

Resource	Projects	Stars	Forks	Issues	Commits	Contributors
Libraries.io [21]	209	141	57	12.6	608.6	12.6
Legunsen et al. [17]	126	244	103	14.5	333.1	11.8
Munaiah et al. [20]	83	2195	685	59.1	545.2	16.6
GitHub trending Java projects [11]	71	2400	923	52.1	1537.5	27.0
Government-sponsored [9]	68	8	7	3.0	648.1	8.1
Government-funded research [10]	22	8	5	2.9	152.0	3.4
Government civic hackers [8]	21	3	3	2.6	2536.4	6.4
Total projects	600					

4 Evaluation

We seek to answer the following research questions (RQs):

- *RQ1: How prevalent are vulnerable dependencies among our projects?*
- *RQ2: How common are security notification policies in open source projects?*
- *RQ3: How available is contact information for open source projects?*

4.1 RQ1: Prevalence of Known Vulnerable Dependencies

Finding Vulnerable Dependency Libraries. We used Snyk [26] to scan each of our 600 Java projects. Snyk maintains a database of libraries and corresponding vulnerabilities for each version of each library. These vulnerabilities are associated with a Common Vulnerabilities and Exposures (CVE) ID or a

Common Weakness Enumeration (CWE) ID. We say a given project contains a *direct vulnerability* if it uses a vulnerable library as reported by Snyk. Similarly, we say that a project has a *transitive vulnerability* if a project’s dependency itself uses a subsequent dependency with a vulnerability as reported by Snyk. We also used the GitHub API to record statistics (e.g., stars, open issues) for each project.

Results. In the 600 open source projects we examined, we found an average of 7.8 direct or transitive vulnerabilities per project. Further, we used the severity scale based on the severity rating provided by Snyk as part of the scanning process (derived from CVE and CWE reports). The average number of high severity vulnerabilities was 4.1 per project, compared to medium severity level vulnerabilities which averaged only 3.6 per project, and low severity which averaged 0.1 vulnerabilities per project.

We further categorize these open source projects containing vulnerabilities based on the severity level of vulnerabilities. A single project may contain multiple vulnerabilities that range from low to high severity. Overall, we obtained the following numbers of projects:

- 266 projects used at least one dependency with a high severity vulnerability;
- 202 projects used at least one dependency with a medium severity;
- 39 projects used at least one dependency with a low severity;
- 215 had no known vulnerability.

Note that some of these sets overlap because some projects contained multiple vulnerabilities with different severity levels.

Overall, 64.2% (385 of 600) of the projects we examined used at least one vulnerable dependency. This result is consistent with a recent report that Java applications are likely to include at least one vulnerable library [29]. Although a vulnerable library does not necessarily mean that a project using that library can be exploited, it does represent a risk to the community overall.¹ Additionally, OWASP’s 2017 report [23] described how such uncertainty could facilitate adapting known exploits to many projects.

We note that not all vulnerabilities are exploited equally—CVE reports for a vulnerability contain a Common Vulnerability Scoring System (CVSS) score [22], a measure of how exploitable that vulnerability is. We leave the consideration of CVSS and exploitability for future work.

Based on our results, the open source community is in need of a standard process of reporting potential security vulnerabilities to open source project owners. This process would allow the open source community and cyber security researchers to privately disclose potential security issues in a standardized way. Such a process could improve the quality and software of open source projects.

¹ This effect can be viewed as a dual of herd immunity with immunization—the more projects use vulnerable libraries, the more risk there is to the community as a whole.

4.2 RQ2: How Common Are Security Notification Policies?

Determining Whether the Open Source Project Has a Security Policy.

We first scanned the project’s README and CONTRIBUTING.md files for the keywords `security`, `vulnerability`, `reporting`, or `disclosure`. We also looked for files whose name contained a keyword. If no such files or keywords existed, we scanned the owner’s user or group account page on GitHub for links to company or user webpages or public Wikis. When such pages existed, we (manually) scanned them for bug bounty or security disclosure policies. In cases where no such information could be gleaned, we considered that project to have no policy for reporting security vulnerabilities.

Results. Recall from Sect. 4.1 that 385 of 600 open source projects contained at least one vulnerable dependency. Of those projects, only 3.4% (13 of 385) had a security vulnerability reporting process. The other 96.6% (372 of 385) open source projects which had a vulnerability had no publicly available security vulnerability reporting process. Overall, out of 600 open source projects, only 3.2% (19 of 600) had some type of security vulnerability reporting process. The remaining 96.8% (581 of 600) had no security vulnerability reporting process based on the aforementioned method.

For the remaining 581 projects, there is no standard recourse for reporting vulnerabilities. Recall from Sect. 2 that opening issues, submitting pull requests, or attempting private contact can result in unpredictable outcomes (including losing a GitHub account to the spam flagging system).

For the 19 open source projects that contained a security notification policy, we manually read through the policy. We broadly categorize these policies as: bug bounty program, email address, or web form. First, 4 of 19 projects had a bug bounty program administered through HackerOne [14], which outlined the process, rules, and scope for an individual reporting a vulnerability. Second, 11 of 19 projects provided an email address to contact in case of a security vulnerability. Some projects provided a more specific security reporting policy that a security researcher might follow in reporting potential issues. Third, 1 of 19 projects contained a web form for submitting vulnerabilities. This project provided a detailed security reporting process. The remaining 3 projects had unique notification policies that did not fit into these three categories.

Security Policies at Scale. Next, we considered projects from popular hosting platforms GitHub, GitLab, and BitBucket. We used two curated lists, one for BugCrowd [1] and one for HackerOne [14], that contained a list of current bug bounty programs [28]. We searched the bug bounty lists for projects contained in the Libraries.io’s dataset [21]. Specifically, we used the repository owner name and project name from Libraries.io to find Bug Bounty programs.

We found that, of the 30,705,634 repositories in Libraries.io’s dataset, only 6,645 open source projects have a bug bounty program. We interpret this as a sign that the open source community does not have a standard process to report

Table 2. Location of contact information

Location of contact information	Count
Account page	339
README	130
Other locations	12
None	119

security vulnerabilities. Additionally, our analysis suggests that significant effort is required to find security policies in projects where they do exist. This has the potential of inducing failures to report vulnerabilities appropriately or at all.

4.3 RQ3: Is Contact Information Available for Open Source Projects?

Approximating Effort to Discern Point of Contact. We manually inspected each open source project using the following steps:

1. Check the project’s README file
2. Check the CONTRIBUTING.md file
3. Check the GitHub Wiki page
4. Check on the repository’s account or group page
5. Check any provided website for the project (e.g., in the project description)
6. Check any provided website on the repository’s account or group page
7. Check whether the Top Contributor for the project has their contact information *publicly* available (not just email addresses in Git commits)

We consider a project to have no contact information available if none of the above steps yield contact information. We used a stopwatch to measure the approximate time taken to find (or fail to find) contact information.

Results. We discovered 19.8% (119 of 600) of open source projects contained no publicly available contact information. Among the remaining 481 open source projects that provided contact information, 27.0% (130 of 481) contained contact information in the README file. The remaining 351 projects required more thorough investigation to determine contact information.

Table 2 shows the breakdown of where we found contact information for open source projects. For the majority, we found the contact information on the repository group’s or top contributor’s account page on GitHub. The next most popular location was within the README file contained in the repository. For the remaining open source projects, we found the contact information in a variety of locations described above.

Table 3 shows the breakdown of different forms of communication we found in these open source projects. In several cases we found multiple forms of communication but no preference or priority associated with each form. The majority

Table 3. Type of contact information found

Form of contact information	Count
Email	312
Website	108
Gitter	29
Google group or forum	22
Twitter	17
IRC	6
Slack	5
LinkedIn	3
Mailing list	3
List of individual contacts	2
Discordapp	1

of cases had an email address. We note a heterogeneity of communication forms associated with open source projects, adding to the potential communication burden associated with reporting vulnerabilities.

We approximated effort required to find contact information by measuring the time taken to search projects as described above. It took us an average of 44 s (± 2.6 s with 95% confidence) to search a project for contact information. Times ranged from 7 s to 300 s. We observed a bimodal effect with times: projects would either take a very short time (e.g., if the top of the README happened to contain contact information) or a very long time (e.g., if it was not clear without searching through many files in the project).

While 44 s may not seem like a significant burden, Liu et al. [18] showed that the first 10 s that users observe a newly loaded web page are critical. These results suggest that a person may lose interest before successfully finding contact information for an open source project.

5 Recommendations and Discussion

In this section, we discuss two potential approaches to addressing the reporting of security vulnerabilities of open source projects. First, we suggest introducing a standardized SECURITY.md file to projects that describes basic contact information and disclosure processes for vulnerabilities. Second, we discuss a potential addition to hosting platforms (such as GitHub) to support private or hidden pull requests that enable developers or security researchers to disclose vulnerabilities.

5.1 SECURITY.md Mechanism for Vulnerability Notification

Given the dearth of security reporting policies among open source projects, we recommend the creation of a SECURITY.md file in open source repositories.

This file would contain contact information and the disclosure policy of an open source project. Of the 19 open source projects that contained such a policy, only one of those projects described the policy in the repository itself, while the remaining 18 projects required additional effort to find the relevant information.

The creation of a `SECURITY.md` file would provide a solution to the open source community that is currently lacking a standard process as shown by both our research and in the Snyk’s report [27]. Additionally, lacking public information can make it difficult to assess the overall commitment to security from an open source project and to understand how to disclose newly discovered vulnerabilities to open source project owners [27].

We suggest an adaptation of an existing RFC, “A Method for Web Security Policies” [6], modified for an open source repository. We suggest using the `SECURITY.md` file in the root of a repository to contain basic contact information (email addresses) and optionally contain text describing the security policy, encryption, and contribution guidelines for the project. Such a file could help inform the community and cyber security researchers with an effective way to report vulnerabilities as they are discovered.

Adding `SECURITY.md` provides a beneficial method for standardization of vulnerability reporting processes. This recommendation also helps to fix an issue that was discovered in the 2017 Open Source Survey by GitHub where one of the largest issues was “Incomplete or confusing documentation” [13]. Additionally, by including this file in an open source project’s repository, it shows that the project has a commitment to improving the security of the project.

Furthermore, Williams and Dabirsiaghi [29] suggest using a vetting process when choosing whether to use a project’s library or source code. Considering the majority of open source projects currently do not have a security reporting process, the addition of the `SECURITY.md` would fulfill Williams and Dabirsiaghi’s recommendation [29] by providing a way to vet projects.

5.2 Adapting Hosts to Facilitate Security Disclosures

As an alternative to `SECURITY.md`, open source hosting platforms could provide new features to facilitate communication of vulnerabilities. For example, based on our experience described in Sect. 2, we suggest the creation of a “verified researcher” tags to be associated with an account that has built a reputation for submitting pull requests or issues that disclose security vulnerabilities. These tags would allow project owners to evaluate contributor account reputation. As another example, hosting platforms could provide features for submitting private/hidden pull requests and issues that would be visible to project owners but not fully public.

Our recommendations seek to increase the interaction between the open source communities and vulnerability research communities. Establishing better interaction would reduce burden on reporting vulnerabilities and vetting contributors (e.g., by focusing on the requests from verified accounts). The increased interaction could help improve security in open source projects.

6 Related Work

We draw inspiration for standardized security policies from several sources.

RFC2142 [3] suggests that organizations maintain a `security@domain` mailbox that is used for security bulletins and questions. However, it does not specify where or how such information should be made known publicly. We build upon this work by suggesting standardized locations for security policy information to be placed in an open source software repository.

Foudil and Shafranovich [6] suggest placing a `security.txt` file in the root of a web server that allows websites to define security policies. Such information could define how owners can be contacted with security concerns or how a bug bounty program would work. We build on this work by suggesting `SECURITY.md` be added to open source repositories.

Snyk's report [27] investigated the top 400,000 public repositories on GitHub to see if there was any documentation for basic security information for the open source projects. No details were made available in the report about the process they used to look for such information, which programming languages they considered, or a definition for basic security information. Additionally, the report did not propose a solution to this problem, but instead highlighted a current problem in the open source community. In our work we focused on one programming language, Java, and outlined how we selected open source projects and gathered information. Additionally, we proposed a recommendation on how to improve the current lack of security vulnerability reporting process in open source projects.

Decan et al. [4] mined statistics about security vulnerabilities from over 600,000 open source projects. They found that 50% of security vulnerabilities survive to 30 months after being introduced. However, they also found that 50% of vulnerabilities were fixed within 1–2 months after discovery. This statistic suggests that having a mechanism in place to disclose vulnerabilities could help contribute to more rapidly fixing vulnerabilities.

GitHub already informs project owners about potentially vulnerable dependencies included in a project by examining commit messages and CVEs [7]. However, GitHub's technique only works for those projects that adopt GitHub's file format for describing dependencies, which is language-specific and limited to Java, JavaScript, .NET, Python, and Ruby. It still does not address the issue of how researchers or other developers (who may have developed other ways to detect vulnerable dependencies) could privately disclose potential vulnerabilities.

7 Limitations

Our recommendation to add a `SECURITY.md` file may be a burden for project developers to properly maintain. Additionally, older or abandoned open source projects may not add new files. In this case, the only option for a security researcher is to leave a public issue or pull request on the open source project so that future users of the project are made aware of the security vulnerability

and have a way to fix it prior to using the project. We note there is currently a limit—but no transparency!—on how many issues or pull requests a security researcher can make before an account is flagged for spam.

Our other recommendation is that open source hosting platforms provide new features for reporting security vulnerabilities. The limitation is that open source project owners cannot do it just themselves as with the `SECURITY.md` file.

8 Conclusion

Both the open source community and the providers of open source repositories need a method to communicate security vulnerabilities, which would both give a voice to project owners about how they want disclosures to occur, and give service providers (e.g., GitHub) better understanding of user needs.

In this study, we evaluated three different aspects of 600 open source Java projects: (1) how frequently open source projects use known vulnerable libraries, (2) whether open source projects contain security vulnerability reporting policies and what kind of policy is it, and (3) how much effort is required to find contact information for an open source project. Our findings showed a high ratio of the open source projects currently contain at least one vulnerable dependency. Although this does not guarantee the vulnerability can be exploited, it does show that there is at least some risk to the project and its users until the dependency can be updated to a non-vulnerable version. Additionally, with the exception of 19 open source projects, the majority of projects lacked some kind of security vulnerability reporting process that was easily accessible. Finally, the majority of open source projects studied did have some kind of contact information, but it was often not easy to find, and there is no guarantee that contact information found would be the correct person or group to contact about potential security vulnerabilities.

To address the current shortcomings of security reporting policies in open source projects, we proposed one recommendation to create a `SECURITY.md` file that would contain the necessary details about the open source project's security reporting policy and who to contact when (or if) a potential security vulnerability is discovered. To encourage the adoption of `SECURITY.md`, we could create a website in similar to securitytxt.org [5] to gain feedback from the open source community to improve the concept of `SECURITY.md`. Another recommendation is for open source hosting platforms to provide new features for private disclosure of vulnerabilities. Many challenges remain in the process of reporting potential security vulnerabilities to the open source community.

Acknowledgments. We thank Snyk [26] for providing us access to their tool and data. This material is based upon work partially supported by the US Air Force Research Laboratory under Contract FA8750-15-2-0075 and US National Science Foundation under Grant Nos. CNS-1646305, CNS-1646392, CNS-1740897, and CNS-1740916.

References

1. BugCrowd: Bugcrowd. <https://www.bugcrowd.com>
2. Cavusoglu, H., Cavusoglu, H., Raghunathan, S.: Efficiency of vulnerability disclosure mechanisms to disseminate vulnerability knowledge. *IEEE TSE* **33**, 171–185 (2007)
3. Crocker, D.: Mailbox Names for Common Services, Roles and Functions. RFC 2142, Internet Engineering Task Force (1997). <http://www.rfc-editor.org/rfc/rfc2142.txt>
4. Decan, A., Mens, T., Constantinou, E.: On the impact of security vulnerabilities in the npm package dependency network. In: MSR (2018)
5. Foudil, E., Shafranovich, Y.: securitytxt.org. <https://securitytxt.org>
6. Foudil, E., Shafranovich, Y.: A method for web security policies. Technical report, Internet Engineering Task Force (2018). <https://datatracker.ietf.org/doc/html/draft-foudil-securitytxt-03>
7. GitHub: About security alerts for vulnerable dependencies. <https://help.github.com/en/articles/about-security-alerts-for-vulnerable-dependencies>
8. GitHub: GitHub and government civic hackers projects. https://government.github.com/community/#civic_hackers
9. GitHub: GitHub and government open source projects. <https://government.github.com/community/>
10. GitHub: GitHub and government research projects. <https://government.github.com/community/#research>
11. GitHub: GitHub trending Java open source projects. <https://github.com/trending/java>
12. GitHub: Octoverse. <https://octoverse.github.com/projects#languages>
13. GitHub: Open source survey. <https://opensource-survey.org/2017>
14. HackerOne: HackerOne. <https://hackerone.com>
15. HackerOne: Vulnerability disclosure policy basics: 5 critical components. <https://www.hackerone.com/blog/Vulnerability-Disclosure-Policy-Basics-5-Critical-Components>
16. Kula, R.G., German, D.M., Ouni, A., Ishio, T., Inoue, K.: Do developers update their library dependencies? *ESE* **23**, 384–417 (2018)
17. Legunsen, O., Hassan, W.U., Xu, X., Roşu, G., Marinov, D.: How good are the specs? A study of the bug-finding effectiveness of existing Java API specifications. In: ASE (2016)
18. Liu, C., White, R.W., Dumais, S.: Understanding web browsing behaviors through Weibull analysis of dwell time. In: SIGIR (2010)
19. Mirhosseini, S., Parnin, C.: Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In: ASE (2017)
20. Munaiah, N., Kroh, S., Cabrey, C., Nagappan, M.: Curating GitHub for engineered software projects. *ESE* **22**, 3219–3253 (2017)
21. Nesbitt, A., Nickolls, B.: Libraries.io open source repository and dependency metadata (2017)
22. NIST: National vulnerability database (2018). <https://nvd.nist.gov>
23. OWASP Foundation: Top ten security risks. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_2017_Project
24. Podjarny, G.: Open source vulnerabilities tripped Equifax, how can you defend yourself? <https://snyk.io/blog/equifax-breach-vulnerable-open-source-libraries>
25. Rapid7: NIST cyber framework updated with coordinated vuln disclosure processes. <https://blog.rapid7.com/2017/12/19/nist-cyber-framework-revised-to-include-coordinated-vuln-disclosure-processes>

26. Snyk: Snyk. <https://snyk.io>
27. Snyk: The state of open source (2017). <https://snyk.io/stateofossecurty>
28. Tetelman, A.: bounty-targets-data (2018). <https://github.com/arkadiyt/bounty-targets-data>
29. Williams, J., Dabirsiaghi, A.: The unfortunate reality of insecure libraries. <https://www.contrastsecurity.com/the-unfortunate-reality-of-insecure-libraries>

Organizational Aspects of FLOSS Projects



EJ: A Free Software Platform for Social Participation

Fábio Macêdo Mendes^{1(✉)}, Ricardo Poppi^{1,2}, Henrique Parra²,
and Bruna Moreira¹

¹ Universidade de Brasilia (UnB), Brasília, Brazil
{fabiomendes, brunamoreira}@unb.br

² Instituto Cidade Democrática, São Paulo, Brazil
{ricardo, henrique}@cidadedemocratica.org.br

Abstract. As the Internet grows on importance as a forum for political activity, it is necessary to occupy it with proper tools for democratic discussion, dialogue and deliberation. Currently, a substantial part of political debate is conducted on social media inside proprietary networks. Those solutions are flagrantly inadequate to build consensus seeking understandings and to mediate the interaction between the government and the citizenry. This work present EJ, a platform for crowd-sourced social participation which uses machine learning based intelligence and gamification techniques to increase engagement and counteract the formation of opinion bubbles and the “echo chamber” effect of social networks.

Keywords: Social participation · E-gouvernement · Web · Free software

1 Introduction

As a growing proportion of the world’s population gains access to the Internet, so grows its importance as a place for public discourse and social organization. An optimistic stance on this fact points to a networked democracy in which information flows horizontally and the common citizen can exert real decision making power and produce solutions to societal problems. On the other hand, the same Internet has allowed mechanisms that support privacy violation, surveillance [1] and spread of misinformation [6] in a scale never seen before. For better or for worse, it is unquestionable that the Internet plays an important role in shaping societies and political systems of the present and of the future [3].

This contradiction of potentialities is perhaps better demonstrated in today’s largest social network, Facebook, with its more than 2 billion users¹. The rise of modern social networks has indeed delivered novel forms of horizontal communication and has given voice to actors that would otherwise be excluded from traditional processes. At the same time, Facebook’s business model is centered

¹ <https://newsroom.fb.com/company-info/>.

around keeping users connected to the platform to deliver targeted marketing content, while gathering any information useful for profiling. This model has been frequently abused to obtain audience and political gain via questionable practices such as inflammatory and sensationalist discourse, disinformation, hate speech², etc; all very detrimental to further democratic values, albeit frequently treated as a tolerable source of revenue by the social network itself³.

We believe that a platform for social participation must be explicitly designed to be resilient to those forms of manipulation rather than arise as an accessory feature of a larger social network. Moreover, participation technologies (digital and otherwise) should be seen as social goods and thus must employ greater levels of transparency and public oversight than for-profit products. An acceptable level of public control of software products is perhaps only attained by using open source tools and technologies.

This report introduces a platform for social participation named EJ (Empurrando Juntas) that aims to fill a specific niche in this universe. We present EJ both at technical level and contextualize it in a more conceptual and perhaps sociological discussion. Technical aspects are considered in Sects. 3 and 4, and technically oriented readers are invited to go there to understand the rationale behind the technology and the most important architectural decisions. The rest of the paper is dedicated to analyzing the role of the platform in promoting participation and the challenges we perceived to creating the networked democracy hinted in the first paragraph.

2 Platforms for Social Participation

Compared to more traditional processes for consultation and public debate, the Internet enables a much faster flow of information which helps making large scale decentralized organizations viable. Those features have certainly sparked many initiatives towards building tools for a networked democracy that aims to bring prominence to the common citizen in shaping collective decisions. Those initiatives can usually be categorized in 4 broader areas (with many possible intersections): transparency⁴, petitions and public awareness⁵, debating⁶ and deliberation⁷. We can also classify initiatives with regard to its relationship with the State, in which some take a decisively anti-government stance (e.g., Wikileaks⁸), while others are directly conducted or sponsored by the State.

While we acknowledge that all of those areas are similarly important, we decided to focus our efforts on debating and consultation. We took this path out

² <https://www.theguardian.com/technology/2018/apr/06/myanmar-facebook-criticise-mark-zuckerberg-response-hate-speech-spread>.

³ <https://www.propublica.org/article/facebook-enforcement-hate-speech-rules-mistakes>.

⁴ <https://www.opengovpartnership.org/>, <https://serenata.ai/>.

⁵ <https://avaaz.org>, <https://www.change.org/>.

⁶ <https://debatemap.live/>, <https://www.artikulate.in/>.

⁷ <https://decidim.org/>, <https://arxiv.org/abs/1707.06526>.

⁸ <https://wikileaks.org/>.

of the belief that the two major features of a platform for social participation should be the possibility of probing opinions and listening to novel inputs that users might add to the debate. The archetypal tools for handling each of those cases are the discussion forum and the poll. The first is useful to construct and refine positions and ideas in an unstructured fashion, in open dialogue; the second probes perceptions of the crowd from a predefined set of options.

Internet forums have (an often deserved) reputation for leading to toxic behaviors. Even when discounting for this possibility, participation in those environments demand users to invest time and knowledge, presenting a substantial barrier for engagement. There are of course many examples of groups of engaged and uniquely qualified individuals that produce rich discussions and communities, but we believe that the discussion forum model is severely limited when applicable to wide audiences.

If only a minority of interested users decide to participate, discussions may feature an unfair representation. This fragility undermines the political weight of those initiatives by making it easier to portray conclusions as instances of group-think and undemocratic. Excluding the rare cases in which a hard consensus is obtained, it is hard to elicit opinions in a structured form. How then should the result of open discussions be used to support a collective decision?

While platforms for open ended discussion produce barriers to participation and engagement, polls suffer from the opposite problem: the cost of participation is very low, but more qualified users are unable to enrich the debate or introduce novel ideas even when willing to do so. Polling make it relatively easy to educe the preferences of the majority since data is created in a quantitative form. However, users might feel alienated from the decision making process if they perceive all options as unappealing or, worst yet, as being designed to fulfill some specific agenda.

2.1 Crowdsourcing

In order to bridge those two approaches, we believe that the design of collective deliberation must consider the pedagogical aspect of interaction. A good example of digital tools with accessible interfaces that take elements from both ends are the so called “crowdsourced discussion” architectures⁹ [4,15]. As a crude approximation, those platforms are based on a polling mechanism in which the list of alternatives can be continuously expanded with user interaction.

Participants of those platforms can control the energy dedicated to each discussion by switching from “voting” to “making new contributions” modes. Since users are constantly evaluating contributions from other users, it should be possible to automate data analysis to understand and identify different opinion profiles.

A good example of this kind of platform is the open source application Pol.is. In its current state, it provides a simple interaction in which administrators propose a theme for discussion and users can express their positions either by

⁹ <https://www.allourideas.org/>.

publishing a simple paragraph of text or by evaluating other user opinions. Pol.is then uses machine learning algorithms to identify affinity groups and present this information to all participants. While this is a promising approach, we believe that there are improvements to be made on how to use this information to promote some kind of effective collective action. We use Pol.is as the main inspiration of our approach, but add an additional layer of gamification in order to put some incentives for user to engage and create prolific debates and deliberation.

2.2 Social Networks

The enormous user base of social network websites (Facebook alone comprises almost a third of human population) guarantees a diverse audience. In spite of that, each user may experience very low diversity of ideas: algorithms that perform content selection end up placing users into “echo chambers” that recirculate a small set of ideas without challenging them with input from different social groups [12]. Diversity of viewpoints are rarely appreciated and debates tend to be contentious. This dynamics shifts the goal of political struggle from seeking democratic compromises and consensus to establishing (real or apparent) numeric majorities to subject the opposition to the rule of the crowd.

Those proprietary networks use their position as keepers of giant volumes of data to control the flow of information and influence user behaviours. One plausible response to those threats is to advocate for decentralization. This crucial aspect can be handled in many different ways, the most simple one is distribution of a software product as open source. This gives the possibility of users and organizations to manage their own deploys and set their own rules of governance.

Beyond that, some projects advocate for protocols and formats that enable different instances to communicate and share data in a federated fashion. Notable projects in this area include PeerTube¹⁰ and Mastodon¹¹. More radical approaches like Tim Berners-Lee’s Solid [16] and the Dat network [11] even calls for an overhaul of the Internet structure to favor local computation, granular access controls and P2P communication. It is conceivable those projects they might experience short term success, but it is unlikely to be in a scale that significantly changes the current panorama of social networks in spite of the very interesting technical achievements.

2.3 The Role of the State in Participation Platforms

It seems to be a clear desire of governmental and civil society organizations to have tools for democratic dialogue and direct communication. There are many examples of governmental entities that sponsored initiative based on collective intelligence, but most of those experiences suffered from difficulties in scaling or generating the desired impact.

¹⁰ <https://joinpeertube.org/>.

¹¹ <https://joinmastodon.org/>.

A notable experience took place in Brazil during the consultations for the Marco Civil da Internet (Brazil Rights Framework for the Internet), in which an existing Open Source Social Network developed in Brazil called Nosfero¹² was used. This led to number of other public consultations using similar technologies for collective deliberation. State’s collaboration with those open source communities to build shared digital goods preserves sovereignty, while ensuring that citizens have autonomy to develop civic technologies independent of a direct custody from a State actor.

3 Conception and Architecture

EJ was conceived to work both as a standalone product and as an embedded component in some existing application. In both cases, we expect that most of our user base will access the platform using mobile devices since that number is growing worldwide and already accounts to most of web traffic, nowadays¹³. Mobile access implies that we have the opportunity to engage our users in a continuous manner, either by expecting constant availability, or by using more active approaches such as push notifications and web apps.

3.1 User Experience

EJ organizes topics of discussion around “conversations”. Each conversation is introduced with a single paragraph of text that should try to frame how users should contribute to the discussion, like in Fig. 1. Users are then able to insert comments associated with a conversation or vote if they agree or disagree with other user’s propositions.

As users interact with some conversation (either proposing new comments or by voting on propositions), it gradually become possible to recognize the different opinion profiles. As soon as the platform is able to classify users into distinct opinion groups, it display results to all participants. That way, not only owners of each conversation can extract useful metrics to guide policy and decision making, but they can also justify their decisions from the shared information available on the platform. Moreover, each participant is able to access how their own opinions, or those from their network of peers and compare with the whole. We believe that this form of transparency towards information is an important factor to counteract the “echo chamber” effect of traditional social networks.

Only minimal personal information is requested from new users and none of it is exposed publicly like a social network. A potential threat of this model is that it

¹² <http://noosfero.org/>.

¹³ StatCounter, a firm specialized on web traffic analysis, shows worldwide mobile usage slightly above desktop at 52.98% vs 47.02% (<http://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide>). A more detailed account (<https://www.stonetemple.com/mobile-vs-desktop-usage-study/>) shows even large gaps depending on the selected metric (e.g., page views, unique users, time on site, etc).



Fig. 1. EJ user interface showing a conversation

provides little safeguards against bots and other malicious users since it lowers the barrier for creating new accounts. We decided not to tackle this difficult problem, but rather to make it possible to exclusively use external authentication via OAuth and rely on an identity provider trusted by deployment.

3.2 Machine Learning

Machine learning enriches the platform in the detection of opinion groups and understanding how each user relates to each group. In Machine Learning jargon, detection of opinion groups is a non-supervised classification problem. We tried a few classical methods (K-means and SVM) with moderate success. More reliable outcomes are obtained when making allowance for prior information. Our approach to classification requires conversations to be created in association with some stereotypical opinion groups. Stereotypes are controlled by the conversation author and behave similarly to regular users when interacting with comments. Those users define their initial centroids in the clusterization algorithm and serve as labels that more reliably identify clusters across different classifications.

EJ encodes votes as +1 (agree), -1 (disagree) and 0 (skip). The set of user interactions is thus a vector of components in that range. In a typical situation, this data set comprises mostly of missing data entries, and requires a good imputation strategy. We used a mean-value based imputation for simplicity and

computational efficiency and better results were obtained by using collaborative filtering [17], but at a significant efficiency cost.

Traditional K-means consists of 4 steps: (1) centroid initialization, (2) labeling elements to the closest centroid, (3) re-calculation of each centroid, and (4) go to 2 until convergence. Our modification uses stereotypes as the initial centroids in step (1) and forces that each stereotype will stay in its own cluster by never re-labeling them in step (2). This simple modification not only is able to introduce some form of prior information, but has the positive side-effect of generating predictable labels for each run of the classification algorithm.

The default visualization display cluster sizes, similarities and diversity in a stylized manner. We also employ classic machine learning algorithms for dimension reduction aimed at other visualization options. For instance, a scatter plot of user inputs displays the opinion of each user together with the stereotypes. Our empirical evaluation selected principal component analysis (PCA) as opposed to more sophisticated alternatives such as t-SNE [9] or MDS [5] for its superior reliability and efficiency. PCA is a rotation based method, which preserves some intuitive geometric relationships between elements, making it more accessible to non-experts.

3.3 Web Platform

As a web native platform, EJ subscribes to a few broad principles that guides all architectural decisions.

Progressive enhancement: Web applications are created using HTML, CSS and JavaScript. Progressive enhancement postulates that the website must be usable in its most basic HTML-only form and each new layer (CSS, JavaScript) should progressively enhance the user experience instead of being a hard requirement.

Mobile first design: UX (user experience) and UI (user interaction) is designed primarily for mobile, and desktop experience is enhanced with CSS media queries. It also implies that the application should be usable from poor connections on underpowered devices. We prioritize small JavaScript size, since loading, parsing and executing implies non-trivial CPU costs which blocks the main application thread.

Progressive web app: EJ only uses functionality available in the web platform. We developed it as Progressive Web App (PWA) in order to avoid the costs of a native implementation. PWAs blur the distinction of native mobile applications and web sites. Users can install an icon on their home screens and it spawns a background process that allows push notifications and offline interaction. Those are not intrusive technologies and we were able to develop most of the platform largely ignoring any specific needs for “PWAs”.

API as a product: Since we expect EJ to be easily embedded on user applications, the REST API is designed as an independent product and should be able to easily expose any information or interaction to third parties. The API also implements business logic regarding authentication and authorization policies in the platform.

3.4 Technology Stack

EJ was initially conceived as fork of the open source project Pol.is¹⁴, which is based on Node.js and Clojure (for a in-house implementation of Machine Learning algorithms). This initial experience was afflicted with a bad deployment story and poor communication with Pol.is core development team. This lead to the decision of abandoning Pol.is and change the stack in favor of Python.

EJ uses the Django Framework¹⁵ for web development and the Pydata stack (Numpy [20], Pandas [10], Scikit-Learn [13]) for building the data analysis pipelines. On the frontend side, it uses Sass to manage CSS and adopts a somewhat traditional web architecture in which pages are rendered server-side rather than by the client. This approach is sometimes touted as ROCA¹⁶ (Resource Oriented Client Architecture) to confront with the recent SPA (Single Page Application) architectural trend, and is perhaps nothing more than adoption of what used to be considered the best practices for web development a few years ago. Following the principles of ROCA, we used a library for unobtrusive JavaScript named Unpoly¹⁷ and decided to avoid client-side frameworks completely.

By following the principle of progressive enhancement, it was possible to create a product that is accessible by many clients, ranging from command line web-browsers, web spiders, screen readers to the evergreen versions of major Browsers. While we expect most users to lie in the last category, progressive enhancement also benefit this audience by providing a workable (but degraded) website even when connection is poor and downloads fail.

4 Development and Results

EJ is still in active development and has released only a single stable version. We do not have data about its real use performance. This section rather discuss how development was organized and which results were obtained.

4.1 Team

EJ has gone through two different development phases, each one carried by a different team. In the first phase, it was developed by HackLab/São Paulo¹⁸, which is a socially oriented Startup that employs and develops free software technologies. This initial phase was focused on a product for the Perseu Abramo Foundation, which is a Brazilian think tank associated with the Partido dos Trabalhadores (Workers' Party, of ex-president Lula da Silva fame). In the second phase, development moved to the University, in which part of the original team

¹⁴ <https://pol.is/>.

¹⁵ <https://www.djangoproject.com/>.

¹⁶ <https://roca-style.org>.

¹⁷ <https://unpoly.com/>.

¹⁸ <https://hacklab.com.br/>.

was replaced by students. This report describes the second phase of development in which all authors took an active role and was a collaboration with the Federal Government to create a participation platform for teenagers via the National Secretary for the Rights of Children and Adolescents.

The team was divided roughly into 4 parts: design/UX, frontend, backend and operations. Since EJ was developed as a University project in a Software Engineering department, we took into account pedagogical outcomes when allocating team members. Every two weeks, students could then negotiate the allocation in each of the last 3 teams in which we encouraged a moderate level of rotation so everyone could experience different aspects of development (sometimes at expense of raw productivity).

4.2 Continuous Delivery and Deployment

Continuous deployment was an important part of the development culture in EJ. After some adjusting, we finally ended up organized our Git repository¹⁹ into two main branches: a master branch which only receives tagged release commits and a development branch that is used to integrate contributions from all developers. Any commit in those branches that pass an automated test suite automatically trigger a deploy script pointing to some specific instance of the platform.

The system is also integrated with Codeclimate²⁰, a static analysis tool that tracks the evolution of code quality metrics, test coverage and can alert to the introduction of defects and regressions. Since most development takes place in feature-specific branches, those warnings can prevent some defects from entering into the main development branch. A negative side effect of requiring manual merging of feature branches with explicit code review is that it burdens team members assigned to those tasks. We decided that changes deemed low risk (e.g., updates to translation strings, documentation, small CSS fixes or improvements to the test suite) should be developed directly into the development branch without any external review step before merging.

Generally speaking, productivity increases when good tools and automations are available [2]. Since we made the initial decision of hosting the code on Github, the choice of tools was largely guided by the options that platform already provides (e.g., issue tracker, project management Kanban board) and have a good story of integration.

Since a large part of the development team is comprised of students with high rotation rate and considerable learning curve, we decided to avoid emphasizing too much the role of processes and culture and tried to rely heavily on automation. The objective is to gain better efficiency and shorter learning curve due to better tooling, at the same time we guarantee all contributions to the development process are crystallized in the form of code. Creation of automation scripts is thus treated as an important part of the development as is writing application code. This includes code responsible for deployment and testing, but

¹⁹ <https://github.com/ejplatform/ej-server/>.

²⁰ <https://codeclimate.com/>.

also automation of everyday chores such as creating new environments, building assets and test databases, managing and executing Docker images, etc.

5 Conclusions

Building effective and constructive social participation mechanisms is a chief problem of contemporary democracies. Citizens are increasingly reporting a sense of misrepresentation and frustration with democracy worldwide, and the perceived efficacy of the government is diminishing [19]. Some scholars have pointed that this growing scepticism on the capacity of democracies to solve social issues may lead us to an unprecedented state of “a recession” [7], in the number of democracies in the world. All this in a time in which humanity faces unique and very pressing challenges in politics [8], economy [14], environment preservation [18], health, education and many others aspects of life.

Of course, none of those problems have simple solutions. We, as technologists, may offer tools that improve the collective capacity for finding answers to those dilemmas. This report introduces EJ, a simple system of participation that can be used to improve the dialogue between the government and autonomous organizations with the population. While the technology has not been adopted in any large deployment yet, we believe that it was built taking into consideration important aspects of collective participation.

Our two fundamental working hypothesis is that EJ is able to promote a higher level of engagement than existing platforms for social participation, and that its mechanisms to counteract polarization are effective for building civil dialogue and discussion. While engagement is easy to measure, polarization requires more indirect observation. Existence of clearly separated opinion groups is not necessarily an indication of polarization, but might rather stem from valid and rational disagreements between different perspectives. Nevertheless, it is still possible to probe some indirect effects of polarization. Polarized discussions arise from unfair and distorted perceptions about the other group’s motivations and points of view and generally produces a mismatch in familiarity with basic facts about a subject and acceptance of fake and incorrect information.

EJ was built on the notion that socially relevant technologies should be created as common goods: making it free (as in freedom) for anyone to use, to study, and to adapt. This provides a good level of control and oversight and protects collectivity from vested interests from corporate owners of proprietary solutions.

Acknowledgements. The authors would like to thank Fundação Perseu Abramo and the former Ministry of Human Rights (now transformed into a secretary by the current Brazilian government) for recognizing the importance of direct social participation in shaping public policy and for financial support.

References

1. Albrechtslund, A.: Online social networking as participatory surveillance. *First Monday* **13**(3) (2008). <https://doi.org/10.5210/fm.v13i3.2142>, <http://journals.uic.edu/ojs/index.php/fm/article/view/2142>

2. Barry, E.J., Kemerer, C.F., Slaughter, S.A.: How software process automation affects software evolution: a longitudinal empirical analysis. *J. Softw. Maint. Evol.: Res. Pract.* **19**(1), 1–31 (2007). <https://doi.org/10.1002/smr.342>
3. Best, M.L., Wade, K.W.: The internet and democracy: global catalyst or democratic dud? *Bull. Sci. Technol. Soc.* **29**(4), 255–271 (2009). <https://doi.org/10.1177/0270467609336304>
4. Brabham, D.C.: Crowdsourcing the public participation process for planning projects. *Plan. Theory* **8**(3), 242–262 (2009). <https://doi.org/10.1177/1473095209104824>
5. Cox, T.F., Cox, M.A.A.: Multidimensional scaling. In: No. 88 in *Monographs on statistics and applied probability*, 2nd edn. Chapman & Hall/CRC, Boca Raton (2001)
6. Del Vicario, M., et al.: The spreading of misinformation online. *Proc. Natl. Acad. Sci.* **113**(3), 554 (2016). <https://doi.org/10.1073/pnas.1517441113>
7. Diamond, L.: Facing up to the democratic recession. *J. Democr.* **26**(1), 141–155 (2015). <https://doi.org/10.1353/jod.2015.0009>
8. Levitsky, S., Way, L.: The rise of competitive authoritarianism. *J. Democr.* **13**(2), 51–65 (2002). <https://doi.org/10.1353/jod.2002.0026>
9. van der Maaten, L., Hinton, G.: Visualizing data using t-SNE. *J. Mach. Learn. Res.* **9**(Nov), 2579–2605 (2008)
10. McKinney, W.: Data structures for statistical computing in python. In: *Proceedings of the 9th Python in Science Conference (SCIPY 2010)*, pp. 51–56 (2010). [https://doi.org/10.1016/S0168-0102\(02\)00204-3](https://doi.org/10.1016/S0168-0102(02)00204-3)
11. Ogden, M., Mc Kelvey, K., Madsen, M.B., et al.: Dat-distributed dataset synchronization and versioning. *Open Sci. Fram.* **10** (2017)
12. Pariser, E.: *The Filter Bubble: What The Internet Is Hiding From You*. Penguin Books Limited (2011). <https://books.google.com.br/books?id=-FWO0puw3nYC>
13. Pedregosa, F., et al.: *Scikit-learn: machine learning in python* (2012). <https://doi.org/10.1007/s13398-014-0173-7.2>. <http://arxiv.org/abs/1201.0490>
14. Piketty, T., Goldhammer, A.: *Capital in the twenty-first century* (2017). oCLC: 1063105013
15. Salganik, M.J., Levy, K.E.C.: Wiki surveys: open and quantifiable social data collection. *PLOS ONE* **10**(5), e0123483 (2015). <https://doi.org/10.1371/journal.pone.0123483>, <https://dx.plos.org/10.1371/journal.pone.0123483>
16. Sambra, A.V., et al.: *Solid: a platform for decentralized social applications based on linked data* (2016)
17. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: *Proceedings of the 10th International Conference on World Wide Web, WWW 2001*, pp. 285–295. ACM, New York (2001). <https://doi.org/10.1145/371920.372071>
18. Masson-Delmotte, V., et al.: IPCC. Global warming of 1.5 $\hat{\text{A}}^{\circ}\text{C}$. An IPCC special report on the impacts of global warming of 1.5 $\hat{\text{A}}^{\circ}\text{C}$ above pre-industrial levels and related global greenhouse gas emission pathways, in the context of strengthening the global response to the threat of climate c, p. 792 (2018). <https://doi.org/10.1017/CBO9781107415324>
19. Van Reybrouck, D.: *Against Elections* (2013)
20. van der Walt, S., Colbert, S.C., Varoquaux, G.: The NumPy array: a structure for efficient numerical computation. *Comput. Sci. Eng.* **13**(2), 22–30 (2011). <https://doi.org/10.1109/MCSE.2011.37>



Introducing Agile Product Owners in a FLOSS Project

Matthias Müller^(✉), Christian Schindler, and Wolfgang Slany

Institute of Software Technology, Graz University of Technology, Graz, Austria
{mueller,cschindler,wslany}@ist.tugraz.at

Abstract. Sponsored Open Source Software projects, driven by various actors, have to balance the needs of volunteer contributors and business objectives. This work presents Catrobat, a FLOSS project established at Graz University of Technology, and how it introduced agile product owners. Product owners communicate the product vision, provide a general direction, decide about features, and prioritize requirements that are implemented by the community, i.e., they are ultimately responsible for the product. This agile approach is intended to ensure a certain outcome, such as business objectives, but also to react to the needs of community members and users on a short-term basis. This paper presents how therefore this role has been defined and the processes have been adapted.

Keywords: FLOSS · Open Source Software ·
Agile Software Development · Sponsored Open Source Communities

1 Introduction

Governing Free/Libre Open Source Software (FLOSS) projects has already been subject of extensive research in the past. Publications on FLOSS [10] pointed out the role of leaders in such communities, that are either fulfilled by the projects' initiators or core-contributors. A main task for them is to provide a shared vision and govern the projects [7, 10]. However, in recent years we observed an increasing involvement of businesses in open source communities. Open source projects are nowadays often situated in ecosystems strongly connected to companies and social communities [5]. This means that businesses and other entities are establishing new, so called sponsored, open source projects [17]. These project not only follow a shared vision, but also business objectives set by the establishing entity. The challenge in governing these projects is to find a balance between the motivation of the contributors and business objectives [12]. Governance must therefore consider several aspects such as decision making, interaction, or release planning [5]. Failing in governing this relation between business objectives and community interests has been identified as a potential reason for the collapse of open source projects [1]. In general, leaders must be trusted by the community [7]. Therefore, governing open source projects becomes increasingly challenging and needs to respect a variety of different aspects.

In this work we present the case of Catrobat, a FLOSS project at Graz University of Technology, developing mobile visual programming frameworks and tools [15]. Although it is based on an open community, the main contributors are students who may participate as part of their curriculum [8]. These students are limited in the freedom of contribution, since a certain academic outcome is mandatory and there is also connected research with certain requirements that must be satisfied. In addition to that, the created software is published on different markets that require certain standards to be fulfilled. Requirements, originating independently of the contributors and users, e.g., by research or cooperation, must be balanced with the community’s interests to keep them actively involved. Therefore, the project’s leaders act as agile product owners since 2018. In this work we line out the process how this role was introduced in this project and how the resulting connected up- and downsides are dealt with.

2 Motivation to Introduce Product Owners

An all-time issue within the project has been that the number of proposed issues, representing user-stories and bugs, grew faster than the number of solved issues. Therefore, the need of prioritization came up to ensure that urgent and important requirements get finished in time. Furthermore, contributors did not thoroughly maintain the publicly reported issues, e.g., bugs or missing functionalities, resulting in a constantly growing issue pool. As also outlined in previous research [4], features requested by external parties often get unrecognized by core-developers. Lacking to meet requests from users can be frustrating for the community and may impact the project negatively [1]. These aspects made it necessary to introduce a role to keep track of, sort, prioritize issues independently from their origin. Catrobat already applies several individual agile methods and various chances and challenges of these methods were already discussed in the past [2,3,9]. Due to the positive experiences with agile principles, also this new role was supposed to be based on agile methodologies.

3 Product Owner Within Catrobat

The introduction of product owners required several organizational changes. Besides defining this formal role, also processes and communication needed to be adapted to the new circumstances, as outlined in the following sections. Although this role is common in industry, special attention is needed in open communities such as Catrobat. Their character requires to focus on the balance between the different needs of contributors, users and external stakeholders, that are all involved for different individual reasons. Also the constant change of the community and direct involvement of users comes along with further challenges.

3.1 The Role “Product Owner”

Leaders in open source projects implicitly perform actions and fulfill responsibilities as they are described for product owners. In Scrum, a product owner

has the responsibility for the backlog to maximize value and represent external interests [6, 13]. Furthermore, they need to communicate the vision of the desired product and be a leader for the team [11]. It is important to note that these responsibilities are based on collaboration with the team, making it necessary to have a common understanding and language [13]. However, decisions by the product owner must be made visible to and be respected by all people involved [14]. *“The product owner is the one person ultimately responsible for the success or failure of the project”* [6]. Therefore, the founder of the project and experienced contributors have been assigned with this role. Although, Scrum defines this role for a single person [11, 14], these product owners form a board, similar to a committee that is common for FLOSS projects [7]. Derivations of Scrum, also having multiple product owners, can be found in successful industry projects too [16]. The decision therefore was based on the constant need of having a product owner available to the contributors, since previous research has shown contributors are working on an irregular schedule [9]. Therefore, constant availability for information exchange must be ensured. This has been identified as a main success factor for Scrum in industry [16]. Furthermore, whereas in industry this role should be performed as full-time position [11], in Catrobat, for a lack of resources, this role can just be fulfilled on a part-time basis, resulting in the need for several people.

Specific parts of the project also have *project owners* that are long-term and experienced contributors, having the same responsibilities and possibilities as the product owners within their specific scope. However, they are not allowed to develop user stories that they have specified themselves. This shall foster collaboration between all involved contributors. The co-structure of product owners representing sponsored goals, e.g., by research, cooperation or user-feedback, and projects owners originating from the community shall increase the commitment to the project. Introducing a governing structure that pays respect to both sides can be considered a main task for such open source communities [17].

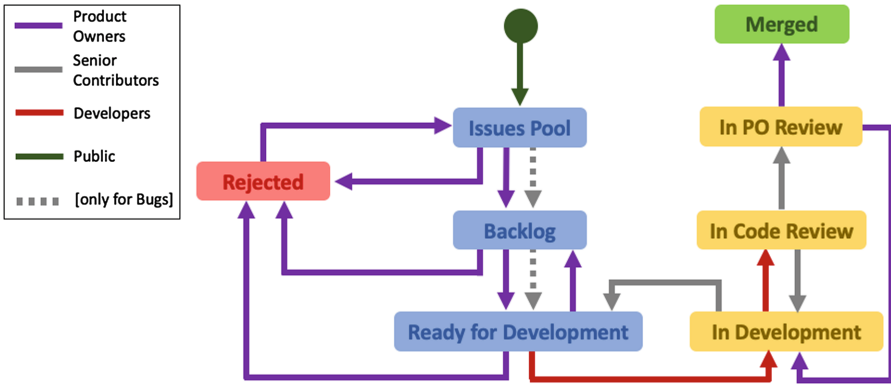


Fig. 1. Catrobat’s development workflow considering product owner interactions.

3.2 Development Workflow

To introduce this new role, a development process, as illustrated in Fig. 1, needed to be introduced. Catrobat's development is managed through a issue tracking system, which is open for all interested contributors. Also non-programmers are enabled by such issue tracking systems to report bugs and feature requests, which reflect the ideas of the users [4]. To prevent duplicate work and ensure the quality standards are met also external contributors are invited to work with this system. Besides that, this workflow is intended to allow frequent and fast releases, which is a challenge for many community driven projects [17]. Therefore, product owners in this workflow have three major tasks:

- Defining and prioritizing requirements and issues for the developers. Whereas, external contributors are not necessarily required to follow these predefined issues (e.g. work freely on ideas), participating students have to choose issues provided in *Ready for development*. However, also common contributors are asked to communicate their ideas to avoid rejection in the acceptance phase. Therefore, the creation of new issues is open for the public, also allowing to consider these ideas in the workflow, avoiding the mentioned rejection afterwards and foster involvement of users and the community.
- Discussing proposed requirements in planning games to clearly communicate the objectives and to get the developers' commitment.
- Functional acceptance of issues and merging them into the main repository. This step is necessary for all contributors to ensure the quality and prevent unfinished, buggy or inappropriate work being published.

The transition of issues from *Backlog* to *Ready for development* happens in a joint planning game. In this, the issues are estimated and developers assure a joint understanding of them. Whereas product owners have the key role in the first and last phase of the process, development is managed entirely by the developers. This includes that issues are not preassigned to contributors, reducing dependencies on certain individuals. Therefore, discussing proposed requirements with the developers and getting their commitment is essential for this agile workflow. Developers are also asked to review the work of others if it complies to the project's quality standards. This shall strengthen the collective code ownership in this process. An exemption in the workflow exists for bugs. They can be claimed by developers at any time without the involvement of product owners. Although, final product owner approval must be granted just like with scheduled issues. This supports the benefit of open source software projects - that bugs can be found, fixed and released quickly.

3.3 Communication

An important requirement for the introduction of product owners has been strengthening communication within the project. This step also focused on encouraging exchange between contributors, e.g., jointly discussing development tasks. Regular on-site meetings with student-contributors get reinforced and

Slack got introduced as main communication platform for all contributors, stimulating discussion in different topic specific channels. This is also intended to document decisions and information for currently absent contributors that might be involved in the following implementation phase. Beside the communication within the team, also product owners need regular meetings and exchange about the project's status. Therefore, following activities have been put into focus:

- Weekly Get-Together of to discuss upcoming features and requirements. This also includes backlog refinement and obtaining feedback from contributors.
- Monthly Planning-Games of product owners with the development team to schedule issues for *Ready for development*. During this event also the specification of issues is discussed, e.g., if developers need further clarifications, or if they are blocked by each other. Product owners hand over the prioritized issues for the next month to the contributors.
- Continuous exchange about current issues on designated Slack-channels and on an individual basis, e.g., via e-mail or comments on Jira and GitHub.

All this is intended to be open and transparent, what is essential for successfully governing FLOSS projects [7]. A challenge is the efficient communication between product owners. Especially for the case when one product owner answers questions from contributors, it is important that all others are informed about decisions made in their absence. This makes documentation and communication essential for this multi-person product owner approach.

4 Discussion

Although this workflow has just been introduced in early 2018, positive feedback is received from contributors. A challenge identified is to centralize communication that got essential for the collaborative nature of the workflow. Whereas contributors before exchanged in a way preferable for them, they must now be streamlined on dedicated channels. This is also true for the introduced product owners. An increasing number of messages and online time by contributors is outlining a preferable progress to tackle this challenge. Since this work solely points out the experience of this specific case, further long-term research on this approach is needed to be able to evaluate its success.

5 Conclusion

This work provides an approach for sponsored open source projects to balance the involved parties' needs and the freedom of contributors in an agile way. A simple workflow with clear responsibilities is provided that might be a response to the growing involvement of businesses or public institutions in open source communities. The introduced role of product owners can be seen as an extension to the already often defined role of leaders governing a FLOSS project. However, by following the proposed workflow and role definition based on Scrum, collaboration and communication in this open setting can be fostered, by ensuring the development of required business objectives at the same time.

References

1. Ehls, D.: Open source project collapse—sources and patterns of failure. In: Proceedings of the 50th Hawaii International Conference on System Sciences (2017)
2. Fellhofer, S., Harzl, A., Slany, W.: Scaling and internationalizing an agile FOSS project: lessons learned. In: Damiani, E., Frati, F., Riehle, D., Wasserman, A.I. (eds.) OSS 2015. IAICT, vol. 451, pp. 13–22. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17837-0_2
3. Harzl, A.: Combining FOSS and Kanban: an action research. In: Crowston, K., Hammouda, I., Lundell, B., Robles, G., Gamalielsson, J., Lindman, J. (eds.) OSS 2016. IAICT, vol. 472, pp. 71–84. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39225-7_6
4. Heppner, L., Eckert, R., Stuermer, M.: Who cares about my feature request? In: Crowston, K., Hammouda, I., Lundell, B., Robles, G., Gamalielsson, J., Lindman, J. (eds.) OSS 2016. IAICT, vol. 472, pp. 85–96. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39225-7_7
5. Kilamo, T., Hammouda, I., Mikkonen, T., Aaltonen, T.: From proprietary to open source—growing an open source ecosystem. *J. Syst. Softw.* **85**(7), 1467–1478 (2012)
6. Lacey, M.: *The Scrum Field Guide: Practical Advice for Your First Year*. Addison-Wesley Professional, Boston (2012)
7. Lerner, J., Tirole, J.: Some simple economics of open source. *J. Ind. Econ.* **50**(2), 197–234 (2002)
8. Müller, M., Schindler, C., Slany, W.: Engaging students in open source: establishing FOSS development at a university. In: Proceedings of the 52nd Hawaii International Conference on System Sciences (2019)
9. Müller, M.: Agile challenges and chances for open source: lessons learned from managing a FLOSS project. In: 2018 IEEE Conference on Open Systems (ICOS), pp. 1–6 (2018)
10. Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., Ye, Y.: Evolution patterns of open-source software systems and communities. In: Proceedings of the International Workshop on Principles of Software Evolution, pp. 76–85. ACM (2002)
11. Pichler, R.: *Agile Product Management with Scrum: Creating Products that Customers Love*. Addison-Wesley Professional, Boston (2010)
12. Rosén, T.: Open source business model: balancing customers and community. Ph.D. thesis, Industrial Marketing and Industrial Economics, The Institute of Technology, Linköping University (2008). Report code: LiU-TEK-LIC 2008:26
13. Schwaber, K.: *Agile Project Management with Scrum*. Microsoft Press, Redmond (2004)
14. Schwaber, K., Sutherland, J.: *The Scrum Guide*. Scrum Alliance (2017)
15. Slany, W., Luhana, K.K., Müller, M., Schindler, C., Spieler, B.: Rock bottom, the world, the sky: catrobat, an extremely large-scale and long-term visual coding project relying purely on smartphones. In: *Constructionsim 2018* (2018)
16. Sverrisdottir, H.S., Ingason, H.T., Jonasson, H.I.: The role of the product owner in scrum—comparison between theory and practices. *Procedia-Soc. Behav. Sci.* **119**, 257–267 (2014)
17. West, J., O’Mahony, S.: Contrasting community building in sponsored and community founded open source projects. In: Proceedings of the 38th Annual Hawaii International Conference on System Sciences, p. 196c (2005)



What Are the Perception Gaps Between FLOSS Developers and SE Researchers? A Case of Bug Finding Research

Yutaro Kashiwa^(✉), Akinori Ihara, and Masao Ohira

Wakayama University, Wakayama, Japan

kashiwa.yutaro@g.wakayama-u.jp, {ihara,masao}@sys.wakayama-u.ac.jp

Abstract. In recent years, many researchers in the SE community have been devoting considerable efforts to provide FLOSS developers with a means to quickly find and fix various kinds of bugs in FLOSS products such as security and performance bugs. However, it is not exactly sure how FLOSS developers think about bugs to be removed preferentially. Without a full understanding of FLOSS developers' perceptions of bug finding and fixing, researchers' efforts might remain far away from FLOSS developers' needs. In this study, we interview 322 notable GitHub developers about high impact bugs to understand FLOSS developers' needs for bug finding and fixing, and we manually inspect and classify developers' answers (bugs) by symptoms and root causes of bugs. As a result, we show that security and breakage bugs are highly crucial for FLOSS developers. We also identify what kinds of high impact bugs should be studied newly by the SE community to help FLOSS developers.

Keywords: Open source software · High impact bug · Interview

1 Introduction

The importance of FLOSS is increasing day by day, not only for personal use but also for enterprises to incorporate it into parts of their products. With the increase in the number of FLOSS users, use cases have also been expanding. As a result, lots of bugs are being reported to FLOSS projects. In the field of software engineering, many methods have been proposed to help FLOSS developers predict faults in modules, localize and repair faults in source code, and so on. However, these methods are too diverse for FLOSS developers to follow and effectively adapt to their projects as needed. Although several studies [28] provided a systematic review to mitigate the difficulty in understanding existing methods, they are neither beyond grouping existing studies nor necessarily in line with FLOSS developers' needs for support. Ideally speaking, we should try to thoroughly understand which bugs cause the most trouble for FLOSS developers and propose solutions to fix the bugs effectively and efficiently. In this

study, we try to reveal the gap between the way that FLOSS developers and researchers perceive bug finding and fixing, through interviews with 322 notable GitHub developers. In the interviews, we do not directly ask the developers about what troubleshooting tools are highly in-demand, but instead we ask them about what kinds of bugs causes them severe troubles, in order to enable us to precisely understand the problems they face. After the interviews, we discuss what kinds of high impact bugs are able to be solved or not by existing studies, in order to provide researchers with insights to come up with new solutions. This work is separate from general bug categorizations, and to the best of our knowledge, our work is the first bug categorization focusing on only high impact bugs. In this paper, we address the following research questions;

RQ1: What kinds of high impact bugs are mainly considered high impact by FLOSS developers?

RQ2: What kinds of high impact bugs do FLOSS developers encounter most frequently?

RQ3: What kinds of high impact bugs should be studied newly by the SE community in order to support FLOSS developers?

Our contributions to the study are as follows;

- We revealed symptoms and causes of bugs considered high impact by FLOSS developers, through interviews and manual inspections of bug reports.
- For FLOSS developers, we identified which kinds of high impact bugs are supported by previous studies.
- For researchers, we showed the area where FLOSS developers’ needs are still not fulfilled.

2 High Impact Bugs

Over the past two decades, the SE community have dedicated considerable efforts to help software developers to predict faults in modules, localize and repair faults in source code, and so on. Although the existing, traditional studies had not thoroughly considered the characteristics nor the impacts of bugs, in recent years they began to tackle with the impacts of bugs on users and the development process. In what follows, we introduce some existing studies on finding and fixing high impact bugs.

A Security bug [8] can cause serious problems which often impacts on uses of software products directly. Since Internet devices (e.g., smartphones) usage is increasing every year, security issues of software products are of interest to many people. In general, security bugs are fixed as soon as possible.

A Performance bug [22] is defined as “programming errors that causes significant performance degradation.” The performance degradation contains poor user experience, laggy application responsiveness, lower system throughput, and waste computational resources [18]. [22] showed that a performance bug needs more time to be fixed compared with a non-performance bug.

A Breakage bug [30] is a type of functional bug which is introduced into a product when the source code is modified to add new features or to fix existing

bugs. Though it is well-known as regression, a breakage bug mainly focuses on regression in functionalities. A breakage bug causes problems which make available functions in one version unusable after releasing newer versions.

A Blocking bug [7] is a bug that prevents other bugs from being fixed. It is often caused due to a dependency relationship among software components. Since a blocking bug inhibit developers from fixing other dependent bugs, it can highly impact on developers' task scheduling since a blocking bug takes more time to be fixed [7] (i.e. a developer would need more time to fix a blocking bug and other developers need to wait for being fixed to resolve the dependent bugs).

A Surprise bug [30] is a new concept of software bugs. It can disturb the workflow and/or task scheduling of developers since it appears at unexpected times (e.g., bugs detected in post-release) and locations (e.g., bugs found in files are rarely changed in pre-release). As a result of a case study of a proprietary, telephony system which has been developed for 30 years, [30] showed that the co-changed files and the amount of time between the latest pre-release date and the release date can be good indicators of predicting surprise bugs.

A Dormant bug [4] is also a new concept on software bugs and is defined as "a bug that was introduced in one version (e.g., Version 1.1) of a system, yet it is not reported until after the next immediate version (i.e., a bug is reported against Version 1.2 or later)." [4] showed that 33% of the reported bugs in Apache Software Foundation projects were dormant bugs and were fixed faster than non-dormant bugs.

3 Study Design

3.1 Overview

In this study, we e-mail and ask notable developers in GitHub [9] to answer a questionnaire about high impact bugs. After aggregating collected responses, we show the developers' demographic information (**Q1**), and the distribution of the bugs that are considered high impact (**Q2-1**). As the questionnaire includes an open question (**Q2-2**) to tell us actual bug reports which caused troubles in the past, we manually inspect the bug reports and categorize them by symptoms.

3.2 Participant Selection

In order to select notable developers to invite to our interview in this study, we use *contribution* which represents the amount of the developer's commit activity to GitHub repositories and can be calculated with GitHub API [10]. First, we make a list of all repositories in GitHub and calculate the total number of contributions for each repository. Note that we only calculate contributions for the most committed repositories if the repositories have the same name, since forked repositories partly (sometimes largely) include the same commits from original repositories and we need to avoid multiple counts for the same contributions by the same developer. Next, the total contributions of each developer is calculated

based on the selected repositories above, and we choose candidates who mark over 100 contributions. Finally, we sent e-mails to 22,228 candidate developers to ask them to participate in our interview.

3.3 Questionnaire

We prepared Google Forms for our interview which consisted of three questions to know the developers demography (**Q1**), one closed question to reveal a distribution of high impact bugs considered important by developers (**Q2-1**), and one open question to collect and further analyze actual reports on high impact bugs (**Q2-2**). The questionnaire has six more questions, but in this paper, we only focus on the five questions above due to the space limitations.

[Q1. Profile]

Q1-1 Your main project

Q1-2 Your experience with FLOSS development

Q1-3 Your motivation to participate in FLOSS development

[Q2. High impact bugs]

Q2-1 What kind of bug would be much more important to be fixed?

- a bug threatening systems' security (Security bug)
- a bug deteriorating system's performance (Performance bug)
- a bug blocking other bug fixes (Blocking Bug)
- a bug found in unexpected timing and location (Surprise bug)
- a bug introduced in older releases and found in a newer releases (Dormant bug)
- a bug introduced in a newer release and breaking functions of older releases (Breakage bug)
- others [free text]

Q2-2 Please tell us high impact bug(s) you encountered in the past.

3.4 Categorization of Bug Symptoms

Based on the responses of **Q2-2**, we collect actual bug reports from developers' projects and confirm the symptoms of the bugs, in order to discuss what techniques have been already proposed or that will be required to find and fix those high impact bugs. The first and second authors independently and manually inspect symptoms of actual high impact bugs and classify them by the card sort technique [23]. After the independent classification, the two authors discuss each classification result together and merge the results by mutual consent to make one classification. Here, the inspectors include one Ph.D. student (first author), who worked at a software company for two years as a full-time developer, and one professor who has been studying FLOSS development over ten years.

4 Interview and Classification Results

4.1 Developer Demography (Q1)

As we described earlier, we invited 22,228 developers to join our interview. During the two weeks interview period, we got responses from 322 developers. Table 1 shows the product domains where the developers participated. We can see “web application” is the most popular domain (7%) but it does not stand out from the others. The product domains spread across a wide area. We can assume that the results of our interviews reflect a wide range of situations across FLOSS development. Table 2 shows the developers’ experience with FLOSS development. The majority of the developers have over five years experiences. It is no surprise because we only invite active developers who have made at least over 100 commits to GitHub repositories. Table 3 shows developers’ motivations to FLOSS development. 59% (126 + 64) of the developers contribute to FLOSS projects as part of work.

Table 1. Product domains where GitHub developers join (Q1-1)

Domain	#	%	Domain	#	%	Domain	#	%
Web application	22	7%	Database	9	3%	Machine learning	5	2%
Development tool	19	6%	Network server	9	3%	UI	5	2%
Analysis	19	6%	Messaging tool	9	3%	Mobile app	5	2%
Language & compiler	17	5%	Education	8	2%	Desktop system	4	1%
OS	15	5%	Simulator	7	2%	Mail	4	1%
Graphic	14	4%	Finance	7	2%	Browser	3	1%
Game	13	4%	Resource monitoring	7	2%	Others	37	11%
Programming tool	12	4%	Image editor	6	2%	No answer	34	11%
Blog	11	3%	Network tool	6	2%			
Embedded OS	9	3%	Security tool	6	2%	Total	322	100

Table 2. Experience with FLOSS development (Q1-2)

Experience	Developers
More than five years	213
Three to five years	63
One to three years	45
Less than one year	1

Table 3. Motivation to participate in FLOSS development (Q1-3)

Motivation	Developers
Hobby	111
Work	126
Both	64
Other	21

Table 4. A distribution of high impact bugs in Q2-1

High impact bugs	#	%
Security bug	171	53%
Breakage bug	72	22%
Performance bug	20	6%
Blocking bug	16	5%
Dormant bug	12	4%
Surprise bug	7	2%
Others	24	7%

4.2 RQ1: What Kinds of High Impact Bugs Are Mainly Considered High Impact by FLOSS Developers?

In Q2-1, we asked the developers to select one of the six kinds of high impact bugs which are introduced in the previous section and have been well studied in the SE community. Table 4 shows the responses from the developers. We can see the FLOSS developers from GitHub attach greater importance on security bugs (53%) and breakage bugs (22%). It was unexpected for us that the other four bugs attract less attention from the FLOSS developers. It partly indicates the perception of gaps between researchers and FLOSS developers. Some researchers in the SE community might misunderstand FLOSS developers' actual needs.

Researchers in the SE community have been studying to help developers find and fix bugs especially in terms of impacts on users' satisfaction and during the development process (release) [13]. Although high impact bugs have been studied individually so far, it was not clear if FLOSS developers are mostly concerned with particular high impact bugs. From the result of Q2-1, we now answer RQ1 as follows.

Answer to RQ1: Researchers have been dedicating to provide a means to find and fix a variety of high impact bugs, but FLOSS developers mainly emphasize a focus on security and breakage bugs.

4.3 RQ2: What Kinds of High Impact Bugs Do FLOSS Developers Encounter Most Frequently?

In Q2-2, we asked the developers to describe the high impact bugs they have encountered in the past. Many of the developers described characteristics of high impact bugs in the free text format and also gave us direct links to actual bug reports which present symptoms discussed among developers and users.

Table 5 shows symptoms of bugs considered high impact by the respondents. In the table, we count multiple times if a developer described several high impact bugs. The percentage in the table is the ratio of developers' answers in each category, but the total percentage does not become 100% due to the above reason. As we described earlier, we manually inspected and categorized the information on high impact bugs by symptom. In what follows, we summarize the classification result.

We had 249 valid answers from 192 developers about symptoms of high impact bugs which actually get FLOSS developers in trouble in the past. In Table 5, the most common symptom was "unexpected processing" responded by 17% of developers (42 cases). As regards "unexpected processing", we could confirm less in common with bug reports. They ranged from different calculation results to unexpected rendering. The next most frequent was "sudden stop" responded by 16% of developers (39 cases). The corresponding bug reports shown by the developers suggested us that it often happened due to null pointer exception, run-time error exception, segmentation error, and overflow. Although the

Table 5. Symptoms described in actual bug reports

Category	Subcategory	Description	#	%	Ref
Behavior	Disable start	Users cannot install, compile or start an application	19	8%	[1, 27, 36]
	Never start function	A function never start once a user clicks a button	21	8%	[5, 30, 32, 34]
	Sudden stop	A program suddenly stops during running	39	16%	[2, 25, 26, 33]
	Unexpected processing	A program does not output or behave as developers expected	42	17%	[31, 32]
	Never finishing state	A process never finish (e.g, hang up and infinite loop)	5	2%	
	Unable to login	Users cannot login a system	4	2%	
	Others	Lack of items in display, wrong warnings, lower user experiences etc.	8	3%	
Effect	Lower performance	A program lowers performance (e.g, too large memory usage)	13	5%	[20–22, 35]
	Damage other systems	A program damages other systems (e.g, OS cannot boot)	5	2%	
	Others	Making a disk full etc.	3	1%	
Security	Vulnerability	Security defects allow an attack to cause an abnormal behavior	22	9%	[6, 17, 29]
	Unauthorized access	An impersonating account accesses to a server, service, or data	28	11%	[12, 16, 19]
	DDoS	Massive accesses from multiple terminals make a service unable	7	3%	
Data	Data loss	A program deletes data in a product (e.g., user data and database breakage)	12	5%	
	Incorrect data	A program produces incorrect or duplicated data	1	0%	
Development	Architecture change	It forces developers to change a architecture or core program in a product	3	1%	
	Reproduce	Developers cannot reproduce a reported bug	3	1%	
	Others	Blocking other bugs fixed etc.	4	2%	
Reputation	Compatibility	Compatibility is broken (e.g, API, hardware and OS)	7	3%	
	Execution env.	A product can not guarantee an execution environment	3	1%	

above two are related to “Behavior” of a program, the third and fourth most common symptoms were “Security” concerns such as “vulnerability” and “unauthorized access.” About “vulnerability,” the corresponding bug reports suggested the developers especially concerned with XSS and SQL injection attacks. The OpenSSL problem (i.e., Heartbleed) and the hidden way of leaking user data were included in bug reports about “unauthorized access.”

In RQ1, 53% of developers think that security is the biggest concern among high impact bugs in the previous studies. However, the result in RQ2 shows that the developers come across high impact bugs about Behaviour more often than the one about security. In fact, one developer said, “*Since the mentioned project is (mostly) a client-side javascript library, security problems aren’t common.*” Based on the results here, we answer RQ2 as follows.

Answer to RQ2: FLOSS developers most frequently encounter bugs relating to behaviors such as unexpected behaviors and sudden stops. Security bugs such as vulnerabilities and unauthorized accesses are also often encountered.

4.4 RQ3: What Kinds of High Impact Bugs Should Be Studied Newly by the SE Community in Order to Support FLOSS Developers?

The percentages in Table 5 are indicated by boldface if the corresponding symptoms account for about 80% of all the symptoms (i.e., the developers frequently encounter the symptoms with boldface.). For the majority of the symptoms, we surveyed existing studies which have tried to find and/or resolve the symptoms and showed references as “Ref” in Table 5.

The percentages of the symptoms in “Behavior” category are relatively high and these have been well-studied by the SE community [1, 2, 5, 25–27, 30–34, 36]. For instance, “never start function” is well studied *breakage bugs* [30] so-called regressions which disable existing functions due to additional changes to software products. Although in the paper we did not introduce this as a high impact bug, “unexpected processing” is well studied as a functionality bug or feature bug [32]. “disable start” and “sudden stop” are also studied as build bugs [36] and crash bugs [2] respectively.

As we confirmed “vulnerability” and “unauthorized access” achieved relatively high attention from the developers, *security bugs* are also considered high impact by researchers and have been well studied [6, 12, 16, 17, 19, 29, 35]. “Lower performance” in “Effect” category is also well studied [20–22, 35] as performance bugs. However, to the best of our knowledge, there is no study on “data loss” in “Data” category which is of relatively high concern to FLOSS developers (5%). For instance, a bug on “data loss” in “Data” category is observed when deleting data related to the operation under a condition. Other data loss bugs occurred due to executing the wrong processing of multi-transaction or by using variables not multi-threaded (e.g., HashMap in Java). In fact, for instance, loss of users’ data such as their pictures was recently reported in the update of Windows 10 [3]. We regard it is one of the perception gaps between FLOSS developers and SE researchers and should be should be studied, allowing us to address the issue. Based on the results here, we answer RQ3 as follows.

Answer to RQ3: Existing studies can cover FLOSS developers’ concerns about high impact bugs, but researchers still have space to further study other kinds of high impact bugs such as “data loss”.

5 Discussions

5.1 Is the Current Support for High Impact Bugs Enough? How Can We Help FLOSS Developers Fix Bugs?

Many studies focus on symptoms of bugs and observe their characteristics and impacts [4, 7, 22, 30]. We classified bugs based on the symptoms of high impact bugs included in the answers from the interviews. However, it is not enough to support fixing bugs because we can not fix them only by knowing the symptoms of high impact bugs. In this section, similar to the classification of the symptoms, we classify causes of high impact bugs obtained in the answers from the interviews. With the classification of the causes, we discuss how we can support fixing the high impact bugs in each category.

Table 6 shows the causes of bugs considered high impact by the respondents. Here, we had 182 valid answers including the root causes of high impact bugs (from 142 developers). In Table 6, the most common root cause was “insufficient processing” reported by 35% developers (49 cases). Furthermore, we broke down the 49 cases of “insufficient processing” and found that they consisted of 13 cases of “insufficient checks for inputs by users”, nine cases of “insufficient checks against malicious inputs”, five cases of “insufficient null checks such as arguments and return values”, four cases of “insufficient authority checks (e.g., database), and “others” (16 cases). The second and third most common root causes were “problem in 3rd party” and “change in 3rd party” respectively which relate to 3rd party libraries and systems.

Figure 1 depicts the relationship between symptoms and causes. The line width shows how symptoms and causes have strong relationships. For ease of reading graph, we only show the relationships that appeared more than two-times (which account for 52% of all the relationship). For further information, we provide the data and the figures showing all the relationships on our online appendix [24]. Looking into the causes of “unexpected processing”, the most frequent causes are “insufficient processing” (11 cases), and “problem in 3rd party” (5 cases), and “usage” (4 cases) are following. Furthermore, we investigated the detail of the bugs and found that nine of 11 cases were happened via regular usages and two cases were occurred by abnormal usage. These bugs should be found by unit test, or developers can utilize many studies (e.g, test case generation) to find them. “Sudden stop” is also caused by “insufficient processing” (11 cases). Eight of the 11 cases are caused by insufficient checks for inputs by users or by insufficient null checks, which are able to be applied by binary analysis [25].

The causes of both “disable start” and “never start function” are related on 3rd parties. The main causes of “disable start” are “changes in 3rd party”. We found two interesting answers indicating difficulty in finding the bugs are brought by “changes in 3rd party”. One developer said, *“It was just to update dependencies but finding bug was hard”*. Moreover, another developer said, *“Because there are too many packages in the repository, nobody can keep an eye on all the packages”*. Even though developers roughly know that the causes of the bugs are caused by changes in 3rd parties, it is difficult to specify which changes by third

Table 6. Root causes described in actual bug reports

Category	Subcategory	Description	#	%
Design	Usage	A program usage unanticipated by developers (e.g., no network)	12	7%
	Architecture	An architecture has an inappropriate algorithm etc.	2	1%
	Wrong processing	A program has incorrect processing	10	5%
	Superfluous processing	A program have superfluous processing	3	2%
	Insufficient processing	A program needs another processing (e.g., null checks)	39	21%
	Others	Incorrect encryption etc.	6	3%
Implement	Exception handling	A program throws a wrong Exception or cannot catch it	11	6%
	Condition	A program has wrong conditions (e.g., for and if)	7	4%
	Usage of variables	A program use wrong variables or data type	5	3%
	Method use	Developers incorrectly use methods in their product or 3rd party library	8	4%
	Memory management	A program cannot appropriately manage memory	9	5%
	Others	Wrong implementation of async, data or race competition etc.	6	3%
Operation and Maintenance	Problem in 3rd party	A program is affected on a problem in 3rd party library or systems	21	12%
	Changes in 3rd party	A program is affected on a change in 3rd party library or systems	13	7%
	Execution env.	Developers did not check if a program can run on some OS or shells	9	5%
	Change effects	A change in the module affects on other modules	5	3%
	Setting	Developers use a wrong setting (e.g, buffer size and database access authority)	5	3%
	Others	Incorrect document, refactoring, or operation etc.	11	6%

party products created the bug. Although Ma [15] et al. investigated common practice to fix cross-project correlated bugs in the GitHub scientific Python ecosystem, there are no approaches to specify the root causes of cross-project bugs.

“Vulnerability” and “unauthorized access” are mostly caused by “insufficient processing” (6 cases each). Among the subcategories of “insufficient processing”, the most common causes are derived from insufficient checks against malicious inputs (3 cases each). Fuzzing techniques [11, 14] are able to be used to find the insufficient checks.

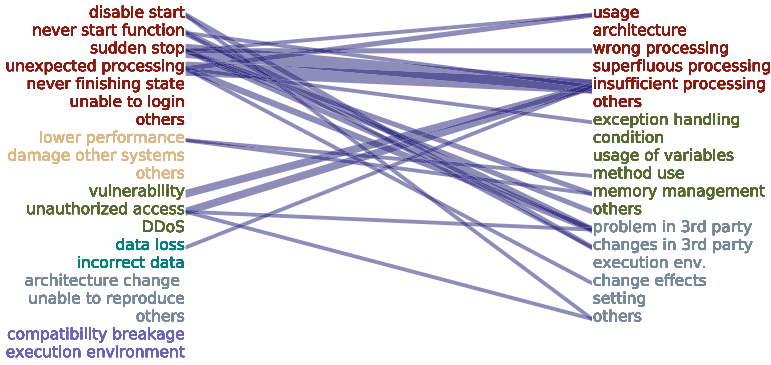


Fig. 1. Relationships between symptoms (left axis) and causes (right axis)

The cause of “data loss” is “insufficient processing”, which can be broken down into three different cases: “insufficient checks for input by users”, “insufficient checks against malicious inputs”, and “others”. Because of no similarities, we could not find any approaches to address “data loss” bugs, therefore it requires further work in the future.

5.2 Threats to Validity

Internal validity: The categorization of Table 5 and 6 may not be perfect. We have a good deal of knowledge about software, but we also recognize the limitations of our knowledge about specific domains. We also might bias in creating the category. **External validity:** Although the developer demography consists of developers working in a wide range of product domains, a judgment if a bug is high impact or not would depend on a product domain. **Construct validity:** To avoid bias in the developers responses, we asked them about high impact bugs without providing rigorous definitions of high impact bugs. Attitudes towards high impact bugs might be different among the developers.

6 Conclusion and Future Work

In this study, we interviewed 322 notable GitHub developers to reveal the perception gaps between FLOSS developers and researchers on bug finding and fixing. We manually inspected and classified actual bug reports which were presented and considered high impact by the developers. As a result, we concluded that security and breakage bugs are highly important for FLOSS developers. We also identified “data loss” bugs should be studied newly by the SE community to support FLOSS developers. Based on the bug report data presented by the developers in this study, we plan to investigate actual impacts of bugs on the size of source code change, resolution time, and so forth in the future.

Acknowledgment. We really appreciate the cooperation of developers in GitHub in completing our survey. This research is conducted as part of Grant-in-Aid for Japan Society for the Promotion of Science Research Fellow and Scientific Research (JP17J03330, JP17H00731, JP18K11243).

References

1. Abate, P., Di Cosmo, R., Gesbert, L., Le Fessant, F., Treinen, R., Zacchiroli, S.: Mining component repositories for installability issues. In: *Proceeding of the 12th Working Conference on Mining Software Repositories*, pp. 24–33 (2015)
2. An, L., Khomh, F., Guéhéneuc, Y.G.: An empirical study of crash-inducing commits in Mozilla Firefox. *Softw. Qual. J.* **26**(2), 553–584 (2018)
3. Ars Technica. <https://arstechnica.com/gadgets/2018/10/microsoft-suspends-distribution-of-latest-windows-10-update-over-data-loss-bug/>
4. Chen, T.H., Nagappan, M., Shihab, E., Hassan, A.E.: An empirical study of dormant bugs. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*, pp. 82–91 (2014)
5. Felsing, D., Grebing, S., Klebanov, V., Rümmer, P., Ulbrich, M.: Automating regression verification. In: *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, pp. 349–360 (2014)
6. Gao, F., Wang, L., Li, X.: BovInspector: automatic inspection and repair of buffer overflow vulnerabilities. In: *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pp. 786–791 (2016)
7. Garcia, H.V., Shihab, E.: Characterizing and predicting blocking bugs in open source projects categories and subject descriptors. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*, pp. 72–81 (2014)
8. Gegick, M., Rotella, P., Xie, T.: Identifying security bug reports via text mining: an industrial case study. In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, pp. 11–20 (2010)
9. GitHub. <https://github.com/>
10. GitHub API. <https://developer.github.com/v3/>
11. Jiang, B., Liu, Y., Chan, W.K.: ContractFuzzer: fuzzing smart contracts for vulnerability detection. In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pp. 259–269 (2018)
12. Kafali, O., Jones, J., Petruso, M., Williams, L., Singh, M.P.: How good is a security policy against real breaches? A HIPAA case study. In: *Proceedings of the 39th International Conference on Software Engineering*, pp. 530–540 (2017)
13. Kashiwa, Y., Yoshiyuki, H., Kukita, Y., Ohira, M.: A pilot study of diversity in high impact bugs. In: *Proceedings of the 30th International Conference on Software Maintenance and Evolution*, pp. 536–540 (2014)
14. LibFuzzer. <https://llvm.org/docs/LibFuzzer.html>
15. Ma, W., Chen, L., Zhang, X., Zhou, Y., Xu, B.: How do developers fix cross-project correlated bugs? A case study on the GitHub scientific python ecosystem. In: *Proceedings of IEEE/ACM 39th International Conference on Software Engineering*, pp. 381–392 (2017)
16. Mathis, B., Avdiienko, V., Soremekun, E.O., Bohme, M., Zeller, A.: Detecting information flow by mutating input data. In: *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, pp. 263–273 (2017)

17. Meng, N., Nagy, S., Yao, D., Zhuang, W., Argoty, G.A.: Secure coding practices in Java: challenges and vulnerabilities. In: Proceedings of the 40th International Conference on Software Engineering, pp. 372–383 (2018)
18. Molyneux, I.: The Art of Application Performance Testing: Help for Programmers and Quality Assurance, 1st edn. O’Reilly Media Inc., Newton (2009)
19. Near, J.P., Jackson, D.: Finding security bugs in web applications using a catalog of access control patterns. In: Proceedings of the 38th International Conference on Software Engineering, pp. 947–958 (2016)
20. Nguyen, T.H.D., Nagappan, M., Hassan, A.E., Nasser, M., Flora, P.: An industrial case study of automatically identifying performance regression-causes. In: Proceedings of the 11th Working Conference on Mining Software Repositories, pp. 232–241 (2014)
21. Nistor, A., Chang, P.C., Radoi, C., Lu, S.: CAMEL: detecting and fixing performance problems that have non-intrusive fixes. In: Proceedings of the 37th International Conference on Software Engineering, vol. 1, pp. 902–912 (2015)
22. Nistor, A., Jiang, T., Tan, L.: Discovering, reporting, and fixing performance bugs. In: Proceedings of the 10th Working Conference on Mining Software Repositories, pp. 237–246 (2013)
23. Nurmuliani, N., Zowghi, D., Williams, S.: Using card sorting technique to classify requirements change. In: Proceedings of 12th International Requirements Engineering Conference, pp. 224–232 (2014)
24. Online_Appendix. <https://github.com/YutaroKashiwa/OSS2019>
25. Pham, V.T., Ng, W.B., Rubinov, K., Roychoudhury, A.: Hercules: reproducing crashes in real-world application binaries. In: Proceedings of the 37th International Conference on Software Engineering, vol. 1, pp. 891–901 (2015)
26. Seo, H., Kim, S.: Predicting recurring crash stacks. In: Proceedings of the 27th International Conference on Automated Software Engineering, p. 180 (2012)
27. Seo, H., Sadowski, C., Elbaum, S., Aftandilian, E., Bowdidge, R.: Programmers’ build errors: a case study (at Google). In: Proceedings of the 36th International Conference on Software Engineering, pp. 724–734 (2014)
28. Shafiq, H., Arshad, Z.: Automated debugging and bug fixing solutions: a systematic literature review and classification, M.Sc thesis, Blekinge Institute of Technology (2014)
29. Shar, L.K., Beng Kuan Tan, H., Briand, L.C.: Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis. In: Proceedings of the 35th International Conference on Software Engineering, pp. 642–651 (2013)
30. Shihab, E., Mockus, A., Kamei, Y., Adams, B., Hassan, A.E.: High-impact defects: a study of breakage and surprise defects. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, pp. 300–310 (2011)
31. Sullivan, M., Chillarege, R.: Software defects and their impact on system availability—a study of field failures in operating systems. In: Proceedings of the Fault-Tolerant Computing: The Twenty-First International Symposium, pp. 2–9 (1991)
32. Tan, L., Liu, C., Li, Z., Wang, X., Zhou, Y., Zhai, C.: Bug characteristics in open source software. *Empir. Softw. Eng.* **19**(6), 1665–1705 (2014)
33. Tan, S.H., Dong, Z., Gao, X., Roychoudhury, A.: Repairing crashes in Android apps. In: Proceedings of the 40th International Conference on Software Engineering, pp. 187–198 (2018)

34. Tan, S.H., Roychoudhury, A.: Relifix: automated repair of software regressions. In: Proceedings of the 37th International Conference on Software Engineering, vol. 1, pp. 471–482 (2015)
35. Zaman, S., Adams, B., Hassan, A.E.: Security versus performance bugs: a case study on Firefox. In: Proceedings of the 8th Working Conference on Mining Software Repositories, pp. 93–102 (2011)
36. Zhao, X., Xia, X., Kochhar, P.S., Lo, D., Li, S.: An empirical study of bugs in build process. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing, pp. 1187–1189 (2014)

FLOSS Adoption



Fifteen Years of Open Source Software Evolution

Francis Bordeleau¹, Paulo Meirelles², and Alberto Sillitti³(✉)

¹ Ecole de Technologie Supérieure (ETS), Montreal, Canada

`francis.bordeleau@cmind.io`

² Federal University of Sao Paulo, Sao Paulo, Brazil

`paulo.meirelles@unifesp.br`

³ Innopolis University, Innopolis, Russian Federation

`a.sillitti@innopolis.ru`

Abstract. The Open Source Software (OSS) ecosystem and community has evolved enormously from the first edition of the OSS conference that took place in Genoa (Italy) in 2005. Such evolution happened in every aspect of OSS including research, technology, and business pushing its adoption to an unpredictable scale. Nowadays, it is almost impossible for people not using OSS in every interaction they have with technology. This fact is a tremendous success for OSS but such evolution and adoption has not always followed the intended path and some relevant deviations have occurred during such long journey.

This paper provide an overview of the evolution of OSS in the three mentioned areas (research, technology, and business) highlighting the main aspects and identifying the current trends that will be the basis for its future evolution.

Keywords: OSS evolution · OSS research · OSS technology · OSS business

1 Introduction

In an very famous article published in *The Wall Street Journal* in August 2011 titled “Why Software Is Eating The World” [1], Marc Andreessen provided a deep analysis of the IT industry and analyzed how value was moving out from hardware while it was increasing in software helping a new generation of companies and entrepreneurs to build new business at a fraction of the costs compared to the beginning of the century.

The identified trend is still in place and become even more complex with the development of synergies between hardware devices and software in many application areas (e.g., smart home assistants, autonomous vehicles, etc.) inspired by the Apple’s iOS ecosystem.

However, what the article of the 2011 was not discussing is a similar evolution that was already happening inside the software domain. As the commoditization

of hardware (also through virtualization and the introduction of cloud services such as Amazon AWS) reduced the costs for running the computing infrastructure, the extensive usage of quality OSS reduced the costs for running the basic software infrastructure (e.g., operating system, database, web server, application server, etc.), the development environments (e.g., IDEs), and applications (e.g., office automation tools, browser, data analysis tools, etc.). This trend was and still is a powerful support to the creation of new startup companies that require a fraction of the budget compared to the early 2000s. However, such effect has not only positive aspects but also negative ones when we consider the entire software ecosystem enabling *free riders* taking advantage of the OSS community without giving back.

Moreover, with the development of IoT, Smart Cities, etc., the value is still moving upwards leaving even the software domain towards the data that actually enable a whole new category of applications that are software-based but not implementable with just software (e.g., applications based on machine learning algorithms).

The paper is organized as follows: Sect. 2 and subsections analyze the OSS evolution according to the different perspectives identified: research (Sect. 2.1), technology (Sect. 2.2), and business (Sect. 2.3); Sect. 3 discusses the current trends in the above mentioned areas; finally, Sect. 4 draws the conclusions and introduces possible future work.

2 Open Source Software Evolution

OSS has been evolved in the last 15 years from several perspectives that are partially related to each other. We have decided to focus on the following ones:

1. **Research:** how the research has evolved considering the problems investigated, the types of activities performed, and papers published.
2. **Technological:** how the technology has changed considering the popularity of the projects, how people contribute, and how the technology has been adopted by users.
3. **Business:** how business models have changed, which ones emerged and which ones has been dismissed.

Such perspectives are not the only ones that can be identified but they are just a starting point for a deeper investigation.

2.1 The Research Perspective

The research environment has changed dramatically in the last 15 years from several points of view:

- **Publication venues:** the number of papers published in top venues dealing with OSS was very limited. As an example, just look at the proceedings of the *International Conference of Software Engineering (ICSE)* in 2005 [11]

and 2019 [12] for comparison. This could be related to the fact that results of research activities sponsored by companies were rarely released to the community and public sponsored research grants were not pushing researchers to release their results other than through publications. Nowadays, most of the papers published in top venues deals with OSS since the research behind them is using OSS as a source of data (e.g., in terms of code, developers activity, etc.), use OSS as a tool for extracting and/or analyzing data (e.g., as in machine learning tools for building models), release tools as OSS, etc. Moreover, most companies have an OSS strategy that allows or even force researchers to release OSS and public sponsored research grants often require the release of results and tools as OSS.

- **Investigated topics:** the investigated topics were often related to the adoption of OSS in different contexts (e.g., in the public administrations), the migration from proprietary solutions to OSS [13–15], the quality of OSS from the point of view of the code produced and the development process [8–10], licensing aspects. Nowadays, the focus is more oriented to areas in which OSS enable research activities that are not possible in other environments such as the investigations related to the analysis of large code repositories and their evolution [2–4], the analysis of communities [5, 7], resource usage [6], security issues, etc. As a general trend, the focus now is more on the data and its analysis with machine learning approaches present almost everywhere.
- **Specialized venues:** the OSS conference series has evolved since OSS-related papers are now published everywhere and not in specialized venues anymore. OSS is now a first class citizen in any research-oriented conference or journal and not the exception. The current trend requires authors to release their code and datasets to enable other researchers to replicate their findings and justify if this is not possible for any reason. This is a fundamental paradigm shift that is not completed yet but it has started and it is spreading across a number of top quality conferences and journals.
- **Open access:** it is now a common practice for authors to pay for allowing people to access their papers for free to increase the dissemination of their work. This practice is often supported by public research grants to assure the maximum exposition of the research results they have sponsored. Even if the approach is controversial for many reasons (e.g., conflicts of interests of publishers that may lead to lower quality), it has the advantage of making research outcomes available to everybody without any paywall.

2.2 The Technological Perspective

The technology has evolved in many different areas but some have been affected more in depth:

- **Software technologies:** OSS is part of almost any software product due to the popularity of powerful and high quality libraries (e.g., the ones from the Apache Foundation, the ones released by major IT companies such as Google, Facebook, IBM, etc.) published according to licenses that allows their

integration in both open and closed products (e.g., BSD, MIT, LGPL, etc.). Moreover, many basic components of the overall infrastructure are powered by OSS (e.g., databases, web servers, etc.).

- **Hardware technologies:** also any product that include a software component nearly always include OSS. TVs, cars, IoT devices, etc. often include stripped down versions of open operating systems, libraries, web servers to provide easy to use user interfaces, etc.
- **Software-as-a-Service (SaaS) technologies:** with the development of this approach to the delivery of software systems, a new set of problems have been identified for OSS. Open licenses were not ready to cope with this novel approach for releasing software and many companies took advantage of OSS breaking the basic ideas behind openness. As an example, the GPL license requires that the entire code of a product that use a GPL component needs to be released as GPL if it is released. With SaaS, the code is never released but only used through the Internet. This enabled many companies (including giants like Google, Facebook, Amazon, etc.) to build closed services taking advantage of the open source communities without giving anything back violating the spirit of the GPL. For this reason, a new version of the AGPL license have been developed. This license forces developers to release their code to the community even if the developed product is not released but just delivered as SaaS.
- **Mobile technologies:** with the introduction of the Android operating system, OSS has heavily penetrated the consumer environment becoming the most used operating system. Other open mobile operating systems have been proposed such as Tizen or Sailfish OS but they have a very small market share. Moreover, the popularity of OSS powered hardware platforms such as Arduino and Raspberry PI have made OSS the de facto standard for the development of prototypes, custom IoT projects, etc.
- **Cloud and big data technologies:** many technologies for running the cloud infrastructure has been released as OSS due to changes in the business models (see Sect. 2.3). A similar approach has been adopted in big data technologies where OSS is the major player with contributions provided by many kinds of companies that cooperate in the development of the technology (e.g., Apache Hadoop ecosystem) but compete in providing solutions to the customers.
- **Machine learning technologies:** all the major IT companies has released their tools as OSS (e.g., Google with Tensorflow, Facebook with PyTorch, Microsoft with CNTK, etc.) offering developers their cloud services for running them. Moreover, such tools are not very useful without a wide amount of data that is needed to build the models able to be used in actual applications.

2.3 The Business Perspective

Business models have evolved from focusing on selling software to focusing on providing on-demand services based on OSS. The business of software has changed in many aspects including the following:

- **Software commoditization:** the value in the software business is continuously moving upwards transforming the basic infrastructure and applications into a commodity. Nowadays, leading applications include a relevant set of features powered by machine learning algorithms that are based on the analysis of a huge amount of data that is not available to the community while software is. For this reason, the business models are changing moving towards the data that will be able to provide competitive advantages to companies that own them. This is a relevant problem for the open source communities that currently do not have the ability to collect and exploit such amount of data preventing the creation of cutting edge applications.
- **Value of data:** machine learning algorithms require a huge amount of data for the definition of reliable models. Such data is continuously collected and enhanced by large IT companies that base their products on them. Data are collected to improve the models that make the products more useful and reliable from which additional data are collected. For this reason data complement software enabling the creation of a new generation of products.
- **Introduction of new OSS licenses:** the introduction of the AGPL license was required to keep the spirit of the GPL license valid also in a world where software is not delivered anymore but offered as a service. This change in the software delivery paradigm requires an adaptation to the licensing approaches as it happens with the increasing value of data.

3 Current Trends

According to the three perspectives analyzed in Sect. 2, it is clear that OSS has penetrated all the business and research environments helping in shifting the focus from the basic software to higher levels that provide more value to the final customers. Moreover, it is clear that pure software is not enough anymore since data is the new king and applications cannot be tuned and work as customers expect without analyzing a massive amount of data extracting useful information that can be used to enhance the user experience.

Open source communities have to deal with these two new aspects and define a proper way to manage them. About SaaS, the community has introduced the AGPL that is able to keep the spirit of openness also in such environments. However, currently, there are no well established approaches to deal with data and assure that they stay open for the usage of the entire community. This is a new challenge that needs to be addressed as soon as possible to keep value in OSS.

4 Conclusions and Future Work

The paper presents an initial qualitative investigation of the evolution of the OSS in the last 15 years under three perspectives: research, technological, and business. The main challenges that OSS have to deal with are related to the

delivery of SaaS and managing the value of data that is the key enabler for the development of a new generation of applications.

A more detailed analysis is needed through the collection and the analysis of quantitative data to provide a more complete overview of the identified trends and how they are connected to each other.

References

1. Andreessen, M.: Why software is eating the world. *Wall Street J.* (2011). <https://a16z.com/2011/08/20/why-software-is-eating-the-world/>
2. Ciancarini, P., Poggi, F., Rossi, D., Sillitti, A.: Improving bug predictions in multicore cyber-physical systems. In: Ciancarini, P., Sillitti, A., Succi, G., Messina, A. (eds.) *Proceedings of 4th International Conference in Software Engineering for Defence Applications. AISC*, vol. 422, pp. 287–295. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-27896-4_24
3. Ciancarini, P., Sillitti, A.: A model for predicting bug fixes in open source operating systems: an empirical study. In: *28th International Conference on Software Engineering and Knowledge Engineering (SEKE 2016)*, Redwood City, San Francisco Bay, CA, USA (2016)
4. Ciancarini, P., Poggi, F., Rossi, D., Sillitti, A.: Analyzing and predicting concurrency bugs in open source systems. In: *30th International Joint Conference on Neural Networks (IJCNN 2017)*, Anchorage, AK, USA (2017)
5. Di Bella, E., Sillitti, A., Succi, G.: A multivariate classification of open source developers. *Inf. Sci.* **221**, 72–83 (2013)
6. Georgiev, A., Sillitti, A., Succi, G.: Open source mobile virtual machines: an energy assessment of Dalvik vs. ART. In: *10th International Conference on Open Source Systems (OSS 2014)*, San Jose, Costa Rica (2014)
7. Jermakovics, A., Sillitti, A., Succi, G.: Exploring collaboration networks in open-source projects. In: Petrinja, E., Succi, G., El Ioini, N., Sillitti, A. (eds.) *OSS 2013. IAICT*, vol. 404, pp. 97–108. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38928-3_7
8. Petrinja, E., Sillitti, A., Succi, G.: Overview on trust in large FLOSS communities. In: Russo, B., Damiani, E., Hissam, S., Lundell, B., Succi, G. (eds.) *OSS 2008. ITIFIP*, vol. 275, pp. 47–56. Springer, Boston, MA (2008). https://doi.org/10.1007/978-0-387-09684-1_4
9. Petrinja, E., Nambakam, R., Sillitti, A.: Introducing the open maturity model. In: *2nd Emerging Trends in FLOSS Research and Development Workshop at ICSE 2009*, Vancouver, BC, Canada (2009)
10. Petrinja, E., Sillitti, A., Succi, G.: Comparing OpenBRR, QSOS, and OMM assessment models. In: Ågerfalk, P., Boldyreff, C., González-Barahona, J.M., Madey, G.R., Noll, J. (eds.) *OSS 2010. IAICT*, vol. 319, pp. 224–238. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13244-5_18
11. Roman, G.-C., Griswold, W., Nuseibeh, B.: *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*. ACM (2005)
12. Atlee, J., Bultan, T., Whittle, J.: *Proceedings of the 41st International Conference on Software Engineering (ICSE 2019)*. ACM (2019)
13. Rossi, B., Scotto, M., Sillitti, A., Succi, G.: Criteria for the non invasive transition to OpenOffice. In: *1st International Conference on Open Source Systems (OSS 2005)*, Genoa, Italy (2005)

14. Rossi, B., Scotto, M., Sillitti, A., Succi, G.: An empirical study on the migration to OpenOffice.org in a public administration. *Int. J. Inf. Technol. Web Eng.* **1**(3), 64–80 (2006)
15. Russo, B., Braghin, C., Gasperi, P., Sillitti, A., Succi, G.: Defining the total cost of ownership for the transition to open source systems. In: 1st International Conference on Open Source Systems (OSS 2005), Genoa, Italy (2005)



Open Source Software Community Inclusion Initiatives to Support Women Participation

Vandana Singh^(✉) and William Brandon

School of Information Sciences, University of Tennessee-Knoxville,
1345 Circle Park Drive, Suite 451, Knoxville, TN 37996-0332, USA
vandana@utk.edu, wbrandol@vols.utk.edu

Abstract. This paper focuses on the inclusion initiatives of Open Source Software (OSS) Communities to support women who participate in their online communities. In recent years, media and research has highlighted the negative experiences of women in OSS and we believe that could be detrimental to the women of OSS. Therefore, in this research, we built upon the research that demonstrates the value of Codes of Conduct for minorities in an online community. Additionally, we focus on women only spaces in OSS, because past research on women and IT shows that women perform better when they can build connections and mentoring networks with other women. We investigated 355 OSS websites for presence of women only spaces and searched for, collected and analyzed the Codes of Conduct on the websites of these OSS. Qualitative content analysis of the websites show that only 12 out of 355 websites have women only sections. Less than ten percent (28) of the analyzed websites had a code of conduct.

Keywords: Open Source Software · Gender and IT · Women in open source · Code of conduct · Women only spaces

1 Introduction

Open Source Software (OSS) communities depend on the participation and contribution of voluntary members to ensure sustainable growth and day-to-day management. In more than a decade, OSS communities have shown no growth in the percentages of women participating in these communities [5]. Women form more than half of the workforce in the United States, but the participation from women in Information Technology is not even thirty percent [13]. In addition, when we zoom in deeper into OSS communities; we find that the percentage of women in OSS communities is a measly three to five percent [15] or even one percent according to some sources [OS4W.org]. It behooves OSS communities to understand and address the issues related to the under representation and mistreatment of women. In order to encourage more women to participate in OSS communities, it is important to take measures to demonstrate that the communities are inclusive, welcoming and supportive to the women who participate and contribute.

Anecdotal evidence about the negative experiences of women in the OSS communities has been in the spotlight via media and on the internet [6, 8, 17] and highly

controversial empirical research [18] that show that the contributions of women receive inferior treatment than the contributions of men. It is important for OSS communities to demonstrate that they take efforts to regulate the behavior of their community and that they are committed to providing an inclusive collaborative-shared environment where all the people and their views are welcome. The inclusion initiatives of OSS communities should be visible to the newcomers and should be available to community members who can use these to call out unwanted behavior.

Codes of conduct and women only spaces have shown to facilitate women engagement and retention in online communities. These safe spaces also allow women to be engaged and be creative [11]. In past research, we surveyed and interviewed women from across the globe to document their experiences of participating in OSS communities. Most of the surveyed and interviewed women indicated that along with mentoring, codes of conduct and women only spaces played an important role in their experiences within the open source communities. Other research had similar findings [2] regarding the value of Codes of conduct and therefore, this research aims to investigate the presence of Codes of Conduct in OSS communities and find out if any of these codes of conduct specially address issues or gender equality. We are also interested in finding out what types of women only spaces exist in OSS communities and the purpose and the scope of activities for these women only spaces. Cumulatively, these seem to create a visibly welcoming, inclusive and safe collaborative environment for women of OSS.

In the following sections of this paper, we will present the relevant past research on this topic, the research methods that we used to approach the aims of this study, the results from the research and a discussion of the results, limitations and future research areas.

2 Literature Review

Open source software communities are often seen as political communities, and the members are often viewed as ‘citizens.’ Carillo, Huff, and Chawner [1] address this aspect, noting how important the proper ‘socialization’ of new members is to both their performance of particular tasks and their participation in maintaining the community as a whole, the latter of which they refer to as ‘citizenship behaviors.’ When it comes to ‘socialization,’ the authors note that ‘codes of conduct’ are one important means of community maintenance (p. 338). Coelho and Valente [4] discuss whether lacking a ‘code of conduct’ is one of the reasons open source projects fail when they do. Although they do conclude that the adoption of a code of conduct is statistically negligible when it comes to differentiating the failed projects from the most popular ones in their sample set, they note that not one of the failed projects had a code of conduct. Neither did any included in the ‘least popular’ category. Thirteen percent of the most popular projects, on the other hand, implemented codes of conduct, and at least some of the developers of the failed projects considered cited ‘conflict among developers’ as a reason for failure.

Online communities including OSS groups often rely on ‘codes of conduct’ to regulate behavior among members. Many scholars have addressed codes of conduct in

passing as they explored various aspects of OSS and other online communities such as how they develop [2, 9, 10], how these ‘collaborative’ projects work in general (e.g. Scacchi, Grundy, Hoek, and Whitehead [16]) and the role of a ‘leader’ or ‘manager’ in such a community [2, 12, 16]. Others studied, the values that these communities embody [20], what other types of organizations can learn from them e.g. [14], and how integrating women in software development leads to a more balanced “community smell” [3]. Recently, however, scholars have become interested in the code of conduct as an object of study in its own right. What kinds of things does it include? Tourani, Adams, and Serebrenik [19] offer a sustained discussion of online communities’ ‘codes of conduct’ in the literature to date. The authors investigate a number of different codes of conduct through both web searches for the codes themselves and interviews with some members of the communities who adopt them. The questions they attempt to answer with this empirical research concern the kinds and number of codes there are, what they say, and how they are used (pp. 24–25).

In this research, we built upon the research on Codes of Conduct by Tourani, et al. [19] and started a new inquiry into the women only spaces of OSS to explore the OSS communities and their inclusion initiatives for women.

3 Research Methods

Research Objective: The main objective of this research is to investigate the inclusion initiatives of OSS communities in the form of women only spaces and codes of conduct that are available on the websites of OSS.

Research Questions

1. How many OSS communities have women only spaces on their website?
2. What is the nature and scope of these women only spaces?
3. How many OSS communities have Codes of Conduct on their website? What are the common and differentiating elements in these codes of conduct?
4. How do the Codes of Conduct demonstrate support for women in OSS communities?

3.1 Data Collection and Analysis

For this exploratory qualitative research, we used the list of OSS available from Wikipedia at https://en.wikipedia.org/wiki/List_of_free_and_open-source_software_packages. The list was collected on March 1, 2018. On that date, there were 355 OSS listed on that Wikipedia entry and included 22 categories of open source software. The list is a good starting point of representative open source software and provided a functional sample of popular and not so popular OSS.

A spreadsheet was created with the 355 software and the corresponding website for each of the software. Each website was searched for women only spaces and Codes of Conduct. We use the following keywords to search the websites with a search feature:

Women, Female, Codes of Conduct, Guidelines and Values. The websites that did not have a search feature for the overall website were examined by reviewing all the individual sections of the website, these sections included, Blogs, Forums, Mailing Lists, IRC, Community FAQs, About Us, News, Announcements, Wiki, GitHub Link and Support section. Very few websites had women only spaces and/or Codes of Conduct. All the Codes of Conduct that were available on these sites were collected and stored in the spreadsheet for further analysis. Notes were created for women only spaces and a URL of the women only space was recorded in the spreadsheet.

For examining the women only spaces on the OSS websites, we investigated the types and the purpose of these spaces. We reviewed the content on these spaces to understand their objectives and activities.

For analyzing the codes of conduct, we built upon the five components developed by Tourani et al. [19] by adding three new components specific to our research objectives. The three new components that we added are, name used to refer to the code of conduct; the intended audience of the code of conduct and the explicit mention of specific groups (women). The five components from the Tourani study are purpose; honorable behavior; unacceptable behavior; enforcement and scope. We used these collective eight categories to analyze the elements of all the codes of conducts for presence of these elements. Then we qualitatively analyzed the text for the eight elements using NVIVO software to develop common themes across the elements of the Codes of Conduct. This gave us a deeper understanding of the focus and intent of each of the eight elements of the Codes of Conduct.

4 Results

In this section, we present results starting with the inquiry into the women only spaces, the purpose of these spaces and the activities of these spaces. In the next sub-section, we discuss the results about the Codes of Conduct. We present results about the number of codes found, the names of these codes and an analysis of the key elements of these codes.

4.1 Women Only Spaces in Open Source Software Communities

Out of the 355 OSS websites listed on the Wikipedia page (https://en.wikipedia.org/wiki/List_of_free_and_open-source_software_packages), only twelve websites had women only spaces or they provided a link to an external website that was for women using that software. Some examples of the external websites are Facebook groups and local chapters of meet up groups. Table 1 presents a list of the websites that have women only spaces and the names and URL of these spaces.

Table 1. OSS websites with women only spaces.

Software package	Name of the space	URL
ArchLinux	Arch Linux for Women	http://archwomen.org
Bitcoin	Women in Bitcoin Madchenabend in Berlin	https://www.facebook.com/womeninbitcoin/
BonitaSoft	Blog Post about Community efforts for encouraging women	https://community.bonitasoft.com/behind-scenes-bonita-21-27-feb-2011
Debian	Debian Women	https://www.debian.org/women
Drupal	Women in Drupal	http://www.womenindrupal.org/
Fedora	Fedora Women	http://fedoraproject.org/wiki/Women
FreeNX	IRC Channel for Women	https://archwomen.org/wiki/aw-org:irc
GNOME	GNOME Women	https://wiki.gnome.org/GnomeWomen http://gnome.org/opw/
KDE	IRC Channel for Women	https://userbase.kde.org/IRC_Channels
Mozilla	WoMoz	http://www.womoz.org/blog/
PHP	PHP Women	http://phpwomen.org/
Ubuntu	Ubuntu Women Project	https://wiki.ubuntu-women.org/

4.2 Purposes of Women Only Spaces on OSS Communities

The different types of women only spaces that we found on OSS communities to support women in participating in the community range from completely dedicated websites for women, to women only IRC Channels, dedicated blogs, collection of resources for women in a blog post, dedicated Facebook pages and local meet-up groups. The purposes explicitly reported on each of these spaces are presented in Table 2.

Table 2. Purposes of the women only spaces on OSS websites

Name of the space	Purpose of the space
Arch Linux for Women	Helping more women become involved
Women in Bitcoin Madchenabend in Berlin	Networking and promoting women in bitcoin
Bonita Soft	Recruiting more women developers
Debian Women	Engaging and promoting women
Women in Drupal	To offer women only space, promote women
Fedora Women	Foster involvement of women
IRC Channel for Women	Talk about women issues, etc.
GNOME Women	Female only space, encouraging women of GNOME
KDE IRC Channel	Women only chatting space
WoMoz Blog	Dedicated to women
PHP Women	Online community, events and mentoring for women
Ubuntu Women Project	Fostering women contributions through mentoring and inspiration

Some of the women only spaces include blog posts and describe the initiatives that the community is taking to support women. This type of posts are good minimal investment ways to make the community efforts visible to newcomers. Bonita Soft, for example, presented the statistics related to women participating in their community and the growth in that participation. The Debian women community explains the roles in which women can start contributing and offer a safe place to ask questions. Drupal women site expresses a sense of community by creating a place for sponsoring women, solving issues that women face in Drupal or outside and have fun and celebrate life together. Fedora women page was very clear in being inclusive for women, trans and genderqueer people. Fedora Women also does a great job at highlighting the existing women in the community. The IRC channels support women by having technical channel and an off topic channel, so that women can feel free to connect with other women about “off topic” concerns. The Arch Linux IRC channel for women was interesting for multiple things, including explicit messages about the unacceptable behavior including “If you make us feel uneasy or “creeped out”. This is both not quantifiable and not negotiable. It is a judgment call and entirely at the discretion of the Op requesting your exit.” In addition “Looking for dating advice. This is not a dating service. This is also not a lonely hearts channel. We like sex and relationships fine. But that’s not what we’re here for.” And “Discussion of self harm.” This channel also listed resources for people feeling depressed and/or suicidal.

4.3 Codes of Conduct

Out of the 355 OSS packages listed on the Wikipedia page, only 28 websites had Codes of Conduct or other similar guidelines for regulating the behavior of participants in the communities. After this observation, we refer to the general name as “community rules” and reserve the use of “Code of Conduct” for the community rules that are called “Codes of Conduct”. We collected these community rules by searching each of the website using the keywords “Code of Conduct”, “Values” and “Guidelines” or by browsing all the sections of the website. Interestingly, not all the communities used the specific term “Code of Conduct” even when they had some rules for the community. The different names under which community rules were found varied significantly and included terminology such as guidelines, principles, rules, netiquette, etc.

Out of the 28 websites, 14 websites used the term Code of Conduct, three websites used Guidelines and two websites used Principles to refer to their community rules. The rest of the communities used the following one time occurring terms when referring to their community rules - Community guidelines, forum guidelines, fundamental freedoms and values, ListEtiquette, Netiquette guidelines, Rules, Values, and Goals. Within the code of conducts, there were sections such as Board rules, Chatiquette, common code, common etiquette rules, core values, event policy, forum code, mailing list code, Policy (Welcoming Policy, Anti-harassment policy, and complaint procedure), Statutes, Values and Goals, IRC rules and “Principles, guidelines and actions”.

Community rules were collected from the following 28 websites: 3DSlicer, Apache, Apereo, Asterisk, Chromium, CiviCRM, Compiere ElasticSearch, ERPNext, Evergreen, IRC, Fiji, FreeNX, FreePBX, GIMP, HandBrake, Inkscape, ITK, KitenK-MyMoney, LedgerSMB, Moodle, Natron, OpenAFS, OpenBravo, Opencog, OpenSSL, Pencil2D, PM and Tryton.

4.4 Key Elements of the Community Rules

In order to identify the common and differentiating elements in the 28 community rules, we conducted content analysis and coded each one for common elements. We used the five common elements as outlined by Tourani et al. [19] they derived these elements from an analysis of seven Codes of Conduct, which they found were basis of a large number of codes of conduct in OSS communities. We used these five elements as an initial starting point, added more element categories, and customized the definition of the eight elements according to our data set. We analyzed the 28 community rules for the following eight elements.

1. Terms Used – We added this new category because our searches on the website were not limited to “Codes of Conduct” keyword, but included any initiatives that were about community regulation and laid guidelines for the behavior of members of the community. This category allowed us to see the diverse range of terminology that the communities use to refer to their rules. It is important to know about the different ways in which communities express their approach and acknowledging this allows us to do a more comprehensive inquiry into inclusion initiatives.
2. Length of the rule – This was also a new element category that we added in order to record the varying lengths of the rules. The length of the rule can be an influencing factor in the usage of the rule. If it is too long, if it is ever read completely and if it is not read, then does it serve any purpose? The purpose for this category element is to guide understanding what would be good length for a code of conduct.
3. Purpose of the community rule – This element category is defined as the reason for the community rule to exist and the intent of this code for the community. What is it that the community wants to achieve and what do they focus on in order to achieve that goal or intent.
4. Honorable Behavior Examples – This element category is defined as the section of community rules that expresses what type of behavior is expected from the community members. These are positive examples of how the community expects members to behave.
5. Unacceptable Behavior Examples – This element category is defined as the section of the community rules that describes the behavior that is negative, discouraged and is to be avoided in the community
6. Enforcement – this elements category is defined as the section of the community rules that outlines consequences of not following the community rules, the actions that the community will take if the community rules are violated and it includes clear articulation of what a member of the community should do if they witness violation of community rules.

7. Scope of the code – this element category is to record the applicability of the code of conduct – if it is application to a section of the website (such as forums), or if it is for the entire community or if it is for online and offline community interactions and if it included the intended audience for the community rules.
8. Specific mention of minority groups/gender – Since we are specifically interested in the inclusion activities of the communities per their community rules, we added an element category to record the community rules that specifically include gender and generally any minority groups.

Table 3. The elements of community rules

CoC Element	No. of Instances out of 28
Terms Used	28/28 – 11 different terms were used in 28 examples
Length of the Code	Range of Length – 146 words to 7080 words
Purpose	28/28
Honorable Behavior Examples	27/28
Unacceptable Behavior Examples	25/28
Enforcement	28/28
Scope	28/28
Gender/Minority	13/28

As presented in Table 3, all the 28 community rules that were analyzed had a term for the rules, a purpose of the rules, an articulation of the scope of the rules and enforcement of the rules, in case of a violation. Most of the codes also included examples of honorable and unacceptable behavior, though not all. Only 13 (less than half) of the community rules that were analyzed had an explicit mention of gender and/or minority. Even though, the purpose of all the communities reflected the goal of an inclusive community, they all do not address gender or minorities in the language of the code.

4.5 Main Themes for Each Key Element of Community Rules

In depth, qualitative analysis of each of the element category gave insights into the approach of the respective OSS community. The *length of the rules of the community* also varied greatly, with OpenSSL code being the shortest in length with 146 words and the ITK code with more than 7000 words. The explicit *purpose* of most of these codes of conduct can be summarized as that of creating a shared collaborative environment, ensuring good community health, creating a positive community vibe, sharing and ensuring a good safe community environment. They express their goal to achieve their purpose by focusing on activities such as conversations, collaboration, welcoming, getting involved, leveraging diverse perspectives, willingness to learn and having a good time. Interestingly, many communities specifically outline the goal to be happy, stress free, friendly and helpful in this productive learning environment. The *honorable behavior examples* include two types of behavior; one is about how each community

member should behave with others in the community e.g. being open, considerate, polite, friendly, respectful, kind, empathetic and taking responsibility of their own actions and speech. In addition, the second type of examples include the responsibility of each community member to support other community members by flagging bad behavior and engaging when others are being disrespected. One code of conduct specifically discussed resolving disagreements constructively and asking for help when unsure what to do. In the *unacceptable behavior examples* elements category, we found a range of approaches being used by the OSS communities, including not mentioning any unacceptable behavior to giving very specific definitions and examples of unacceptable behavior. Some common themes that most of these examples included are harassment, verbal abuse, sharing private information about others, spamming, being rude or offensive, exclusionary behavior, trolling and personal attacks. Some Codes of conduct use this section of the code to condemn any acts of harassment related to gender or otherwise. A standard template language was often used to explain these unacceptable behaviors; this language is from a standard template and was found across multiple websites. E.g. “Examples of harassment: Verbal taunting, racial and ethnic slurs, comments that are degrading or unwelcome regarding a person’s nationality, origin, race, color, religion, gender, sexual orientation, age, body disability or appearance”. Specific examples against sexual harassment were also found such as; “sexual harassment of any kind, defined as unwelcome sexual advances, verbal sexual innuendos, suggestive comments, jokes of a sexual nature, requests for sexual favors, distribution or display of sexually suggestive graphic materials, and other verbal or physical conduct of an unwanted sexual or intimidating nature and contributing to an unwelcoming environment.” In the *enforcement* category, we found examples that describe how the community will enforce its code, how the community members can ensure that the code is followed and who should be contacted to inform about the violation of the code. Some of the consequences of violation outlined in the code of conducts are removal of content, verbal warning, public censure of the member and expulsion from the community. Removal of offensive content or illegal content is often the first act after a report is made. In most codes, this section also explains the process of registering a complaint and the steps taken by the community. Only one community mentions legal actions against community members if they post illegal content. Community moderators contact information is provided in most of the codes and specific names were provided in some instances. In the *scope* element category, all the codes of conduct defined their community and distinguished the spaces based on being online or offline. Online included forum, mailing list, wiki, web site, code repository, IRC channels, and private correspondence and public meetings. In offline events, it included meetups, hackathons, events, conferences and public meetings. The *intended audience* for all the community was specified as the users of the spaces and members who belonged to the community. In the *specific mention of gender and/or minority* category, we found examples of template-based language that was repeated in less than half of the total codes of conduct. A common excerpt for this category is, “We are committed to making participation in this project a harassment-free experience for

everyone, regardless of level of experience, gender, gender identity and expression, sexual orientation, disability, personal appearance, body size, race, ethnicity, age, religion, or nationality.”

5 Discussions and Future Research

The results show that different types of women only spaces are present in very few OSS communities. These spaces are considered “safe” spaces for women and are focused on mentoring, supporting, helping, guiding, advancing women in these communities. The objectives of these spaces are very much in line with the research that shows that when women feel safe, they are able to participate, engage and create better [11]. Some of the spaces we studied also had very specific information about sexual harassment, self-harm issues and “no defined topic” areas. These “no defined topic” areas are for any discussion that the women would like to have with other women and was not restricted to OSS or technology. We believe the concept of peer parity (when an individual can identify with at least one other member when interacting in a community) [7] would be valuable in OSS communities to improve the participation and retention of women. Fedora Women does that by identifying and posting profiles of other women who are successful in the Fedora community. More OSS communities should consider creating women only spaces in order to become more inclusive for women and to provide resources to women who participate. The expenses for creating these women only spaces would be mitigated by the larger number of women who will participate and the improved sense of safety and equality that the women will experience. The impact of these safe spaces for women has been studied in other areas [10], but not specifically in OSS communities. The authors of this study will continue this research by studying a larger number of such women only spaces in OSS communities and by interviewing the members of these communities to assess the perceived impact. Codes of Conduct investigation revealed very few OSS communities have a Code of Conduct. Recent research has shown an increasing trend [20], but there needs to be more awareness about the value of Codes of Conduct for fair treatment of all participants of a community. Specially, since there are several existing templates for Codes of conduct, we urge the creators of the OSS communities to adopt one of the existing ones, if they are not able to revise the existing ones for their own community. The codes of conduct that are present also do not always highlight women and or minority equality and inclusion. This might be due to a number of reasons, but having these explicit gives an opportunity to the community members to cite these whenever they have to address a conflict or have to flag out a violation of the code. A combination of women only spaces and community rules that explicitly include women will create a positive and welcoming environment for newcomers and veteran women members of the community. The results from the women only spaces show a variety of positive ways in which OSS communities are supporting women. The code of conduct can benefit from adopting some of the same language and activities. The authors of this study acknowledge that codes are conduct are often time added to communities, when they are already successful. We believe that adopting a Code of Conduct early on in the life of a project,

will be a contributing factor for the success of a project. Presence of a code of conduct will demonstrate an interest in building a fair and collaborative space.

We note that this exploratory study is limited in its sample selection and the sample size is not representative of the entire OSS environment. In future research, we will expand our sample size to include trending projects on GitHub and continue to evaluate the role of community rules and women only spaces in an OSS community. As a next step to this research, we are analyzing the women only spaces for membership, longevity and topics. We believe that women only spaces and inclusive community rules will provide a strong foundation for OSS communities to attract more women and retain the existing women contributors.

References

1. Carillo, K.D.A., Huff, S., Chawner, B.: What makes a good contributor? Understanding contributor behavior within large Free/Open Source Software projects – a socialization perspective. *J. Strateg. Inf. Syst.* **26**(4), 322–359 (2017)
2. Carillo, K.D.A., Huff, S., Chawner, B.: It's not only about writing code: an investigation of the notion of citizenship behaviors in the context of Free/Libre/Open Source Software communities. In: 2014 47th Hawaii International Conference on System Sciences, Waikoloa, HI, pp. 3276–3285 (2014)
3. Catolino, G., Palomba, F., Tamburri, D.A., Serebrenik, A., Ferrucci, F.: Gender diversity and women in software teams: how do they affect community smells? In: 41st International Conference on Software Engineering, (ICSE 2019). *Software Engineering in Society* (2019)
4. Coelho, J., Valente, M.T.: Why modern open source projects fail. In: Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany, 4–8 September 2017 (ESEC/FSE 2017), 11 p. (2017)
5. Department of Labor Force Statistics from the Current Population Survey - 2017 (2017). <https://www.bls.gov/cps/cpsaat11.htm>
6. Finley, K.: Diversity in open source is even worse than in tech overall. *Wired Magazine Website* (2017) <https://www.wired.com/2017/06/diversity-open-source-even-worse-tech-overall/>
7. Ford, D., Harkins, A., Parnin, C.: Someone like me: how does peer parity influence participation of women on stack overflow? In: 2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Raleigh, NC, pp. 239–243 (2017)
8. IRINEP: Is open source open to women (2017). <https://wotc.crn.com/blog/is-open-source-open-to-women>
9. Karpf, D.: Open source political community development: a five-stage adoption process. *J. Inf. Technol. Politics* **8**(3), 323–345 (2011)
10. Kilamo, T., Hammouda, I., Mikkonen, T., Aaltonen, T.: From proprietary to open source—growing an open source ecosystem. *J. Syst. Softw.* **85**(7), 1467–1478 (2012)
11. Lewis, R., Sharpe, E., Remnant, J., Redpath, R.: 'Safe spaces': experiences of feminist women-only space. *Soc. Res. Online* **20** (2015). <https://doi.org/10.5153/sro.3781>
12. Michlmayr, M.: Community management in open source projects. *Upgrade* **10**(2), 22–26 (2009)
13. NCWIT National Council of Women in Information Technology (2016). https://www.ncwit.org/sites/default/files/resources/btn_03092016_web.pdf

14. Raasch, C., Herstatt, C., Abdelkafi, N.: Open source innovation: characteristics and applicability outside the software industry. Working Paper 53 (2008)
15. Robles, G., Reina, L.A., González-Barahona, J.M., Domínguez, S.D.: Women in Free/Libre/Open Source Software: the situation in the 2010s. In: Crowston, K., Hammouda, I., Lundell, B., Robles, G., Gamalielsson, J., Lindman, J. (eds.) OSS 2016. IAICT, vol. 472, pp. 163–173. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39225-7_13
16. Scacchi, W.: Collaboration practices and affordances in Free/Open Source Software development. In: Mistrík, I., Grundy, J., Hoek, A., Whitehead, J. (eds.) Collaborative Software Engineering, pp. 307–327. Springer, Berlin (2010)
17. Taft, D.: Amidst Bias, Women Work to Find a Place in Open Source Communities (2017). <https://thenewstack.io/women-open-source-still-fighting-good-fight/>
18. Terrell, J., et al.: Gender differences and bias in open source: pull request acceptance of women versus men. *PeerJ Comput. Sci.* **3**, e111 (2017). <https://doi.org/10.7717/peerj-cs.111>
19. Tourani, P., Adams, B., Serebrenik, A.: Code of conduct in open source projects. In: IEEE 24th International Conference on Software Analysis, Evolution, and Reengineering (SANER), pp. 24–33 (2017)
20. Weller, M.: The distance from isolation: why communities are the logical conclusion in e-learning. *Comput. Educ.* **49**(2), 148–159 (2007)



Predicting Popularity of Open Source Projects Using Recurrent Neural Networks

Sefa Eren Sahin^(✉), Kubilay Karpat^(✉), and Ayse Tosun^(✉)

Faculty of Computer and Informatics Engineering, Istanbul Technical University,
34469 Istanbul, Turkey
{sahinsef,karpatk,tosunay}@itu.edu.tr

Abstract. GitHub is the largest open source software development platform with millions of repositories on variety of topics. The number of stars received by a repository is often considered as a measure of its popularity. Predicting the number of stars of a repository has been associated with the number of forks, commits, followers, documentation size, and programming language in the literature. We extend prior studies in terms of input features and algorithm: We define six features from GitHub events corresponding to the development activities, and additional six features incorporating the influence of users (followers and contributors) on the popularity of projects into their development activities. We propose a time-series based forecast model using Recurrent Neural Networks to predict the number of stars received in consecutive k days. We assess the performance of our proposed model with varying k (1, 7, 14, 30 days) and with varying input features. Our analysis on five topmost starred repositories in data visualization area shows that the error rate ranges between 19.76 and 70.57 among the projects. The best performing models use either features from development activities only, or all metrics including all the features.

Keywords: Open source projects · Predicting stars · Recurrent Neural Networks

1 Introduction

GitHub is a web-based collaborative software development platform built on the *git* version control system; it is also considered as a social community of developers [13]. In GitHub, users can follow the projects that they are interested in through GitHub stargazers [14]. GitHub keeps millions of repositories on variety of topics. The number of projects are generally high on popular topics such as machine learning, visualization libraries, web application platforms. Since there are many similar projects specialized on the same topic that a user can follow, it is necessary to make a decent analysis on the existing alternatives in order to

find the most suitable library or tool for a specific requirement of a user. The number of GitHub stars can be an appropriate criterion for this selection, since it is viewed as a measure of others developers' approval on the suitability of a project for a specific task, or in other words, a project's popularity [17].

Long-term support is an important factor while evaluating both OSS and commercial software projects, however OSS projects do not offer any binding contract about any future updates and maintenance. At this point, forecasting the number of stars of those OSS projects could be helpful. For example, Borges et al. [3] built a forecast model for GitHub projects based on the number of forks, commits and contributors, code features (age, programming language) and documentation of the projects. Incorporating other features from issues and users [1, 6] into the development activity of those projects produced more valuable insights in other studies. Models on predicting the number of stars are often built with time-series data using different regression techniques [3, 17]. We extend those models by adding both events taken from GitHub API such as number of forks, releases, commits, and issues, and weighting them based on the users' (developers and followers) influence. We also use a special variation of Recurrent Neural Networks (RNNs) which has been popularly used in time-series forecasting (e.g. [7, 8, 18]). We have built prediction models for the five topmost-starred data visualization projects in GitHub, and assessed the performance of the models for the following research questions:

- RQ1: How does the proposed model predict the number of stars with varying days (k)?
- RQ2: What are the effects of user-score based, weighted metrics on the predicting the number of stars?

Using contemporary forecasting techniques and a combined set of features, we believe our proposed model could guide users to understand the evolution of OSS projects, and to select which projects to contribute in their future development activities [5]. In the rest of the paper, we summarize related work (Sect. 2), describe our methodology for collecting the projects, extracting metrics and model construction and evaluation (Sect. 3), discuss our results in Sect. 4 and finally, we share our conclusions in Sect. 6.

2 Related Work

As GitHub became the widespread for OSS development, investigating its different aspects has become inevitable for our community. Secondary studies on GitHub (e.g. [9, 13]) report that researchers mine GitHub repositories to understand software development such as pull requests and forks, characterize projects with respect to popularity, collaboration and transparency, and model social factors through users and developers. In predicting the popularity of projects, studies investigate the number of stars, forks, followers and found strong relations among those [9]. Having more consistent documentation as well as having many contributors are found to be other factors explaining popularity of a project [9].

We found many studies on understanding a project’s popularity in GitHub, but discussed three studies on *predicting* the popularity of projects in terms of number of stars [3,4,17] below.

Borges et al. built models for predicting the number of stars of 2500 GitHub repositories in two of their studies [3] and [4]. In [4], different features from around 2500 GitHub repositories were extracted, such as the project age, the number of forks, commits, contributors, application domain, programming language and repository owner. They analyzed the relationship between these features and the number of stars, and found that there is a weak correlation between the age, the commits, and its star count, however there is a strong correlation between forks and stars [4]. After major releases, projects tend to get a lot of stars. Furthermore, projects are likely to receive more stars right after the first public release. Following that research, [3] expanded their initial model by clustering repositories by their growth rate, making in-cluster predictions and comparing those in-cluster predictions with generic predictions. They built a multiple linear regression model to predict the number of stars of GitHub repositories using previous star counts as their model input. The authors were able to predict the number of stars six months ahead with high confidence using star events of past six months, and they mostly observed more accurate results with in-cluster predictions [3]. Weber and Luo [17] mined two snapshots of GitHub projects, and extracted 38 features, 20 of which were from Python abstract syntax trees (in-code features) and 18 of which were related to project volume, documentation volume, presence of supporting files, and standard library usage. The GitHub snapshots were divided into two even groups, 1000 low popularity and 1000 high popularity projects. Using decision tree classifier, [17] reports the most important features for distinguishing a low popular project from a high popular one as readme file size, continuous integration file size, class definitions, and comment ratios. The division of projects into two classes and the features are different from general practices in this area, and the decision tree classifier is used for finding the importance of features rather than predicting the stars using these features.

In our study, we propose a time-series based model to predict the number of stars received in consecutive k days. Previous works on predicting the number of stars [3,4,17] and on predicting the number of forks [6] are similar to our research goal, although they differ from our work in terms of methodology, algorithm and collected data. [4] did not build a predictive model but by using descriptive analytics, it is reported that forks and release dates are important factors for predicting the popularity. We use the forks, commits, releases as some of the features in our model. We also follow prior studies’ findings and extend the set of features by incorporating the effect of users (contributors and followers of the projects) [5,16] and issues [2] on the star counts. [3] uses multi linear regression model to predict star counts, and reported accurate results in most projects, yet considered as unreliable for repositories with non-linear growth by authors themselves. [6] predicts the number of forks after T months since the project’s start date and uses a stepwise linear regression model for all projects. Different

from both these works, we utilize the power of deep learning methods which do not depend on the linearity constraints. We use a special variation of recurrent neural networks for time series forecasting as these techniques were successfully applied to other problems in [7, 12, 18], especially when incorporating many input features in the form of time-series. Furthermore, our model predicts the number of stars received in consecutive k days. We believe predicting cumulative star count is more challenging because it either increases over days or gets stabilized after the project activity degrades. For the projects with high number of stars, the number of stars received over a week/month may not affect the total number of stars and therefore, forecast models may produce low error rates which is misleading for the star count prediction. Therefore, we aim to find the evolution of the number of stars in GitHub projects using recurrent neural networks.

3 Methodology

In this section, we describe our dataset, our methodology of extracting and interpreting metrics, the training algorithm and the model construction.

3.1 Dataset

Our research focuses on a set of interchangeable repositories which operate in the same application area, namely *data visualization*, since the earlier work report that the distribution of the number of stars by application area is significantly different [4]. The dataset used in this study includes historical data of the selected area. All the data was collected through the public API provided by GitHub. For each data-visualization repository, we collected commit, issue opening, issue closing, starring, forking and publishing a release events in terms of who and when performed the event. All these events can be interpreted as potential features for predicting the stars. Commits are the main activities in GitHub. Hence, an actively developed repository with frequent commits may attract the community more than a repository which does not receive commits on a regular basis. Issue tracking systems are significant parts of software development as they lead to the participation of community while reporting bugs and requesting new features [2, 10]. Thus, issue opening/closing activities can be associated with the interest of community. Forks on the other hand, is highly correlated with the number of stars [4]. Publishing releases also may attract the community. We built a time-series data of each repository. The time-series datasets contain daily number of received stars and other metrics extracted from the events. Even though we collected the whole timeline of the repositories, we limit our dataset to the beginning of 2013. We also selected five topmost starred repositories as our dataset for the experiments. Table 1 shows the selected repositories and their number of stars as of the date their data were crawled from GitHub.

Metrics. We have selected six metrics based on the GitHub events collected from data visualization repositories. In order to see the effect of the users, we also

Table 1. Five topmost starred repositories (as of 2017-11-14).

Repository	No. of stars
d3/d3	70049
chartjs/Chart.js	33416
ecomfe/echarts	21785
Leaflet/Leaflet	19944
gionkunz/chartist-js	10151

derived a time-aware user scoring method, calculated six additional weighted metrics based on user scoring, and computed 12 metrics in total. Since we have built a daily time-series data, all of the metrics are computed on a daily basis. The weighted metrics, on the other hand, are aggregated by summing scores of all the users related with the repository until day t . We listed the metrics and their brief descriptions in Table 2.

Table 2. Metrics and their descriptions.

Metric	Description
Number of stars	The number of stars received by the repository at day t
Number of forks	The number of forks received by the repository at day t
Number of commits	The number of commits made on the repository at day t
Number of opened issues	The number of issues opened on the repository at day t
Number of closed issues	The number of issues closed on the repository at day t
Number of releases	The number of releases published by the repository at day t

User Scoring. Previous models in the area of measuring developer influence propose metrics like the number of followers in GitHub [2, 10]. [2] also proposed a method for ranking developers through an influence propagation over a network of users. Our scoring mechanism also calculates an influence score for a developer by adhering to the development and community activities within the repositories on data visualization area. However, weighting a metric on day t_1 with user score on day t_2 may cause incorrect results. Thus, instead of using static user scores, we have created a time-series data of user scores such that the metrics on day t are weighted based on the scores of the users in data-visualization repositories until day t . We have seven main user-scoring metrics. The user-score metrics and their descriptions can be seen in Table 3. Equation 1 shows our heuristic formulation for calculating the user score on day t . We weighted each metric based on its importance value and normalized the metric by dividing it to the sum of the importance values. With this calculation, the impact of a user increases over time as the user actively contributes to the projects in the selected application area.

Table 3. User-score metrics, their descriptions and importance (I) values.

Metric	Description	I
Number of commits	The number of commits made by the user until day t	7
Number of closed issues	The number of issues closed by the user until day t	6
Number of releases	The number of releases published by the user until day t	5
Number of contributed repositories	The number of repositories the user made contributed until day t	4
Number of opened issues	The number of issues opened by the user until day t	3
Number of forks	The number of repositories forked by the user until day t	2
Number of stars	The number of repositories starred by the user until day t	1

$$score_{(u,t)} = \frac{\sum_i metric_i(u,t) * importance_i}{\sum_i importance_i} \quad (1)$$

3.2 Algorithm

Even though there are linear regression models and ARIMA to model time series data, we use Long Short Term Memory (LSTM) as our forecast model. LSTM is a special variation of Recurrent Neural Network (RNN) as a solution for the problem of blowing up and vanishing of recurrent back-propagation in traditional RNNs [11]. A simple LSTM unit consists of a memory cell and gates [11]. In memory cell, important information of the input sequence is stored. The gates determine whether the information will be stored or not. Thus, this makes it applicable for predicting time-series. The architecture of LSTM makes it suitable for training steps of time-lag sequences with different input and output steps. Our motivation is to build a flexible forecast model which is capable of making predictions for k consecutive days by learning from historical time-series data, and hence, LSTM fits perfectly to this requirement.

There have been many applications of LSTM for time series forecasting (e.g. [7, 12, 18]). Previous studies show that LSTM outperformed models built with Support Vector Regression on both univariate and multivariate data for phone price prediction in [7], and with ARIMA for CPU usage prediction [12]. It is also proven that multiple features can be fed into time-series forecasting with LSTM [18].

3.3 Model Construction

As LSTM Networks are capable of learning sequences in different fields, this algorithm allow us making multi-step predictions. To accomplish that, we formatted our time-series dataset as input sequences which have historical data and future data consists of each sample corresponding to a day. To answer RQ2, while creating sequences at each time sample, we took data of input metrics in previous 180 days as input, and stars in next 30 days as output. This means that our model predicts stars received daily in the next 30 days given the previous 180 days' data. Then, we split the dataset as 70% for training and 30% for test. Instead of a general training model which involves all the data visualization

repositories, we have specialized and trained a LSTM Network for each repository individually. These constructed LSTM networks are formed by one input layer, two LSTM layers each of which contain 500 hidden units and an output layer.

3.4 Performance Evaluation

We evaluate our models with Mean Absolute Percentage Error (MAPE) which is widely used for evaluating the accuracy of forecast models. It is defined as follows:

$$MAPE = 100 * \left(\frac{1}{n} \sum_{t=1}^n \frac{|Y_t - F_t|}{Y_t} \right) \quad (2)$$

where Y_t denotes the actual value, F_t denotes the forecasted value and n denotes the number of samples. Even though MAPE is widely used for evaluating forecast accuracy, outliers drastically increases the error [15]. Therefore we also report the box plots of Absolute Percentage Error (APE) for each model over multiple iterations.

4 Results

We built different models having different combinations of metrics as their inputs. The input combinations are as follows: **Model 1.** Number of stars only. **Model 2.** Repository based metrics (Table 2). **Model 3.** User-Score based, weighted metrics (Metrics in Table 2 weighted based on Eq. 1). **Model 4.** All metrics. Our prediction models generated forecasts for the k consecutive days for each instance in our dataset. We chose k to be 1, 7, 14 and 30 indicating predictions made for 1 day ahead, 7 days ahead, 14 days ahead and 30 days ahead, respectively. To answer RQ1 and RQ2, we obtained predictions for consecutive 30 days for each model constituted by different input combinations, and evaluated the input combinations to see how different input metrics impact the performance of the model. Table 4 shows the calculated MAPE values for the input combinations across all repositories.

RQ1 aims to evaluate the performance of the proposed models on predicting the number of stars for different days. Our experiments were made for $k=1, 7, 14, 30$ days. It seems there is not a generic pattern in the change of MAPE values over different k . The models perform almost the same for different k days. There exists an irregular increase in MAPE values for Models 2 to 4, as the predictions are made for further days for *gionkunuz/chartist-js*. But the increase is not statistically significant. *d3/d3* is the best performing repository in all cases with MAPE values less than 30%. We also computed the distribution of absolute percentage error (APE) values of the repositories over multiple iterations of model evaluation. Figure 1 shows APE distribution for *d3/d3* project only, as the pattern is the same for all the projects. For all cases, it is observed that the outlier APE values cause the increase in MAPE.

Table 4. MAPE values for four models across all repositories. The column k indicates the predicted day.

Repositories	Model 1				Model 2				Model 3				Model 4			
	k				k				k				k			
	1	7	14	30	1	7	14	30	1	7	14	30	1	7	14	30
d3/d3	28.56	28.56	28.52	28.76	20.34	20.4	21.06	19.76	20.46	21.26	22	19.79	21.07	21.13	21.23	20.45
chartjs/Chart.js	32.89	32.89	33.56	30.03	33.27	32.2	31.57	37.02	32.09	29.87	29.8	33	31.82	27.93	29.49	31.87
ecomfe/echarts	42.99	42.99	44.05	47.23	39.89	45.07	45.02	48.11	53.13	53.3	53.32	53.67	44.44	50.84	47.72	49.22
Leaflet/Leaflet	35.39	35.39	35.57	37.99	32.95	33.15	33.35	34.62	37.89	37.45	35.56	36.42	36.95	34.69	34.9	36.32
gionkunz/chartist-js	73.25	73.25	69.78	70.48	51	57.31	59.82	66.83	65.98	54.5	58.03	70.57	48.8	48.9	48.85	59.5

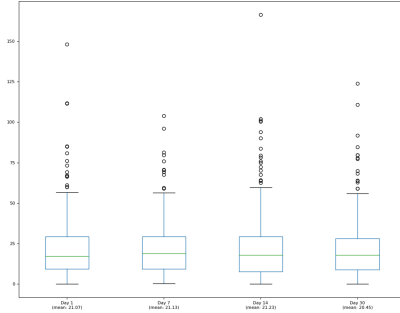


Fig. 1. APE values for Day 1, Day 7, Day 14 and Day 30 for *d3/d3*.

RQ2 aims to assess the performance of predictions using different metric sets in terms of MAPE values reported in Table 4. For all repositories except *ecomfe/echarts*, adding metrics in addition to number of stars leads to more accurate predictions. Especially for *d3/d3* and *gionkunz/chartist-js*, error values significantly decreases when the metrics are included to the model. Even though adding user-score based, weighted metrics slightly decreases the error values for *gionkunz/chartist-js* in all days, and for *d3/d3* for Model 4, repository based metrics (Model 2) or the combination of all (Model 4) perform much better for *d3/d3*, *chartjs/Chart.js*, *Leaflet/Leaflet*, *gionkunz/chartist-js*.

Discussion. In RQ1, the performance of the models did not improve as they generated predictions for bigger k days ($k = 14$, $k = 30$). This finding supports the conclusions in [3]: For smaller k values, average relative squared error (RSE) rates are above 30, whereas for $k = 52$ weeks, RSE rates decrease down to 5 in top starred repositories [3]. Among all the models, we achieved MAPE between 19.76 and 70.57 for $k = 30$ days. However, predictions at $k = 1$ day are between 20.4 and 73.25. But the k values picked in our study are still below the k 's picked in [3], and hence, it is expected to observe higher error rates than the prior work. Findings of RQ2 also support the previous findings that events in GitHub in terms of development and users contributed to the repositories are good indicators of number of stars. In three of the projects (*d3/d3*, *ecomfe/echarts*, *Leaflet/Leaflet*), only repository metrics produce an average MAPE of 20% to 44%, whereas in the two projects (*chartjs/Chart.js*, *gionkunz/chartist-js*) all metrics perform better with an average MAPE of 30% and 51%. When we look at the projects

in detail, each project has a different characteristics and hence, a different best-performing model. The best-performing models are mostly achieved in Model 2 (*d3/d3*, *ecomfe/echarts*, *Leaflet/Leaflet*) with varying k . The other two projects (*chartjs/Chart.js*, *gionkunz/chartist-js*) have better predictions with Model 4 and $k=7$ and $k=14$. The deviations in MAPE values show that predictions for recent days are still challenging, and there might be other factors such as social media attention of a project [5] causing sudden increases in star counts for the projects. In order to focus on the area expertise of the developers we only use their contributions to the data visualization area. However, the features and the LSTM model are independent of the application area. Thus, our method can be applied to different domains.

5 Threats to Validity

In this study we evaluated only one application area in GitHub, namely data visualization, and assessed the performance of deep learning techniques on the prediction of star counts. Fluctuations on growth trends and user interactions particularly in this application area are likely to influence our findings. As this study depends on development activities and community interactions of repositories, our methodology could be applied to other application areas using suitable projects with active development activities and high star counts. Since we used several user-score weighting metrics based on aggregated findings from literature, we had to derive a formula to come up with a single user-score weight at a particular snap shot. The formula used is weighting all metrics based on their importance values decided by votes of the three researchers. These importance values may change, and impact the overall weights of users at any time t , however the values are assigned based on our experience in using GitHub and previous studies' highlights on the important factors influencing a user's impact on the community. We used MAPE to assess the performance of our model as it is stated as a better measure to assess the forecast errors in [15]. We also observed that MAPE is sensitive to outliers, and hence plotted APE values over all iterations of the model.

6 Conclusion

This study proposes a methodology to predict the number of stars received at the consecutive k days in GitHub using LSTM. Our results show that predictions for recent days (up to 30 days) can be made with varying error rates as the performance of LSTM differs among the selected projects. The best performing model is achieved in *d3/d3* project with a MAPE value of 19.76. As a future work, we would like to add user-score metrics using a different heuristics into the model, and observe the effect of these heuristics on the model performance. We would also like to investigate the effects of individual features on predicting the star counts. We believe LSTM performed very well on time-series GitHub events, as it captures changes on development and contributions at the lowest granularity

(daily) and adds multiple metrics' time series data into a single model. Therefore it is very powerful for analysis on time-series forecasting problems, and can be configured in various forms in the future. Our proposed LSTM model can also be applied on other repositories.

Acknowledgments. This research is supported in part by Scientific Research Projects Division of Istanbul Technical University with project number MGA-2017-40712 and Scientific and Technological Research Council of Turkey with project number 5170048.

References

1. Badashian, A.S., Stroulia, E.: Measuring user influence in GitHub: the million follower fallacy. In: International Workshop on CrowdSourcing in Software Engineering, pp. 15–21 (2016)
2. Bissyande, T.F., Lo, D., Jiang, L., Reveillere, L., Klein, J., Traon, Y.L.: Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub. In: International Symposium on Software Reliability Engineering, pp. 188–197 (2013)
3. Borges, H., Hora, A., Valente, M.T.: Predicting the popularity of GitHub repositories. In: International Conference on Predictive Models and Data Analytics in Software Engineering, pp. 1–10 (2016)
4. Borges, H., Hora, A., Valente, M.T.: Understanding the factors that impact the popularity of GitHub repositories. In: IEEE International Conference on Software Maintenance and Evolution (2016)
5. Borges, H., Valente, M.T.: What's in a GitHub star? Understanding repository starring practices in a social coding platform. *J. Syst. Softw.* **146**, 112–129 (2018)
6. Chen, F., Li, L., Jiang, J., Zhang, L.: Predicting the number of forks for open source software project. In: International Workshop on Evidential Assessment of Software Technologies, pp. 40–47 (2014)
7. Chniti, G., Bakir, H., Zaher, H.: E-commerce time series forecasting using LSTM neural network and support vector regression. In: International Conference on Big Data and Internet of Thing, pp. 80–84 (2017)
8. Connor, J.T., Martin, R.D., Atlas, L.E.: Recurrent neural networks and robust time series prediction. *IEEE Trans. Neural Netw.* **5**(2), 240–254 (1994)
9. Cosentino, V., Izquierdo, J.L.C., Cabot, J.: A systematic mapping study of software development with GitHub. *IEEE Access* **5**, 7173–7192 (2017)
10. Grammel, L., Schackmann, H., Schröter, A., Treude, C., Storey, M.A.: Human aspects of software engineering, pp. 1–6 (2010)
11. Hochreiter, S., Schmidhuber, J.: Long short-term memory. *Neural Comput.* **9**(8), 1735–1780 (1997)
12. Janardhanan, D., Barrett, E.: CPU workload forecasting of machines in data centers using LSTM recurrent neural networks and ARIMA models. In: International Conference for Internet Technology and Secured Transactions (2017)
13. López, A.R.: Analyzing GitHub as a collaborative software development platform: a systematic review. MSc thesis, University of Victoria (2017)
14. Neath, K.: Notifications & stars, August 2012
15. Shcherbakov, M.V., Brebels, A., Shcherbakova, N.L., Tyukov, A.P., Janovsky, T.A., Kamaev, V.A.: A survey of forecast error measures. *World Appl. Sci. J.* **24**(24), 171–176 (2013)

16. Tsay, J., Dabbish, L., Herbsleb, J.: Influence of social and technical factors for evaluating contribution in GitHub. In: 36th International Conference on Software Engineering, pp. 356–366 (2014)
17. Weber, S., Luo, J.: What makes an open source code popular on GitHub? In: International Conference on Data Mining Workshop, December, pp. 851–855 (2014)
18. Zhang, L., Liu, P., Gulla, J.A.: A neural time series forecasting model for user interests prediction on Twitter. In: 25th Conference on User Modeling, Adaptation and Personalization, pp. 397–398 (2017)



What Attracts Newcomers to Onboard on OSS Projects? TL;DR: Popularity

Felipe Fronchetti¹(✉), Igor Wiese², Gustavo Pinto³, and Igor Steinmacher⁴

¹ University of São Paulo, São Paulo, São Paulo, Brazil
fronchetti@usp.br

² Federal University of Technology, Campo Mourão, Paraná, Brazil
igor@utfpr.edu.br

³ Federal University of Pará, Belém, Pará, Brazil
gpinto@ufpa.br

⁴ Northern Arizona University, Flagstaff, Arizona, USA
igor.steinmacher@nau.edu

Abstract. Voluntary contributions play an important role in maintaining Open Source Software (OSS) projects active. New volunteers feel motivated to contribute to OSS projects based on a set of motivations. In this study, we aim to understand which factors OSS projects usually maintain that might influence their new contributors' onboarding. Using a set of 450 repositories, we investigated mixed factors, such as the project age, the number of stars, the programming language used, or the presence of text files that aid contributors (e.g., templates for pull-requests or license files). We used a K-Spectral Centroid (KSC) clustering algorithm to investigate the newcomers' growth rate for the analyzed projects. We could find three common patterns: a logarithmic, an exponential, and a linear growth pattern. Based on these patterns, we used a Random Forest classifier to understand how each factor could explain the growth rates. We found that popularity of the project (in terms of stars), time to review pull requests, project age, and programming languages are the factors that best explain the newcomers' growth patterns.

Keywords: Open Source Software · Newcomers · Attractiveness

1 Introduction

Voluntary contributions play an important role in maintaining Open Source Software (OSS) projects active [29]. This is because OSS projects work in a symbiotic way. While communities need to motivate, engage, and retain new developers to remain sustainable [19], a large, globally distributed community of developers wants to contribute for a variety of reasons, including learning, the necessity to fix a bug, and reputation [13, 24, 33].

However, as shown before in several studies [28–30], the newcomers face several barriers while joining to OSS projects. This can lead to demotivation and,

ultimately, dropouts. Given the importance of the newcomers to the projects and the barriers they face, it is important to study the different aspects of the joining process. As previously stated by Steinmacher et al. [30], joining a project is a complex process composed of different stages and a set of forces that push newcomers towards (motivation and attractiveness) or away (onboarding barriers) from the project. While motivation is something that is usually inherent to the developers, attractiveness is a force that—to some extent—can be managed by the projects.

Some previous studies analyzed project attractiveness by analyzing its relationship with license [22], source code attributes [15,16], and code base [4]. However, the existing literature does not analyze the temporal aspect of the newcomers' joining, nor consider the recent phenomenon of social coding environments and their characteristics, which introduced a more standardized way to contribute [9]. This perspective is important, since, according to Capiluppi and Michlmayr [3] “the success of a project is often related to the number of developers it can attract”.

In this paper, we start filling this gap by investigating which projects' characteristics are related to the increase of newcomers growth temporally in OSS projects maintained on GitHub. To achieve that, we selected a set of factors inherently from the OSS projects that might explain the increase of newcomers. We place these factors in context, measuring their effects on 72 weeks of growth of newcomers in 450 OSS projects. Our approach included clustering the OSS projects in terms of newcomers' joining growth, aiming to identify different growth patterns. Based on the patterns identified, we leveraged the Random Forest [2] classifier to measure the effects of the projects' characteristics aiming to explain each pattern.

The contributions of this study include: (i) empirical evidence of different patterns of newcomers growth in OSS projects, adopting a time series analysis; and (ii) identifying the factors that could potentially lead to attraction of newcomers in OSS projects. Ultimately, the results of this work may benefit project maintainers who can get acquainted with ways to make their project's more attractive, creating a more welcoming environment for newcomers.

2 Methodology

To guide our research towards this goal, we designed the following research questions:

- **RQ1. What are the newcomer joining rates in OSS projects?** The answer to this question is relevant to understand if projects receive constant rates of newcomers temporally, or if there are different trends for different projects. In case we find different trends of temporal joining rates, it is worth understanding how these different trends can be classified. By having a comprehensive classification of newcomers joining rate per time, it is possible to go in-depth and explore the reasons for the differences.

- **RQ2. What are the project’ factors that may influence the joining rate?** The answer to this question aims to explain what are the factors that may influence different rates of newcomer joining, which may bring to light potential ways to attract more developers to OSS projects.

2.1 Curating the Corpus of OSS Projects

To explore the attractiveness of OSS projects, we retrieved data from 450 OSS projects hosted on the GitHub coding platform. To improve the variety of projects, we first selected the fifteen most popular programming languages used on GitHub [20]. The set of programming languages was composed of C, Clojure, CoffeeScript, Erlang, Go, Haskell, Java, JavaScript, Scala, Objective-C, Perl, PHP, Python, Ruby, and TypeScript. To avoid unmaintained projects, for each selected programming language, we filtered the 30 most starred OSS projects. We followed recent related work that considers stars as a measure of attractiveness in OSS projects [1]. We ended up with 450 OSS projects, written by 15 popular programming languages. The set of sampled projects included:

- SCALA/SCALA: The Scala programming language was released in 2001 and has received more than 31 K commits. Mostly written in Scala.
- DJANGO/DJANGO: High-level web application framework. Release in 2005, it has received more than 26 K commits. Mostly written in Python.
- VIM/VIM: Highly configurable text editor. Released in 1991, it has received more than 9 K commits. Mostly written in C.

As one can see, Fig. 1 suggests that even selecting projects with the highest number of stars, the distribution of stars varies significantly in the dataset, ranging from 6 to 39,990 stars (Q1: 3,648; Median: 10,470; Q3: 18,900). The data was collected using the GitHub API, and it was conducted in October 2018. To access the complete dataset and the source code of the tools used in this research are publicly available in our repository¹.

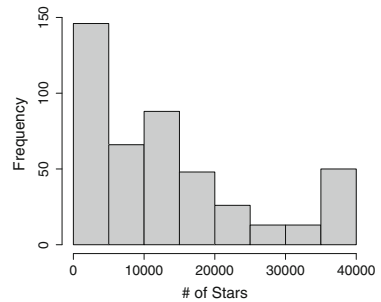


Fig. 1. Stars per project

In our analysis, we collected a set of factors from the selected projects’ repositories, including project popularity, maturity, receptivity/welcoming features. We used these factors in our model, and their descriptions are presented in Table 1. We also calculated the Spearman correlation coefficients [12] on the factors in order to remove the strongly correlated ones ($\rho > 0.7$). The number of forks was removed from the list of factors since it has a high correlation with the number of stars.

¹ <https://github.com/fronchetti/OSS-2019>.

Table 1. Factors extracted from repositories

Factor	Description
Age	Number of years since the repository creation
Main language	Most used programming language
Time to merge	Average of days for pull requests (PR) to be merged
Account type	If the OSS is hosted on an Organization or User account
Domain	The software application domain
# of stars	Number of stars
# of languages	Number of used programming languages
# of integrators	Number of contributors with rights to merge pull requests
Has PR template	Repository has a standard template for new pull requests
Has issue template	Repository has a standard template for new issues
Has license	Repository has the LICENSE file
Has code of conduct	Repository has CODE_OF_CONDUCT file
Has readme	Repository has the README file
Has contributing	Repository has the CONTRIBUTING file
Has wiki	Repository has WIKI+ section

It is important to note that some of these factors are not straightforwardly available in the GitHub repository, which is the case of the domain of the repositories. For this particular factor, we followed the Borges et al. methodology to define projects domain [1], and manually added it by doing a qualitative analysis over the website and documentation of the projects. As an example, we manually evaluate the Linux website² to define it as part of the system software domain.

To characterize the project attractiveness temporally, we collected the newcomers' growth rate for each project, considering one week as the observation unit. The growth of newcomers is represented by a time series, which associates the evolution of the number of newcomers with the number of weeks existent in each project (newcomers per week). To this end, we define a newcomer as any contributor that submitted their very first contribution (commit) to the master branch. It is worth mentioning that each contributor was considered a newcomer only once, in the particular week that they submitted their first commit.

2.2 Identifying Growth Patterns

To identify the different newcomers' growth patterns, we clustered the sampled OSS projects growth according to the time series mentioned before. We used the K-Spectral Centroid (KSC) clustering algorithm [32] to create the clusters. The KSC algorithm finds clusters of time series that share distinct temporal

² <https://www.linux.com/what-is-linux>.

patterns, following a similar approach as the used in the classical K-means clustering algorithm [11]. We chose to apply the KSC clustering algorithm because the clustering is performed independently of shifts (i.e., dates) and scale (i.e., volume), focusing rather on the overall shape of the time series [7]. Moreover, the KSC algorithm was applied in well-established papers that follow a similar approach in different contexts [1] and domains [8].

The KSC algorithm requires that all the time series used during the clusterization have the same length. For this reason, we used only the time frame comprising the last 72 weeks of newcomers inflow for each project, considering the date of the dataset creation (October 2018). We use the length of 72 weeks because all projects are at least 72 weeks old. The KSC algorithm also requires the definition of a specific number of k clusters. To decide the best number of k clusters, we used the β_{CV} [17] heuristic. The β_{CV} heuristic is defined as the ratio of two coefficients: variation of the *intracluster* distances and variation of the *intercluster* distances. The smallest value of k after which the β_{CV} ratio remains roughly stable should be selected, as a stable β_{CV} implies that new splits affect only marginally the variations of *intracluster* and *intercluster* distances [8].

Figure 2 presents an association between the β_{CV} ratios and the k clusters for the newcomers time series. When considering the β_{CV} ratios and the number of projects per cluster, we decided to use $k = 3$ clusters in the K-SC algorithm. Note that $k = 4$ clusters could represent a better number of clusters in terms of ratio stability. By testing the clustering algorithm for $k = 3$ and $k = 4$, we found that the results were similar, and two groups could clearly be merged. Moreover, having a small number of projects per cluster would affect the future classification of patterns presented in Sect. 2.3. Thus, we kept with $k = 3$ clusters.

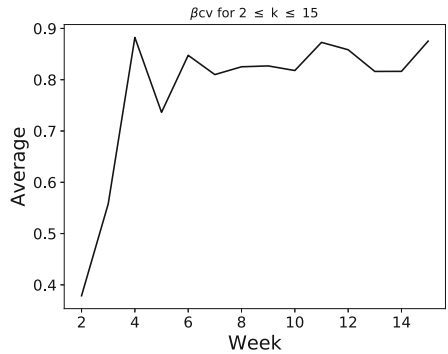


Fig. 2. The β_{CV} ratios

2.3 Identifying Explanations to the Growth Patterns

Our next step was to explore the different newcomers’ growth patterns. To achieve this goal, we used a Random Forest [2] classifier to measure the effects of the independent variables (factors) in the explaining of the dependent variable (growth patterns). We selected Random Forest because it is fast [23], robust in the presence of noise and outliers [21], and has a great performance with numerical and categorical data [25].

To develop our model, we used the `RandomForestClassifier` class from the `scikit-learn` framework. The predictors used were the factors defined in Table 1, and the target variables were the growth patterns found. Using a measure called Mean Decrease Impurity (MDI) [14], `RandomForestClassifier` also

provides a ranking of the most important features in the prediction of the target variables. The higher the score of the feature, the greater is its importance. We used the scores obtained from the classifier to understand the relationship between the factors and the growth patterns. Finally, we measured the effectiveness of the classifier using three commonly used metrics of Machine Learning: Precision, Recall, and F-measure. The code related to the clusterization and classification of the growth patterns are publicly available in our repository (See footnote 1).

Table 2. Description of the clusters

Cluster	Pattern	# repositories	Growth (%)
C1	Logarithmic	71 (15.7%)	12.6%
C2	Exponential	57 (12.6%)	27.6%
C3	Linear	322 (71.5%)	19.9%

3 RQ1. On the Newcomer Joining Rates in OSS Projects

Figure 3 presents the growth patterns found, which we named as *logarithmic*, *exponential* and *linear* growth of newcomers. We chose these names based on the centroids' trends, defined as the average growth of the clusters (also presented in Fig. 3). In Table 2 we present a clusters overview, including the number of repositories per cluster and the percentage increase of newcomers obtained from the

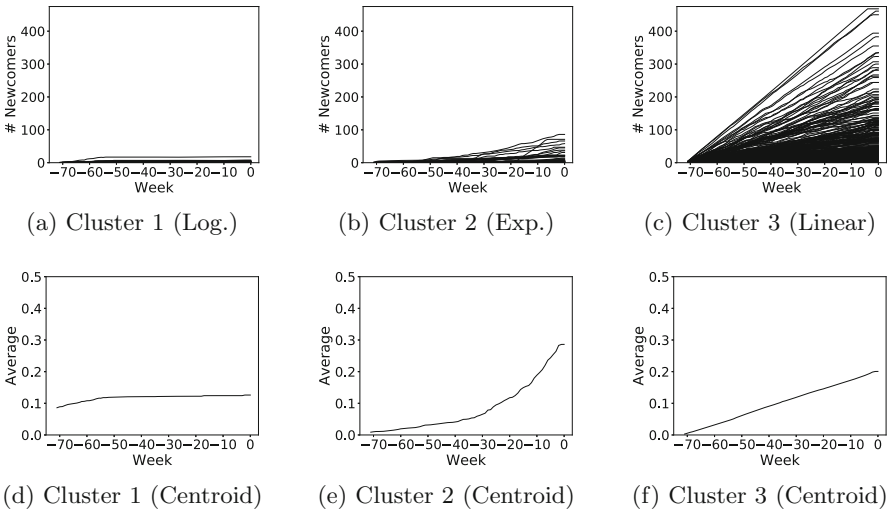


Fig. 3. Growth patterns clusters and centroids.

centroids in 72 weeks. *Linear* growth includes the highest number of repositories (71.5%), with a percentage increase of newcomers of 19.9%. The *Logarithmic* growth is the second one with most repositories (15.7%), but different from the *Linear* growth, it has the lowest increase of newcomers (12.6%). The *Exponential* growth is the less representative cluster, represented by only 12.6% of the repositories. On the other hand, its percentage increase of newcomers is the highest one (27.6%).

To put these results in a better context, we investigated OSS projects for each one of the growth patterns. In the *linear* growth, we perceived that projects DEFINITELYTYPED/DEFINITELYTYPED, RAILS/RAILS, and SYMFONY/SYMFONY, follow this tendency from the very beginning. We perceived a similar trend when analyzing the projects that fit on the *logarithm* growth. The projects ALPACALANG/ALPACA and CHAPLINJS/CHAPLIN are interesting samples because both are active projects that received their first newcomer only after the first two months of analysis. On the other hand, the projects IONIC-TEAM/IONIC, SYNRC/N2O, and DRKLO/TELEGRAM are the ones who fit in the *exponential* growth. In particular, project Ionic grew from 25 to 45 newcomers in only ten weeks of activity.

RQ1 Summary. Three different growth patterns represent the entry of newcomers into OSS projects: Linear, exponential and logarithmic. Linear growth represents the majority of repositories with an intermediate growth; exponential growth represents the smallest number of repositories, but holds the highest growth of newcomers; and logarithmic growth represents an intermediate number of repositories, but has the lowest growth among the three patterns.

4 RQ2: On the Factors that May Influence the Joining Rate

After running the Random Forest classifier and calculating the MDI for the factors in our prediction model, we rank the projects' factors. Table 3 presents factors ordered by importance in predicting of newcomers growth patterns. As one could see, the highest score is the number of stars. This finding is particularly interesting because, contrary to well-known beliefs that suggest that newcomers may be more tempted to contribute to OSS projects that are written in a programming language that they are more familiar with, than a popular one. Time to merge appears next in the top factors explaining the newcomers' growth rate. This is interesting, because it shows a relationship between a newcomers onboarding and the good practice of giving timely review, feedback, and closing pull requests. Completing the list of the top factors (with scores higher than 0.10), we have factors that are intrinsically related to the project, such as age and the number of programming languages used.

Moreover, the presence of text files such as the CONTRIBUTING file, the LICENSE file, and the CODE OF CONDUCT file, which are even recommended













Table 3. Ranking of the most important factors

Ranking	Factor	Score
1	# of stars	0.1753
2	Time to merge	0.1535
3	# of languages	0.1278
4	Age	0.1027
5	# of integrators	0.0995
6	Main language	0.0946
7	Domain	0.0708
8	Has contributing	0.0396
9	Has wiki	0.0308
10	Has issues template	0.0260
11	Owner type	0.0252
12	Has license	0.0236
13	Has PR template	0.0183
14	Has code of conduct	0.0118

as community best practices³, are among the worst ranked factors in our model (they scored 0.0396, 0.0236, and 0.0118, respectively). Still, having issues and pull requests templates, which are also recommended to welcome newcomers, presented very low scores (0.0260 and 0.0183, respectively).

We also investigated the effectiveness of the classifier in predicting the growth patterns. We used three metrics from `scikit-learn`⁴ to investigate the effectiveness: Precision, Recall, and F-measure [26]. Precision measures the correctness of the classifier in predicting growth patterns. Recall measures the effectiveness of the classifier in identifying the growth patterns. F-measure is the harmonic mean of precision and recall. Table 4 shows the metrics’ results divided by cluster, along with an overall result based on the micro-average of each metric [27].

Table 4. Precision, recall, and f-measure of the classification model. Divided by clusters, and an overall.

Growth Pattern	Precision	Recall	F-measure
Logarithmic	0.44 	0.19 	0.27 
Exponential	0.33 	0.06 	0.10 
Linear	0.75 	0.95 	0.84 
Overall	0.72 	0.72 	0.72 

³ <https://opensource.guide>.

⁴ <https://scikit-learn.org>.

In general, the Random Forest classifier obtained a significant performance, with a micro-average of 72% for precision, recall, and F-measure. The *Linear* growth group presented the highest results, and almost all its instances were correctly classified (Precision: 75%, Recall: 95%, F-measure: 84%). On the other hand, we could not observe good results for the *Logarithmic* and *Exponential* groups. Only 6% of the *Exponential* instances were identified correctly (Precision: 33%, Recall: 6%, F-measure: 10%). The bad results may be justified by the number of instances analyzed (We only used 450 projects), the number of instances per growth group (since 71.5% of the instances belong to the *Linear* pattern), and the characteristics used during the prediction (Other characteristics may also affect the distinction of patterns). However, to understand these differences accurately, a broader study is needed.

RQ2 Summary. Popularity of the project (in terms of stars), time to review pull requests, and project characteristics like age and programming languages are the factors that best explain the newcomers' growth patterns. In addition, GitHub recommended community standards (<https://github.com/github/opensource.guide/community>) have a lower influence on the observed growth patterns.

5 Related Work

Several studies focus on how newcomers join OSS projects [18, 30, 31]. Von Krogh et al. [31] propose a joining script for developers who want to take part in a project. Similarly, Nakakoji et al. [18] proposed an onion based structure (the onion patch) to explain the OSS joining process. Steinmacher et al. [30] proposed a joining model, in which they represent motivation and attractiveness as forces that influence outsiders to become newcomers to OSS project. In this study, we focus on the characteristics of the project that may explain the attraction of newcomers in terms of temporal onboarding growth.

The attractiveness topic had also been previously studied. For example, Santos et al. [22] defined a theoretical cause-effect model for attractiveness to OSS projects, proposing its typical causes (license type, intended audience, type of project, development status), indicators (hits, downloads, members), and consequences (number of open tasks, time for task completion). They found that projects for end-users and developers have higher attractiveness, that application domain impacts attractiveness, and that projects licensed under most restrictive licenses tend to be less attractive. These results contradict Colazo and Fang's [5] results, which analyzed 62 projects from SourceForge and found that restrictively licensed projects are more attractive to volunteer OSS developers.

From a different perspective, Meirelles et al. [16] applied the same model as Santos [22], inserting source code metrics as a typical attractiveness causes. They observed that structural complexity and software size (lines of code and number of modules), indicating that structural complexity negatively influences attractiveness, whereas software size positively influences it. Chengalur-Smith et al. [4]

analyzed whether codebase size, project age, and niche size (a measure borrowed from ecology) influenced project attractiveness, finding that these three characteristics indeed influence the project’s ability to attract and retain developers.

Although the attractiveness topic has been explored from different perspectives, the studies mentioned do not consider temporal growth trends in newcomer’s onboarding. Moreover, the aforementioned studies do not analyze the attractiveness after the social coding environments become commonplace in OSS development. One exception is the paper by Gupta et al. [10], who analyzed how the adoption of continuous integration impacts developer attraction. However, they analyzed this single intervention, without considering any other project characteristic.

6 Limitations

In a study such as this, there are always many limitations and threats to validity. First, we considered only a limited number of attributes to explain the phenomenon of newcomers onboarding growth. Although we understand that different attributes and different ways to compute the factors should be used, we focused on factors that cover project popularity, maturity, skills required, receptivity/welcoming features.

Second, we focused only on GitHub OSS projects—the largest OSS hosting environment to date —, which means that our findings may not generalize to other platforms with different contributing characteristics. Moreover, we diversified our sample including 30 projects for each of the 15 most popular programming languages in GitHub, which naturally increases the diversity of our sample. However, our focus on the most popular OSS projects may have influenced factors such as the “# of Stars”. Nevertheless, as depicted at Fig. 1, the selected OSS are greatly diverse, when it comes to the number of stars. Still, although large, our dataset clearly does not comprehend the whole universe of OSS projects available. Also, we did not distinguish spare time contributors from employees hired by a software company to contribute to OSS. We are aware that they may have different contributing behaviors [6], but a comprehensive analysis of their joining rate is left this for future work.

Third, the decisions regarding the observation unit (in our case, one week) can also be seen as a limiting factor. However, it is important to note that, when exploring our data, and we found similar behavior for higher time windows. Similarly, we also considered that the use of 72 weeks to represent the joining rate of newcomers may not be sufficient. However, this is a limitation of the KSC algorithm, which requires an equivalent time window for all analyzed series.

Finally, we understand that only three clusters may not represent the diversity of joining rates found in OSS projects. However, we tried to overcome this limitation by using the β_{CV} heuristic. Yet, one might argue that our cluster may not strictly follow the function curves we indicated (exponential, linear, and logarithm). However, to make sense of this, we investigated the coefficients that indicate a strong correlation with the centroids and the growth patterns.

Figure 4 shows this relationship. In each figure, there is a black line (the centroid line) and a red dotted line (the trend line that represents the function curve). As indicated, both lines follow roughly the same curve (the closer R^2 is to 1 indicates the stronger is the correlation).

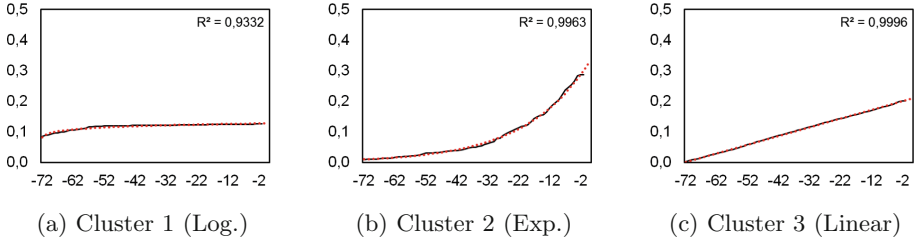


Fig. 4. Comparing the growth patterns with the default curves. (Color figure online)

7 Conclusion

In this paper, we investigated the projects' characteristics that may explain the different patterns of newcomers growth in OSS projects. Through a sequence of quantitative and statistical analyses based on data mined from 450 OSS projects, we were able to uncover several so far unknown behaviors of OSS projects. For instance, we perceived that there are three main onboard growth patterns (namely a logarithm growth pattern, a linear growth pattern, and an exponential growth pattern). Moreover, we also shed some light on the factors that might encourage newcomers to onboard on the OSS projects. The Top-3 ranked factors were: the number of stars, the time to merge a pull-request, and the number of programming languages used. For future work, we plan to leverage qualitative analysis to better understand what external reasons (e.g., new release, recently open sourced, the first page on HackerNews, etc.) might lead some projects to an exponential growth of newcomers.

Acknowledgment. This work is partially supported by CNPq (#430642/2016-4 and #406308/2016-0), Fundação Araucária and FAPESP (#2015/24527-3).

References






1. Borges, H., Valente, M.T.: What's in a GitHub star? Understanding repository starring practices in a social coding platform. *J. Syst. Softw.* **146**, 112–129 (2018)
2. Breiman, L.: Random forests. *Mach. Learn.* **45**(1), 5–32 (2001)
3. Capiluppi, A., Michlmayr, M.: From the cathedral to the bazaar: an empirical study of the lifecycle of volunteer community projects. In: Feller, J., Fitzgerald, B., Scacchi, W., Sillitti, A. (eds.) *OSS 2007*. ITIFIP, vol. 234, pp. 31–44. Springer, Boston, MA (2007). https://doi.org/10.1007/978-0-387-72486-7_3

4. Chengalur-Smith, I.N., Sidorova, A., Daniel, S.L.: Sustainability of free/libre open source projects: a longitudinal study. *J. Assoc. Inf. Syst.* **11**(11), 657–683 (2010)
5. Colazo, J., Fang, Y.: Impact of license choice on open source software development activity. *J. Am. Soc. Inf. Sci. Technol.* **60**(5), 997–1011 (2009). <https://doi.org/10.1002/asi.v60:5>
6. Dias, L.F., Steinmacher, I., Pinto, G.: Who drives company-owned OSS projects: internal or external members? *J. Braz. Comp. Soc.* **24**(1), 16:1–16:17 (2018)
7. Figueiredo, F.: On the prediction of popularity of trends and hits for user generated videos. In: *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, pp. 741–746. ACM (2013)
8. Figueiredo, F., Almeida, J.M., Gonçalves, M.A., Benevenuto, F.: On the dynamics of social media popularity: a YouTube case study. *ACM Trans. Internet Technol. (TOIT)* **14**(4), 24 (2014)
9. Gousios, G., Pinzger, M., Deursen, A.: An exploratory study of the pull-based software development model. In: *36th International Conference on Software Engineering, ICSE 2014*, pp. 345–355 (2014)
10. Gupta, Y., Khan, Y., Gallaba, K., McIntosh, S.: The impact of the adoption of continuous integration on developer attraction and retention. In: *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pp. 491–494, May 2017
11. Hartigan, J.A.: *Clustering algorithms* (1975)
12. Hauke, J., Kossowski, T.: Comparison of values of Pearson’s and Spearman’s correlation coefficients on the same sets of data. *Quaestiones geograph.* **30**(2), 87–93 (2011)
13. Ke, W., Zhang, P.: The effects of extrinsic motivations and satisfaction in open source software development. *J. Assoc. Inf. Syst.* **11**(12), 784–808 (2010)
14. Louppe, G., Wehenkel, L., Sutura, A., Geurts, P.: Understanding variable importances in forests of randomized trees. In: *Advances in Neural Information Processing Systems*, pp. 431–439 (2013)
15. Maalej, W., Happel, H.J., Rashid, A.: When users become collaborators: towards continuous and context-aware user input. In: *Proceeding of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, OOPSLA 2009*, pp. 981–990. ACM (2009)
16. Meirelles, P., Santos, C., Miranda, J., Kon, F., Terceiro, A., Chavez, C.: A study of the relationships between source code metrics and attractiveness in free software projects. In: *2010 Brazilian Symposium on Software Engineering, SBES 2010*, pp. 11–20. IEEE (2010)
17. Menasce, D.A., Almeida, V.A.: *Capacity Planning for Web Services: Metrics, Models, and Methods*. Prentice Hall PTR, Upper Saddle River (2002)
18. Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., Ye, Y.: Evolution patterns of open-source software systems and communities. In: *Proceedings of the International Workshop on Principles of Software Evolution, IWPSE 2002*, pp. 76–85. ACM, New York (2002)
19. Qureshi, I., Fang, Y.: Socialization in open source software projects: a growth mixture modeling approach. *Organ. Res. Methods* **14**(1), 208–238 (2011)
20. Ray, B., Posnett, D., Filkov, V., Devanbu, P.: A large scale study of programming languages and code quality in GitHub. In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 155–165. ACM (2014)

21. Robnik-Šikonja, M.: Improving random forests. In: Boulicaut, J.-F., Esposito, F., Giannotti, F., Pedreschi, D. (eds.) ECML 2004. LNCS (LNAI), vol. 3201, pp. 359–370. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30115-8_34
22. Santos, C., Kuk, G., Kon, F., Pearson, J.: The attraction of contributors in free and open source software projects. *J. Strategic Inf. Syst.* **22**(1), 26–45 (2013)
23. Segal, M.R.: Machine learning benchmarks and random forest regression (2004)
24. Shah, S.K.: Motivation, governance, and the viability of hybrid forms in open source software development. *Manag. Sci.* **52**(7), 1000–1014 (2006)
25. Shi, T., Horvath, S.: Unsupervised learning with random forest predictors. *J. Comput. Graph. Stat.* **15**(1), 118–138 (2006)
26. Sokolova, M., Japkowicz, N., Szpakowicz, S.: Beyond accuracy, f-score and ROC: a family of discriminant measures for performance evaluation. In: Sattar, A., Kang, B. (eds.) AI 2006. LNCS (LNAI), vol. 4304, pp. 1015–1021. Springer, Heidelberg (2006). https://doi.org/10.1007/11941439_114
27. Sokolova, M., Lapalme, G.: A systematic analysis of performance measures for classification tasks. *Inf. Process. Manag.* **45**(4), 427–437 (2009)
28. Steinmacher, I., Conte, T., Gerosa, M.A., Redmiles, D.: Social barriers faced by newcomers placing their first contribution in open source software projects. In: Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work and Social Computing, CSCW 2015, pp. 1379–1392. ACM (2015)
29. Steinmacher, I., Conte, T.U., Treude, C., Gerosa, M.A.: Overcoming open source project entry barriers with a portal for newcomers. In: Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, pp. 273–284. ACM, New York (2016)
30. Steinmacher, I., Gerosa, M.A., Redmiles, D.: Attracting, onboarding, and retaining newcomer developers in open source software projects. In: Proceedings of the Workshop on Global Software Development in a CSCW Perspective, CSCW 2014 Workshops (2014)
31. von Krogh, G., Spaeth, S., Lakhani, K.R.: Community, joining, and specialization in open source software innovation: a case study. *Res. Policy* **32**(7), 1217–1241 (2003)
32. Yang, J., Leskovec, J.: Patterns of temporal variation in online media. In: Proceedings of the Fourth ACM International Conference on Web Search and Data Mining, pp. 177–186. ACM (2011)
33. Ye, Y., Kishida, K.: Toward an understanding of the motivation open source software developers. In: 25th International Conference on Software Engineering, ICSE 2003, pp. 419–429. IEEE Computer Society, Washington (2003)



Why Do Developers Adopt Open Source Software? Past, Present and Future

Valentina Lenarduzzi¹ , Davide Tosi² , Luigi Lavazza²  ,
and Sandro Morasca² 

¹ Tampere University, Tampere, Finland
valentina.lenarduzzi@tuni.fi

² Università degli Studi dell'Insubria, Varese, Italy
{davide.tosi, luigi.lavazza,
sandro.morasca}@uninsubria.it

Abstract. Free/Libre Open Source Software has evolved dramatically in the last twenty years and many open source products are now considered similar, or even better than proprietary counterparts. Given the evolution of software – both concerning its development and its usage – it is likely that the motivations for adopting an open source rather than a proprietary product have changed over time. The goal of this work is to identify the current motivations for adopting open source software, and compare them with the motivations that held in the past. We conducted a set of interviews among software practitioners, asking them to rank motivations for the adoption of open source software, and we compared these new results with the motivations elicited in previous surveys published in 2010 and 2013. The results show that motivations have actually changed over time.

Keywords: Open source software · Free software · Adoption motivations

1 Introduction

Free and Libre Open Source Software (FLOSS) is nowadays integrated in several commercial software products. Companies commonly use FLOSS libraries and products as components, or customize FLOSS for delivering new services.

In the last ten years, several researchers have proposed FLOSS adoption models or investigated the motivations that lead to the adoption of FLOSS instead of other types of software [3, 4, 6, 8, 10, 11]. The goal of this work is to take a snapshot of the current motivations that lead companies to integrate FLOSS in their products, and to support FLOSS producers in understanding which factors their users commonly look into when they are selecting software components. We replicated the surveys published by Del Bianco et al. in 2010 [1] and Taibi in 2013 [2] by interviewing FLOSS adopters in the October 2015–December 2016 period. We interviewed 64 practitioners, to understand the actual trend of motivations that drive FLOSS adoption. Results show that motivations have changed over time and nowadays developers do not care mostly about quality, ethic and economic issues, as they did in the past, but are more interested in modifiability and professional support.

The paper is structured as follows. Section 2 describes the related work and the background of this study. Section 3 presents the new survey. In Sect. 4, we illustrate and discuss the results. Section 5 discusses the threats to validity of this work and Sect. 6 draws some conclusions and outlines future work.

2 Related Work

Previous research on the adoption of FLOSS has mainly focused on adoption models, which suggested that potential adopters take into account economic factors, license, development process, product quality, while some other work highlighted economic motivations, such as the total cost of ownership (TCO) and the return on investment (ROI) [4, 10], or technological reasons [1, 5]. Qualification and Selection of Open Source Software (QSOS) [4], Business Readiness Rating (BRR) [11], and OpenBQR [3] also consider customer related factors, such as to what degree a product satisfies customer requirements. Some evaluation models, such as the Model of Open Source Software Trustworthiness (MOSST) [7], are based on the evaluation of a set of factors, weighted according to their importance, and aim at predicting the trustworthiness of a specific FLOSS product and the likelihood of its adoption. Instead, other models are usually considered by potential users when they select a new FLOSS product [5, 9]. A few studies empirically investigated the motivations considered during the adoption of FLOSS by different organization [1, 2, 6, 8, 9, 14, 15]. In 2005, Glynn et al. highlighted personal interest and relative advantage as important factors [14].

In 2009, Del Bianco et al. provided an evidence-based models for the evaluating OSS trustworthiness based on objective measures of OSS [19, 21–23]. They collected 100 questionnaires, containing 722 product evaluations [8]. In 2007–2009, Del Bianco et al. [1] ran a survey collecting motivations for adopting FLOSS from 151 participants. Product reliability and the degree to which a FLOSS product satisfies functional requirements turned out to be the most important adoption drivers. In 2012, Del Bianco et al., while investigating marketing and communication strategies of three FLOSS producers, highlighted that personal opinion and the product websites play an important role in FLOSS adoption [6]. In 2011 Basilico [26] and Lavazza [25] proposed an OSS evaluation model to recommend OSS providers the information they should publish on their portals, based on the information required by OSS identified in [1]. The same information has been used to support developers in generating the OSS testing documentation [27], and to certify the testing process [19, 28].

In 2013, Li et al. [9] conducted a survey among 294 FLOSS adopters and 212 non-adopters in Asia, identifying as main motivations personal interest, regulations & political influence, accomplishment and experiencing stimulation emerged as relevant factors. In 2013, Taibi [2] replicated the study [1] by interviewing 38 participants. He identified 22 adoption motivations, fourteen of which had already been found in [1]. The ease of customization and ethical motivations, not included in [1], were considered the most important drivers for the adoption of FLOSS. In 2015, Yamakami [15] proposed a set of OSS migration strategies identifying cost, coordination, and development process as main adoption drivers.

In 2017, Wasserman et al. [18] presented the OSSpal model, as the successor to the BRR model [11]. OSSpal is a generic FLOSS adoption model, which aims to be applicable to any kind of user. In OSSpal, the evaluation accounts for functionality (how well the software meets the user's requirements), operations (namely, security, performances, scalability, usability, configuration and ease of maintenance), support and services, documentation availability, technology attributes (software architecture, modularity, flexibility, portability, extensibility, integration easiness, completeness, faultiness), development process. The aforementioned characteristics have been proposed as elements of a guideline for FLOSS evaluation based on the authors' experience, not elicited empirically based on what criteria companies adopt during the adoption of open source code or products.

Sbai et al. classified the information considered by the OSS adopters, focusing on the information that can be automatically extracted from different platforms [24].

3 The Replicated Study

We carried out this study to investigate the current motivations that drive practitioners when selecting a FLOSS products to be integrated in the software they develop, and to outline motivation trends in the last 6 years by comparing current motivations with those identified by previous studies [1, 2]. We formulated our goal as:

Analyze FLOSS adoption process, for the purpose of understanding, with respect to motivations from the point of view of developers, custom integrators and project managers, in the context of development companies integrating FLOSS or extending FLOSS in their software products.

It is important to notice that we considered only motivations for the selection of FLOSS that can be integrated into existing software development processes, such as libraries, components, frameworks, or any tool including IDEs and Databases and others. Standalone products used for generic purposes, such as office suites or other tools were not considered in this work.

Based on the main goal, we defined the following research questions:

RQ1: What are the most common motivations for choosing a specific FLOSS product over proprietary software?

RQ2: How did motivations evolve over time?

We followed the guidelines proposed by Carver for reporting replications [12], and we designed the study as an exploratory, descriptive survey carried out my means of a questionnaire, as a replication of previous studies [1, 2]. The survey consists of closed questions based on the results reported in [1, 2]. The interview was designed to be carried out in person, to ease communication and get a better understanding of the answers provided.

To accurately replicate the previous works, our questionnaire had the same structure of the ones used in the previous studies, and consisted of three main sections:

- **Background and Skills of Respondents.** We collected the profile of the respondents: age, country and the predominant role in the company, the experience with FLOSS products, and the level of adoption in the organizational unit.
- **Company Profile:** We collected information about the type and size of the company and industrial sector.
- **Adoption Motivations:** We asked the interviewees to rank the motivations for the adoption of FLOSS software identified in [1] and [2] based on their importance, on a scale from 0 to 10, where 0 meant “totally irrelevant” and 10 meant “fundamental”. We also invited the participants to add and rank new motivations.

As in the two previous surveys, the interviewees were not selected according to any specific criterion. We interviewed 64 developers and professionals. All the interviews were collected by the same interviewer, who also took care of considering synonyms so as to group similar motivations. During interviews, we did not provide a set of motivations; instead, we let the participants mention their own motivations and, if some of the motivations provided in [1] and [2] were not mentioned, we asked to rank their importance. The interviewer took note of the explanation of the motivation, to understand and clarify possible misunderstandings.

Before analyzing the collected responses, we partitioned them into homogeneous groups, based on demographic information. Ordinal data were not converted into numerical equivalents, since using a conversion from ordinal to numerical data entails the risk that subsequent analysis will give misleading results if the equidistance between the values cannot be guaranteed. Moreover, analyzing each value of the scale allows us to better identify the possible distribution of the answers. We ranked each answer based on the median of the importance reported in the interviews.

4 Results

As reported in Table 1, more than half of the interviewees were software developers. All the participants had experience in evaluating OSS, and have the power to decide if integrate a FLOSS component or adopt a FLOSS tool in the development process (IDE, Database, ...).

4.1 Motivations for Adopting FLOSS (RQ1)

We collected 22 different motivations. The medians of the importance expressed by respondents are given in Table 2 and Fig. 1. Results are presented for all the interviewees (column “All Participants”) and grouped by role.

Evaluations by the whole set of participants range from level 1 (least important) to level 8 (most important). For instance, Ease of Customization is ranked at level 8, so it is deemed more important than Quality and Flexibility, which are ranked at level 7.

It can be observed that there is substantial agreement between Developers and Custom Integrators, while, as could be expected, managers tend to give greater importance to economic and organizational aspects. Figure 2 shows the box plots representing the distributions of motivation importance provided by respondents. It can

Table 1. Characteristics of respondents.

Respondents' organization role	%	Company size	%
Developers	51.6	Medium-sized enterprises	53.1
Custom integrator	23.4	Large corporations	31.3
Project manager	12.5	SMEs	15.6
Project manager and developers	7.8		
Project manager and custom integrators	4.7		
<i>Organizations' industrial sector</i>	%	<i>Experience with FLOSS</i>	%
Hardware/software development	32.8	Less than 2 years	20.3
Security	12.5	Between 2 and 5 years	37.5
Finance	7.8	More than 5 years	42.2
Public administration	7.8		
Avionics	6.3		
Telecommunications	3.1		
Other domains	29.7		

Table 2. Importance of motivations for adopting FLOSS (medians) (RQ1).

Motivation	All participants	Developers	Custom integrators	Project managers
Ease of customization	8	8	8	7
Community support	8	8	8	5
Professional support	7	7	8	8
Quality	7	7	6	7
Flexibility	7	7	6	5
Maturity	7	7	7	5
Reliability	7	7	7	8
Innovation	6	6	6	3
Multiplatform Development	6	6	6	4
Partnership	5	5	5	6
Competitiveness	5	4	5	6
No vendor lock-in	5	3	5	5
Ethics	4	4	4	5
Personal productivity	4	4	4	3
Economic aspects	4	4	4	6
Freedom	4	4	4	1
Free updates	3	3	3	3
Security	2	2	3	2
Customer requirements	2	2	2	3
Training	2	2	2	1
Reuse	2	2	2	3
Imposed by the company	1	1	1	1

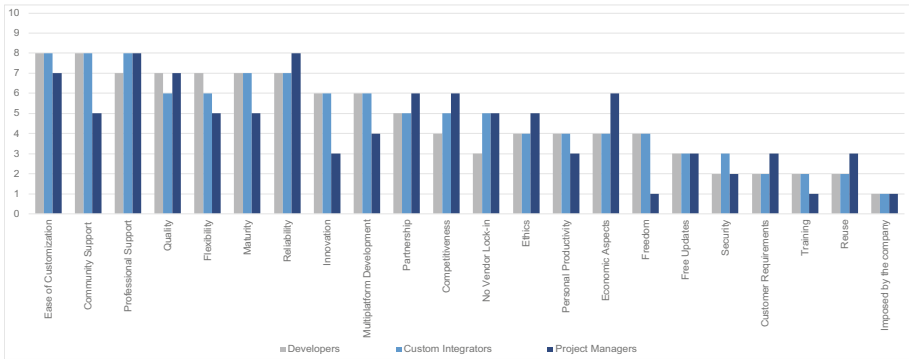


Fig. 1. Importance of motivations for adopting FLOSS in 2016 (medians) (RQ1).

be observed that there is a strong agreement among respondents on the most important motivations: for instance, the majority of the evaluations concerning Ease of Customization, Community Support, Professional Support, Quality and Flexibility were in a 2-grade range. The data in Table 2 provide the answer to our research question RQ1.

4.2 Motivations: Trend Over 6 Years (RQ2)

The results of our survey and those from previous surveys are given in Table 3. No new motivations emerged in the 2016 survey with respect to the union of those identified in the 2013 and 2010 studies. In the 2016 survey, all respondents specified the importance of all motivations previously detected, whereas in [1] and [2] respondents were free to mention and rank only the motivations they considered relevant. Hence, there are some motivations—such as Flexibility, Maturity, Ethics, etc.—that do not appear in the “2010” column, since nobody mentioned those motivations in the 2010 survey. Similarly, nobody mentioned Professional Support in the 2013 survey.

In Table 3, arrows represent changes in the importance of a motivation comparing the first survey (2010) with the last one (2016). For example, a downwards arrow shows that the importance of Reliability decreased (from 8 in 2010 to 7 in 2016). The data in Table 3 provide a first answer to our research question RQ2; however, the following observations appear useful to get a complete view of the motivations for FLOSS adoption through years.

In 2016, Developers considered Ease of Customization, and Community Support as the most important motivations, while in 2013 they considered Ethics, together with Ease of Customization, as the most important motivations; back in 2010, Customer Requirements were the main adoption driver for developers.

In 2016, Custom Integrators considered at the highest importance level also Professional Support, together with Ease of Customization and Community Support, while in 2013, Quality was considered by Custom Integrators as the most important motivation with Ease of Customization; back in 2010, Reliability was the main driver for adoption according to Custom Integrators.

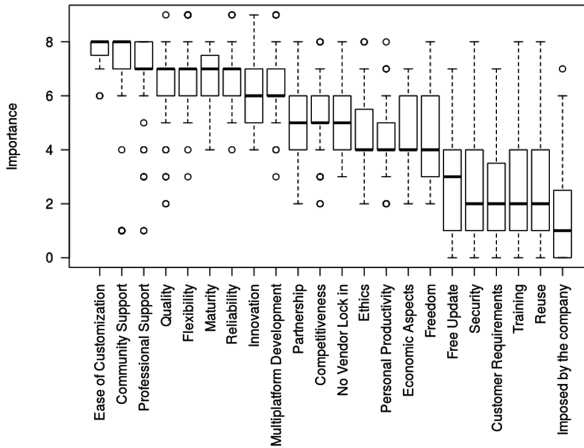


Fig. 2. Importance of motivations for adopting FLOSS in 2016: boxplots (all participants).

Table 3. Importance of motivations for adopting FLOSS (medians) (RQ1).

Motivation	All Participants			Developers			Custom Integrators			Project Managers		
	2016	2013	2010	2016	2013	2010	2016	2013	2010	2016	2013	2010
Ease of Customization	8↑	8	4	8↑	8	4	8↓	8	3	7↑	7	3
Community Support	8↑	4	6	8↓	3	5	8↑	6	6	5↓	2	6
Professional Support	7↑		5	7↑		5	8↑		5	8↑		6
Quality	7↑	6	5	7↑	6	5	6	8	6	7↑	6	6
Flexibility	7↑	2		7			6↑	2		5		
Maturity	7↑	1		7↑	1		7↑	2		5		
Reliability	7↓	1	8	7		7	7↓	1	8	8	1	8
Innovation	6↑	2		6			6↑	2		3	3	
Multiplatf. Develop.	6↓	2	4	6↑	3	4	6↑		5	4↑	2	3
Partnership	5	5		5			5↑	4		6↓	7	
Competitiveness	5↑	2		4			5↑	2		6↑	3	
No Vendor Lock-in	5↑	1	1	3↑	2	1	5↑		2	5↑	1	
Ethics	4↓	7		4↓	8		4↓	7		5↓	7	
Personal Productivity	4↓	6		4↓	7		4↓	7		3↑	1	
Economic Aspects	4↓	6	2	4↑	2	2	4	4	1	6↑	9	3
Freedom	4	4		4↓	5		4↑	3		1		
Free Updates	3↓	1	4	3↑		2	3↓		4	3↑	1	3
Security	2↓	2	5	2↓	2	4	3↓	3	5	2↓		5
Customer Reqs	2↓	1	8	2↓		8	2↓		7	3↑	1	8
Training	2↑	1	2	2		2	2↓		4	1	1	1
Reuse	2↓	1	4	2↓	2	5	2↓		5	3↑	2	4
Imposed by company	1	1	1	1	1	1	1		1	1	1	1

Finally, in 2016 Project Managers provided indications that are partly different with respect to the other roles: Professional Support and Reliability are deemed most important. In 2013, Economic Aspects were considered by Project Managers as the most important motivation, while in 2010, Reliability and Customer Requirements were their main drivers for adopting FLOSS (Fig. 3).

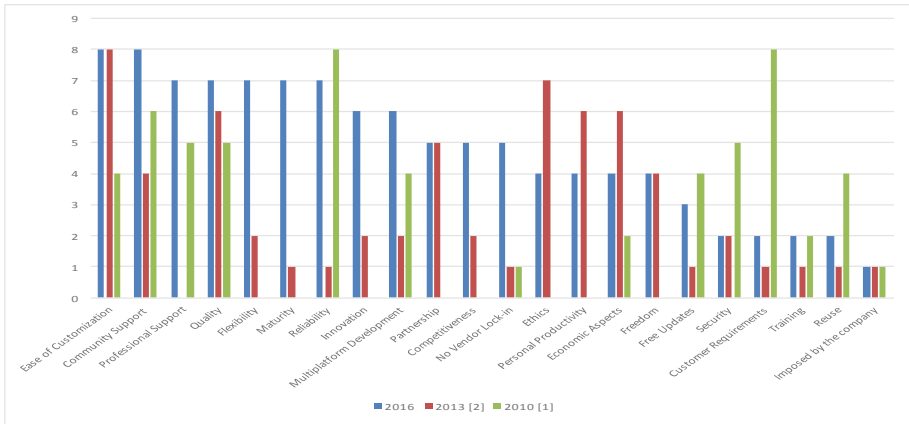


Fig. 3. Importance of motivations in 2010, 2013, and 2016 (median for all participants).

Our results confirm that – as natural and expected – Project Managers continue to focus on factors that can impact the management process of a project, while developers mainly focus on factors that affect the development phases.

As for the evolution of the motivations in the last ten years, we can see big changes from several points of views. Several motivations kept growing: for instance, the importance of Quality increased from level 5 in 2010 to 6 in 2013 to 7 in 2016. Similarly, the importance of Community Support kept growing from 2010 to 2016, resulting in one of the most important motivations in 2016. Flexibility, Maturity, Multiplatform Development, and Innovation dramatically increased their relevance in 2016 compared to 2013, not having been mentioned in 2010. Other motivations appear to have an oscillating importance: for instance, in 2010, FLOSS Reliability was among the most important adoption drivers, then its importance dropped to level 1 in 2013, and raised back at level 8 in 2016. It is very difficult to draw conclusions about these oscillating motivations.

Some motivations were constantly considered relevant: for instance, Ease of customization, Professional support, and Partnership received the same evaluation in 2013 and 2016. Some motivations' importance decreased since 2010. Other motivations, such as Training, Reuse, and Company imposition, appear definitely not relevant, having received low grades through the three surveys.

Considering role-specific evaluations, the importance of Economic aspects for managers, was very high (level 9) in 2013, but descended to level 6 in 2016, showing that the managers pay more attention to the effectiveness of the whole FLOSS-using development process, rather than to sheer costs.

4.3 Discussion

The first result of the study is that nowadays FLOSS appears to be selected by using a different approach than in 2010. The adoption drivers have changed, and economic aspects are no longer as important as in 2010 and 2013. FLOSS was initially perceived

as a free product while now it is correctly perceived as recommended by the Free Software Foundation as “free as in free speech, not as in free beer” [16]. Therefore, developers are now aware that FLOSS is not free of charge and are paying less attention to cost issues, as researchers had already predicted back in 2007 [17]. Similarly, ethical issues are no longer considered that important, probably because the ethical debate on FLOSS appears to have been settled by now.

Our interviewees preferred FLOSS since they can easily customize it, without having to deal with proprietary issues, and can provide the highest possible value to their customers. Therefore, our interviewees were highly interested in Community and Professional support, with the importance of Professional Support growing sensibly since 2010 and almost equaling Community Support. Nowadays, companies appear willing to pay for technical support from FLOSS providers – as would be the case with proprietary software – but with the freedom to access the source code and modify it. In fact, being the ease of customization a dominant motivation for adopting FLOSS, the availability of the source code is extremely important; nonetheless, having just the code is not enough: support from the community and professionals is also needed.

As expected, Quality is always considered very important by all roles, and its importance has increased over time. Other quality aspects, such as project Maturity, Reliability, and Multiplatform Development are also definitely important, thus supporting the idea that non-functional aspects of FLOSS are increasingly relevant.

Personal Productivity and potential Partnerships, which were first detected in 2013 survey, are still considered drivers of medium importance. For Personal Productivity, interviewees appear to behave as end users (as opposed to developers): they do not care for FLOSS or non-FLOSS tools, they ask for (black-box) tools and apps that help their every-day tasks. As for potential Partnerships, commercial solutions appear to be currently considered as more apt to favor the creation of business partnerships than FLOSS communities.

The results from our survey partially confirm the evaluation categories proposed by the OSSpal evaluation model [19]. OSSpal consider qualities – such as Professional and Community Support, and Ease of Customization – that ranked as important by the developers we interviewed. OSSpal also accounts for motivations considered as relevant by software end users. However, OSSpal considers several factors (such as performances and usability) that are of low importance to our interviewees, and other characteristics (such as installation and configuration easiness) never mentioned by our respondents.

5 Threats to Validity

In this section, we discuss the threats to validity and explain the adopted tactics [13].

Concerning internal validity, we identified the following issues.

Participants Selection: We selected participants with a similar background. In order to avoid any bias due to different roles, we tried to have as equal as possible frequency of roles (Developers, Custom Integrator, Managers) in the three studies. Only for Managers role we have proportionally fewer participants in the 2016 replication.

Testing: We avoided that the pre-testing (first survey) could affect the scores on the post-test, since, first we asked to the participant what they considered during the FLOSS adoption process, then, in case the answers were different from the previous surveys, we asked to express an opinion also on the motivations emerged from previous surveys.

Instrumentation: During the study we avoided changing the way data were collected and analyzed.

Design Contamination During the Different Surveys: We avoided any possible design contamination during the different surveys.

Concerning external validity, we identified the following issues.

Population Validity: The selected samples are representative enough of developers and project managers, but not enough of top management roles such as CEOs. From the results of the 2010 survey [1], we only considered the answers provided by developers and custom integrators and ignored the ones obtained from the end users.

Study Results: This survey is – at most– representative for developers using FLOSS.

Concerning reliability, in this survey, we adopted the same questionnaire used in [1] and [2]. The Questionnaire was checked by empirical studies experts.

6 Conclusions

In this paper, we investigated the motivations for the adoption of FLOSS up to 2016. In 2010, the vast majority of users was interested in getting FLOSS as-is without paying any license fee. More recent results show that ethical and economic motivations are not driving the choice of FLOSS over proprietary software: already in 2013, economic aspects and type of license were no longer considered important. New motivations, like the ease of customization, have emerged, because developers started perceiving FLOSS as means to build better products more easily.

References

1. Del Bianco, V., Lavazza, L., Morasca, S., Taibi, D.: A survey on open source software trustworthiness. *Software* **28**, 67–75 (2011)
2. Taibi, D.: An empirical investigation on the motivations for the adoption of open source software. In: 10th International Conference on Software Engineering Advances - ICSEA, Barcelona (2015)
3. Taibi, D., Lavazza, L., Morasca, S.: OpenBQR: a framework for the assessment of OSS. In: Feller, J., Fitzgerald, B., Scacchi, W., Sillitti, A. (eds.) OSS 2007. ITIFIP, vol. 234, pp. 173–186. Springer, Boston, MA (2007). https://doi.org/10.1007/978-0-387-72486-7_14
4. Origin, A.: Method for qualification and selection of open source software (QSOS). Version 1.6. <https://www.qsos.org>
5. Buffett, B.: Factors influencing open source software adoption in public sector national and international statistical organizations. In: Meeting on the Management of Statistical Information Systems (MSIS 2014), Dublin, Ireland and Manila, Philippines (2014)

6. del Bianco, V., Lavazza, L., Lenarduzzi, V., Morasca, S., Taibi, D., Tosi, D.: A study on OSS marketing and communication strategies. In: Hammouda, I., Lundell, B., Mikkonen, T., Scacchi, W. (eds.) OSS 2012. IAICT, vol. 378, pp. 338–343. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33442-9_31
7. del Bianco, V., Lavazza, L., Morasca, S., Taibi, D.: Quality of open source software: the QualiPSo trustworthiness model. In: Boldyreff, C., Crowston, K., Lundell, B., Wasserman, A.I. (eds.) OSS 2009. IAICT, vol. 299, pp. 199–212. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02032-2_18
8. Del Bianco, V., Lavazza, L., Morasca, S., Taibi, D.: The QualiSPo approach to OSS product quality evaluation. In: Workshop on Emerging Trends in FLOSS Research and Development (FLOSS-3), pp. 23–28 (2010)
9. Li, Y., Tan, C.H., Xu, H., Teo, H.H.: Open source software adoption: motivations of adopters and amotivations of non-adopters. ACM SIGMIS Database **42**, 76–94 (2011)
10. Lenarduzzi, V.: Towards a marketing strategy for open source software. In: 12th International Conference on Product Focused Software Development and Process Improvement, pp. 31–33 (2011)
11. Business Readiness Rating for Open Source - A Proposed Open Standard to Facilitate Assessment and Adoption of Open Source Software, BRR 2005 RFC 1. <http://www.openbrr.org>
12. Carver, J.: Towards reporting guidelines for experimental replications: a proposal. In: 1st International Workshop on Replication in Empirical Software Engineering Research (2010)
13. Yin, R.K.: Case Study research. Applied Social Research Methods Series. Design and Methods. Sage Publications, London (2009)
14. Glynn, E., Fitzgerald, B., Exton, C.: Commercial adoption of open source software: an empirical study. In: International Symposium on Empirical Software Engineering, pp. 17–18. (2005)
15. Yamakami, T.: Open source software adoption patterns and organizational transition stages for software vendors. In: Conference on Information Sciences and Interaction Sciences (2010)
16. GNU: What is free software? <https://www.gnu.org/philosophy/free-sw.html>
17. Lavazza, L.: Beyond total cost of ownership: applying balanced scorecards to open-source software. In: 2nd International Conference on Software Engineering Advances – ICSEA (2007)
18. Wasserman, A.I., Guo, X., McMillian, B., Qian, K., Wei, M.-Y., Xu, Q.: OSSpal: finding and evaluating open source software. In: OSS 2017 (2017)
19. Morasca, S., Taibi, D., Tosi, D.: OSS-TMM: guidelines for improving the testing process of open source software. Int. J. Open Source Softw. Process. **3**(12), 1–22 (2011)
20. Lavazza, L., Morasca, S., Taibi, D., Tosi, D.: An empirical investigation of perceived reliability of open source Java programs. In: Symposium on Applied Computing - SAC (2012)
21. del Bianco, V., Lavazza, L., Morasca, S., Taibi, D., Tosi, D.: An investigation of the users' perception of OSS quality. In: Ågerfalk, P., Boldyreff, C., González-Barahona, J.M., Madey, G.R., Noll, J. (eds.) OSS 2010. IAICT, vol. 319, pp. 15–28. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13244-5_2
22. Lavazza, L., Morasca, S., Taibi, D., Tosi, D.: Predicting OSS trustworthiness on the basis of elementary code assessment. In: ESEM 2010 (2010)
23. Taibi, D., del Bianco, V., Carbonare, D.D., Lavazza, L., Morasca, S.: Towards the evaluation of OSS trustworthiness: lessons learned from the observation of relevant OSS projects. In: Russo, B., Damiani, E., Hissam, S., Lundell, B., Succi, G. (eds.) OSS 2008. ITIFIP, vol. 275, pp. 389–395. Springer, Boston, MA (2008). https://doi.org/10.1007/978-0-387-09684-1_37

24. Sbai, N., Lenarduzzi, V., Taibi, D., Sassi, S.B., Ghezala, H.H.B.: Exploring information from OSS repositories and platforms to support OSS selection decisions. *Inf. Softw. Technol.* **104**, 104–108 (2018)
25. Lavazza, L., Morasca, S., Taibi, D., Tosi, D.: OP2A: how to improve the quality of the web portal of open source software products. In: Filipe, J., Cordeiro, J. (eds.) *WEBIST 2011*. LNBIIP, vol. 101, pp. 149–162. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28082-5_11
26. Basílico, G., Lavazza, L., Morasca, S., Taibi, D., Tosi, D.: Op2A: assessing the quality of the portal of open source software products. In: *WEBIST 2011* (2011)
27. Morasca, S., Taibi, D., Tosi, D.: T-DOC: a tool for the automatic generation of testing documentation for OSS products. In: Ågerfalk, P., Boldyreff, C., González-Barahona, J.M., Madey, G.R., Noll, J. (eds.) *OSS 2010*. IAICT, vol. 319, pp. 200–213. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13244-5_16
28. Morasca, S., Taibi, D., Tosi, D.: Towards certifying the testing process of open-source software: new challenges or old methodologies? In: *Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development* (2009)



Why Do People Give Up FLOSSing? A Study of Contributor Disengagement in Open Source

Courtney Miller^{1(✉)}, David Gray Widder^{2(✉)}, Christian Kästner^{2(✉)},
and Bogdan Vasilescu^{2(✉)}

¹ New College of Florida, Sarasota, USA
courtney.miller17@ncf.edu

² Carnegie Mellon University, Pittsburgh, USA
{dwidder,kaestner,vasilescu}@cmu.edu

Abstract. Established contributors are the backbone of many free/libre open source software (FLOSS) projects. Previous research has shown that it is critically important for projects to retain contributors and it has also revealed the motivations behind why contributors choose to participate in FLOSS in the first place. However, there has been limited research done on the reasons why established contributors disengage, and factors (on an individual and project level) that predict their disengagement. In this paper, we conduct a mixed-methods empirical study, combining surveys and survival modeling, to identify the reasons and predictive factors behind established contributor disengagement. We find that different groups of established contributors tend to disengage for different reasons; however, overall contributors most commonly cite some kind of transition (e.g., switching jobs or leaving academia). We also find that factors such as the popularity of the projects a contributor works on, whether they have experienced a transition, when they work, and how much they work are all factors that can be used to predict their disengagement from open source.

1 Introduction

Contributor disengagement in open source is widely known as a costly and critical issue [9, 19, 49], as it can directly affect the sustainability of projects. For example, in a recent study Coelho et al. reported that 41% of failed open source projects cited a reason involving the developer team, such as lack of interest or time of the main contributor [9]. Such local (project-level) sustainability issues in open source can have cascading effects on the entire ecosystem because of project interdependencies [12, 53]. So-called “core”, i.e., established, contributors are particularly critical for the sustainability of open source projects [19, 57].

C. Miller—Part of this work was carried out during the author’s REU program at CMU.

There are many reasons why established contributors disengage. Some may be unavoidable, whereas others could perhaps be prevented through interventions or by providing better community support. Likely there are various dynamics in play, including the role of volunteers as compared to corporate employees [44], the role of external events such as family planning and job changes, and the role of perceived purpose, community support, and stress. Effects might include abruptly leaving the project, but also slow disengagement, or causing ripping frustrations through delays or cynicism.

The goal of our research is to better understand disengagement factors and which established contributors are at risk and when; this will enable us to build and validate a conceptual framework and theory. Moreover, we pursue a data-driven approach, operationalizing uncovered factors based on publicly available trace data. This way, we can identify at-risk open source contributors and communities, and help guide resources (e.g., volunteers, sponsors) toward projects and contributors in need, enhancing the sustainability of the overall ecosystem.

We identify potential disengagement factors from literature on turnover and open source retention, cross-validate them with results from a survey among contributors who recently stopped all open source activities on GitHub, operationalize select factors with public trace data, and finally conduct survival modeling among a set of 206 GitHub users to triangulate the survey results.

Among others, we identify the degree to which contributors work outside of typical office hours and to what degree they engage in support activities as important moderating factors. According to Claes et al. [8], 33% of open source contributors do not follow typical working hours, but instead work nights and weekends. Our survey shows that contributors who work nights and weekends proportionally tend to disengage for different reasons than those working regular hours. In addition, our survey reveals that the most common reasons for complete disengagement relate to transitions in employment, such as graduating from academia, changing employers, and changing roles.

To validate disengagement factors beyond our survey, we model to what degree hypothesized factors—such as working hours, engagement in support activities, and team size, which can be measured in public trace data of contributor activities—can predict the later disengagement of those contributors. To that end, we use the quantitative statistical method of survival modeling. As a key factor in our model, derived from our survey results, we incorporate transitions identified from public CVs of developers. Specifically, we analyze which contributor populations are more resilient to transitions such as job changes.

We find that working predominantly during office hours and experiencing a transition both increase a contributors risk of disengagement. Conversely, we find that increased levels of activity and working on more popular projects both decrease a contributors risk of disengagement.

In summary, we contribute (1) a survey revealing the reasons behind contributor disengagement; (2) a comparison between different groups of contributors; (3) measures to differentiate between groups, which could be used to help identify at-risk groups and better target support interventions; (4) a novel operational-

ization of transition data; and (5) a survival model demonstrating which factors are able to predict contributor disengagement.

2 Related Work

Turnover. Prior work has shown that the turnover rate of a project profoundly affects its survival probability [33, 46] and code quality [21]. Approximately 80% of open source projects fail due to contributor turnover related issues [46]. Even within projects that do not outright fail, contributor turnover has a significant adverse effect on software quality [21]. On a project level, contributor disengagement results in knowledge loss, which is a particularly expensive issue [33].

Employee turnover and retention have been broadly studied across many fields [31, 35]. In professional settings, early turnover research has focused often on personal characteristics (e.g., ability, age) and employee satisfaction, measured with hiring tests and surveys, whereas later research has explored many more nuanced factors, such as labor market (e.g., job opportunities), non-work values, and organizational commitment [31]. Research has shown that, while far from all turnover can be explained by dissatisfaction and similar factors [38], there are positive and negative factors that can buffer against shocks such as external job offers [6, 20]. Turnover among volunteers is less explored: Although some research suggests that similar personal and environmental factors influence their decisions to quit [41], other researchers point out that satisfaction and achievement, compatible working hours, training, challenging work, and role identity may play particularly strong roles [25, 34, 50].

Whereas reasons for joining open source [5, 24, 37, 44, 48, 54, 55] and interventions to improve the onboarding experience for new developers [7, 18, 30, 52] have been studied in depth, studies of contributor retention are rarer. Prior research has focused primarily on testing basic attributes [11, 39, 40, 46, 49, 53, 58]. For example, they have shown that retention is higher for contributors that have participated longer [39, 49], contributed more code changes [11, 39], and communicated more [11]. However, there has been a limited amount of prior research has also explored more nuanced factors, like whether a developers gender and social network effect their risk of disengagement [43]. Using surveys, researchers further associated ratings of general dissatisfaction and lack of community identification with higher perceived turnover and turnover intentions [32, 56]. Zhou et al.’s case study of three projects further suggests that commercial participation can crowd out volunteers [58].

Long working hours, lack of sleep, and lack of recovery on weekends are often discussed as stressors. Many studies confirm the importance of “mentally switching off” [1, 4, 51]. In software engineering, several studies have shown the influence of time-related factors, such as late-night commits and long working sessions being more likely to contain bugs [17, 45], sleep deprivation reducing code quality [22], Monday commit comments using more negative language [29], and time pressure is often seen as an important stressor [36].

Open Source Practitioners Reporting Stress. In addition to the academic literature, open source practitioners also spoke out about frustrations, funding concerns, stress, and even burnout. Often, there are high expectations and copious amounts of pressure placed on established open source contributors.

Many stories via blog posts from maintainers who disengaged have a similar narrative that describes the growing pressures and responsibilities they experienced that lead to their disengagement. One such blog post describes how “as [my project’s] popularity rose and rose, my drive to continue to create new projects, fell. All while the burden of supporting the needs of the massive user bases of my successful projects and the pressure of maintaining those projects grew.”¹

In addition to blog posts, there were also participants from the survey we ran who explicitly cited a lack of support as a reason for their disengagement. For example “[The open source project] is increasingly depended upon by other projects, but very few external developers are interested/willing enough to [understand the company] let alone contribute improvements/fixes. The support burden is a good problem to have (people are finding [the project] useful), but it does impose a productivity (and sometimes a motivation) burden.” (P35)

Contributors are broadly expected to maintain their projects. Having a seemingly never-ending list of tasks is another commonly cited reason for disengagement among the aforementioned blog posts and survey respondents. As described in a blog post by a now-retired developer, “working long hours for endless months” was a critical reason for their disengagement.²

3 Overview: Mixed-Method Research

Our mixed-method empirical study follows a sequential exploratory design [14], combining qualitative and quantitative analysis of survey and GitHub trace data.

Step 1: Survey (Sect. 4). Although the turnover literature (Sect. 2) provides several starting points for potential disengagement factors, there has been only limited research on the actual reasons *why* open source contributors disengage. Therefore, we decided to ground our research by conducting an open-ended survey among developer who recently disengaged from all public GitHub activities. We furthermore analyze the frequency of self-reported reasons for disengagement regarding whether different populations disengage for different reasons.

Step 2: Survival analysis (Sect. 5). We test to what degree the potential disengagement factors identified statistically explain disengagement. To that end, we operationalize several disengagement factors, including when and what contributors worked on as well as job transitions in historic trace data and public CVs, and use survival modeling [42] to test their significance.

¹ <https://www.kennethreitz.org/essays/the-reality-of-developer-burnout>.

² <https://hackernoon.com/what-is-programmer-burnout-651aa48984ef>.

4 Self-reported Reasons for Disengagement (Survey)

4.1 Survey Methodology

To ground our analyses, we surveyed a sample of open source contributors who recently disengaged from all public GitHub activities, asking about their reasons.

Recently Disengaged Established Contributors. We invited open source contributors who stopped all public activity on GitHub after being active for at least 18 month. We identified such contributors from GHTorrent [26] trace data (version 2018-08). We then constructed six-month panels aggregating contributions (commits and issue/pull request events) per person, and selected those contributors who contributed at least 100 commits per six-month period for three consecutive periods, but at most 5 commits in the following period (the five commit threshold allows for some residual activity). This way, we identified a total of 702 contributors who disengaged (i.e., stopped contributing publicly) within the last year and had public email addresses listed on their GitHub profile pages.

We specifically sampled only previously active contributors with at least 100 commits per period across all of GitHub. Previous research has shown that within a single project, there are many different kinds of contributors, with one of the most popular models being the onion model [15]. With our threshold we target contributors who are likely very active in at least one project, rather than more peripheral or episodic contributors, which may have different motivations [2].

Survey Design. We designed a simple, single-question, open-ended survey, asking “*Could you help us understand your reasons for reducing your contributions to GitHub projects?*” We chose the open-ended format to avoid priming the participants to ensure organic but relevant responses. We use the single-question format without external survey software, because it reduces the barrier to participation. We invited all 702 identified candidates and received 151 valid answers (21.5% response rate). Our response rate is in line with other GitHub surveys, e.g., [27].

Card Sorting Analysis. We used card sorting, a qualitative content analysis [47] method, to analyze the survey answers. Two researchers reviewed the cards and organized them into mutually agreed upon categories using a ground-up process resulting in 17 subgroups. These subgroups were then further grouped into three overarching themes: Technical, Social, and Occupational. Note that many participants cited multiple reasons, resulting in 239 reasons from 151 responses.

Quantitative Analysis. In addition to identifying common self-reported reasons for disengagement from the survey responses, we additionally explore whether different populations report different reasons. Based on the literature and reports from open source practitioners (cf. Sect. 2), we specifically investigate whether contributors (a) working mostly “regular” office hours or (b) performing more support activities report disengaging for different reasons.

Working Hours: Analyzing GitHub data, we measure what percentage of contributions are made between 7am and 7pm local time, Monday through Friday, captured as *indexWorkHours* (the slightly wider interval than the traditional 9am to 5pm increases robustness to daylight savings [8]). To detect the contributor’s local time, we adjusted the UTC times in GHTorrent with the average time zone offset for each developer, collected from a small random sample of their commits after cloning repositories locally. We then separate our survey participants into two groups, *Office Hours* (more likely paid contributors) and *Nights and Weekends* (more likely volunteers), based on whether they perform more or less relative amount in the office hour window described above than average (average *indexWorkHours* = 0.6; design following prior research [39]).

Support Activity: We also measured *indexSupport* as the percentage of support activities among all activities, i.e., all non-commit GHTorrent events related to managing issues and pull requests. We distinguish between *High Support Work* and *Low Support Work* relative to the mean (*indexSupport* = 0.2).

Note that given the different ways in which we aggregate the survey responses and the relatively small sample size overall, we cannot draw sound statistical conclusions about differences between the (sub)groups. While we report exact numbers, readers should focus on qualitative differences.

Threats to Survey Validity. As usual for surveys, our results may be affected by a selection bias: contributors who did not answer may have had different

Table 1. Self-reported reasons for disengagement in survey

Subgroup	Count	Office Hrs vs Nights&We	More Support vs Less
<i>Occupational reasons</i>			
Got new job that doesn't support FLOSS	37		
Changed role/project	25		
Left job where they contributed to FLOSS	16		
No time: new job	15		
No time: existing job	10		
Left school where they contributed to FLOSS	12		
No time: in school	12		
FLOSS in school, now job doesn't support FLOSS	7		
Too much coding at work	4		
<i>Social reasons</i>			
Lost interest in FLOSS	24		
No time: personal	23		
Lack of peer support	16		
No time: nondescript	15		
<i>Technical reasons</i>			
Issues w GitHub or industry	14		
Individually moved to private repos	12		
Changed platform	10		
Feature complete project	3		

reasons for disengaging. To identify contributors who had disengaged, we used public GitHub data, which covers much but not all open source activities, as also visible in 10 (of 151) survey responses that indicate changing platforms. Deriving the survival model data from survey participants enabled modeling only contributors confirmed to have disengaged. Note that we consider moving to private repositories (12 answers) still as disengagement from public open source activities. Furthermore, our approach to identify disengagement looks for sudden disengagement (within a six-month window) and results may not generalize to contributors who disengage more gradually. Contributors may also deliberately or unconsciously self-censor in their answers, providing socially acceptable reasons rather than real—a common concern in turnover research [31]. Note however, that our survival model (discussed later) is built entirely on historic trace data rather than self-reported answers, and thus reduces this threat.

4.2 Results from Survey

In Table 1, we show the survey results. The most common self-reported reason for disengagement was changing jobs to a job that does not support open source work and occupational reasons were generally the most frequent.

Furthermore, we observe differences across populations: *Contributors who work nights and weekends tend to disengage for different reasons than those who work during office hours*: contributors who worked nights and weekends most commonly cited social reasons, whereas those who worked during office hours most commonly cited occupational reasons; the largest difference is between those who cited *Left job where they contributed to OSS*, with 19% and 0% citing it respectively.

Next, we turn to the aggregation by type of work, noting *Contributors who do less support work tend to disengage for different reasons than those who do more*: In particular, only 67% of the *More Support Work* group cited at least once Occupational reason, compared to 72% of the *Less Support Work* group. The difference between these two groups may be because since they are less stressed when major life changes occur (i.e., getting a new job or leaving school), they are better able to cope with transitions.

Finally, we emphasize a surprising result. For all contributors, occupational reasons such as major life changes (e.g., getting a new job or leaving school) were the most cited (with 106 citations), significantly more than lacking peer support or losing interest that are more commonly discussed in the literature. This motivated us to consider transitions explicitly in our survival analysis below.

5 Modeling Disengagement Factors (Survival Analysis)

5.1 Survival Model Methodology

We use survival analysis to triangulate the survey results and model the relative strengths of the effects of the three main factors emerging from the survey

analysis on the risk of disengagement from public GitHub activity (*Work Hours vs Nights and Weekends*; *High Support Work vs Low Support Work*; and *Job Transitions*). Survival analysis is a statistical modeling technique that specializes in time-to-event data [42], particularly suited for modeling right censored data. In our study, the event is public GitHub *disengagement*; right censorship can occur for contributors whose last recorded event may be very close to the end of the observation period, for which it is not clear whether they will return to contribute more. In particular, we use a Cox Proportional Hazards regression model [13]. The estimated regression coefficients describe each variable’s *hazard ratio* (HR), which is analogous to an odds ratio in for multiple logistic regression analysis. Briefly, an $HR > 1$ indicates an increased risk of observing the event, and an $HR < 1$ indicates a decreased risk, relative to a one unit change in a predictor variable (or flipping the value, in case of binary variables), while holding all other predictors constant.

Data. We collect GitHub data on several variables for the open source contributors who disengaged and responded to our survey (the ‘treatment’ group), as well as for an equal sized ‘control’ group of contributors who did not disengage. With this design, a survival model estimates which factors are statistically useful for distinguishing groups.

For job transition data, we collect publicly available CV data from contributors by following links on their GitHub profiles. Since our data collection is not yet fully automated, we can currently only assemble a dataset of moderate size, therefore we only collected data for our survey participants (plus the control group), because their survey answers validate that they actually disengaged. For non-CV data, we use GHTorrent (Sect. 4). We discard 34 participants for which we cannot find CVs or similar information from which we can deduce past transitions, leaving us with a dataset of 206 contributors of which 103 disengaged. By construction, both groups contributed actively for 18 months (at least 100 commits per six-month period for three consecutive periods; Sect. 4); the ‘control’ group contributors then remained active for at least another six months at similar levels or higher, while the ‘treatment’ group contributors made at most five commits in the following period, i.e., they *disengaged*.

Model Factors and Operationalization. We compute:

- *Activity level:* Prior work has shown that more active contributors are less likely to disengage [11], hence we control for the average quarterly activity level by counting all activities (commits and support) per person.
- *Working hours and support:* We use the two factors *indexWorkHours* and *index-Support* as introduced in Sect. 4.1 to characterize the degree of work outside regular working hours (more likely volunteers) and the degree of support activities, both identified as stressors by practitioners (cf. Sect. 2). We compute dummy variables indicating being above or below the mean.
- *Organizational affiliation:* Previous research has shown that on a project scale, having an organizational affiliation can help increase developer retention rates [58]. We test whether organizational affiliation has the same affect

on engagement on an individual scale as it does on a project scale. Using GHTorrent, we record whether contributors had an Organizational Affiliation listed on their GitHub public profile.

- *Team size*: Turnover research regularly reveals social embedding in a team as an antidote to turnover [19]. We operationalize this as the number of contributors per project. Since a contributor may be part of multiple projects, we consider only their main projects (for a contributor, taking all projects with the highest number of contributions that together constitute at least 50% of all contributions) and record the average team size among those projects. ‘Teams’ comprise everyone who authored at least one commit.
- *Project popularity*: To control for whether contributors are more likely to disengage from small or very popular projects, we use the number of stars a project has on GitHub as a proxy for its popularity (standard measure in GitHub research [16]). We model popularity in addition to activity level because previous research has shown that the popularity of a project influences its survival probability [53], and we are interested in whether the popularity of a project also affects the survival probability of its contributors on an individual level. For contributors working on multiple projects, we consider the max popularity of the contributor’s active projects (see team size).
- *Transition found*: Finally, to operationalize a contributor’s transition data, identified as very important in our survey, we went to their linked publicly available CV and created a binary variable that recorded whether there was a transition present in the last year or not. We considered a transition to be either the stopping or starting of a job or educational program.

Model Diagnostics. We performed the standard model diagnostics: We log transformed variables with highly skewed distributions, as necessary, to reduce heteroscedasticity [23]. We tested for multicollinearity using the variance inflation factor ($VIF < 3$) [10]. We also inspected Schoenfeld residual plots to graphically diagnose Cox regression modeling assumptions [28].

Threats to Model Validity. Regarding the survival model, statistical power is limited by the small sample size, which is limited by our design of modeling only survey participants with public CV data (due to confirming disengagement with the survey and manual effort required, as discussed). Since our treatment group was limited to the survey respondents, our survival model also has the risk of suffering from selection bias. As usual, our operationalization of factors in our survival model can only capture part of the concept to be measured. While we experimented with different operationalizations of our factors to ensure construct validity and robustness, one needs to be careful in generalizing our results beyond our specific operationalizations.

5.2 Results from Survival Modeling

Table 2 presents the results from the two survival models created; a base model without the novel transition found variable, and a full model with.

Table 2. Survival models for contributor disengagement.

	Base model		Full model	
activity	0.36 (0.21)***	29.00***	0.36 (0.21)***	27.92***
orgAffiliation	0.90 (0.21)	0.27	0.92 (0.21)	0.17
maxTeamSize	1.17 (0.08)	3.59	1.17 (0.08)	3.41
maxNumStars	0.85 (0.05)**	10.01**	0.86 (0.05)**	9.08**
highSupportWork_TRUE	1.29 (0.26)	0.96	1.43 (0.27)	1.74
workHours_TRUE	1.56 (0.21)*	4.52*	2.20 (0.30)**	5.59*
jobTransition			2.48 (0.31)**	8.15**
workHours:jobTransition			0.55 (0.42)	
R ²	0.21		0.25	

*** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$

The *base model* had a goodness of fit of $R^2 = 0.21$. The controls behave as expected. *Total activity* had a hazard rate of 0.36, meaning it decreases a contributor’s risk of disengaging by a factor of 0.38. Similarly, contributors who work on more popular projects are less likely to disengage (*Max number of stars* has a hazard ratio of 0.85).

As predicted based on previous research, the *workHours* dummy affects a contributor’s risk of disengaging, having a hazard ratio of 1.56. This suggests that working during business hours more than the average contributor increases the risk of disengaging by a factor of 1.56. Surprisingly, we do not observe any statistically significant effects of doing more support work than average (the *highSupportWork* dummy), perhaps due to our operationalization or relatively small sample size.

The *full model* fits the data better ($R^2 = 0.25$), meaning that adding in the *jobTransition* variable helped increase the explanatory power of the model. The *jobTransition* variable has a hazard ratio of 2.48, meaning, as suggested by the survey results, that experiencing a transition significantly increases a contributor’s risk of disengagement by a factor of 2.48.

6 Discussion and Conclusions

In this research, we have looked at the reasons why established open source contributors disengage, using a survey with 151 responses and a survival model to quantify factors which predict disengagement. From the grouped analysis of survey results, we learned that the *Nights and Weekends* and *Office Hours* groups tend to cite different reasons for their disengagement, and so do more the *Less Support Work* and *More Support Work* groups.

Importantly, our study shows that operationalizations of different disengagement risk factors using publicly observable trace data are plausible. For example, since occupational reasons were the most commonly cited, we used online public CVs to operationalize the *jobTransition* variable; however, other commonly

cited reasons from the survey may also be operationalizable. Another commonly cited reason was ‘no time, personal circumstance’, more specifically people often cited having children or getting married. Such circumstances may be observable on social networking platforms. This suggest that a data-driven systems could be developed to help identify at-risk groups on a significantly larger scale, instead of having to rely on relatively expensive survey data. This information could be useful to different stakeholders, such as open source foundations and other funding agencies, looking to target support interventions. Overall, support interventions targeted more appropriately could significantly increase the sustainability of open source ecosystems.

We aim to work on these extensions of the research and more, to better understand the reasons why different kinds of established contributors disengage, since defining the problem is the first step to solving it [3].

Acknowledgements. This work was supported through CMU’s REU in SE, NSF (1318808, 1552944, 1717022, and 1717415), and AFRL and DARPA (FA8750-16-2-0042). We thank our survey participants, and colleagues at CMU, especially Jim Herb-
sleb, Chris Bogart, Marat Valiev, and Sophie Rosas-Smith.

References

1. Bannai, A., Tamakoshi, A.: The association between long working hours and health: a systematic review of epidemiological evidence. *Scand. J. Work Environ. Health* **40**, 5–18 (2014)
2. Barcomb, A., Kaufmann, A., Riehle, D., Stol, K.-J., Fitzgerald, B.: Uncovering the periphery: a qualitative survey of episodic volunteering in free/libre and open source software communities. *IEEE Trans. Softw. Eng.* (2018)
3. Bardach, E., Patashnik, E.M.: *A Practical Guide for Policy Analysis: The Eightfold Path to More Effective Problem Solving*. CQ Press, Washington D.C. (2015)
4. Binnewies, C., Sonnentag, S., Mojza, E.J.: Recovery during the weekend and fluctuations in weekly job performance: a week-level study examining intra-individual relationships. *J. Occup. Org. Psychol.* **83**(2), 419–441 (2010)
5. Bonaccorsi, A., Rossi, C.: Comparing motivations of individual programmers and firms to take part in the open source movement: from community to business. *Knowl. Technol. Policy* **18**(4), 40–64 (2006)
6. Burton, J.P., Holtom, B.C., Sablinski, C.J., Mitchell, T.R., Lee, T.W.: The buffering effects of job embeddedness on negative shocks. *J. Vocat. Behav.* **76**(1), 42–51 (2010)
7. Canfora, G., Di Penta, M., Oliveto, R., Panichella, S.: Who is going to mentor newcomers in open source projects? In: *Proceedings of International Symposium Foundations of Software Engineering (FSE)*, p. 44. ACM Press, New York (2012)
8. Claes, M., Mäntylä, M.V., Kuutila, M., Adams, B.: Do programmers work at night or during the weekend? In: *ICSE*, pp. 705–715. ACM (2018)
9. Coelho, J., Valente, M.T.: Why modern open source projects fail. In: *ESEC/FSE*, pp. 186–196. ACM (2017)
10. Cohen, P., West, S.G., Aiken, L.S.: *Applied Multiple Regression/Correlation Analysis for the Behavioral Sciences*. Psychology Press, London (2014)

11. Constantinou, E., Mens, T.: An empirical comparison of developer retention in the RubyGems and NPM software ecosystems. *Innov. Syst. Softw. Eng.* **13**(2–3), 101–115 (2017)
12. Constantinou, E., Mens, T.: Socio-technical evolution of the Ruby ecosystem in GitHub. In: SANER, pp. 34–44. IEEE (2017)
13. Cox, D.R.: *Analysis of Survival Data*. Routledge, Abingdon (2018)
14. Creswell, J.W., Clark, V.L.P.: *Designing and Conducting Mixed Methods Research*. Wiley, Hoboken (2007)
15. Crowston, K., Annabi, H., Howison, J., Masango, C.: Effective work practices for software engineering: free/libre open source software development. In: *Proceedings of the 2004 ACM Workshop on Interdisciplinary Software Engineering Research*, pp. 18–26. ACM (2004)
16. Dabbish, L., Stuart, C., Tsay, J., Herbsleb, J.: Social coding in GitHub: transparency and collaboration in an open software repository. In: *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, pp. 1277–1286. ACM (2012)
17. Eyolfson, J., Tan, L., Lam, P.: Do time of day and developer experience affect commit bugginess? In: *Proceedings of the 8th Working Conference on Mining Software Repositories*, pp. 153–162. ACM (2011)
18. Fagerholm, F., Guinea, A.S., Borenstein, J., Münch, J.: Onboarding in open source projects. *IEEE Softw.* **31**(6), 54–61 (2014)
19. Fang, Y., Neufeld, D.: Understanding sustained participation in open source software projects. *J. Manag. Inf. Syst.* **25**(4), 9–50 (2009)
20. Feldman, D.C., Ng, T.W.H.: Careers: mobility, embeddedness, and success. *J. Manag.* **33**(3), 350–377 (2007)
21. Foucault, M., Palyart, M., Blanc, X., Murphy, G.C., Falleri, J.-R.: Impact of developer turnover on quality in open-source software. In: *ESEC/FSE*, pp. 829–841. ACM (2015)
22. Fucci, D., Scanniello, G., Romano, S., Juristo, N.: Need for sleep: the impact of a night of sleep deprivation on novice developers' performance. *IEEE Trans. Softw. Eng.* (2018)
23. Gelman, A., Hill, J.: *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press, Cambridge (2006)
24. Ghosh, R.A., Glott, R., Krieger, B., Robles, G.: Free/libre and open source software: survey and study - part 4: survey of developers. Technical report, International Institute of Informatics, University of Maastricht, Maastricht (2002)
25. Gidron, B.: Predictors of retention and turnover among service volunteer workers. *J. Soc. Serv. Res.* **8**(1), 1–16 (1985)
26. Gousios, G.: The GHTorrent dataset and tool suite. In: *MSR*, pp. 233–236. IEEE (2013)
27. Gousios, G., Zaidman, A., Storey, M.-A., Van Deursen, A.: Work practices and challenges in pull-based development: the integrator's perspective. In: *ICSE*, pp. 358–368. IEEE (2015)
28. Grambsch, P.M., Therneau, T.M.: Proportional hazards tests and diagnostics based on weighted residuals. *Biometrika* **81**(3), 515–526 (1994)
29. Guzman, E., Azócar, D., Li, Y.: Sentiment analysis of commit comments in GitHub: an empirical study. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*, pp. 352–355. ACM (2014)
30. Hannebauer, C., Gruhn, V.: On the relationship between newcomer motivations and contribution barriers in open source projects. In: *Proceedings of International Symposium on Open Collaboration, OpenSym 2017*, pp. 2:1–2:10. ACM, New York (2017)

31. Hom, P.W., Lee, T.W., Shaw, J.D., Hausknecht, J.P.: One hundred years of employee turnover theory and research. *J. Appl. Psychol.* **102**(3), 530 (2017)
32. Homscheid, D., Schaarschmidt, M.: Between organization and community: investigating turnover intention factors of firm-sponsored open source software developers. In: *Proceedings of Conference Web Science (WebSci)*, pp. 336–337. ACM, New York (2016)
33. Izquierdo-Cortazar, D., et al.: Using software archaeology to measure knowledge loss in software projects due to developer turnover. In: *HICSS*, pp. 1–10. IEEE (2009)
34. Jamison, I.B.: Turnover and retention among volunteers in human service agencies. *Rev. Publ. Pers. Adm.* **23**(2), 114–132 (2003)
35. Kim, H., Kao, D.: A meta-analysis of turnover intention predictors among us child welfare workers. *Child. Youth Serv. Rev.* **47**, 214–223 (2014)
36. Kuutila, M., Mäntylä, M.V., Claes, M., Elovainio, M.: Reviewing literature on time pressure in software engineering and related professions: computer assisted interdisciplinary literature review. In: *2017 IEEE/ACM 2nd International Workshop on Emotion Awareness in Software Engineering (SEmotion)*, pp. 54–59. IEEE (2017)
37. Lakhani, K., Wolf, R.G.: Why hackers do what they do: understanding motivation and effort in free/open source software projects. Technical report, MIT Sloan Working Paper (2003)
38. Lee, T.W., Mitchell, T.R.: An alternative approach: the unfolding model of voluntary employee turnover. *Acad. Manag. Rev.* **19**(1), 51–89 (1994)
39. Lin, B., Robles, G., Serebrenik, A.: Developer turnover in global, industrial open source projects: insights from applying survival analysis. In: *2017 IEEE 12th International Conference on Global Software Engineering (ICGSE)*, pp. 66–75. IEEE (2017)
40. Midha, V., Palvia, P.: Retention and quality in open source software projects. In: *AMCIS 2007 Proceedings*, p. 25 (2007)
41. Miller, L.E., Powell, G.N., Seltzer, J.: Determinants of turnover among volunteers. *Hum. Relat.* **43**(9), 901–917 (1990)
42. Miller Jr., R.G.: *Survival Analysis*, vol. 66. Wiley, Hoboken (2011)
43. Qiu, H.S., Nolte, A., Brown, A., Serebrenik, A., Vasilescu, B.: Going farther together: the impact of social capital on sustained participation in open source. In: *International Conference on Software Engineering, ICSE*. IEEE (2019)
44. Roberts, J.A., Hann, I.-H., Slaughter, S.A.: Understanding the motivations, participation, and performance of open source software developers: a longitudinal study of the Apache projects. *Manag. Sci.* **52**(7), 984–999 (2006)
45. Rodriguez, A., Tanaka, F., Kamei, Y.: Empirical study on the relationship between developer's working habits and efficiency. In: *Proceedings of Conference on Mining Software Repositories (MSR)* (2018)
46. Schilling, A., Laumer, S., Weitzel, T.: Who will remain? An evaluation of actual person-job and person-team fit to predict developer retention in FLOSS projects. In: *HICSS*, pp. 3446–3455. IEEE (2012)
47. Schreier, M.: *Qualitative Content Analysis in Practice*. Sage Publications, Thousand Oaks (2012)
48. Shah, S.K.: Motivation, governance, and the viability of hybrid forms in open source software development. *Manag. Sci.* **52**(7), 1000–1014 (2006)
49. Sharma, P.N., Hulland, J., Daniel, S.: Examining turnover in open source software projects using logistic hierarchical linear modeling approach. In: Hammouda, I., Lundell, B., Mikkonen, T., Scacchi, W. (eds.) *OSS 2012*. IAICT, vol. 378, pp. 331–337. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33442-9_30

50. Skoglund, A.G.: Do not forget about your volunteers: a qualitative analysis of factors influencing volunteer turnover. *Health Soc. Work* **31**(3), 217 (2006)
51. Sonnentag, S., Binnewies, C., Mojza, E.J.: Staying well and engaged when demands are high: the role of psychological detachment. *J. Appl. Psychol.* **95**(5), 965 (2010)
52. Steinmacher, I., Conte, T., Gerosa, M.A.: Redmiles, D.: Social barriers faced by newcomers placing their first contribution in open source software projects. In: *Proceedings of Conference on Computer Supported Cooperative Work (CSCW)*, pp. 1379–1392. ACM, New York (2015)
53. Valiev, M., Vasilescu, B., Herbsleb, J.: Ecosystem-level determinants of sustained activity in open-source projects: a case study of the PyPI ecosystem. In: *ESEC/FSE*. ACM (2018)
54. Von Krogh, G., Haefliger, S., Spaeth, S., Wallin, M.W.: Carrots and rainbows: motivation and social practice in open source software development. *MIS Q.* **36**(2), 649–676 (2012)
55. West, J., Gallagher, S.: Challenges of open innovation: the paradox of firm investment in open-source software. *R&D Manag.* **36**(3), 319–331 (2006)
56. Yu, Y., Benlian, A., Hess, T.: An empirical study of volunteer members' perceived turnover in open source software projects. In: *Proceedings of Hawaii International Conference on System Sciences (HICSS)*, pp. 3396–3405. IEEE (2012)
57. Zhou, M., Mockus, A.: Developer fluency: achieving true mastery in software projects. In: *ESEC/FSE*, pp. 137–146. ACM (2010)
58. Zhou, M., Mockus, A., Ma, X., Lu, Z., Mei, H.: Inflow and retention in OSS communities with commercial involvement: a case study of three hybrid projects. *ACM Trans. Softw. Eng. Methodol. (TOSEM)* **25**(2), 13 (2016)

FLOSS Cost and Licences



Open Source for Open Source License Compliance

Oliver Fendt and Michael C. Jaeger^(✉)

Siemens AG, Corporate Technology,
Otto-Hahn-Ring 6, 81379 Munich, Germany
{oliver.fendt,michael.c.jaeger}@siemens.com

Abstract. Today, many software systems are of a level of complexity that no single company can implement modern solutions alone. Thus many companies engage in the open source software (OSS) ecosystem to keep the development costs manageable. But the usage of third-party components (both OSS and commercial) also mandates the need of a license compliance process supported by suitable tools. This paper is focused on using open source tools and relevant processes for open source license compliance. OSS license compliance is a very important topic, and requires appropriate processes, culture, and tools.

This work is based on extensive practical industrial experience and broad use at Siemens AG. We first describe the process and culture, then a set of tools. We complement this with related work in the community and future directions.

Keywords: License compliance · License scanning · Component inventory · Open source management

1 Elements for an OSS Management Process

The clearing of components and involved licenses is part of an OSS management process that covers the handling of 3rd party software. As described in [1] the following main goals have to be achieved by the OSS management process:

- Assurance that only suited components are approved for integration – after the involved licensing has been determined and understood.
- Assurance of license requirement fulfillment – determining the involved licenses and understanding involved terms has the purpose to actually implement those license requirements in order to provide a compliant product or delivery.
- Storing and tracking of OSS components. An organization takes advantage from keeping track of 3rd party software use: on one hand, it serves the purpose of documentation if questions or inquiries arise about OSS usage for example. On the other hand, an organization wants to prevent redundant clearing work and reuse clearing results for future uses of the same component.

The management process requires different elements in an organization for its implementation. On one side there are organizational aspects, which can be responsibilities, roles, contact persons and a decision board. These aspects are for example summarized

by the OpenChain (<https://www.openchainproject.org>) standard which describes which basic organization elements should be present.

And of course, a culture that is “open” to the use of open source is a key element of any open source management. Mentioning our organization as an example, we have regular internal events on OSS and a broadly-used web-based training educating all employees not only in software development related roles, but also beyond e.g. procurement and product management. Since its introduction in 2015, several ten thousands have attended the Web-based training so far. As an example for a dedicated role with decision responsibility, “third-party software experts” of the various business units regularly meet to discuss and agree on common approaches, best practices and challenging cases of 3rd party software usage.

In addition to company-internal activities described here, we also actively engage in world-wide activities of the OSS community. Such engagement is not only limited to re-using OSS components, and re-using OSS clearing results, but also other aspects such as agreeing on standard formats like SPDX (<https://spdx.org/>) and joining activities like the OpenChain projects that promote a high level of license clearing processes and enable sharing of cleared components.

This paper focuses on the use of tools and services which are available as open source from the community for implementing license compliance. Despite commercial tools being available, our work shows that OSS tools can be adopted by organizations and provide an effective and open approach. Among many advantages, OSS software allows for modification and adaptation to own needs; can better use the latest innovative approaches from the OSS community; do not require the establishment of a commercial contractual relation; and it allows for using software without spending monetary resources, which is an advantage also for non-commercial organization.

An open source management process needs to count on different artifacts for a successful implementation. These include a tool for analyzing the licenses present in 3rd party software components, a database that holds license interpretation of used licensing, a catalogue application that keeps track of 3rd party software in products and services, an application that keeps track of the progress of clearing tasks as well as providing an overview of all involved 3rd party component w.r.t. clearing status, a source code and component and repository that stores used software for analyses and reference to clearing results, an application which generates the licensing documentation for distribution as well as for internal approval purpose, product OSS code collector, and a code verifier.

The management of software component takes place inside the software engineering process – there are software repositories, dependency management systems and packing tools which already create various software artifacts for distribution. Naturally, the open source management tools must be integrated with the already existing software engineering facilities, for example continuous integration tools.

Figure 1 shows a general setup and context of software management, including basic elements for a software development tool chain. On the left there are different sources for 3rd party software as well as information about them. All these elements are outside of the organization and are relevant sources for OSS management. In addition to inbound 3rd party software, public repositories and databases holding information about 3rd party software components exist which can hold relevant information for component and license clearing.

In the middle of this diagram, the integration of OSS management tools with existing software engineering tools is outlined: internal repositories and tools for building the software are integrated with clearing tools. Organizational internal repositories for software exist that contain information used for component clearing, but more importantly, checks and analyses tool for a clearing are triggered by the build and software production automation.

In basic terms, the input for managing 3rd party components, originates from the software building infrastructure. Going further right, the software is prepared for distribution which involves the generation of distribution documentation as well as checks if the license terms are fulfilled. Around all this, organization internal repositories, such as internal git servers, but also artifact servers are not only relevant for software development, but also for OSS management. In additional, a central element is a component catalogue application which acts as a central inventory capturing component usage. It is fed by analysis information of 3rd party components as well as usage information in products and services.

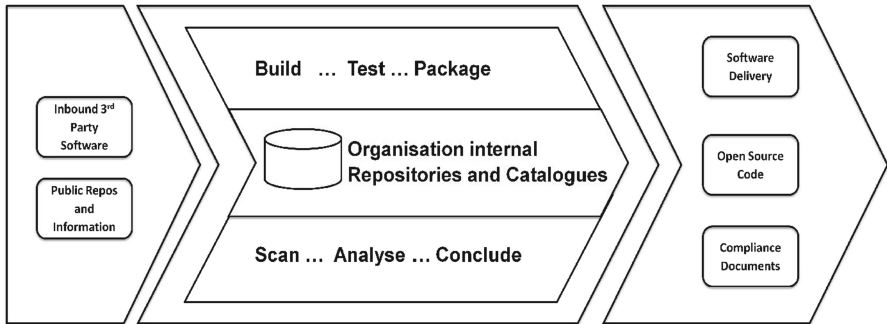


Fig. 1. Context of 3rd party software management

2 OSS Software Projects and Tools

For the elements depicted in the Fig. 1, different OSS tools exist for the implementation of the given elements. We would like to introduce a minimal set for implementing a OSS management process with the goals given above. The first part on the right hand side refers, of course, to OSS published software (e.g. Sourceforge, or nowadays Github or git servers of OSS foundations), but also software package servers, such as those for Linux distributions as well as those for developing software for a particular languages (e.g. Nuget for C# components). In addition, software can be obtained from commercial vendors and as such, represents also inbound 3rd party software.

As a second element that an organization can use are the now growing offerings for information about 3rd party software. Some companies offer concluded licensing information, some analyses and metadata describing how sane the software project is. Moreover, public information relevant for an OSS management process are license (text) collections and interpretations about licenses. A license collection is very useful

for identifying license texts as found in OSS. As for the interpretation of licenses, there are also libraries in the form of offering interpretations for licenses, as well as information pages about articles that discuss licenses cases from a legal point of view.

There are a number of open source tools for various aspects of open source license clearing and management. In our organization, the two main OSS tools we use for this are FOSSology (<https://www.fossology.org/>, <https://github.com/fossology/fossology>) and SW360 (<https://eclipse.org/sw360>, <https://github.com/eclipse/sw360>). FOSSology is a Linux Foundation project which has a meanwhile over 10 year project history providing a solution for license scanning [2]. It has a very precise license scanning facility that allows us to identify licenses as well as copyright information well. SW360 is a project hosted by the Eclipse foundation. It provides both a web application and a repository to collect, organize and make information available about software components.

For the “Assurance of license requirement fulfillment” step in the license clearing process all of the OSS licenses of the approved component have to be thoroughly examined by FOSSology to clearly identify the requirements (license obligations) to fulfill as well as to define the ways of fulfilling the obligations. For the “Storage and tracking of OSS components” step we use SW360. With it, the usage of the approved components are typically registered and uniquely tracked for future reference and reuse. The goal is not only to have a database of already approved OSS components with all their associated information (e.g. suitability ranking, copyright holders, applicable licenses, set of requirements the products have to fulfill derived from the license situation, etc.). An additional goal is also to provide a means for internal (and external) knowledge sharing on how to use, integrate and analyze the component.

Figure 2 shows a high level overview of an integrated compliance tool chain, which represents also a more detailed diagram compared to Fig. 1. On the left side, the incoming software is depicted. For successful use of OSS, also contributions to the project should be considered. The right side shows the deliverables, products, or the software which the organization conveys. In the middle part, a collection of elements is shown which play a role license compliance and OSS management. The connection of all the elements together, integrated with build infrastructure provides us with a compliance tool chain.

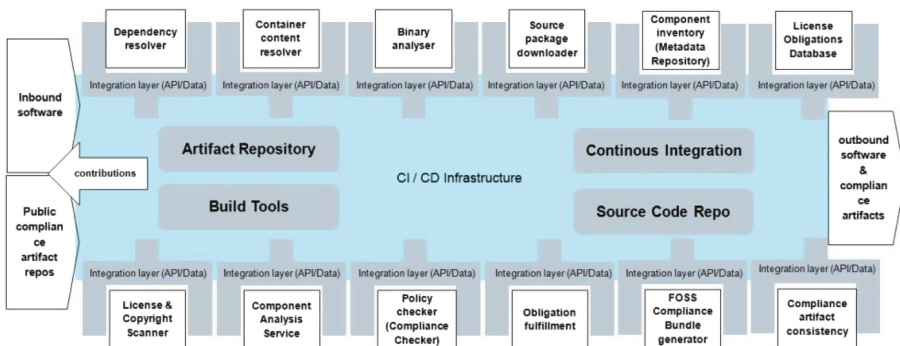


Fig. 2. Overview of an integrated compliance tool chain

3 Related Work

We are part of a vibrant and growing community in the area of open source tools of open source license compliance. Numerous members of this community presented at a forum organized by the German BITKOM organization [3] and the community strives to find ways to further foster the ecosystem of OSS-based tools for OSS license compliance and related topics. In addition to our own participation (FOSSology, SW360), other tools include Quartermaster (<https://qmstr.org>, <https://github.com/qmstr/qmstr>) and OSS Review Toolkit (<https://github.com/heremaps/oss-review-toolkit>).

More relevant OSS tools and activities include: ScanCode, another OSS license clearing tool, Tern which analysis containers for their used OSS for compliance (<https://github.com/vmware/tern>), and ClearlyDefined (<https://clearlydefined.io>, <https://github.com/clearlydefined>). ClearlyDefined and its hosting foundation, the Open Source Initiative, are on a mission to help OSS projects by being, clearly defined in terms of compliance relevant information. According to ClearlyDefined, a lack of clarity around licenses and security vulnerabilities reduces engagement and this can lead to fewer user, contributors and a smaller community. Although not being a tool, the Software Heritage project (<https://www.softwareheritage.org/>) is an important part of the OSS ecosystem. Ideally, it could be a central source that the various tools could use as the central repository of the OSS source code which would be of significant benefit to the entire community.

4 Summary and Future Work

Modern software engineering seems to be increasingly use OSS including many of the aspects that have been a hallmark of OSS – transparency, improvement, sharing, and collaboration. As open source becomes more and more prevalent in our products, and also in the tools that help create those products, it is logical to also increasingly use open source tools and processes to do OSS license compliance. The open source community on this topic is active and includes well-established tools with a long history such as FOSSology, but also a set of numerous other tools as well. We encourage the community to continue to work together to further extend the scope and the increased use of these tools in practice.

We are firmly convinced that an open source approach is the best way to able to keep up with the fast and ever-faster changing software world. Some of the future directions and areas for future work for the international OSS community are:

- Reuse of (a subset of) clearing results across an external ecosystem,
- License compliance in the context of continuous delivery/DevOps ecosystem identification of dependencies, automatic download of the source code packages of the used packages (incl. dependencies), license analysis – and licenses determination, copyrights and automatic generation of license compliance artifacts. To realize a fully functional DevOps setup the license compliance as well as cyber-security processes need to be seamless integrated in the environment,
- Enhancing automatic container analysis for license compliance with the goal to identify all applicable licenses.

References

1. Fendt, O., Jaeger, M.C., Serrano, R.J.: Industrial experience with open source software process management. In: IEEE COMPSAC (2016)
2. Jaeger, M.C., et al.: The FOSSology project: 10 years of license scanning. *Int. Free Open Source Softw. Law Rev.* **9**(1), 9 (2017)
3. Forum Open Source 2018 - BITKOM 2018. <https://www.bitkom.org/-Themen/Technologien-Software/Open-Source/Forum-Open-Source-2018.html>



Opportunity Costs in Free Open-Source Software

Siim Karus^(✉) 

University of Tartu, 51009 Tartu, Estonia
siim.karus@ut.ee

Abstract. Opportunity cost is a key concept in economics to express the value one misses out on when choosing one alternative over another. This concept is used to explain rational decision making in a scenario where multiple mutually exclusive alternative choices can be made. In this paper, we explore this concept in the realm of open-source software. We look at the different ways for measuring the cost and these can be used to support decisions involving open-source software. We review literature on opportunity cost use in decision support in software development process. We explain how the opportunity cost analysis in the realm of open-source software can be used for supporting architectural decisions within software projects. We demonstrate that different measures of costs can be used to mitigate problems (and maintenance complexity) arising from the use of open source software, allowing for better planning of both closed-source commercial and open-source community projects alike.

Keywords: Code churn · What-if analysis · Scenario analysis · Coding effort · Alternative cost · Opportunity cost · Software cost

1 Introduction

A large subdivision of software engineering research is focused on software estimation. Software estimation is concerned with devising and validating models that can be used to predict or estimate some aspect of software. These estimation models can be used for detecting trends in software development or to support decision making process or planning processes in software development. As such, the aim of many of these models is to provide decision makers with comparable values for different alternatives. Despite much effort put into evaluating and improving the accuracy of different models [1], the applicability of these models for actual comparison of alternatives is often overlooked. Hereby, we aim to collect the different approaches to measure cost in software development and review the approaches' applicability for comparing alternative actions.

In order to accomplish the set task, we perform an analytical review of objective comparison options. That is, we give emphasis on data-driven options instead of subjective expert (or community) opinion or evaluation. As a limitation, the options are only readily applicable on open source software, which makes relevant data available. We start by reviewing options for estimating the cost of software development. This is then complemented with an overview of literature reporting opportunity cost or alternative cost use in software engineering. In particular, we look for the following aspects:

1. What is the measure of the cost used?
2. Is the estimation model objective? That is: can the estimation be made solely based on the data?
3. Does the literature confirm the generalisability of the estimation model on different projects?
4. Does the literature report the confidence and accuracy of the model?

The first aspect is for classification of the approaches and identification of different measures of cost used in software engineering. The second and the third aspect describe the models' independence from specialized knowledge of experts that might not be available or have comparable skill level to the experts used in the original studies. As such, objective data-based models that have been evaluated on multiple projects are preferred due to their wider applicability and higher theoretical reproducibility. The last aspect evaluates whether the model can be used for comparison of alternatives in general. It is important to know the accuracy of the estimations in order to assess, whether two estimations are significantly different from one another. If two estimations differ less than the estimation error at given confidence level (or, to a limit, to given p-value), then the estimations can not be considered significantly different from each other.

The literature overview is followed by examples of how decision making in or involving free open-source software setting can benefit from the comparison of alternatives based on the estimation models. We highlight the challenges of extending the process of comparing alternatives to closed-source commercial software development (as opposed to FOSS) and list the main limitations of opportunity cost analysis. Finally, we conclude the study by listing opportunities for the future.

2 Measures of Cost

Cost is an amount that has to be paid or given up in order to get something [2]. In business, cost is usually a monetary valuation of (1) effort, (2) material, (3) resources, (4) time and utilities consumed, (5) risks incurred, and (6) opportunity forgone in production and delivery of a good or service [2]. In software projects, cost can be measured using several units. In commercial environments, the value of different goods is made comparable by introducing a conversion rate for all goods into a common good - currency. Consequently, currency is commonly used to evaluate costs.

2.1 Monetary Value

Even though the monetary value is most commonly used for cost measurement, it has many variations. A contemporary good cost estimation model is supposed to take into account any development and delivery effort until a release of the software [3]. Sometimes the monetary cost is also considered to include sales effort and expenses, efforts to adjust to changes, maintenance efforts, operational efforts and expenses, brand-related turnover, or other expenses or revenue opportunities. As such, the different monetary cost estimation methods can lead to very different estimations and need to be evaluated separately.

In free open-source software development, the monetary cost is usually of negligible interest, as the product is not sold for profit. In some extreme cases the projects may be sponsored, in which case the monetary dimension comes to play as the product needs to keep the sponsors interested and the sponsorship is an opportunity to allow higher expenses in the project.

2.2 Effort (Person-Hours)

As noted before, cost is usually measured in monetary evaluation. However, cost can be measured using other metrics as well. As the cost of workforce is highly volatile, a proxy of monetary cost, effort in worktime (usually person-hours or person-months) is used instead [1]. This measure is much more stable as the cost is measured directly and therefore does not differ as much in time and space as currency [3]. Nevertheless, such cost is highly dependent on the individual assigned to the task, as different developers can need different amount of time to complete the same task. To make the situation even more complicated, reliable measuring of time spent on task is very difficult. As software development requires planning and substantial mental work, it is not limited to a location or registered by automatic means. Consequently, time measurements are often based on self-reporting of the developers, which is not very accurate.

2.3 Time (Calendar)

The release cycle and release timings have become an important aspect of software development. It is the need to publish early that competes with the time available for development and testing [4, 5]. Consequently, the value measured by the cost can be the earliness of the next release. The cost does not have to be linear or even monotonic in relation to calendar time as certain dates or weekdays can be considered more valuable or less valuable than others.

2.4 Size

Sometimes the amount of code is used as a measure of cost. The size reflects the amount of effort put into developing the software – at least in the actual writing effort. Nevertheless, sometimes smaller size can be more valuable as it means less code to go through and to understand. As such, the size of the software can have a positive or negative value depending on the evaluator's objective.

2.5 Code Churn

Code maintenance effort is often evaluated based on code churn. Whilst code churn can be measured in several different units from modules and files to tokens and symbols, it is most commonly measured in lines of code (LOC). Code churn in lines of code is calculated as the sum of the number of lines of code added, modified and deleted between two revisions of a software module [6]. Some studies only count lines that are written in certain programming language and affect program logic. However, all source code needs to be maintained.

In addition to providing an indicator of code maintenance effort, code churn has also been shown to be correlated with software defects [7, 8]. In those cases, code churn is usually aggregated over a certain period of time (commonly a year [9–11]).

Naturally, code churn estimates over a certain period of time show how stable is the code-base. Projects with stable code-base can be considered more mature as there is less on-going development and fewer bug fixing going on. Thus, in situations where high reliability and low maintenance costs are important, projects with low code churn estimates would be generally preferable. Of course, this is on the premise that the stability of the project is not confused with the abandonment of the project.

Code churn is often estimated based on object-oriented metrics of the project [9, 12, 13]. Nevertheless, process-based metrics and organisational metrics have been shown to provide far better results [14–16].

2.6 Other Cost Measures

In addition to these more common cost measures, sometimes the relevant and substantial costs can be of very specific or difficult to measure kinds. For example, costs of brand, morale or public opinion can be detrimental. When discussing technical debt, effects towards monetary cost are always to be considered together with potentially longer term effects on morale, productivity, risk, and quality [17]. The practical limitation for uniform opportunity cost assessment is that some of these costs (e.g. morale) are very difficult to quantify. In those cases, the opportunity cost evaluations will need to be multi-dimensional in order to take account of all the different costs and scales they can be measured on.

3 Opportunity Cost in FOSS

The comparison of alternative actions requires the valuation of the outcomes from making a decision. The basis for such rational value-based decision-making in economics is opportunity cost. **An opportunity cost** (also known as **alternative cost**) is a benefit, profit, or value of something that must be given up to acquire or achieve something else [2]. That is, an opportunity cost can be expressed as:

$$\text{Opportunity cost} = \text{value of an option not chosen} - \text{value of chosen option.}$$

Table 1. Search results and topics. Search results shows ‘number of results from keywords “opportunity cost” “software”’ + ‘number of results from keywords “alternative cost” “software”’.

Database	IEEE Xplore	ACM Digital Library	ScienceDirect	Total
Search results	13 + 4	8 + 3	51 + 11	90
Software engineering articles	10	6	19	35
Relevant articles	4	2	2	8
Topics	Maintenance, testing, planning	Testing, planning	Testing, planning	
Measures	Monetary, effort	Monetary	Monetary	
Data-based articles	1	0	0	1

In rational value-based decision-making, the chosen option should always be the highest-valued option. As such, one can look at decision-making as an optimisation process.

Even though the formula is simple, the evaluation of alternatives rarely is. The main complexity stems from the fact that the “value” of an alternative is often difficult to define precisely. The cost estimation models are generally focusing just a few of the measures of the true value of an option (e.g. some models try to estimate monetary gain or loss, others focus on delivery times), which might not even be directly relatable to one-another. Consequently, if multiple factors apply the optimal decision can be found using Pareto optimisation [18]. This also means that the best option may have sub-optimal value in some respect. For example, faster delivery date may be more important than lower monetary expense for some projects.

We searched three major repositories of research relating to software engineering: IEEE Xplore¹, ACM Digital Library² and ScienceDirect³. As keywords, we used “opportunity cost” and “alternative cost”. The search was refined by adding a keyword “software” to both searches to eliminate the bulk of studies not relating to software. Nevertheless, the results still contained largely studies that did not discuss topics relating to software development as studies discussing opportunity cost in other fields commonly listed the software packages used for opportunity cost estimation and comparison. In summary, the search from the databases yielded 90 results with only 35 relating to software development processes⁴. The 55 results were rejected based on the review of titles and abstracts, which revealed these to be discussing opportunity costs in agriculture, medicine, (non-software) economics, or other fields. The remaining 35 papers were manually reviewed to filter out papers where opportunity costs were discussed only in reference (e.g. as future work, introduction or only in abstract) or did not relate to decision-making in any way (e.g. algorithm optimisation). This left us with eight papers (four from journals, four from proceedings with the earliest from 1990 and latest from 2016) relevant to our current focus. The review also identified additional related keyword “options analysis”, but searching using this keyword did not reveal additional articles of relevance to our study. The statistics of the search results can be found in Table 1.

We found three studies discussing opportunity costs in the planning of software development activities: [19, 20], and [21]. All of these studies were conducted in commercial environment and had monetary cost included in the analysis. Spinola et al. defined indexes that can be used for balancing effort with monetary cost when assessing resource (developers) reallocation options [19]. Whilst the paper shows a nice

¹ <https://ieeexplore.ieee.org/>.

² <https://dl.acm.org/>.

³ <https://www.sciencedirect.com/>.

⁴ We also looked into Google Scholar search, but found its tools for filtering the results too limiting for practical use. The original search keywords would get about 40,000 hits due to the popularity of “opportunity cost” in many fields. Unfortunately, Google Scholar does not provide means to limit search to specific fields and by adding exclusion keywords for more frequent non-related fields, we could limit the search results down to about 4000 before hitting the limitations of Google Scholar query complexity.

example of how to derive the indexes needed for cost computation and comparison, the practical application of the analysis process would require expert input in order to identify appropriate index values. Similarly, Cai et al. proposes a framework that can be used to justify architectural decisions by minimising the risks and monetary costs of potential changes [21]. Özogul et al., on the other hand, presents a case study of a formula used to evaluate alternative investments in a hospital information system development and operation [20]. The investment value estimation is based on sub-estimates from experts or past experience.

Papers relating to software testing were the most common among the software engineering papers on opportunity cost: 4 out of 8 studies looked at aspects of testing. Two of these were trying to answer the question of when should one stop testing and release the software: [4] and [5]. Both of these looked at monetary cost balance of the cost of testing (and fixing bugs found during testing) and the cost of resolving bugs in released software. In [4], the authors try to improve a previously developed model for assessing the optimal time to stop testing by adding measures to handle uncertainty of the values of some components of the model. In overall, their model is robust but rather simplified with only the efficiency of the testing team (that is, how fast do they detect bugs) left to the experts to evaluate. [5] offers a more detailed model for calculating the costs and introduces the concept of patching to the model. The other two articles look into prioritization of tests in order to balance testing expenses with the risks from missed bugs [22], and finding the balance between the number of automated test cases and the number of human test executions [23].

Finally, one paper discussed opportunity cost in software maintenance: [24]. In there, the authors were looking at the opportunity costs from the inner-company client's perspective. They concluded that in their case study the costs for the IT department were not affected by the time bugs remained open (the "lead time"). However, they did find that it does influence the opportunity costs of the users of the information systems and could lead to sub-optimal distribution of costs within a company. As the analysis was performed on actual historic data, it does not rely on the availability and skill of experts, which makes it stand out of all the papers on opportunity cost reviewed here.

Even though "opportunity cost" is a term rarely used in current software engineering research, a properly reported study of "cost estimation" can be used for assessing opportunity costs as well. This is an advantage as "cost estimation" is a popular topic with a combined total of over more than 2000 search hits in the three databases used in this study.

4 Opportunity Cost in Decision Making

Opportunity cost can be used in several different decision-making scenarios. For example, it is useful for picking the platforms or libraries for use in a project potentially avoiding the need to reverse such decisions (e.g. reverting from Angular JS due to maintenance costs incurred by its fast pace of changes); or evaluation of options for refactoring a project's component and/or dependency structure. Hereby we give examples on how opportunity cost analysis can be used in these scenarios depending on the available cost measure.

4.1 Choosing a Product

Let us assume a scenario where a project manager needs to choose, which software library or framework to use in the project. There are several approaches to solving this task depending on the aim of the project. Given two or more alternative options, the project manager can make a preliminary choice of one library that seems most suitable (alternatively, he could choose to not use a third party library). Based on the choice, the project manager can calculate the costs of adopting alternative libraries. If the opportunity cost of adopting an alternative library is favourable, the project manager can change the choice to become the better alternative.

The monetary opportunity cost should be easy to understand – positive opportunity cost means that the monetary gains from adopting the alternative library outweigh the gains from the current choice. Accordingly, negative monetary opportunity cost favours current choice.

The opportunity cost in effort (person-hours) is a bit more complex to act upon. As effort is by its nature a loss (the team spends effort) and the opportunity cost is defined as difference of gains, the opportunity cost in effort is the negative of the difference of the team's efforts. Thus, a positive opportunity cost in effort means that the alternative requires less work than the current choice. This means that if the project is mostly voluntary, the minimisation of effort might be preferred (thus, the choice should be changed if opportunity cost is positive). In some cases, both the effort and monetary costs need to be considered together as separate dimensions. This can be a case if there are license or infrastructure costs linked to the adoption of libraries. We highlight that the comparison is not linear as the difference in monetary cost can be insignificant if the budget is high or the monetary cost difference can be limiting due to budget limitations. Thus, the monetary cost component tends to behave in a stair-like fashion rather than linear fashion.

The opportunity cost in time is even more complicated to calculate, as the value of the timing of actions is rarely monotonic. For example, optimal release opportunities tend to depend on the weekly and monthly cycles. In addition, special external events or deadlines from clients may make certain periods (e.g. releasing consumer-oriented services ahead of major holidays) more favourable than others. The same applies to the team members' ability to react to other team members' changes in the source code.

As mentioned in the description of the measure of size, size can be either a positive or a negative attribute of the software. Consequently, it needs to be handled accordingly when calculating the opportunity cost.

As code churn is an indicator of maintenance effort, it can provide interesting insights into the quality and development stage of the library. A high code churn estimation will indicate that the library will be actively developed or maintained. A low estimation means that the module is either stable or abandoned. Consequently, when looking to adopt a more stable library, one would prefer libraries with lower code churn estimation (code churn is a negative feature). If one would like to use more changing and agile libraries, high code churn would be preferred (code churn is a positive feature). In general, commercial companies focused on reliability are more interested in using stable libraries and frameworks in their projects [25].

This example of choosing a library or a framework can be extended to a scenario of choosing a software application from multiple alternatives. For example, software with lower estimated code churn can be expected to be more stable than other alternatives.

4.2 Refactoring Options

Software component maintainability can be an important factor when deciding how to refactor the software. The refactoring options can include replacement of a component, integration of a component into other source code, or preservation and continued maintenance of the component. Using a framework for assessing a module's or component's quality (e.g. via a method proposed by Upadhyay et al. [26]), will give general quality assessment on the module, but does not give an accurate assessment of the cost of replacing or integrating the component as the maintenance of software components is highly dependent on one-another (as shown by Mari et al. [27]). Understanding this, we can devise a following method for assessing the alternative cost of a software module.

Let us assume that a project manager or architect is wondering whether certain sections of the software should be replaced or removed from the project. We can find out, whether removing (or replacing) the modules has positive effect on the maintenance of the software by simulating the removal (or replacing) of the module and estimating the opportunity cost in maintenance effort.

From the general definition of opportunity cost, we construct the definition of opportunity cost of maintaining a software module. Opportunity cost or alternative cost of not having a module m in project p (A_m^p) is the difference between the cost of maintaining or developing the project without the module (E_m^p) and the cost of maintaining or developing the project with the module (C^p).

$$A_m^p = E_m^p - C^p$$

It is clear from the definition, that in order to calculate the opportunity cost of maintenance of the project without a module, we need to know both the cost of developing the project with and without a module. These two costs never occur in the same project at the same time, which rules out using actual data for calculating the opportunity cost. Therefore, we need to use an oracle that would tell how much the development in one or another scenario would cost. The part of the oracle can be performed by a cost estimation model, which allows reproducible objective analysis of options, or by experts (or a combination of experts and cost models), which is less generalizable.

Having found out that the opportunity cost of removing the module is significantly⁵ negative, we know that we should keep the module. If the opportunity cost would have turned up significantly positive, you would know that the module is safe to remove. If the opportunity cost is near zero, the removal of the module is likely to have little to no effect on the maintainability of the project. Similarly, one can simulate any refactoring

⁵ Significantly means that the probability of being mistaken is less than acceptable by your confidence threshold. Commonly confidence of 0.95 or higher is used meaning the probability of being mistaken is 5% or less.

option and calculate corresponding opportunity cost. Therefore, it would be possible to evaluate all refactoring options before taking action.

As the possible simulations are not limited by anything but the oracle's ability to sense the changes, the same process can be used for more accurate evaluation of implementation alternatives. The oracle has another limitation – one needs to know whether the oracle's accuracy is below or above the significance threshold (confidence) of the developer. Thus, only estimation models with published error and confidence (or p-value) can be used as an oracle.

4.3 Commercial Applications

In commercial applications, the monetary cost of the project becomes more relevant than in free software. The opposite is often true for the effort as the availability of workforce is less volatile. Simplistically, the monetary value can be assumed to stem from effort alone [1]. In practice, this can be an oversight as depending on the project, the infrastructure costs or even public image can have higher impact on the monetary costs than developer salary. Commercial evaluation also needs to take into account inflation, which affects monetary costs [20].

On the other hand, a more rigid structure of commercial projects will allow more precise cost estimation methods to be used for opportunity cost estimation. This can lead to much more accurate opportunity cost analysis and better options for finding the optimal behaviour. In this study, we focus on free software as the data on free software has better availability and lends itself to easier replication of the study.

5 Limitations of Cost Estimation

An important factor that was highlighted by Özogul et al. is that the value of the measurement unit changes in time [20]. As they were studying the monetary costs, they resolved that limitation by calculating the present value of any future values. They could do that on the assumption of knowing the approximate average inflation rate for the prices in the studied currency. In general, it is not that simple to compare values or costs from different time periods. This also applies to code churn as the efficiency of software developers is increasing due to the introduction of frameworks, higher level languages and design-time code generation. However, we are currently not aware of any standard methods or discounting the effects of such advancements in code churn analysis.

As all estimations, cost estimation are not ideal and exact. Thus, one needs to consider the limitations of the accuracy of cost estimation models used. We highlighted that as a requirement for any cost estimation model to be generally useable as it is possible to identify significant cost differences only with models that have known error ranges and confidence levels. Not understanding and controlling for the significance of difference can easily lead to invalid conclusions. As in our experience similar software modules often have similar cost estimations, this can lead to project managers switching software modules and components too often (even when there is no significant benefit of cost) causing more cost to incur due to instability of the project architecture.

6 Conclusions and Future Perspectives

Opportunity cost is a key concept in economics when describing rational decision-making. A literature search of opportunity cost in software in three larger software-related research databases found only a handful of relevant results. Therefore, it is clear that the value-based decision making rationales in software engineering are not well established. Even more, we found that only 1 of 8 papers demonstrated opportunity cost calculation without relying on expert opinion. Only 1 out of 8 papers claimed that the approach used was tested on more than 2 projects and none of the studies claimed verification from more than 2 experts when expert opinion was needed. As such, the generalisability of the current studies is weak. None of the studies reported the accuracy and confidence of the proposed methods.

On positive note, we found that there is at least one cost estimation method that can be used for opportunity cost analysis. Furthermore, the identified method uses a model that does not require any expert input – the estimations were based solely on the data available in a source code repository history. This allows the approach to be used even in situations where experts with sufficient competence are not available without needlessly exposing the estimation process to a potential human error. The technique presented in this paper shows how the effects of decisions made during software development process can be simulated. More specifically, we offer an approach for better identification of the drivers of software coding effort in open source software projects. As such, we cater for the future-oriented needs of software development analysts as defined by Buse et al. [28]. As an added benefit of using easy-to-use end-user-oriented analysis platform, our analysis technique can be easily deployed without deep understanding of statistics, databases or software source code management systems.

The obvious future perspective is the need for more studies of opportunity cost modelling in software engineering. Some of the scarcity of opportunity cost modelling can be due to the performance of the estimation models, which we have seen to be sufficient for limited success. Improving the estimation abilities of the models can lead to significantly better and more detailed what-if analysis.

Acknowledgement. This work was supported by the Estonian Research Council (grant IUT20-55).

References

1. Jørgensen, M., Shepperd, M.J.: A systematic review of software development cost estimation. *IEEE Trans. Softw. Eng.* **33**(1), 33–53 (2007)
2. BusinessDictionary. <http://www.businessdictionary.com/>
3. Pandey, P.: Analysis of the techniques for software cost estimation. In: *Proceedings of 2013 Third International Conference on Advanced Computing and Communication Technologies*, pp. 16–19. IEEE, Rohtak (2013)
4. Dalal, S.R., Mallows, C.L.: Some graphical aids for deciding when to stop testing software. *IEEE J. Sel. Areas Commun.* **8**(2), 169–175 (1990)

5. Kapur, P., Shrivastava, A.: Release and testing stop time of a software: a new insight. In: Proceedings of 4th International Conference on Reliability, Infocom Technologies and Optimization. IEEE, Noida (2015)
6. Hall, G.A., Munson, J.C.: Software evolution: code delta and code churn. *J. Syst. Softw.* **54**(2), 111–118 (2000)
7. Munson, J.C., Elbaum, S.G.: Code churn: a measure for estimating the impact of code change. In: Proceedings of International Conference on Software Maintenance (ICSM), pp. 24–31 (1998)
8. Nagappan, N., Ball, T.: Use of relative code churn measures to predict system defect density. In: Proceedings of the International Conference on Software Engineering, pp. 284–329 (2005)
9. Thwin, M.M., Quah, T.-S.: Application of neural networks for software quality prediction using object-oriented metrics. *J. Syst. Softw.* **76**(2), 147–156 (2005)
10. Koten, C.V., Gray, A.R.: An application of Bayesian network for predicting object-oriented software maintainability. *Inf. Softw. Technol.* **48**(1), 59–67 (2006)
11. Karus, S., Dumas, M.: Predicting coding effort in projects containing XML. In: Proceedings of 16th European Conference on Software Maintenance and Reengineering (CSMR), pp. 203–212 (2012)
12. Li, W., Henry, S.: Object-oriented metrics which predict maintainability. *J. Syst. Softw.* **23**(2), 111–122 (1993)
13. Zhou, Y., Xu, B.: Predicting the maintainability of open source software using design metrics. *Wuhan Univ. J. Nat. Sci.* **13**(1), 14–20 (2008)
14. Rahman, F., Devanbu, P.: How, and why, process metrics are better. In: Proceedings of the 2013 International Conference on Software Engineering, pp. 432–441. IEEE Press, San Francisco (2013)
15. Karus, S., Dumas, M.: Code churn estimation using organisational and code metrics: an experimental comparison. *Inf. Softw. Technol.* **54**(2), 203–211 (2012)
16. Nagappan, N., Murphy, B., Basili, V.R.: The influence of organizational structure on software quality: an empirical case study. In: Proceedings of 30th International Conference on Software Engineering (ICSE), pp. 521–530. ACM, Leipzig (2008)
17. Tom, E., Aurum, A., Vidgen, R.: An exploration of technical debt. *J. Syst. Softw.* **86**(6), 1498–1516 (2013)
18. Wierzbicki, A.P.: A mathematical basis for satisficing decision making. *Math. Model.* **3**(5), 391–405 (1982)
19. de Mesquita Spinola, M., de Paula Pessoa, M.S., Tonini, A.C.: The Cp and Cpk indexes in software development resource relocation. In: Portland International Center for Management of Engineering and Technology, pp. 2431–2439. IEEE, Portland (2007)
20. Özogul, C.O., Karsak, E.E., Tolga, E.: A real options approach for evaluation and justification of a hospital information system. *J. Syst. Softw.* **82**(12), 2091–2102 (2009)
21. Cai, Y., Sullivan, K.J.: A value-oriented theory of modularity in design. In: Proceedings of the Seventh International Workshop on Economics-Driven Software Engineering Research, pp. 1–4. ACM, St. Louis (2005)
22. Schwartz, A., Do, H.: Cost-effective regression testing through adaptive test prioritization strategies. *J. Syst. Softw.* **115**, 61–81 (2016)
23. Ramler, R., Wolfmaier, K.: Economic perspectives in test automation: balancing automated and manual testing with opportunity cost. In: Proceedings of the 2006 International Workshop on Automation of Software Test, pp. 85–91. ACM, Shanghai (2006)
24. Chan, T.: Beyond productivity in software maintenance: factors affecting lead time in servicing users' requests. In: Proceedings of International Conference on Software Maintenance, pp. 228–235. IEEE (2000)

25. Raemaekers, S., Deursen, A.V., Visser, J.: Measuring software library stability through historical version analysis. In: Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM), pp. 378–387. IEEE, Trento (2012)
26. Upadhyay, N., Despande, B.M., Agrawal, V.P.: Towards a software component quality model. In: Meghanathan, N., Kaushik, B.K., Nagamalai, D. (eds.) CCSIT 2011. CCIS, vol. 131, pp. 398–412. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-17857-3_40
27. Mari, M., Eila, N.: The impact of maintainability on component-based software systems. In: Proceedings of 29th Euromicro Conference, pp. 25–32 (2003)
28. Buse, R.P., Zimmermann, T.: Information needs for software development analytics. In: Proceedings of the 34th International Conference on Software Engineering, pp. 987–996. IEEE Press, Zurich (2012)

FLOSS Education and Training



Does FLOSS in Software Engineering Education Narrow the Theory-Practice Gap? A Study Grounded on Students' Perception

Debora Maria Coelho Nascimento¹ ,
Christina von Flach Garcia Chavez²  , and Roberto Almeida Bittencourt³ 

¹ Federal University of Sergipe, São Cristóvão, Brazil
deboramcn@dcomp.ufs.br

² Federal University of Bahia, Salvador, Brazil
flach@ufba.br

³ State University of Feira de Santana, Feira de Santana, Brazil
roberto@uefs.br

Abstract. Software engineering education is challenged by the need to convey practical experience in the context of a rich and large body of theoretical knowledge. This study investigates whether the use of open source projects can reduce the gap between theory and practice in undergraduate software engineering courses. Two qualitative case studies were conducted with students performing activities in an open source project, each one in a different course: software testing and software requirements. Results point out that the use of open source projects provides a concrete experience similar to industry experience, allows high cognitive engagement when performing tasks, favors understanding and content retention, and leads to the recognition of the usefulness of software engineering principles, techniques and methods.

Keywords: Education · Theory-practice gap · Qualitative studies

1 Introduction

Free/Libre/Open Source Software has not only been instrumental for education and research in the academia, but also provides a real world object of study (software and its development) for software engineering (SE) researchers. Their use in SE education is becoming more popular, since it provides an opportunity for learning SE principles, techniques and methods and, thus, for narrowing the theory-practice gap usually present in undergraduate courses on the subject.

Software Engineering reference curricula [23] emphasize the need for professional practice and student participation in real projects. Several countries provide guidelines for Computer Science courses that recommend that curricula must leverage the coexistence between theory and practice, so that students

can adapt to new situations of their training area in the future. Nonetheless, examples and exercises presented either in SE courses or in textbooks are usually simple and easier to understand than real world problems. Such limitations sometimes make students remain unaware whether they really need to apply theory from the SE discipline.

On the other hand, the adoption of open source projects (OSP) to perform practical activities in the formal education of SE allows students to live practical experience environments close to the ones they will find in industry [2, 19]. Nandigam *et al.* [17] argue that students learning with OSP acquire a view that allows them to understand the basic principles of SE, not as merely academic knowledge, but as something useful and necessary for practice.

This work investigates whether the adoption of OSP allows students to make connections between theoretical knowledge with practical knowledge and skills. This investigation is relevant to address the lack of motivation that arises when students do not perceive the real usefulness of theoretical knowledge, which may lead students to ignore SE principles, practices and methods. We conducted two case studies in different courses of SE, in which students used an OSP to perform activities related to software testing and software requirements.

The results show that students perceived that the use of OSP: (i) provides a concrete experience equivalent to the situations that they will experience in the labor market; (ii) allows high cognitive engagement with active participation; (iii) favors understanding and retention of content; and (iv) reduces abstraction and allows the object of study to be more concrete and meaningful. Consequently, within the case studies and according to their own perceptions, students were able to connect the theory provided in an academic environment with real-world practice, recognizing the importance of SE knowledge, and therefore, narrowing the theory-practice gap in SE education.

2 Experience, Engagement and Content Significance

Boud *et al.* [1] emphasize that *experience* is the basis and the stimulus for learning. Watching a lecture, reading a textbook, discussing certain content, performing practical activities, visiting a museum are examples of experience. Moon [15] distinguishes between external experience (what is experienced by the learner, whether an object, a concept, an image, etc.) and internal experience (what the learner recovers from his/her cognitive structure to the current learning situation; the set of previous experiences that are important for the present situation). Therefore, learning occurs by comparing external experience with current internal experience, and is the result of variations between them [15]. Jarvis [12] emphasizes that internal/previous experience is what guides how the apprentice responds to present experience. However, the teacher/educator can have influence on learning by specifying external experiences for the learner [15]. Finally, not every experience brings good results for learning [6, 8, 15]. Experience must be vivid, lively, interesting and must be connected with future experience, especially with out-of-school situations [6].

Cognitive engagement refers to the quality of the student's psychological engagement in academic tasks [5]. It comprises the need for the learner to process the content of the lesson, so that this engagement may be superficial or deep [24]. Learning is a function of the student's cognitive engagement, that is, it increases as a result of increased quality of cognitive engagement [24]. Attending a lecture, attending a demonstration, describing certain content in their own words, giving examples, solving exercises, discussing ideas on the subject, presenting their opinion using argumentation, solving real problems are examples of gradual forms of engagement. Engagement strategies should promote the manipulation of information, rather than memorization [9]. Uden and Beaumont [24] emphasize that engaging in complex cognitive activities generates useful and authentic learning. Moreover, engagement through practical activities allows the occurrence of errors and mistakes, which are also important for learning.

Significant content is an element that participates early in the learning process, with the perception or selection of the information to be processed. Only what is significant or meaningful to the learner is captured. Information needs to be perceived or selected so that learning takes place [24]. Lefrançois [13] argues that the more meaningful the content is, the more easily it will be remembered. This would be the explanation for the vulnerability of episodic memory, whereas an event with related meaning is more difficult to be forgotten. Learning is effective when the learner is able to organize information by identifying logical relationships in the content [24]. We highlight two issues with respect to the construction of meaning. First, concrete experiences are critical to meaningful learning [15, 20]. They enable the learner to better understand, analyze the importance of the studied content and evaluate consequences. Second, although the construction of meaning is individual, experience takes place within a social context and therefore, each person is strongly influenced by the social and cultural context of the learning environment [15]. Beliefs and values of the individuals are generated from the social and cultural context in which they live, and influence the interpretation of the facts and, consequently, the construction of meaning.

3 Related Work

Systematic mapping studies identified various initiatives to software engineering education with OSP [2, 19]. Most of the primary studies presented solution proposals or experience reports; other studies presented a more general view of how the approach could be incorporated into the curriculum, or how to bridge the training gaps pointed out by industry.

Recent studies have focused on capturing students' perceptions on the use of OSP in formal education [18, 22]. Nascimento *et al.* [18] investigated whether undergraduate students regarded the activities performed in open source projects as a real-world experience. The results provided evidence based on students' perceptions that OSP have a set of features similar to industrial software. They recognized the closeness of the activities to be carried out in industry and, consequently, of the typical difficulties in working with real projects, and the skills

they need to develop. Pinto *et al.* [22] conducted an exploratory investigation on students' perceptions about the need to contribute to an OSP as a mandatory activity in a software engineering course. The authors highlighted students' recognition of improving their technical skills and increasing their self-confidence. The study also highlighted the complexity and diversity of students' engagement in carrying out the activities. While these studies [18, 22] present evidence of students' perception on the 'cognitive engagement' and 'vivid and real experience' provided by the adoption of OSP in formal education, our study aims to provide evidence grounded on students' perception on the 'significance' of the contents studied with the support of OSP.

Nandigam *et al.* [17] adopted a practical approach to teach a subset of basic software engineering principles by using OSP with the belief that students would perceive how such principles could be used in practice. The authors provided a detailed report on the activities performed and concluded that the expected results were achieved. However, they collected such perceptions based on students' oral presentations and follow-up discussions [17]. Our study provides evidence based on students' own perceptions about the connection between theory and practice when performing activities with the OSP.

4 Methodology

We conducted two case studies in the academic setting to explore students' perceptions on the use of OSP and whether it provides a connection between theory and practice. The first case study (CS1) was executed in a software testing course, while the second one (CS2) was executed in a course on software requirements. In both CS1 and CS2 students used **JabRef**¹, a software for managing bibliographic references in the Bibtex format, to perform practical activities. The research team was responsible for defining the activities to be carried out and supporting students with **JabRef** issues.

In CS1, students could choose to either perform or not the requested activities with **JabRef**. The instructor used classical lectures followed by exercises, with theory presented independent from the OSP. We added an optional activity to implement automated tests for **JabRef** that consisted of two steps. First, each team of students was responsible for building automated unit and integration tests for some assigned feature module. Within the team, students discussed and decided which features should be tested and who should write the tests. In the second step, they should build automated functional tests based on the application interface, analyze coverage of the implemented tests, and develop and run at least one regression test plan. In CS2, the use of **JabRef** to perform activities was compulsory. Students split into teams performed two practical assignments. In the second assignment, they should reverse-engineer the requirements of a legacy software, **JabRef**. Students identified functional and non-functional requirements, created a requirements traceability matrix, developed system sequence diagrams for use cases, identifying classes and methods

¹ <http://www.jabref.org/>.

involved; and proposed improvements to the built-in help function of the use cases.

We used interviews and questionnaires as instruments of data collection. The research team had no relation to the students or their grading. Questionnaires were used in the first contact with the students, before and after the intervention using the OSP. In the first one, the goal was to identify their previous experience with real software projects. In the second, the goal was to gather students' perceptions about the knowledge and skills regarding the subject, before applying the OSP approach. In the third questionnaire, the goals were twofold: to capture students' perceptions on the knowledge and skills acquired during intervention, and to identify whether they perceived the interaction with the OSP as a means to bridge the gap between theory and practice. We used semi-structured interviews after the intervention. In each case study, we invited at least one member from each team with different levels of previous experience with real projects. The interviews were recorded, transcribed and reviewed. A total number of 30 students participated in the case studies and answered the questionnaires provided by the research team. The characterization of the participants in each case study and other supplementary material are available at [21].

For data analysis, we used descriptive statistics for the quantitative data, and the inductive-deductive process described by Merriam [14] for the qualitative data. We used open coding to discover information that might be relevant to the research. After we reached a certain number of codes, we performed axial coding simultaneously with open coding, grouping the interrelated codes and creating a hierarchy of themes. As coding proceeded, we verified existing codes that could be used or we created new codes whenever needed. After completing coding of all data, we reviewed the resulting codes, refined the hierarchy of themes, and generate memos for the key themes. Throughout the process, we sought to identify relationships between codes and themes created. In the following, the term "categories" also refers to the created codes. After performing data analysis and interpretation for each study, we triangulated the results to identify intersections between studies and the particularities of each study.

5 Results

We identified two key themes related to students' perceptions on the connection between theory and practice with the use of OSP: 'the importance of practice for learning the theory' (Sect. 5.1) and 'the importance of theory to practice' (Sect. 5.2).

5.1 Importance of Practice for Learning the Theory

Practice Provides Concrete Examples. In the two case studies, students pointed out that practice with the OSP provided a 'concrete example' of application of the theory so that the object of study was no longer just an account of the instructor, and reduced the abstraction level by reifying the theory.

“You only catch a glimpse in class, you ... grasp the subject, study and imagine, right? ... but when you examine it in practice ... it strengthens more” (ST1).

“As I had already said, it was not just in the ... knowledge was not just in theory. It ... let’s say it got out of the mind and actually happened” (SR5).

“Because what has been done (...) it moved from abstract to real” (SR2).

Practice Helps the Student to Understand Theory. Possibly by providing a concrete example, we found evidence in both CS1 and CS2 studies that practice with OSP helped students to ‘understand the concepts’ studied, a situation also reported by Hepting et al. [10].

“(.) practice helped a lot to understand the test types, white box, black box, interface tests, so ... you take a closer look at the tests ... integration ... ” (ST1).

“The tests clarify further, when you look at what you did and see how it fits into that (...) when you have a range of tests to differentiate a group of tests from another group, such as integration and unit tests. (...) When you just have an explanation, you don’t ... oh, that’s it, and it’s a very similar thing, that has a different question. ’ (...) when you start writing a test, you already ... ’oh, okay, I need to do this, so this is an integration test’ (...) You are already differentiating by groups ... It kind of enriches your ... theoretical knowledge” (ST2).

“It was something that greatly improved my understanding, and I think of my friends’ too, at least in my team” (SR2).

“(.) JabRef ... I think we could apply a lot of topics (...) and put into practice (...) as I said, there were unclear things, topics of the course that were not so clear. And there with the project, (...) trying to do ... to carry the work out, things got clear” (SR4).

“(...) several theoretical topics that I believed I had understood, then when I went to practice and used that understanding, I realized it was not quite that ... That eases theory understanding ” (SR7).

Practice Consolidate Contents. In CS2, students emphasized that the use of the OSP in practical activities enabled them to ‘consolidate knowledge’.

“(.) we study a lot of topics, and we understand, but ... with the project, I think it further consolidates the knowledge” (SR3).

“(.) I was able to ... connect the dots and ... fill in the gaps, let’s say ... to consolidate the knowledge (...) I think it became more ... let’s say so ... concrete. I could actually ... consolidate this kind of information” (SR8).

“And with practice ... it’s as if it [the project] creates a box and keep it into the person’s knowledge, into the mind, which ... turns it into ... something already known, it is no longer something new for you, and ... [with respect to] the knowledge of the course contents ... it has enriched a lot” (SR5).

In these last two extracts, students mention that, with the OSP, they were able to ‘connect the dots and fill in the gaps’ (SR8) or that the execution of the project allowed ‘to create a box and keep it into the person’s knowledge, into the mind’ (SR5). Without realizing it, students illustrated the process of knowledge assimilation and accommodation in their cognitive structure, according to the constructivist understanding of the learning process [13].

Practice Helps Content Retention. Students recognized the relevance of practice with the OSP for content retention. For CS1 participants, practical activities with JabRef were more meaningful than simply studying for an exam, and possibly forgetting the content soon afterwards: *“let’s say ... it’s better for retaining knowledge. Because if you give something more concrete to keep what was taught in class, it is not just for passing the exams, for example”* (ST5).

“Because if you only focus on theory (..), in a month, you will forget it, but not with practice, when you need it, it will be there” (ST3).

A student explained that retention happens because it is necessary to understand the content to be able to do the practical activity: *“It was good to remember because ... you see a thousand concepts (..) and you stay there with no attention sometimes ... then you have to remember what is an unit test, what is a regression test ... I’ll have to write one (..) how do you write it, you have to understand ... doing helped me remember it all”* (ST4).

Practice Promotes Student’s Active Participation. In CS2, two students indicated as a relevant factor for content retention, the need for ‘active participation’: *“(..) when you go to class to watch ... just slides (..) that knowledge will vanish at a certain point in time. There will come a time that you will no longer remember ... what you’ve seen, what you’ve heard, if you do not consolidate through practice ... you take ... from theory and put into practice what you see, what you hear, what you’re learning, and I think that contributed a lot to knowledge”* (SR5).

“This project was very important because we could do, with our own hands, what we learned in class or with third parties, you know? It’s much better for you to learn by doing than by listening ... Because one thing is for you to use slides and to be there speaking, blah, blah, blah ... But it’s neither our fault nor the instructor’s fault. It’s because the model applied in class is totally different from what you do. You learn much more by doing than by seeing” (SR2).

In traditional classroom, the student only ‘receives’ information and the knowledge ‘will vanish at a certain point in time’ (SR5). Active participation is distinct from such passive attitude and *‘you learn a lot more by doing than by listening’* (SR2). Budd [3] adds that active participation in any OSP will require that students become self-taught because they will be continuously challenged to learn some tool or develop some skill.

Practice Confirms the Applicability of Theory. In CS2, students were able to perceive that the concepts and principles studied were actually applied in practice, that is, to confirm the ‘applicability of theory’: *“It contributed because we really did practice. We set out to practice and realized that they really are applied ... the whole concept, the whole theory that was presented in class”* (SR1).

“At least I did not have that skill. (...) we had to run after stuff we didn’t know and put into practice in the project” (SR2).

“Practice helped a lot to figure out how the diagrams and stuff works” (SR7).

Practice is Essential to Learning Some Subjects. In CS1, students recognized that ‘lack of practice leads to partial learning’. Lectures enabled understanding about tests, but doubts emerged with practical activities: *“You*

understand visually, but when you put into practice what was presented in class, then the doubts begin to arise” (ST7).

To ‘learn how to do’, one needs to exercise the testing techniques: *“I think a unit testing without practice for me would be ... wouldn’t be good, I think ... I would learn half of it ...” (ST1).* The learner will only be sure if he or she knows or does not know, after experiencing [8].

Practice Helps Students to Grasp the Problem. By working with projects that are close to reality, students could ‘see how the problem really is’ and confirm that things are not as simple as they seem, when they consider only theory. According to Morelli *et al.* [16], students witness that challenging problems rarely yield to solutions presented in textbooks. Students who do not have experience with real projects have unrealistic expectations regarding the quality of the source code and are surprised when they do not find an elegant code like those presented in textbooks.

“(..) you only see the problem when you have the problem at hand. Knowing the theory on how to do the tests, what is needed, is nice, but you will only really know what is needed, when you are there in the situation (..). So, I think theory is good, but practice makes it worthwhile” (ST2).

“(..) because in theory everything is simpler ... The examples we get are always simple ... it’s an integer, then a number between zero and ten. Now this is not a number [commenting on the project], it’s a string, there’s a database that, with an input, with something else as input, so it stays ... it was difficult to know ... what exactly to use from theory” (ST6).

“(..) we see how complicated it is (..) eliciting requirements ... knowing the use cases and everything ... knowing a little bit of what the guy who developed had to know ... It’s pretty cool, because sometimes we see the theory, but with practice, it’s when you see that, it’s more difficult than in theory ... it’s something that we get the content and think we know, but when you see it in practice, there the doubt arises ... it helped a lot” (SR3).

Practice Enables Discussion. Concrete examples, a more realistic view of problems to be faced, judgments of which techniques should be applied, and the experience with a common project ‘enable discussion’. From the practical project, students have their own experience and elaborate their point of view. In addition, all students experience a little of the project, different perspectives can be exposed, favoring discussions that take place on concrete examples. In CS1 and CS2, specific classes were devoted to discuss the activities carried out in the project, and students reported their satisfaction with the learning in the light of such discussions: *“I think it contributed a lot, the issue of different perspectives regarding the project ...” (ST8).*

“... you do, implement things, see results and have your opinion. When you see that from another person as well, you increase ... you see more things too, that you have not seen and that other person has” (ST1).

A student reported that discussions in class allowed him to compare his solution with those of his classmates: *“And we can also check whether what we’ve done is right (..) With discussion, listening to other people’s opinions is*

also interesting because sometimes they point out things that we haven't perceived, and that should be there ... I found it quite interesting" (SR3).

5.2 Importance of Theory to Practice

Theory Is Necessary for Practice. In CS1 and CS2, this category emerged from several excerpts: *"You can't do testing without knowing the theory ... Even if you haven't learned it previously in the course, you come back, because you have to see the strategies, study and implement them (...) practice without theory does not exist ... for me, it's important" (ST1).*

"When I actually moved to practice, that demanded me to come back to theory to get more knowledge about that subject ..." (ST3).

"Things I've seen in class ... a few, I managed to put them in the project. Other stuff, I had to get from third parties, in websites and articles, so I could be able to do them in the project. I had to complement [my knowledge] about this, so I could implement it in the project" (SR2).

Theory Enables Planning the Things to be Performed. Theory is important to practice because it allows 'planning practice' (CS1) and defines 'what' and 'how': *"I had to come back to what I've seen in class, to know what had to be done: oh, I have to do such thing with such a JabRef class ... How does that class access that method? How do I test it?" (ST4).*

"(...) when you write the tests ... sometimes you don't know exactly what you're doing there ... When you have the theoretical background ... you need to plan to do the tests and then you already have a sense of what you're doing, so it helps a lot, you get an idea of how you're testing" (ST2).

In this last extract, the student emphasized that without theory, he tests with no clear purpose. But when he knows the theory, he knows how to plan what should be tested and how.

5.3 Discussion

Practice with an OSP lets one learn how to do something, which leads to the development of technical skills such as testing or systematically changing software. Dealing with a real problem can develop pro-activity and creativity in the search for alternative solutions. The real-world context allows students to see problems as they really are and enables them to perceive the applicability of the theory, promoting reflection and development of critical thinking. Practice with the OSP enables discussions about the project which allow students to express their opinions and broaden their views from their classmates' opinions. Thus, several skills linked to professional practice can be developed. Students also recognize that theory is necessary for practice and enables planning things to be performed.

Other studies corroborate our results. In the study by Chen *et al.* [4], one of the students stated that participation in the project allowed putting into practice the concepts learned in class. Hislop *et al.* [11] conclude that after practice with OSP, students can see the reasons for applying the theory. Ellis *et al.* [7] complement that principles of SE incorporated into the project become evident to

students as the project is executed, that is, principles are learned by experience and not by the instructor's voice. For Nandigam *et al.* [17], activities with OSP provide a solid background for the discipline of SE, and the lectures and discussions on design metrics, code maintainability, documentation and concern with design-code synchronization, with the articulation of their own points of view, have become more significant for students after the activities in the project.

On the other hand, our results cannot be generalized to other contexts. Given the methodological rigor suggested for qualitative research and followed in this work, CS1 and CS2 results can only be extrapolated to similar conditions. We believe, however, that even particular results may be useful to researchers and practitioners. Finally, because of space constraints, we presented only a few excerpts from interviews to give consistency to our findings.

6 Conclusions

This study investigated whether the adoption of open source projects in SE education enables students to connect software engineering theory with practice, under students' own perceptions. Three elements from learning theories provided the background for our study: the experience lived by the student, the depth of the student's cognitive engagement throughout this experience, and the significance of the contents studied.

We reported the results of two case studies conducted in two different SE courses, in which students used `JabRef` to perform activities related to software testing and reverse engineering of requirements. These studies brought grounded evidence to support that the adoption of OSP in practical activities in formal SE education (i) stands for a concrete experience equivalent to industry-like situations to be later experienced by graduates; (ii) allows high cognitive engagement with active student participation while analyzing, testing, modifying, and documenting the source code, among other activities that can be performed; (iii) favors content understanding and retention; and, finally, (iv) leads to the recognition of the importance of principles of software engineering, that is, the construction of meanings that allow more effective learning.

In addition, our study revealed that the OSP approach allowed students to experience that reality is more complex than how theory is usually presented; it promoted the development of technical skills; it supported discussions based on each student's practical experience, and promoted the development of critical thinking, broadening their view to other possibilities. It is worth highlighting the evidence of a student describing the process of assimilation and accommodation of knowledge supported by constructivist theories for the learning process.

From the point of view of our initial research question, we conclude that within the two case studies that used `JabRef`, the object of study became less dependent on instructors' accounts and less abstract, and became something real, concrete and meaningful. Therefore, in that context, the use of OSP through practice, supported students to make connections between theory and practice, narrowing the gap between them.

References

1. Boud, D., Cohen, R., Walker, D.: Understanding learning from experience. In: Boud, D., Cohen, R., Walker, D. (eds.) *Using Experience for Learning*, pp. 1–18. Open University Press (1993)
2. Brito, M.S., Silva, F.G., Nascimento, D.M.C., Chavez, C.F.G., Bittencourt, R.A.: FLOSS in software engineering education: an update of a systematic mapping study. In: *Proceedings of the 32nd Brazilian Symposium on Software Engineering, SBES, Sao Carlos, Brasil*, pp. 250–259 (2018)
3. Budd, T.A.: A course in open source development. In: *Integrating FOSS into the Undergraduate Computing Curriculum, Free and Open Source Software (FOSS) Symposium, Chattanooga* (2009). <http://www.cs.trincoll.edu/~ram/hfoss/Budd-FOSS-Course.pdf>
4. Chen, Y., Roytman, A., Fong, P., Hong, J., Garcia, D., Poll, D.: 200 students can't be wrong! GamesCrafters, a computational game theory undergraduate research and development group. In: *AAAI Spring Symposium, Technical report* (2008)
5. Davis, H.A., Summers, J.J., Miller, L.M.: *An Interpersonal Approach to Classroom Management: Strategies for Improving Student Engagement*. Corwin, Thousand Oaks (2012)
6. Dewey, J.: *Experience and Education*. Macmillan, London (1938). <http://ruby.fgu.edu/courses/ndemers/colloquium/experienceducationdewey.pdf>
7. Ellis, H.J., Morelli, R.A., de Lanerolle, T.R., Hislop, G.W.: Holistic software engineering education based on a humanitarian open source project. In: *20th Conference on Software Engineering Education & Training, CSEET 2007*, pp. 327–335. IEEE, Dublin, July 2007. <https://doi.org/10.1109/CSEET.2007.26>
8. Gentry, J.W.: What is experiential learning?, Chap. 2. In: *Guide to Business Gaming and Experiential Learning*, p. 370. Nichols Pub Co (1990)
9. Hannafin, M.J.: Instructional strategies and emerging instructional technologies: psychological perspectives. *Can. J. Educ. Commun.* **18**, 167–179 (1989)
10. Hepting, D.H., Peng, L., Maciag, T.J., Gerhard, D., Maguire, B.: Creating synergy between usability courses and open source software projects. *ACM SIGCSE Bull.* **40**(2), 120–123 (2008). <https://doi.org/10.1145/1383602.1383649>
11. Hislop, G.W., Ellis, H.J., Morelli, R.A.: Evaluating student experiences in developing software for humanity. *ACM SIGCSE Bull.* **41**(3), 263–267 (2009). <https://doi.org/10.1145/1595496.1562959>
12. Jarvis, P.: *The Paradoxes of Learning*. Jossey-Bass, San Francisco (1992)
13. Lefrançois, G.R.: *Teorias da Aprendizagem*, 5th edn. Cengage Learning, São Paulo (2012)
14. Merriam, S.B.: *Qualitative Research: A Guide to Design and Implementation*. Jossey-Bass, San Francisco (2009)
15. Moon, J.: *A Handbook of Reflective and Experiential Learning: Theory and Practice*. RoutledgeFalmer, London (2004)
16. Morelli, R., et al.: Revitalizing computing education through free and open source software for humanity. *Commun. ACM* **52**(8), 67–75 (2009). <https://doi.org/10.1145/1536616.1536635>
17. Nandigam, J., Gudivada, V.N., Hamou-Lhadj, A.: Learning software engineering principles using open source software. In: *Proceedings of the 38th Annual Frontiers in Education Conference (FIE)*, pp. 18–23. IEEE, October 2008. <https://doi.org/10.1109/FIE.2008.4720643>

18. Nascimento, D.M.C., Chavez, C.F., Bittencourt, R.A.: The adoption of open source projects in engineering education: a real software development experience. In: Proceedings of the 48th Annual Frontiers In Education Conference (FIE), San Jose, pp. 1091–1100 (2018)
19. Nascimento, D.M.C., Bittencourt, R.A., Chavez, C.F.: Open source projects in software engineering education: a mapping study. *Comput. Sci. Educ.* **25**, 67–114 (2015)
20. Northern Illinois University, Faculty Development and Instructional Design Center: Experiential Learning (2012). <http://www.niu.edu/facdev/resources/guide>
21. Nascimento, D.M.C., Chavez, C.F., Bittencourt, R.A.: Does FLOSS in SEE narrow the Theory-Practice Gap? Supplementary Material (2019). <http://sites.google.com/site/oss2019osp/>
22. Pinto, G., Ferreira, C., Souza, C., Steinmacher, I., Meirelles, P.: Training software engineers using open-source software: the students' perspective. In: Proceedings of the International Conference on Software Engineering, Software Engineering Education and Training (SEET), Montreal, Canada (2019)
23. IEEE/ACM Joint Task Force on Computing Curricula: Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering (2015)
24. Uden, L., Beaumont, C.: Technology and Problem-Based Learning. Information Science Publishing, London (2006)



Faculty Development for FLOSS Education

Becka Morgan¹(✉), Gregory W. Hislop², and Heidi J. C. Ellis³

¹ Western Oregon University, Monmouth, OR, USA
morganb@wou.edu

² Drexel University, Philadelphia, PA, USA
hislop@drexel.edu

³ Western New England University, Springfield, MA, USA
ellis@wne.edu

Abstract. With the recent upsurge in the development, use, and adoption of free/libre open source software (FLOSS) across all sectors of business, it is critical that graduates of computing degree programs gain an understanding of FLOSS development tools, processes, and culture. However, many faculty members are not fluent in FLOSS development and have little experience in teaching FLOSS. This paper reports on a faculty development program designed to bring instructors up to speed on how to support student learning within FLOSS projects. The paper discusses the challenges to FLOSS education from the instructor's perspective, describes the Professors' Open Source Software Experience (POSSE) workshop, and presents the results of a study into the impact of POSSE on instructors based on semi-structured interviews. This work is part of a larger study into instructor experiences when incorporating Humanitarian Free Open Source Software (HFOSS) into their curriculum.

Keywords: Faculty development · FLOSS education

1 Introduction

Free/libre open source software (FLOSS) is rapidly becoming mainstream in industry today as evidenced by Microsoft's acquisition of GitHub, IBM's purchase of Red Hat, and the wide adoption of FLOSS by the majority of enterprise, mid-market, and small businesses [1]. Results of recent surveys indicate that 92% of respondents identified that their company's applications contain open source libraries [2] and over 50% of respondents belong to an organization that has an open source software program or has plans to establish one [3].

One reason for the increase in adoption of FLOSS is that it is the source of significant innovation for businesses. Many emerging technologies and approaches that are modernizing computing including cloud computing, containerization, and serverless computing originated with FLOSS. Examples include tools such as Docker and Kubernetes, and FLOSS frameworks such as node.js and JUnit. Indeed processes such as continuous integration and DevOps are heavily based on FLOSS concepts.

This increasing use and development of FLOSS creates a clear need for more software professionals who can develop open source software, as well as providing a strong motivation for students to gain an understanding of FLOSS tools and practices

as a part of their undergraduate degree programs. Indeed, FLOSS experience can give students an edge in the hiring process [4]. However, many instructors are not fluent in FLOSS. This paper discusses an assessment of a professional development program for instructors intended to bring them up to speed in FLOSS culture and tools.

2 Challenges

FLOSS practices and tools are not included in typical computing undergraduate degree requirements. In the ACM Curriculum Guidelines, the term “open source” is mentioned only twice in the CS2013 Computer Science Curricula [5] and twice in the IT2017 Information Technology Curricula [6]. The SE2014 Software Engineering Curricula [7] identifies the need for student understanding of FLOSS, yet does not require coverage of FLOSS in the guidelines. This lack of emphasis on FLOSS in professional computing curricula creates a barrier to including FLOSS in the classroom.

Instructors who desire to educate students in FLOSS projects face several additional hurdles. First, many instructors are not familiar with FLOSS culture or tools and face steep learning curves for these, as well as for the FLOSS project domain knowledge. Instructors face an additional learning curve related to how to support student participation in FLOSS projects. Assignment creation, assessment, and development of rubrics are all challenges in the environment of an active FLOSS project and must be undertaken within the constraints of student privacy and intellectual property issues.

Instructors need to learn how to support student learning in the non-traditional environment of a FLOSS project including helping students adjust to a less-structured learning environment and one that has significant size and complexity. Instructors also need to help students with the potentially steep learning curves of FLOSS culture and technologies. Student expectations about the role of the faculty member as a “guide by the side” rather than the “sage on the stage” must be set early and frequently reinforced. Instructors may also need to serve as a liaison between their students and the FLOSS community, which may take additional time.

Another challenge faced by instructors is the selection of an appropriate FLOSS project for use in a course [13]. Ideal FLOSS projects are open to student contributions, welcoming, and have clear communication channels, all while also fulfilling course learning objectives. Several efforts are investigating this problem [8–11].

The FLOSS project environment presents additional challenges. The FLOSS project schedule and instructor’s academic calendar may not align well. The ability of a FLOSS project to quickly make major changes in direction may also impact student participation. In addition, instructors must plan for contributions over the limited timespan of a term, typically 10 or 15 weeks, once or twice a year.

Results of a research effort [12] that surveyed faculty members as to the challenges faced when involving students in FLOSS projects indicated that lack of time to prepare course materials was the largest hurdle to student involvement, with lack of time in the course for FLOSS as a close second. Additional hurdles included lack of an appropriate course, difficulty in installing the FLOSS environment, and lack of comfort with the FLOSS learning materials.

3 Professors' Open Source Software Experience Workshops

The faculty development program used in the study is the Professors' Open Source Software Experience (POSSE) workshop. POSSE was originated by Red Hat in 2009, with the goal to help instructors learn about FLOSS so that they can incorporate it into their courses.

In 2013, a team of instructors received National Science Foundation funding to update POSSE contents to include information on pedagogical and curricular considerations, and to provide funding for faculty attendance. In addition, the focus of POSSE was shifted to Humanitarian Free and Open Source Software (HFOSS) projects as there is some evidence that humanitarian projects are more attractive to women and other under-represented groups. A typical POSSE involves 15–25 instructors new to FLOSS and 3–6 workshop facilitators. POSSE workshops are held in three stages. Stage 1 consists of 8 weeks of online learning activities and three IRC meetings are held to answer questions. Stage 2 is a 2.5 day face-to-face meeting held after stage 1 where instructors learn more about FLOSS culture and gain further understanding of how to assess HFOSS projects and incorporate them into their classes. Stage 3 of POSSE occurs after stage 2 and involves instructors working collaboratively on either common projects or common courses to support student involvement in HFOSS. POSSE is supported by over 100 learning materials that instructors may use, located on foss2serve.org. Further information on POSSE is contained in [14] and the foss2serve.org website.

4 The Study

POSSE workshops are showing promise for expanding the number of instructors able to support student learning within HFOSS projects and several publications have resulted from POSSE alumni teaching efforts [15–18]. As of early 2019, over 150 instructors from over 120 different academic institutions have completed the workshop. These instructors served as the population for a study that involved an online survey followed by a semi-structured interview with the goal of determining the impact of supporting student participation in HFOSS projects on instructors. In this paper, we discuss the impact of POSSE on instructors as observed in interview results.

The participants in the study were selected from instructors who had participated in a POSSE workshop at least one year prior to the study start in order to focus on instructors who had had time to incorporate HFOSS into their classroom. Out of 77 eligible participants, the group was reduced to 41 by selecting only one instructor per academic institution and focusing on participants who were less experienced in FLOSS education at the time of POSSE attendance. The group included instructors who had been successful and unsuccessful in supporting student participation in HFOSS.

Twenty-six instructors completed the survey and 24 of these agreed to participate in the study. Participants were relatively gender balanced, consisting of 11 women and 13 men and participants came from both 2 and 4-year institutions located in 16 different U.S. states. Further details on the study design may be found in [12].

The semi-structured interview included questions such as “What was the biggest hurdle that you encountered and how did you overcome it?” and “What have you

learned from working with a FLOSS community?” The study resulted in a considerable volume of data, which is still being analyzed. The focus of this paper is on the impact of POSSE workshops on instructors.

4.1 Results

The corpus of the 24 semi-structured instructor interviews was searched for all occurrences of the terms “POSSE”, “workshop*”, “roundup” and “meet*”. The occurrences of these terms within their context were analyzed by two authors who coded the answers and organized them into categories. The coding process was performed independently to avoid bias and a conflict resolution meeting was used to come to agreement on the categories. Five main themes, described below, emerged.

Helpfulness. Ten of the respondents identified the POSSE workshop as being helpful using phrases such as “learning activities were immensely helpful”, “very valuable”, and “eye-opening”. One participant stated, “POSSE experience forever changed my perception of what it means to be open source...” An additional three respondents indicated that they had gained a better understanding of FLOSS tools. This feedback clearly indicates that instructors feel that they gain useful knowledge and skills from the POSSE workshop and that the workshop is helpful in supporting student participation in HFOSS projects.

Materials. Nine respondents indicated that they used the POSSE workshop materials in their courses, either directly or with modification. One participant indicated that POSSE was “...quite helpful with learning activities, I got good ideas for activities...” These responses support the observation that POSSE workshops are fulfilling their goal of providing instructors with the skills and materials they need to support student learning within HFOSS projects.

Enhancements. Another major theme that emerged was possible enhancements to the POSSE workshop. Ten participants identified augmentations to POSSE that they thought would help them support student participation in HFOSS. These included:

- Longer workshop meeting with additional hands-on experience with projects
- Tutorials that instructors and students could use to learn tools
- List of approachable HFOSS projects including current community contacts
- Repository of exemplars of student participation in HFOSS
- Curriculum in a box where a project and set of tools are packaged in a VM
- More advanced workshop that builds on current POSSE.

Community. One recurrent theme throughout most of the interviews was the value provided by the POSSE community and the need for it to be an ongoing and healthy community. The use of POSSE “Roundups” and sprints where instructors meet face-to-face to share ideas and work on activities was clearly valued. One respondent indicated “... round ups for me are very beneficial... so I can touch base with other people and hear what other people are doing and exchange ideas.” In addition, some respondents indicated that attending a second POSSE workshop was or would be helpful. Clearly POSSE participants value the community established by the workshops and

desire to see that community continue and grow. POSSE appears to work well as an introduction to the topic and the community, but there needs to be ongoing support for the community.

Shortcomings. Another important theme that emerged from the analysis was identification of the shortcomings of the POSSE experience. Several respondents indicated frustration with not being able to get the HFOSS project development environment installed. One respondent who tried to get an HFOSS project installed while attending POSSE stated, “We spend all of our working time trying to set up the system and never got it set up. That was a kind of negative experience on my part.” Another respondent indicated that the activities in stage 1 and stage 2 needed to be more tightly related stating, “We had a series of HWs prior to workshop, then we didn’t even really discuss them at the workshop...” Another respondent indicated that providing each participant with a concrete take-away appropriate for their class would have been helpful.

Overall, the observations about POSSE indicate that participants are gaining knowledge and skills from the workshops. The majority of participants indicated that they found the workshop helpful and that they used workshop materials in their teaching. The need for continuing community was also a resonant thread throughout the interviews. One respondent noted, “most important takeaway for me is the desire for <the> POSSE organism to remain alive and vibrant moving forward”. Feedback on the shortcomings of the workshop focuses on identification of appropriate course materials and the availability of approachable and tractable HFOSS projects. Instructors involved in carrying out POSSE are currently working to address the shortcomings.

4.2 Limitations

The biggest limitation of the study is that the participants in the study self-selected to participate in the POSSE workshop. A second limitation is the independent coding of the responses. Lastly, the number of professors involved is relatively low, which may limit the generalization of the results.

5 Conclusion and Future Work

The work presented in this paper is a part of a larger study into the impact of student participation in HFOSS on instructors. Results of this study will provide insight into how to increase the number of instructors in the FLOSS education community to fulfill the larger goal of increasing the number of students who graduate ready to contribute to FLOSS projects. Overall, results of analyzing instructor comments on the POSSE workshop experience are positive and appear to indicate that instructors are better able to support student participation in the classroom. The POSSE team is working to address the identified shortcomings of difficulty in project installation and identification of appropriate assignments for those instructors first learning to support FLOSS education. Future areas of study include analysis of instructor expectation, analysis of instructor factors that lead to student success, and how FLOSS education impacts instructor self-perception as a teacher.

References

1. 10th Anniversary of the Open Source Survey (2016). www.northbridge.com/2016-future-open-source-survey-results. Accessed 11 Jan 2019
2. How to make open source work better for everyone, 9 key insights from the 2018 Tidelif professional open source survey, July 2018. <https://tidelif.com/about/2018-Tidelif-professional-open-source-survey-results>. Accessed 20 Dec 2018
3. Hecht, L., Clark, L.: Survey: open source programs are a best practice among large companies, 30 August 2018. <https://thenewstack.io/survey-open-source-programs-are-a-best-practice-among-large-companies/>. Accessed 11 Jan 2019
4. Hansson, D.H.: Reduce the risk, hire from open source. https://dhh.dk//arc/2005_09.html. Accessed 13 Jan 2019
5. Computer Science Curricula 2013 Curriculum Guidelines for Undergraduate Degree Programs in Computer Science, 20 December 2013. https://www.acm.org/binaries/content/assets/education/cs2013_web_final.pdf. Accessed 11 Jan 2019
6. Information Technology Curricula 2017 IT2017 Curriculum Guidelines for Baccalaureate Degree Programs in Information Technology, 10 December 2017. <https://www.acm.org/binaries/content/assets/education/curricula-recommendations/it2017.pdf>. Accessed 11 Jan 2019
7. Software Engineering 2014 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, 23 February 2015. <https://www.acm.org/binaries/content/assets/education/se2014.pdf>. Accessed 11 Jan 2019
8. Gokhale, S.D., Smith, T.M., McCartney, R.: Integrating open source software into software engineering curriculum: challenges in selecting projects. In: Proceedings of the First International Workshop on Software Engineering Education Based on Real-World Experiences (EduRex 2012), pp. 9–12. IEEE Press, Piscataway (2012)
9. Smith, T.M., McCartney, R., Gokhale, S.S., Kaczmarczyk, L.C.: Selecting open source software projects to teach software engineering. In: Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE 2014), pp. 397–402. ACM, New York (2014). <https://doi.org/10.1145/2538862.2538932>
10. Ellis, H.J.C., Hislop, G.W., Burdge, D.: Courseware: HFOSS project evaluation. In: Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2017), pp. 90–91. ACM, New York (2017). <https://doi.org/10.1145/3059009.3072975>
11. Ellis, H.J.C., Hislop, G.W., Purcell, M.: Project selection for student involvement in humanitarian FOSS. In: 26th International Conference on Software Engineering Education and Training (CSEET&T), San Francisco, CA, pp. 359–361 (2013). <https://doi.org/10.1109/cseet.2013.6595279>
12. Postner, L., Ellis, H.J.C., Hislop, G.W.: A survey of instructors' experiences supporting student learning using HFOSS projects, In: Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE 2018), pp. 203–208. ACM, New York (2018). <https://doi.org/10.1145/3159450.3159524>
13. Pinto, G.H.L., Filho, F.F., Steinmacher, I., Gerosa, M.A.: Training software engineers using open-source software: the professors' perspective. In: 2017 IEEE 30th Conference on Software Engineering Education and Training (CSEET), pp. 117–121 (2017). <https://doi.org/10.1109/cseet.2017.27>
14. Ellis, H.J.C., Chua, M., Hislop, G.W., Purcell, M., Dziallas, S.: Towards a model of faculty development for FOSS in education. In: 2013 26th International Conference on Software Engineering Education and Training (CSEET&T), May 2013, pp. 269–273 (2013)

15. Braught, G., et al.: A multi-institutional perspective on H/FOSS projects in the computing curriculum. *ACM Trans. Comput. Educ.* **18**(2), Article no. 7, pp. 7:1–7:31 (2018). <https://doi.org/10.1145/3145476>
16. Crain, S.P.: Open source security assessment as a class project. *J. Comput. Sci. Coll.* **32**(6), 41–53 (2017)
17. Murphy, C., Buffardi, K., Dehlinger, J., Lambert, L., Veilleux, N.: Community engagement with free and open source software. In: *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education (SIGCSE 2017)*, pp. 669–670. ACM, New York (2017). <https://doi.org/10.1145/3017680.3017682>
18. Buffardi, K.: Localized open source collaboration in software engineering education. In: *2015 IEEE Frontiers in Education Conference (FIE)*, El Paso, TX, pp. 1–5 (2015). <https://doi.org/10.1109/fie.2015.7344142>

Author Index

- Bittencourt, Roberto Almeida 153
Bordeleau, Francis 61
Brandon, William 68
- Carlson, Brandon 12
- Ellis, Heidi J. C. 165
- Fendt, Oliver 133
Fronchetti, Felipe 91
- Hislop, Gregory W. 165
- Ihara, Akinori 44
- Jaeger, Michael C. 133
- Karpat, Kubilay 80
Karus, Siim 139
Kashiwa, Yutaro 44
Kästner, Christian 116
Kicin, Sébastien 3
- Lavazza, Luigi 104
Leach, Kevin 12
Lenarduzzi, Valentina 104
- Marinov, Darko 12
Meirelles, Paulo 61
Mendes, Fábio Macêdo 27
Miller, Courtney 116
Morasca, Sandro 104
Moreira, Bruna 27
- Morgan, Becka 165
Müller, Matthias 38
- Nagappan, Meiyappan 12
Nascimento, Debora Maria Coelho 153
- Ohira, Masao 44
- Parra, Henrique 27
Pinto, Gustavo 91
Poppi, Ricardo 27
Pradhan, Dipesh 3
Prakash, Atul 12
- Sahin, Sefa Eren 80
Schindler, Christian 38
Schork, Sebastian 3
Schwichtenberg, Antonia 3
Sillitti, Alberto 61
Singh, Vandana 68
Slany, Wolfgang 38
Steinmacher, Igor 91
- Tosi, Davide 104
Tosun, Ayse 80
- Vasilescu, Bogdan 116
von Flach Garcia Chavez, Christina 153
- Widder, David Gray 116
Wiese, Igor 91
- Zahid, Feroz 3