

FLOSS FAQ chatbot project reuse - how to allow nonexperts to develop a chatbot

Arthur R. T. de Lacerda*

Carla S. R. Aguiar*

arthurrtl@gmail.com

caguiar@unb.br

UnB Faculty in Gama - University of Brasilia
Brasilia, Brazil

ABSTRACT

FAQ chatbots possess the capability to provide answers to frequently asked questions of a particular service, platform, or system. Currently, FAQ chatbot is the most popular domain of use of dialog assistants. However, developing a chatbot project requires a full-stack team formed by numerous specialists, such as dialog designer, data scientist, software engineer, DevOps, business strategist and experts from the domain, which can be both time and resources consuming. Language processing can be particularly challenging in languages other than English due to the scarcity of training datasets.

Most of the requirements of FAQ chatbots are similar, domain-specific, and projects could profit from Open Source Software (OSS) reuse. In this paper, we examine how OSS FAQ chatbot projects can benefit from reuse at the project level (black-box reuse). We present an experience report of a FLOSS FAQ chatbot project developed in Portuguese to an e-government service in Brazil. It comprises of the chatbot distribution service, as well as for analytics tool integrated and deployed on-premises. We identified assets that could be reused as a black-box and the assets that should be customized for a particular application. We categorized these assets in architecture, corpus, dialog flows, machine learning models, and documentation. This paper discusses how automation, pre-configuration, and templates can aid newcomers to develop chatbots in Portuguese without the need for specialized skills required from tools in chatbot

*Both authors contributed equally to this research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

OpenSym '19, August 20–22, 2019, Skövde, Sweden

© 2019 Association for Computing Machinery.

ACM ISBN 978-1-4503-6319-8/19/08...\$15.00

<https://doi.org/10.1145/3306446.3340823>

architecture. Our main contribution is to highlight the issues non-English FAQ chatbots projects will likely face and the assets that can be reused. It allows non-chatbot experts to develop a quality-assured OSS FAQ chatbot in a shorter project cycle.

CCS CONCEPTS

- Computing methodologies → Discourse, dialogue and pragmatics;
- Software and its engineering → Open source model; Reusability.

KEYWORDS

FLOSS, Open source, OSS, FLOSS FAQ chatbot, Black-Box reuse, Portuguese chatbot, experience report, e-government, conversational agents.

ACM Reference Format:

Arthur R. T. de Lacerda and Carla S. R. Aguiar. 2019. FLOSS FAQ chatbot project reuse - how to allow nonexperts to develop a chatbot. In *The 15th International Symposium on Open Collaboration (OpenSym '19), August 20–22, 2019, Skövde, Sweden*. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3306446.3340823>

1 INTRODUCTION

Chatbots are conversational software agents that must have natural language processing (NLP) to understand users intentions [18] and has become increasingly popular. They conduct conversations with humans, integrating services, users, and communication channels. Examples of chatbot applications are notification, FAQ [11, 23], virtual host [15], and personalized assistants [17].

The most common type of conversational agent is FAQ chatbots, which possesses the capability to provide answers to frequently asked questions of a particular service, platform or system [23]. A consequence of using FAQ chatbots in organizations services is the decrease of the demand on other communication channels, such as calls and e-mails [10].

OSS chatbots allow to keep all conversation data on-premises and facilitate their compliance with data regulations, such as the General Data Protection Regulation (GDPR) [9, 28]. They can benefit from collaborative modeling [21], and other bot

data, such as personality, vocabulary, and chit chat datasets. Chatbot architecture, composed by the integration of both building/creating a chatbot and distributing/messaging service, algorithms, and analytics is transparent in FLOSS. However, most of the FLOSS chatbot frameworks are domain-agnostic, where they handle one specific layer of the chatbot architecture and they leave the developer to configure algorithms and parameters for their given application. The documentation found in these communities is mostly technical, done by and to experts of the field. It thus discourages nonexperts practitioners to choose which FLOSS chatbot framework to reuse for a specific application reliably.

In this work, we investigate the assets necessary to FLOSS FAQ chatbot project to be reused as a black-box. We chose the application domain of FAQ to restrain the scope since the requirements specificity affects software positively reuse success [20]. Not only code reuse, but also architecture, content (dialogues, intentions), and machine learning conversation models. We present an experience report, where we conducted a project of a FLOSS FAQ chatbot developed during 15 months of a government-academia collaboration in Brazil, with best practices based on FLOSS ecosystems, agile and DevOps. Our main contribution is to identify and trace assets to maximize reuse in this application domain, regarding toolset, architecture, algorithms, training dataset, and documentation. We collect and analyze data from the project repository and data from analytics. We contribute to guide nonexperts practitioners to develop their FAQ chatbot in reduced project time and effort.

2 RELATED WORK

According to Lebeuf in [11], a chatbot project is composed of distribution and creation services. Distribution services are employed to manage the conversation data, it is the interface between the chatbot and the user, while creation services are used to develop the chatbot knowledge and behavior. Optionally, the business analytics service helps the improvement and evolution of the chatbot. The present work is focused on OSS chatbot distribution services.

The distribution service itself is composed of the following components [8]: Natural Language Understanding (NLU), communication channels, voice user interfaces, dialog management.

A vast number of chatbot frameworks is available in OSS, such as Botkit [11], Rasa [2], Botpress [3], among others. Typically, OSS chatbot frameworks are designed for developers and specialists, and an issue is to enable nonexperts to use them as a black box [28]. An exception is botpress, a complete chatbot architectural integrated solution, with a giving user interface that guides nonexperts from bot content creation to deployment. Limitations of this solution it is being both agnostic-domain and restrains the algorithms

used. Dialog management that handles dialog context and decides the next action for the agent to take have drifted from pure rule-based to data-driven in recent years [8, 19].

Code reuse allows for previously tested and quality-assured code to be implemented in another system and it provides benefits regarding simply adding and enhancing system features [22]. In chatbot projects, both white-box and black-box reuse are common. However, most of the research done focus on the reuse of training datasets [7, 14, 19].

One crucial part of the chatbot is to understand user messages. According to Liu et al. [13], Natural Language Understanding Services perform similarly in both OSS and proprietary solutions. Another grand challenge is the effectiveness of chatbots in non-English speaking countries [14]. When the objective is to build a chatbot in other languages than English, dialogue datasets are the most significant difficulty [7], but it still possible as shown in [24], a German chatbot was built to guide an idea submission process. ParlAI is an open-source platform that aims to minimize this dataset issue by providing a unified framework for sharing, training, and testing dialog models [16]. They aim to reinforce reuse of training datasets by sharing a repository of the corpus, utterances, and machine learning models. Another alternative is the adoption of crowdsourcing to write utterances. Paetzel et al. in [19] evaluate how untrained crowd workers (crowdsourcing) can scale chatbots contents and still maintain coherence in the chatbot personality, affective behavior, and vocabulary.

3 THE EXPERIENCE REPORT

The project to develop an FAQ chatbot is a partnership between government and academia started in 2017 [1], and it is still ongoing. The Ministry of Citizenship decided to join the University of Brasília (UnB) to develop a FLOSS FAQ chatbot to help citizens to understand better their law of incentive to the culture and to answer the frequently asked questions of its service. However FAQ chatbot presence is growing in private sectors, it is still rare in government agencies [10].

One of the significant project requirement that it should be based on existing FLOSS, and the Ministry technical staff should handle its evolution and maintenance. The technical staff is composed mainly by software engineers, journalists, and experts in cultural law, and none of them had any prior experience in developing chatbots. Therefore, project documentation, configurations, and automation should be designed to shorten the team learning curve and to render unnecessary the need for chatbot specialists in the team. The main characteristics of this project are depicted in Table 1.

Table 1. Characteristics of our FLOSS chatbot project.

Team Members from all needed expertise	14 members
Releases in the last 12 months	14 releases
Number of Intentions	72 annotated intents
Number of Utters	147 utters
Original Size of the FAQ	35 questions

Throughout the project, we develop incrementally and employ a set of best practices based on FLOSS, agile and DevOps values, with lessons learned from previous government-academia collaboration projects [27]. We focused on facilitating reuse by the design of a modular architecture, modular corpus, and integration of highly dynamic OSS projects, up-to-date documentation, containerization, and heavy use of continuous integration tools to orchestrate several automated actions that, together, enabled the deployment pipeline.

Based on our practical experience, we present the FLOSS chatbot for FAQ e-government services, the lessons learned, and how the black-box reuse can reduce the time of new projects of FAQ chatbots to mature.

4 THE CHATBOT PROJECT

In the following sections, we present our solution and how significant decisions were made toward reuse. While throughout the project, we had experts in the teams or developed the necessary abilities in-house, this work serves as an experience report that enables black-box reuse to empower nonexperts in developing FAQ chatbots.

Full-stack Team

Choosing the most appropriated frameworks, tools, algorithms and dialogue models require expertise in fields such as machine learning, DevOps, language, User Experience (UX), project management, business strategy, among others [5].

A typical full-stack bot team can be divided into organization, development and beta test members, and each group has one particular involvement with the project (Figure 1). It is essential to have context specialist to give the necessary content and business specialists to guide towards the business goals. Also, it is crucial to have IT specialists involved.

In the present project, the team was formed by undergraduate interns, IT professionals and professors. The DevOps role is necessary to manage automation, services integration, and continuous deploy configuration. The UX Specialists will plan all the dialog flows, chatbot personality, and vocabulary.

Finally, Data Scientists choose the techniques to process natural language, dialogues and calibrate the hyperparameters of such models.

Beta testers are essential to validate new features of the chatbot, provide feedback before it is deployed into production.

Although we have established this team structure, an essential requirement was to guarantee that a "traditional" team structure could do the maintenance, as depicted in Figure 1 (D),(E),(F). The following sections will detail how it was achieved with a focus on black-box reuse.

Overview

The complete overview of our project is depicted in Figure 2, and every component is OSS and integrated. It guarantees that the entire solution is on-premises, and how data is processed, stored, and distributed are transparent and customized.

Figure 2(A) represents the Communication Channel, and it is the interface where the user will send messages to the chatbot, and it will receive answers. Rocket.Chat¹ is a communication tool used. It enables a higher degree of automation, and the capability to store all conversations. Figure 2(B) is the core of the chatbot, where both natural language understanding and dialog management is performed. Rasa NLU and Rasa Core² are our distribution services. They implement state-of-art NLP algorithms and aim to bridge the gap between research and application. The quality of annotated intents is asserted by analyzing the dataset in Jupyter Notebooks. In Figure 2(C) is displayed the business analytics, the stack elasticseach and kibana provide dashboards to gather all information about the users, like the most asked questions, chatbot access, timestamp, and unmapped questions, which can be interpreted by the Organization and the Chatbot Team. We did not use any creation service, and the entire bot knowledge and behavior were done directly on markdown files of the distribution services.

¹<https://rocket.chat>

²<https://rasa.com>

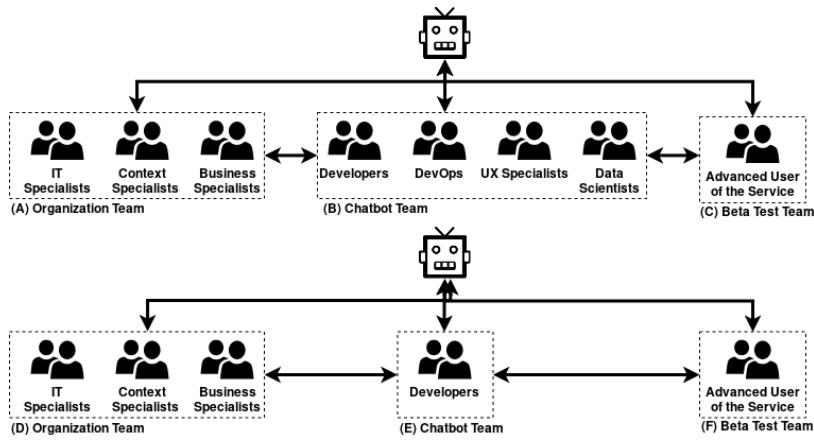


Figure 1: Chatbot team with experts (a),(b),(c) and non experts with project reuse (d),(e),(f).

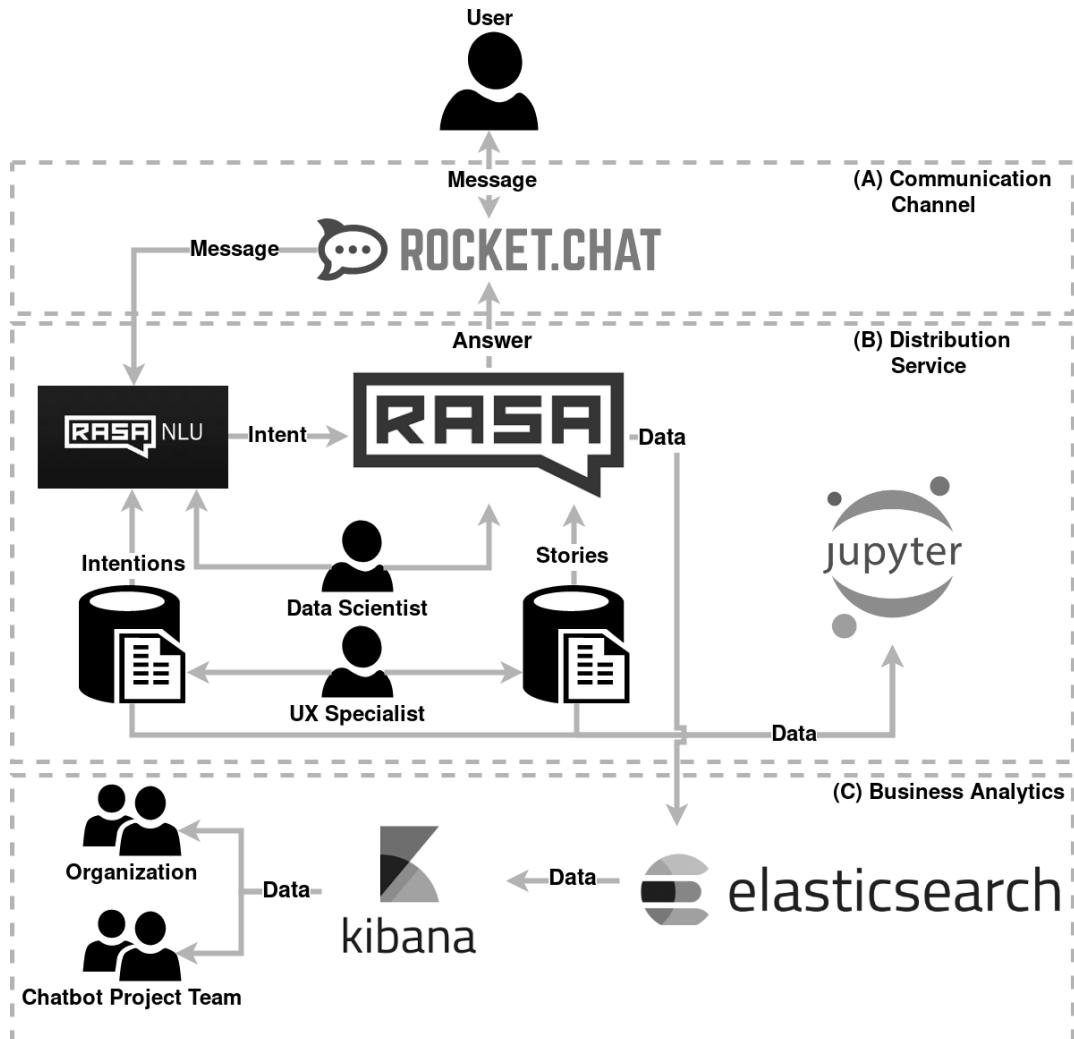


Figure 2: Project overview - technologies, artifacts, and stakeholders.

The automation of the deployment pipeline facilitates reuse of the architecture solution. We employ containerization to each service separately, and configuration files to customize production, homolog, and training environments. Finally, each component of the architecture is loosely coupled and well-encapsulated so that it can be both customized or even replaced.

Interaction Design

Designing the corpus of the chatbot knowledge base and strategy to manage the dialog is time-consuming, hand-authored by UX and interaction designer experts, which does not scale to large domains [19]. While crowdsourcing can help to overcome the problem of scale, it is only possible when it is clear the chatbot personality, vocabulary and writing style.

We addressed two main interaction design challenges: (a) a guideline of how to convert FAQ texts into annotated intents, dialog arcs and dialog flows, and (b) separate chatbot specific dataset content from general dialogue elements, such as chitchat, general intents, fallback strategy. This later could be black-box reused in any FAQ chatbot, and interaction design would be reduced to create training data to the specific FAQ domain, following provided tutorials.

We have documented the entire interaction design process, and we create tutorials, best practices, and guidelines to write annotated intents from FAQs, suggested a personality, vocabulary, and repeated the same procedure to create dialogue contents. Finally, chatbot utterances were separated in files to explicit domain-specific content from general bot behavior.

Natural Language Understanding and Dialog Management

Chatbots are based on machine learning techniques that (a) process natural language from input message from users, (b) identify the message meaning (Natural Language Understanding), and (c) select the most appropriate next action to perform (Dialog management) [8]. The choice of machine learning techniques, their pre and post-processing impact the user experience directly. For example, it influences how well the chatbot identifies user intent on a given language, its ability to infer context, to treat noncooperative behavior, and to manage complex dialog flows [8][10]. Therefore, the proper choice of Natural Language Understanding (NLU), dialog management pipeline and hyper-parameters impose restrictions on what can be done by the chatbot.

FAQ chatbots basically must identify users intents and select the correct action to execute from a closed list. The choice of the most suitable NLU and dialog management techniques depends on the amount of dataset available for training and validation. This dataset is composed by the

list of utterances annotated with intents, entities, language annotated dataset, and dialogues examples [16]. It implies that techniques execution pipeline can be borrowed across projects with similar requirements, like FAQ chatbots, as a form of generalization and reuse.

We have chosen data-driven algorithms that worked well for Brazilian Portuguese language and medium-size utterances (over 1000). For intent classification, we chose the supervised embed model Starspace [2], and we utilize annotated dialogues to train a Long short-term memory (LSTM) network [25]. This pipeline can be reused as a black-box to any FAQ chatbot in Portuguese, where the only configuration necessary is the calibration of the hyperparameters.

To validate the hypothesis of algorithms pipeline reuse, we wrote a tutorial on hyperparameters calibration, and ask a group of software engineering undergraduates, with no prior knowledge in machine learning, to develop FAQ chatbots to the University using the same pipeline. In three months, they were able to develop a chatbot and correctly calibrate the hyperparameters, guided by our tutorials.

Analytic and Improvements

According to Michaud [15], monitoring is an important step while developing a chatbot. Analytic tools track messages, distinguishes guests and users text behavior, and show in dashboards an infinity of graphics that display metrics and data from the actual chatbot use. This information enables the management of the knowledge base and guides the development efforts aligned with the business strategy.

The lesson learned from adding analytics to our chatbot project is the importance of determining the quality of the utterances and dialogue contents [15], in providing accurate data on users real questions and real behavior. This work is unfeasible manually. Metrics guide the development team to improve the quality of the interaction, and they also guide managers to draw data-driven business strategies. Table 2 summarizes some of the data provided by analytics, showing some results from our chatbot data after sixty days in the production environment.

5 DISCUSSION

Throughout this paper, we presented a report of a FLOSS FAQ chatbot project. The objective is to present some of the issues engineers, experts, managers, or researchers will likely be confronted with when developing an OSS FAQ chatbot, some core concerns they should have, and to evidence how they can benefit of black-box reuse of similar project.

Although white-box reuse is an essential practice in Open Source Projects, chatbots can benefit from black-box reuse regarding (a) machine learning models, and techniques pipeline,

Table 2. Business analytics data collected after 60 days of the chatbot in production.

Average Number of Messages sent by single user	10.2
Number of users interacting with the chatbot per day	44
Number of Default fallback occurrences in 60 days	72
Average number of times each user asks for help per conversation	2
Messages per day on week days	738
New questions not present in utterances	121
Number of intents added after analysing the data	14

(b) datasets to train both intent and dialog flows, (c) automation, configuration files, and (d) analytics tools and dashboards. Consequently, with proper documentation, tutorials, and guidelines, untrained workers/ nonexperts could build a mature FLOSS FAQ chatbot from scratch in a short period. Table 3 a list of assets that could be reused as black-box, and the ones that should be customized if our project is used to develop a FAQ chatbot in Portuguese in a different context than ours.

A common problem in chatbot projects is to find, to chose, and to configure the most appropriate set of technologies. In OSS, this problem is accentuated, once most projects only provide advanced technical documentation [4], which creates an adoption barrier to nonexperts. Additionally, some of the most advanced OSS frameworks cover only partially a chatbot architecture, and it imposes an integration effort which is also an adoption barrier to nonexperts. Finally, the common dialogue flows like greeting and chit chat are typically re-implemented every new application. Although it is readily available this corpus in English, the same is not correct to other languages. Therefore, all these barriers make a typical FAQ chatbot project to be time-consuming, filled with mistakes, waste, and to deploy immature chatbots in the production environment. We believe these projects errors can shadow the efficacy of FAQ chatbots, and by poor user experience lead by immature chatbots [14].

In 1996, Weizenbaum [26] called Eliza as a program, in 1997 Lieberman [12] called Letizia as a computer agent, but both Eliza and Letizia are classified today as chatbots. Nowadays, there is still some divergence to similar chatbot concepts, such as in [6] that evidences synonyms like conversational systems and virtual agents, chatbot. The lack of a common vocabulary added disturbance to this study. The discussion and definition of these concepts is an opportunity for academic contribution.

This study has a few apparent limitations. Around 5 groups of much more inexperienced teams, composed by undergraduate students, developed a chatbot to answer FAQ of the administration services at the university in 3 months,

reusing ours as presented in the paper. However, this validation should be done to a broader number of FAQ projects, and data about collected to validate our hypotheses fully. Future work is needed to explore more sophisticated approaches to this problem, and compare reuse in contexts other than FAQ chatbots.

6 CONCLUSIONS

In this paper, we presented the experiment report of a FLOSS FAQ chatbot project developed in the context of e-government services. We highlight the main challenges encountered when developing a chatbot distribution service and analytics for non-English dialogue agents. Several OSS projects compose the chatbot architecture. Their selection is the first difficulty, and automation is fundamental to integrate these services and enable continuous delivery. The choice of Natural Language Understanding (NLU) and Dialogue Management techniques demands the expertise of data scientists, and we found that the OSS community gives little support to non-English applications. We configured a set of techniques that performed well for the Portuguese Language with small training datasets. We provided tutorials to facilitate the reuse and hyperparameters calibration to different contexts and scopes. These automation, customizations, and tutorials enable to use this project in other FAQ chatbots in black-box reuse.

Finally, we have outlined a set of good practices, tutorials, and documentation to empower untrained workers/ nonexperts to build a FLOSS FAQ chatbot from scratch in a short period for non-English contexts.

7 ACKNOWLEDGEMENTS

The authors are indebted to the users and collaborators of this chatbot project for providing invaluable feedback and creating a supportive ecosystem. Special acknowledgment is owed to the Lab team and external contributors. The up-to-date list of contributors may be visited at <https://github.com/lappis-unb/tais/graphs/contributors>.

The Brazilian Ministry of Citizenship has financially supported this research in the TAIS project. The authors are

Table 3. Black-box reuse in FLOSS FAQ Chatbot - list of assets.

Activity	Assets to be reused as black-box	Assets to be customized for each application
Interaction Design	Language corpus General dialogue contents Dialogue tutorials of how to convert FAQ file into intents and utters Chatbot personality tutorials	Write intents, utters and dialogue flows to the desired context Follow dialogue tutorials Follow personality tutorials
Architecture	Distribution service Creation service Analytics service Continuous integration Development containers Experimental environment for dataset validation Configuration tutorials Architecture customization tutorials	Follow configuration tutorials Follow architecture customization tutorials
Natural Language Understanding	Technics pipeline to brazilian portuguese NLU hyperparameters calibration tutorial	Follow tutorial to calibrate NLU hyperparameters
Dialog Management	Dialog Management policy techniques Dialog Management policy hiperparameters calibration tutorial	Follow tutorial to calibrate the hyperparameters of the dialogue management policy
Monitoring	Dashboards templates Dashboards customization tutorials documentation	Follow dashboards customization tutorials

grateful for the stimulating collaboration and support from colleagues and partner organization.

REFERENCES

- [1] Research Advanced Laboratory of Production and Innovation in Software Engineering (LAPPIS/UnB). 2019. Tais - a FLOSS FAQ Chatbot for the Ministry of Culture. <https://github.com/lappis-unb/tais>
- [2] Tom Bocklisch, Joey Faulker, Nick Pawlowski, and Alan Nichol. 2017. Rasa: Open Source Language Understanding and Dialogue Management. *CoRR* (12 2017).
- [3] Botpress. 2019. Botpress - A Chatbot Maker & Development Framework. <https://botpress.io/>
- [4] Noel Carroll, Lorraine Morgan, and Kieran Conboy. 2018. Examining the Impact of Adopting Inner Source Software Practices. In *Proceedings of the 14th International Symposium on Open Collaboration (OpenSym '18)*. ACM, New York, NY, USA, Article 6, 7 pages.
- [5] Jessica Falk, Steven Poulakos, Mubbasis Kapadia, and Robert Sumner. 2018. PICA: Proactive Intelligent Conversational Agent for Interactive Narratives. 141–146.
- [6] B Filipczyk. [n.d.]. Chapter 12 - Success and failure in improvement of knowledge delivery to customers using chatbotâ€”result of a case study in a Polish SME. ([n. d.]), 15.
- [7] Mingkun Gao, Wei Xu, and Chris Callison-Burch. 2015. Cost Optimization in Crowdsourcing Translation. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2015)*. Denver, Colorado.
- [8] J. Harms, P. Kucherbaev, A. Bozzon, and G. Houben. 2019. Approaches for Dialog Management in Conversational Agents. *IEEE Internet Computing* 23, 2 (March 2019), 13–22.
- [9] Eric von Hippel and Georg von Krogh. 2003. Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science. *Organization Science* 14, 2 (March 2003), 209–223.
- [10] P. Kucherbaev, A. Bozzon, and G. Houben. 2018. Human-Aided Bots. *IEEE Internet Computing* 22, 6 (Nov 2018), 36–43.
- [11] C. Lebeuf, M. Storey, and A. Zagalsky. 2018. Software Bots. *IEEE Software* 35, 1 (January/February 2018), 18–23.
- [12] Henry Lieberman. 1997. Autonomous interface agents. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '97*. ACM Press, Atlanta, Georgia, United States, 67–74.
- [13] Xingkun Liu, Arash Eshghi, Paweł Świętajowski, and Verena Rieser. 2019. Benchmarking Natural Language Understanding Services for building Conversational Agents. *arXiv* 1903.05566 (mar 2019), 13.
- [14] R Lowe, N Pow, Iulian Serban, L Charlin, C.-W Liu, and J Pineau. 2017. Training end-to-end dialogue systems with the Ubuntu Dialogue Corpus. *Dialogue and Discourse* 8 (01 2017), 31–65.
- [15] L. N. Michaud. 2018. Observations of a New Chatbot: Drawing Conclusions from Early Interactions with Users. *IT Professional* 20, 5 (Sep. 2018), 40–47.
- [16] Alexander H. Miller, Will Feng, Adam Fisch, Jiasen Lu, Dhruv Batra, Antoine Bordes, Devi Parikh, and Jason Weston. 2017. ParlAI: A Dialog Research Software Platform. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (2017).
- [17] Mycroft. 2019. Mycroft AI Open Source Voice Assistant. <https://mycroft.ai>
- [18] M. Nuruzzaman and O. K. Hussain. 2018. A Survey on Chatbot Implementation in Customer Service Industry through Deep Neural Networks. In *2018 IEEE 15th International Conference on e-Business Engineering (ICEBE)*, 54–61.
- [19] Maike Paetz, James Kennedy, Ginevra Castellano, and Jill Lehman. 2018. Incremental Acquisition and Reuse of Multimodal Affective Behaviors in a Conversational Agent. In *International Conference on Human-Agent Interaction 2018*, 92–100.
- [20] Maria-Eleni Paschali, Apostolos Ampatzoglou, Stamatia Bibi, Alexander Chatzigeorgiou, and Ioannis Stamelos. 2017. Reusability of open source software across domains: A case study. *Journal of Systems and Software* 134 (2017), 211 – 227.
- [21] S. Perez-Soler, E. Guerra, and J. de Lara. 2018. Collaborative Modeling and Group Decision Making Using Chatbots in Social Networks. *IEEE Software* 35, 6 (November/December 2018), 48–54.

- [22] Ruben Prieto-Diaz. 1993. Status Report: Software Reusability. *Software, IEEE* 10 (06 1993), 61 – 66.
- [23] B. R. Ranoliya, N. Raghuwanshi, and S. Singh. 2017. Chatbot for university related FAQs. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 1525–1530.
- [24] Navid Tavanapour and Eva A C Bittner. 2018. Automated Facilitation for Idea Platforms: Design and Evaluation of a Chatbot Prototype. *Conference: Thirty Ninth International Conference on Information Systems (ICIS)* (2018), 9.
- [25] Vladimir Vlasov, Akela Drissner-Schmid, and Alan Nichol. 2018. Few-Shot Generalization Across Dialogue Tasks. *CoRR* (2018).
- [26] Joseph Weizenbaum. 1966. ELIZA—A Computer Program for the Study of Natural Language Communication Between Man and Machine. *Commun. ACM* 9, 1 (Jan. 1966), 36–45.
- [27] Melissa Wen, Paulo Meirelles, Rodrigo Siqueira, and Fabio Kon. 2018. FLOSS Project Management in Government-Academia Collaboration. In *International Conference on Open Source Systems*. 15–25.
- [28] Mairieli Wessel, Bruno Mendes de Souza, Igor Steinmacher, Igor S. Wiese, Ivanilton Polato, Ana Paula Chaves, and Marco A. Gerosa. 2018. The Power of Bots: Characterizing and Understanding Bots in OSS Projects. *Proc. ACM Hum.-Comput. Interact.* 2, CSCW, Article 182 (Nov. 2018), 19 pages.

A Survey of DevOps Concepts and Challenges

LEONARDO LEITE, University of São Paulo, Brazil

CARLA ROCHA, University of Brasília, Brazil

FABIO KON, University of São Paulo, Brazil

DEJAN MILOJICIC, Hewlett Packard Labs, Palo Alto, USA

PAULO MEIRELLES, Federal University of São Paulo, Brazil

DevOps is a collaborative and multidisciplinary organizational effort to automate continuous delivery of new software updates while guaranteeing their correctness and reliability. The present survey investigates and discusses DevOps challenges from the perspective of engineers, managers, and researchers. We review the literature and develop a DevOps conceptual map, correlating the DevOps automation tools with these concepts. We then discuss their practical implications for engineers, managers, and researchers. Finally, we critically explore some of the most relevant DevOps challenges reported by the literature.

CCS Concepts: • **Software and its engineering** → **Software development process management; Programming teams; Software post-development issues;**

Additional Key Words and Phrases: DevOps, continuous (delivery, deployment, integration), release process, versioning, configuration management, and build process

ACM Reference Format:

Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. 2019. A Survey of DevOps Concepts and Challenges. *ACM Comput. Surv.* 1, 1, Article 1 (January 2019), 35 pages. <https://doi.org/10.1145/3359981>

1 INTRODUCTION

Even though the DevOps movement has been discussed for nearly a decade, it lacks a widely accepted definition. By consolidating the most cited definitions of DevOps, we crafted our definition, similar to the one proposed by Dyck *et al.* [16], which we adopt throughout this paper:

DevOps is a collaborative and multidisciplinary effort within an organization to automate continuous delivery of new software versions, while guaranteeing their correctness and reliability.

Desiring to improve their delivery process [120], enterprises are widely adopting DevOps [7, 18, 20]. Although in discordance with most of the academic definitions, the software industry also uses the word “DevOps” to describe a well-paid job title [23][112]. Becoming a DevOps engineer is an attractive opportunity for software professionals. DevOps is also an important phenomenon studied by software engineering researchers and already a mandatory topic in software engineering courses.

Authors' addresses: Leonardo Leite, University of São Paulo, São Paulo, Brazil, leofl@ime.usp.br; Carla Rocha, University of Brasília, Brasilia, Brazil, caguiar@unb.br; Fabio Kon, University of São Paulo, São Paulo, Brazil, kon@ime.usp.br; Dejan Milojicic, Hewlett Packard Labs, Palo Alto, Palo Alto, USA, dejan.milojicic@hpe.com; Paulo Meirelles, Federal University of São Paulo, São Paulo, Brazil, paulo.meirelles@unifesp.br.

ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

© 2019 Association for Computing Machinery.

0360-0300/2019/1-ART1 \$15.00

<https://doi.org/10.1145/3359981>

DevOps and its **challenges** can be discussed from three perspectives: engineers, managers, and researchers. **Engineers** benefit from (1) qualifying themselves for a DevOps position, a technically hard task guided by over 200 papers, 230 books, and 100 tools [63]. Engineers also need to (2) learn how to re-architect their systems to embrace continuous delivery. **Managers** want to know (3) how to introduce DevOps into an organization and (4) how to assess the quality of already-adopted DevOps practices. Managers and engineers, also referred to as *practitioners*, share the necessity of choosing the best automation toolset. Finally, academic **researchers** (5) conduct studies to determine the state of practice in DevOps, thereby contributing to discussions among engineers and managers, and (6) educate a new generation of software engineers on DevOps principles and practices.

In this context, our *research problem* is devising a conceptual framework to guide engineers, managers, and academics in the exploration of DevOps tools, implications, and challenges. There are several studies and a few surveys tackling DevOps challenges. However, with few exceptions [33], they focus on a single perspective to address a given problem. In this context, this survey contributes to the field by investigating and discussing DevOps concepts and challenges from multiple perspectives: engineers, managers, and researchers. Our review also explores a much broader range of sources and is more up-to-date than previous studies. We exploit technical implication and complexity of adopting DevOps such as automation, tightly versus loosely coupled architectures, containerization versus configuration management to deployment automation, and toolset management. Previous surveys did not cover these practical aspects and implications of DevOps.

We first survey and analyze an academic bibliography. We explain the selection and analysis procedures of these works in Section 3. The selected studies are categorized and described in Section 4. Then, we complement our discussions with non-academic sources, such as books and posts from practitioners' blogs; we describe these other sources in Section 4. Also in this section, we compare our work to other reviews on DevOps.

Based on the reviewed literature, Section 5 presents the construction of a conceptual framework [99] on DevOps composed of five conceptual maps. We relate these concepts to the engineer and manager perspectives. We also look at the practical side of DevOps by analyzing the DevOps tools under the perspective of the concepts in Section 6. We categorize these tools, correlate them to concepts, and discuss which roles in the organization should use which tools.

The discussion of DevOps concepts and tools lays the basis to raise essential implications for engineers, managers, and researchers, which we present in Section 7. Although most of the implications are straightforward, some challenges are not entirely illuminated by the current literature. We summarize and discuss some of the main DevOps challenges in Section 8, debating limitations and even contradictions found in the literature.

We close with the limitations of this survey in Section 9 and with concluding remarks in Section 10. In the next section, we briefly introduce DevOps history, motivations, and goals.

2 DEVOPS

DevOps is an evolution of the agile movement [8]. Agile Software Development advocates small release iterations with customer reviews. It assumes the team can release software frequently in some production-like environment. However, pioneer Agile literature puts little emphasis on deployment-specific practices. No Extreme Programming (XP) practice, for example, is about deployment [57]. The same sparse attention to deployment is evident in traditional software engineering processes [92] and books [105, 115]. Consequently, the transition to production tends to be a stressful process in organizations, containing manual, error-prone activities, and, even, last-minute corrections [82]. DevOps proposes a complementary set of agile practices to enable the iterative delivery of software in short cycles effectively.

From an organizational perspective, the DevOps movement promotes closer collaboration between developers and operators. The existence of distinct silos for operations and development is still prevalent: operations staff are responsible for managing software modifications in production and for service levels [48]; development teams, on the other hand, are accountable for continuously developing new features to meet business requirements. Each one of these departments has its independent processes, tools, and knowledge bases. The interface between them in the pre-DevOps era was usually a ticket system: development teams demanded the deployment of new software versions, and the operations staff manually managed those tickets.

In such an arrangement, development teams continuously seek to push new versions into production, while operations staff attempt to block these changes to maintain software stability and other non-functional concerns. Theoretically, this structure provides higher stability for software in production. However, in practice, it also results in long delays between code updates and deployment, as well as ineffective problem-solving processes, in which organizational silos blame each other for production problems.

Conflicts between developers and operators, significant deployment times, and the need for frequent and reliable releases led to inefficient execution of agile processes. In this context, developers and operators began collaborating within enterprises to address this gap. This movement was coined “DevOps” in 2008 [67].

In the *Continuous Delivery* book [82], Humble advocates for an *automated deployment pipeline*, in which any software version committed to the repository must be a production-candidate version. After passing through stages, such as compilation and automated tests, the software is sent to production by the press of a button. This process is called *Continuous Delivery*. A variant is the *continuous deployment* [80], which automatically sends to production every version that passes through the pipeline. Many authors closely relate DevOps to continuous delivery and deployment [42, 46, 47, 49]. Yasar, for example, calls the deployment pipeline a “DevOps platform” [49].

Besides automating the delivery process, DevOps initiatives have also focused on using automated runtime monitoring for improving software runtime properties, such as performance, scalability, availability, and resilience [2, 3, 8, 40]. From this perspective, “Site Reliability Engineering” [58] emerged as a new term related to DevOps work at runtime.

3 STUDY DESIGN

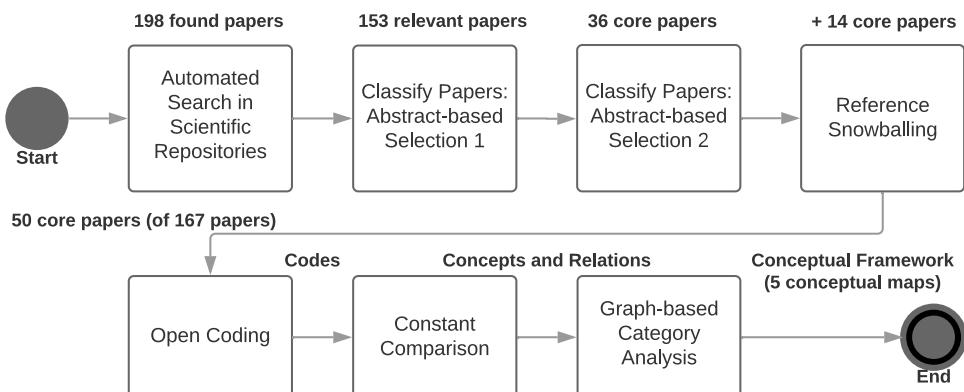


Fig. 1. Sequence of steps taken in our study

The recent growth of the DevOps community prompted a rapid increase in publications, imposing a barrier to a thorough analysis of every published work on the matter. We adopted a qualitative procedure to limit the number of analyzed studies and the information extracted from them. We established a search and selection protocol, inspired by the procedures of systematic literature reviews [62], and set analysis procedures based on Grounded Theory strategies [64]. Our primary outcome is the construction of a conceptual framework, composed of a set of conceptual maps that describe the main concepts of DevOps and how they relate to each other (Section 5). In this section, we detail our search and selection protocol, as well as our procedures to build our conceptual maps. This sequence is outlined in Figure 1.

3.1 Search and selection protocol

We searched for articles in major scientific papers repositories, namely: ACM Digital Library; IEEE Xplore Digital Library; the Spring Link. We also explored Google Scholar, but its outcomes did not provide additional relevant results. The search was conducted according to the following criteria:

- Title or abstract must contain the word “DevOps.”
- Papers must be published in journals, conferences, congresses, or symposiums (workshop papers were not considered).
- Papers must be written in English.
- Papers must have at least three pages.
- Literature reviews were not included. We present them as related work in Section 4.2.

This search resulted in 198 papers. We classified these papers into the following categories: irrelevant, relevant, and core papers. To perform the classification process, we read the abstract of all 198 papers and, in some cases, read their content too. Irrelevant papers used the keyword DevOps only for their initial background, not being indeed papers about DevOps. Using this criterion, we discarded 45 irrelevant papers, reducing to 153 the number of relevant papers to be analyzed. Then, we applied a second filter employing another set of criteria to identify papers that would be unlikely to generate any further concepts, handling theoretical saturation [?]. For example, we identified papers with overlapping topics, that would only reinforce already existing concepts. We then proceeded with our selection process and, from the 153 relevant papers, we obtained a subset of 36 core papers, which were thoroughly read. The remaining 117 ones were classified just as relevant papers.

To reduced the subjectivity of this selection process, at least two authors had to agree on each outcome; the authors produced their classifications independently and when there were different classifications, each paper was individually examined until a consensus was reached. Moreover, even though there were no hard rules for the selection process, we followed some guidelines:

- If a paper discusses DevOps peculiarities or challenges in some specific context, it is a core paper.
- If a paper is primarily centered on a single tool and does not discuss DevOps itself, it is classified as relevant.
- If a paper has an extended or more cited version, only the most relevant version is a core paper, whereas the other one is just a relevant paper.
- If several papers are about the same subject, only some of them are selected as core papers, whereas the others are classified as relevant.
- If a paper is about applying DevOps in a specific context, but does not elaborate beyond the state-of-the-art, it is discarded.
- If a paper discusses some topic (e.g., big data) and uses DevOps only as background, it is discarded.

- If a paper only presents a simple proposal or opinion, without validation, it is discarded.
- In exceptional circumstances, studies become core papers without following all the above rules if at least three authors agree.

We also applied a snowballing process [122] to cover important work not found with the query string “devops”. We explored historical, seminal, highly-cited, and recent references. From this criterion, we considered 14 additional core papers (i.e., equaling 167 relevant papers with 50 core papers).

We list all the 50 core papers on Page 31 at the end of this paper. We identify each paper with a unique reference code, composed of a sequence number and a letter indicating how we found the article: “A” for the ACM Digital Library; “I” for the IEEE Xplore Digital Library; “S” for Springer Link; and “B” for the snowballing process. This reference code is later used to display the papers on the conceptual maps.

To support our searching process and the analysis of the papers, we used some tools such as Mendeley, Libre Office spreadsheets, and SQLite database. Moreover, we developed Python scripts, for example, that generated the bar plots presented in Section 4, using Pandas and Pyplot.

3.2 Producing the conceptual framework

According to Miles and Huberman, a conceptual framework explains graphically the main things to be studied in a given field, including elements such as key factors and their relationships [99]. Our conceptual framework is composed of conceptual maps, which are diagrams structured as graphs in which nodes depict concepts and arrows represent relationships among concepts [76]. Our process of building the conceptual maps was based on Grounded Theory, which is a recognized method for conducting emergent qualitative research [65] and whose adoption in software engineering has grown considerably in recent years [117]. According to Charmaz, it reduces preconceived ideas about the research problem and helps researchers remain amenable to alternative interpretations and apprehend the data in different ways [64]. Grounded Theory is usually applied to primary sources, such as interviews and observational field-notes [111]. In this paper, we adopted Grounded Theory strategies in the literature review to handle the inherent subjectivity of conceptual analysis.

A core strategy of Grounded Theory is *coding* [64], which breaks down and labels data into smaller components [111]. In the *initial coding*, also called *open coding*, researchers avoid injecting previous assumptions, biases, and motivations [117]. Researchers then apply *constant comparison* by constantly comparing and correlating codes from different sources to produce *concepts* and group concepts into *categories* [111, 117]. In our study, we read the 50 core studies and applied *open coding* on them to produce codes. Then, through *constant comparison*, we analyzed, abstracted, and grouped the codes into concepts and categories, presenting each category in a conceptual map.

The open coding phase was carried as follows: while reading a core paper, the researchers used the Mendeley software to highlight excerpts containing potential DevOps concepts. In the following, we provide six examples of codes and their corresponding highlighted text.

DevOps emphasizes collaboration: “DevOps is a new phenomenon in software engineering, emphasizing collaboration (...) that bridge software development and operations activities (...)” [31].

DevOps advocates collaboration: “DevOps is a set of practices that advocate the collaboration between software developers and IT operations (...)” [10].

DevOps emphasizes collaboration and communication: “DevOps is a culture, movement or practice that emphasizes the collaboration and communication of both software developers and Information Technology (IT) professionals (...)” [9].

DevOps shares knowledge and tools: “DevOps was introduced to include cross-functional roles of teams with four perspectives: (...), and 4) sharing of knowledge and tools (...)” [9].

DevOps has a culture of collaboration / Collaboration means sharing knowledge and tools: “Achieving these benefits within enterprises requires the discipline of DevOps: a culture of collaboration between all team members; (...) ; sharing of knowledge and tools (...)” [22].

DevOps emphasizes working together / working together by sharing tools, processes, and practices: “That is why the DevOps movement’s recent emergence is so heartening. It emphasizes development and operations staff working together as early as possible – sharing tools, processes, and practices to smooth the path to production (...)” [48].

By applying constant comparison in these codes coming from different sources, we defined the concept “Culture of collaboration” and linked it to the “DevOps” concept in the form “(DevOps) → is a → (culture of collaboration),” with concepts represented within parenthesis. From the excerpts we also understood that a “culture of collaboration” means sharing “knowledge, tools, processes, and practices”, so we grouped these last concepts, from different sources, to state that “(DevOps) → is a → (culture of collaboration) → based on sharing → (knowledge, tools, processes, and practices),” as we can read from Figure 6 in Section 5.

As concepts emerged, we assembled a single conceptual map, linking concepts nodes in a graph. The categories emerged afterwards mostly from observing the following connectivity patterns in the graph: *i) hubs*: a single concept is linked to many other concepts, as it is the case of “teams” in Figure 6, which is linked to another 7 concept nodes, with 13 references supporting such connections; and *ii) cycles in the corresponding undirected graph*, such as the one present in Figure 6: “(DevOps) → is a → (culture of collaboration) → shared across different types of → (teams) → may or may not have a → (DevOps role) → requires certain → (knowledge, skills, and capabilities for DevOps) ← must teach ← (programming education) ← imposes challenges for ← (DevOps).” The presented hub and cycle examples led to the emergence of the *people category*. We were driven by data to identify the following categories: *process, people, delivery, and runtime*. We then fit other concepts into these categories. If some concepts did not fit in the found categories, new categories could be created, but this did not happen.

The process of building our conceptual maps also obeyed the following rules and restrictions:

- Nodes in the map contain DevOps concepts.
- The map can link nodes to describe relations among concepts.
- A link can be read as a sentence binding the linked concepts, such as “(DevOps) → is a → (culture of collaboration).”
- Each core paper received a unique reference code.
- Every concept and link must be supported by at least one core paper.
- The link’s label contains reference codes.
- A link can have different labels, each one with its references.
- Some concepts, with common links, are grouped to save space in the figure.
- A concept can belong to multiple categories.

Usually Grounded Theory procedures end with *theoretical coding* leading to the formulation of a new theory by the development of hypotheses linking concepts and categories [117]. In this work, we do not aim at the formulation of a new DevOps theory nor to answer open questions posed by the literature. Instead, we use our conceptual framework on DevOps, built with open coding and constant comparison, as input to: *i) study and classify DevOps tools; ii) identify practical DevOps implications and segment such implications for engineers, managers, and researchers; and iii) discuss the main DevOps open challenges.*

Moreover, by providing a global view of the DevOps field, we expect our conceptual maps to be used by engineers and managers to guide their continuing education, by teams to support reflection

and continuous improvements of their DevOps journeys, and by academics to identify research topics, influences, and implications.

In the next section, we give an overview of articles analyzed to build our conceptual maps, as well as other relevant sources of knowledge on DevOps. In Section 5, we present the conceptual maps we produced to structure a set of relevant DevOps concepts grounded in the literature.

4 SOURCES OF KNOWLEDGE

A diverse community of academics and practitioners around DevOps has been formed in recent years, and it is not very obvious where they publish their reports, research, and relevant information since there are very few DevOps-specific venues. In this section, we present the sources of knowledge with the most impact, which include peer-reviewed papers, books, talks, and others. We also categorize our core papers and provide an overview of the existing surveys of DevOps, providing a list of related readings on the subject. We extract from this list the readings the DevOps community consolidated as their references, which provides a reading guide for newcomers.

4.1 Peer-reviewed literature

This survey primarily relies on evaluating peer-reviewed papers, which is standard practice in the academic community for guaranteeing the quality and credibility of published works.

Figure 2 depicts the evolution of published works on DevOps according to the criteria defined in our study design (papers collected in September 2018). Similar to most research areas, conferences generally publish preliminary results. In contrast, journals publish more mature studies. Academic journals primarily reach an academic audience, while magazines also have practitioners as their readers.

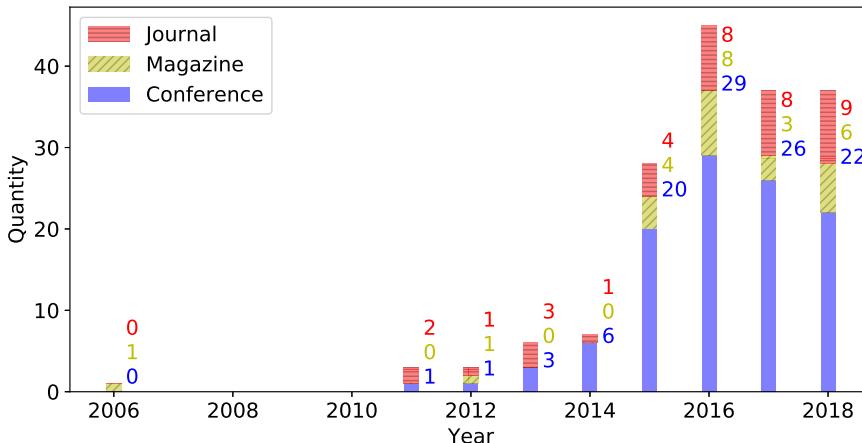


Fig. 2. Publications by source types and publication year

The “IEEE Software” magazine was a constant source of papers on DevOps, having published 13 of the 50 core papers. They are mostly relatively short papers describing real experiences faced by organizations. Therefore, it is advisable for practitioners to subscribe to this source for the latest updates on DevOps.

We did not find any peer-reviewed journal, magazine, or event dedicated exclusively to the theme. The closest related events we found were the “International Workshop on Release Engineering”

and the “International Workshop on Quality-Aware DevOps.” For researchers beginning a study on DevOps, these are two potential venues for publishing their first results. The absence of a specific venue indicates an opportunity for the academic community to organize scientific events and journal special issues dedicated to this area.

After paper selection, following the protocol presented in Section 3, we classified all 50 papers into seven categories, as shown in Table 1. The categories are described in the sequence.

Table 1. Classification of core papers

Category	Quantity	Papers
Practices	15	[3, 5, 11, 17, 19, 24, 26, 27, 30, 32, 38, 46–48, 50]
Experience report	8	[2, 6, 7, 12, 18, 20, 33, 44]
Impact	7	[25, 39–43, 49]
Concepts	7	[4, 14, 16, 21, 22, 36, 37]
Challenges	6	[10, 15, 28, 29, 31, 35]
Education	4	[1, 8, 9, 23]
Adoption	3	[13, 34, 45]

Practices: provides tools, practices, patterns, and strategies to enhance DevOps. These studies offer guidelines for selecting deployment technology [46], explanation of the use of metrics [19], how to support communication among global peers through specialized social networks [32], and how injecting infrastructure faults at production can support system reliability [3].

Experience Report: refers to reports of organizations adopting DevOps or continuous delivery/deployment. For example, Siqueira *et al.* describe the impact of a DevOps team in the context of a governmental project [44].

Impact: explores how DevOps effects other aspects of software development such as architecture [42], security [49], quality assurance [40], or the impact of DevOps on software development in specific scenarios, such as the development of software for research [12].

Concepts: introduces basic concepts in DevOps and continuous delivery, such as the deployment pipeline [22].

Challenges: covers the challenges entailed in adopting DevOps or continuous delivery/deployment. Some papers focus on specific challenges, such as DevOps in regulated domains [28], embedded systems [31], or communication [15].

Education: investigates the challenges in teaching DevOps. Articles in this category propose teaching methods [8], as well as explore the Knowledge, Skills, and Abilities (KSA) necessary for DevOps professionals [9, 23].

Adoption: covers the different models for DevOps adoption [34] and describes how an organization chooses a set of metrics to evaluate its DevOps adoption process [45]. This subject is further discussed in Section 8.2.

4.2 Related Surveys

Even though DevOps is a recent research topic, several studies have performed a Systematic Literature Review (SLR) on the subject. They aim at finding a final DevOps definition and which aspects influence its adoption on an organization. Table 2 depicts the major research questions of these previous SLR. It is worth to mention most previous SLRs performed their literature review

before 2015. It means they primarily examined the pioneering works on DevOps and did not capture the significant growth of publications depicted in Figure 2; therefore, they miss most of the currently existing literature in the topic. This is evidenced by their conclusion that initial studies have a very low quality [70].

Table 2. Primary research questions from previous DevOps SLR works

Research Question	References
RQ1: What is the meaning of the term DevOps?	[66, 70, 74, 83, 116]
RQ2: What are the issues motivating the adoption of DevOps?	[66, 70, 74]
RQ3: What are the main expected benefits of adopting DevOps?	[66, 70, 74]
RQ4: What are the main expected challenges/impediments of adopting DevOps?	[66, 70, 74, 114]

The most relevant contribution of these previous studies is to correlate the absence of a well-established DevOps definition with its effect on the perception of its benefits, expectations, and adoption challenges [70].

To overcame the lack of the initial studies quality, they deployed methods other than Systematic Literature Review to uncover both academic and practitioners perspectives and to address the novelty of DevOps. Table 3 presents a list of sources used by these other works. Our survey selected works from all cited sources and had a much broader range of analyzed works than previous SLRs. We identified 167 papers and carefully examined their titles and abstracts to select the 50 core papers, which were then extensively analyzed to provide an updated overview of DevOps.

Table 3. Sources of DevOps surveys

Author	Google Scholar	Springer Link	ACMDigital Library	IEEE Explore	Scopus	Web of Science	Others	Papers
França [66]	X					X	X	43
Erich [70]			X	X	X	X		27
Smeds [114]	X	X	X	X		X	X	27
Ghantous [74]	X	X	X	X			X	30
Stahl [116]					X			35
Jabbari [83]			X	X	X			49
Lwakatare [95]	X		X	X	X	X	X	22
This survey	X	X	X	X	X	X	X	167 (50)

Erich *et al.* [69, 70] is the most referenced literature review of DevOps. The authors performed an SLR [69, 70] and conducted focused interviews on DevOps principles and practices at six organizations [70]. From SLR, they extracted seven areas related to DevOps: “culture of collaboration, automation, measurement, sharing, services, quality assurance, and governance” [70]. They conclude that there was, in general, low quality on the academic studies on DevOps and a lack of studies to evidence DevOps’ effectiveness. From focused interviews, they analyzed how organizations implemented DevOps in four dimensions: governance, personal traits, department, and practitioners. They categorized answers from each organization into one of the seven areas extracted from SLR to portray their perceptions. Although both academic studies and organizations stated the benefits of adopting DevOps, Erich *et al.* concluded that no quantitative studies support such a claim [70], and reinforce the necessity of experimental studies to verify the effectiveness of DevOps.

França et al. [66] assumed academic studies were insufficient to address DevOps thoroughly, so they alternatively conducted a Multivocal Literature Review (MLR), in which most of the sources were extracted from gray literature, including books, websites, and industry journals and technical reports. The authors describe DevOps as a collection of principles, practices, required skills, and

organizations' motivations to adopt it. They identified seven areas related to DevOps: social aspects, automation, measurements, sharing, quality assurance, and leanness. Their main contribution was to describe DevOps characteristics associated with the practitioners' community and state-of-the-practice.

Erich *et al.* [70] and França *et al.* [66] indicate the need for more research on DevOps, especially quantitative research to evaluate DevOps' effectiveness. Even with distinct methodologies, these two works arrived at similar conclusions. They concentrate their studies on the *Process* and *People* categories of our conceptual map, to be described in Section 5. Previous surveys [66, 74, 83] did not cover the *Runtime* and *Delivery* categories of our conceptual map; they have only mentioned some of the concepts. None of the previous SLRs have discussed the technical implications and complexity of adopting DevOps practices such as automation, microservices architectures, containerization, and toolset management. The concepts and implications of practical aspects of DevOps not only empower managers to make assertive and strategic decisions but also equip engineers with best practices for DevOps adoption. We provide lessons learned of *i*) using DevOps with legacy systems; *ii*) the complexities and mistakes when adopting the microservice architecture; *iii*) guidelines for automating the delivery pipeline; *iv*) the advantages and drawbacks of teams autonomy in toolset selection. These points form the major contribution of our survey, wherein we deepen the analysis of practical aspects of DevOps adoption with the aid of a conceptual map of DevOps to guide engineers, managers, and researchers toward mastering skills necessary to enable DevOps. Additionally, the quality of studies on DevOps improved in recent years, so we have updated the implications of deploying DevOps in an organization as well as contributed to more technical implications for practitioners and researchers, such as re-designing systems architecture, deployment pipeline, and quantitative assessment metrics. Previous SLRs did not cover these latest subjects.

4.3 Books

Books are the standard source of knowledge on DevOps among practitioners. Although they usually have more pages than academic papers, they are easier to read and to assimilate. We searched Amazon.com on December 2018, for books written in English containing “DevOps” in the title. Figure 3 presents the annual distribution of a total of 238 books found in this search. Comparing to Figure 2, it is possible to see their shapes roughly shift by one year, showing how academic publication precedes book publication.

One of the seminal books on DevOps is “Continuous Delivery” [82], from 2010. It does not explicitly use the word “DevOps,” but it describes in detail the deployment pipeline pattern, which is usually central to DevOps strategies. The oldest book found at Amazon containing the word “DevOps” in its title is “DevOps: High-impact Strategies” [107] from 2011. Some of the books frequently cited by our reviewed literature are “The Phoenix Project” [89] and “DevOps: A Software Architect’s Perspective” [55]. The most popular books at Amazon containing the word “DevOps” in the title are “Accelerate: The Science of Lean Software and DevOps” [71], “The Phoenix Project” [89], and “The DevOps Handbook” [90].

It is worthy to note that three authors of these cited books (Nicole Forsgren, Jez Humble, and Gene Kim) also collaborate with the State of DevOps Reports [53, 120], a vital source of knowledge on DevOps, as discussed next.

4.4 Other sources

State of DevOps Reports: annual industrial research published since 2013 by renowned authors in the DevOps field based on the survey of multiple organizations worldwide [53, 120]. The reports

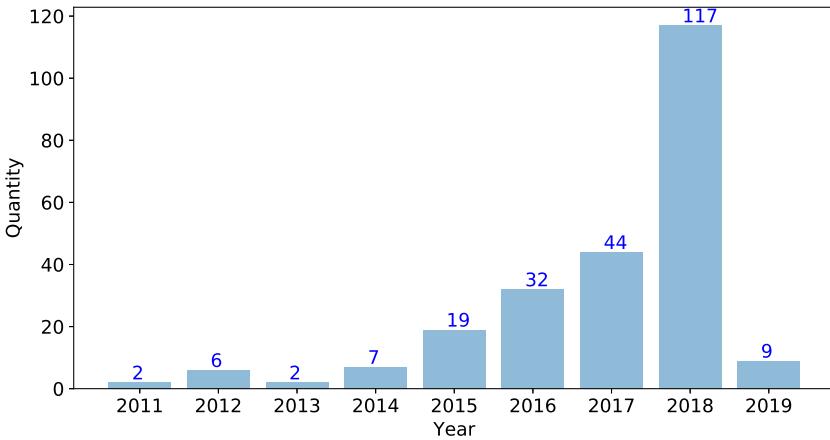


Fig. 3. Distribution of books about DevOps by publication year according to Amazon.com

provide proper usage of quantitative metrics for assessing DevOps initiatives. More about this will be discussed in Section 8.3.

The Twelve-Factor App: a manifest with architectural principles aligned with the DevOps philosophy [121]. It is a reputable source amongst practitioners and is typically used to judge architectural decisions. The present instructions like: (a) keep development, staging, and production environments as similar as possible, and (b) use environment variables to store configuration that varies across environments, among others. A related set of principles is the Reactive Manifesto, which is also used by practitioners as a guideline [59].

Talks and Videos: there are many practitioners' conferences about DevOps and conferences that include DevOps talks. Many practitioners rely on these talks for up-to-date information on DevOps, especially when the speakers are from reputable companies, such as Amazon [60], Microsoft [61], Google [119], Netflix [52], and Opscode [84]. Many of these conference talks are made available on the web, where they join many other short videos about DevOps. On YouTube or other video platforms, dozens of videos have titles matching the “What is DevOps” string.

Global community: engineers leverage global-community knowledge to overcome their daily challenges. This knowledge comes from code repositories, such as GitHub and Chef Supermarket, and from discussion forums like stackoverflow.com [32]. In this way, engineers interact not only with colleagues within the enterprise but also with peers around the world, building a global community.

5 FUNDAMENTAL CONCEPTS

In this section, we present a *conceptual framework* of fundamental concepts of DevOps. The concepts emerged from our systematic analysis of the literature, which followed the protocol presented in Section 3. Our contribution is to provide a relevant and structured set of concepts on DevOps, grounded in the literature, to support analysis and understanding of DevOps challenges.

Our conceptual framework [99] on DevOps is composed of a conceptual map outlining the conceptual categories and of other four conceptual maps that are diagrams structured as graphs in which nodes depict concepts and arrows represent relationships among concepts. There is always

at least one core bibliographic reference supporting a particular relationship. The references are represented by codes that can be identified in the List of Core Papers on page 31.

The concepts are distributed into four major categories: process, people, delivery, and runtime. The *process* category encompasses business-related concepts. *People* covers skills and concepts regarding the culture of collaboration. *Delivery* provides the concepts necessary for Continuous Delivery, and, finally, *runtime* synthesizes concepts necessary to guarantee the stability and reliability of services in a continuous delivery environment.

While the process and people categories relate more to the management perspective, runtime and delivery relate more to an engineering perspective. Moreover, while *delivery* concepts relate more to developers, *runtime* concepts relate more to the traditional operator role. The categories are depicted in Figure 4 and described in this section.

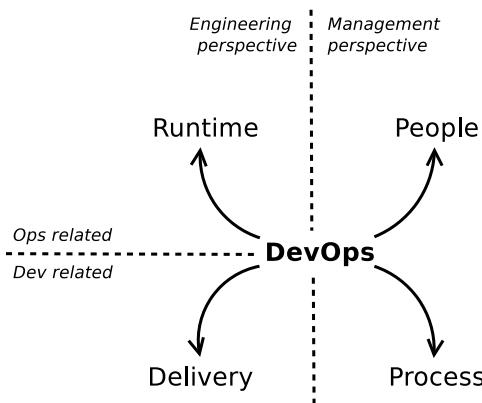


Fig. 4. DevOps overall conceptual map

Before proceeding to the categories descriptions, we briefly present some reasons to support that these categories are enough to frame the DevOps field: *i*) it seems reasonable to split “dev” and “ops” concepts as induced by the categories *delivery* and *runtime*; *ii*) it makes sense to separate more technical concepts from less technical concepts, as caused by the separation between the perspectives of engineering and management. *iii*) These categories match our DevOps definition: “a collaborative and multidisciplinary effort within an organization” regards *people*; “continuous delivery” is a *process* that is “automated” by the *delivery* techniques, which also guarantee software “correctness”; finally, software “reliability” is promoted by the *runtime* concepts.

5.1 Process

DevOps aims to achieve some business outcomes, such as reducing risk and cost, complying with regulations, and improving product quality and customer satisfaction. We grouped these concepts into the *process* category of concepts, presented in Figure 5, which reveals that DevOps achieves such business outcomes through the accomplishment of a process with frequent and reliable releases. In particular, the diagram explicit that continuous delivery leads to product quality and customer satisfaction because of the short feedback cycle it provides.

One could argue that rigorous human and hierarchical approval processes can reduce risk, comply with regulations, and provide product quality. Nevertheless, DevOps is different, once its practices are based on agile and lean principles, which embrace change and shorten the feedback cycle, as depicted in the diagram.

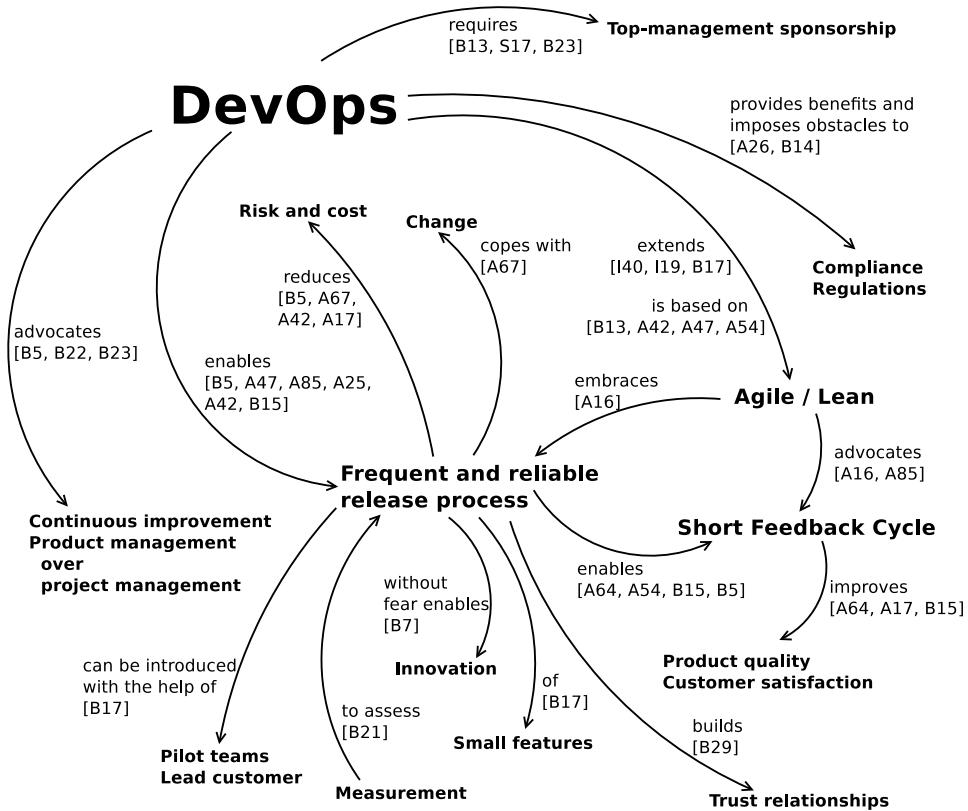


Fig. 5. Process conceptual map

5.2 People

The term “DevOps” centers on the idea of bringing together *people* from development and operations through a culture of collaboration. The concepts around this idea were grouped in the *people* category, which is presented in Figure 6. It depicts that DevOps intends to break down the walls among silos, aligning the incentives across the organization. However, breaking down silos raises many questions concerning concepts depicted in the diagram: how can organizations perform a cultural shift that implies more responsibilities for developers? How do developers acquire operations skills? Can developers and operators work in the same team and still keep different job titles? Should “DevOps” be a role? Should teams be cross-functional and include operators? Should an operator be exclusive to one team? What does it mean being an operator in the DevOps context? Although the concepts of this category are frequent in the literature, the answers to the previous questions are not clear yet, so we debate them in Section 8.2.

5.3 Delivery

The core strategy for achieving a frequent and reliable delivery process is the automation of the *deployment pipeline*, from which DevOps tools and techniques emerge. We grouped the concepts surrounding the deployment pipeline into the *delivery* category, which is presented in Figure 7. Automation tools, as depicted in the diagram, are usually open source and enable core DevOps

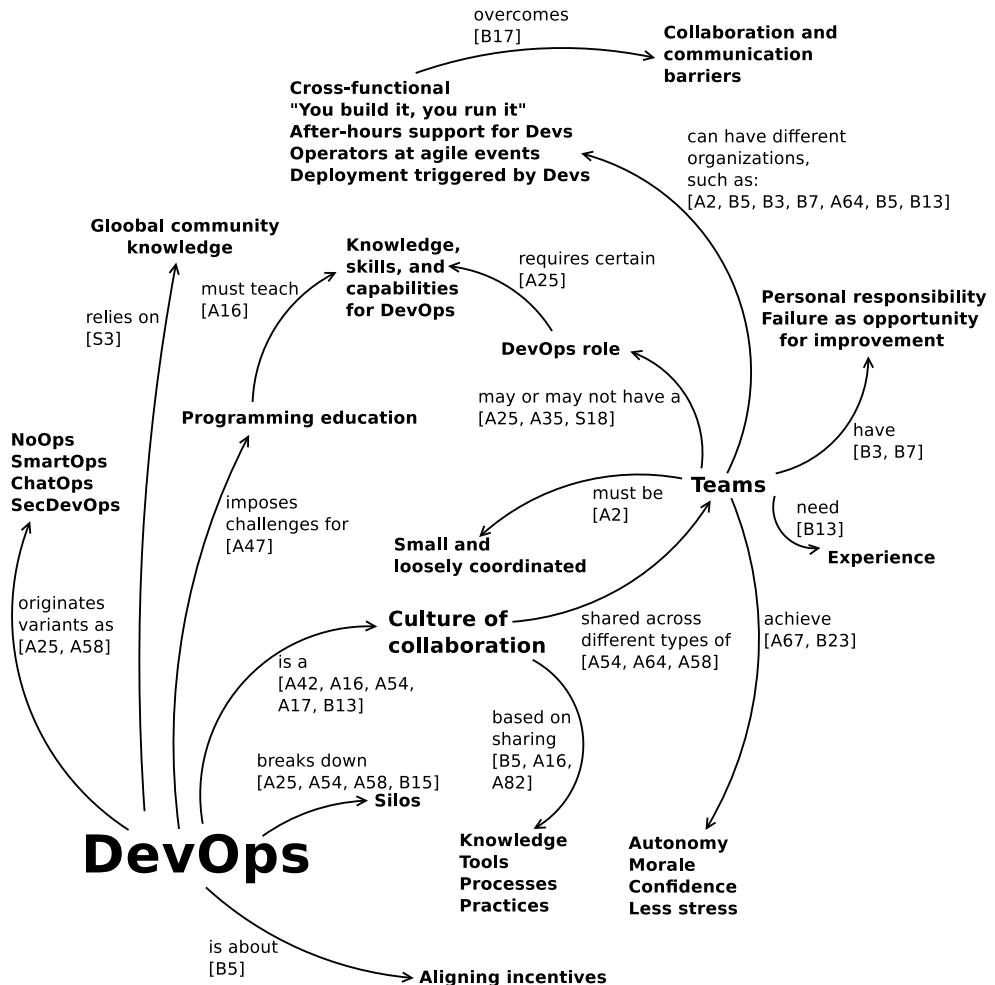


Fig. 6. People conceptual map

practices, such as versioning, testing automation, continuous integration, and configuration management. The diagram also shows the interconnection between DevOps and microservices: one supports the other, as we explore in Section 8.1. Other concepts related to microservices are also depicted, such as backward compatibility and API versioning.

There is still some debate on concrete strategies for tooling, such as the containerized approach leveraged by Docker on the one hand, and continuous configuration convergence, such as in Chef and Puppet, on the other. We discuss such debates in Section 6. However, we claim the *delivery* concepts are much more stable and accepted by the community than the *people* concepts.

5.4 Runtime

It is not enough to continuously deliver new versions, but it is also necessary for each new version to be stable and reliable. Thus, *runtime* concepts are a subsequent and necessary extension of DevOps. The *Runtime* category of concepts is presented in Figure 8, which shows the desired outcomes, such as performance, availability, scalability, resilience, and reliability. The figure also

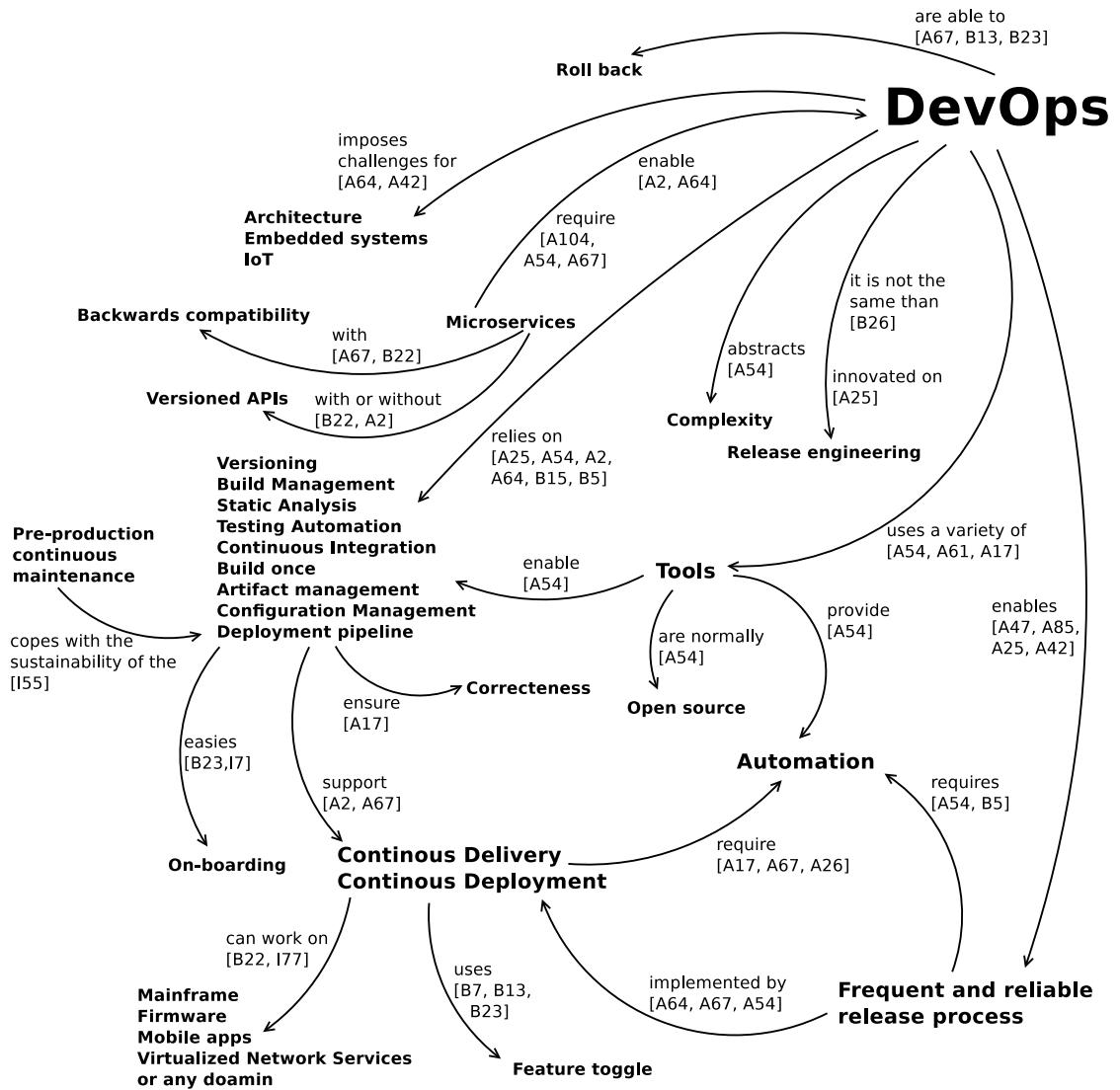


Fig. 7. Delivery conceptual map

shows paths to achieve the mentioned outcomes, like the use of infrastructure-as-code, virtualization, containerization, cloud services, and monitoring. As depicted by the diagram, DevOps can monitor high-level business metrics or low-level resource metrics. Another topic, also shown in the figure, is to run experiments in the production environment, like injecting failures to ensure software reliability, as advocated by the chaos engineering approach [3]. All of this reduces human intervention to ensure software reliability, which is another factor that challenges the traditional role of operators, as we argue in Section 8.2.

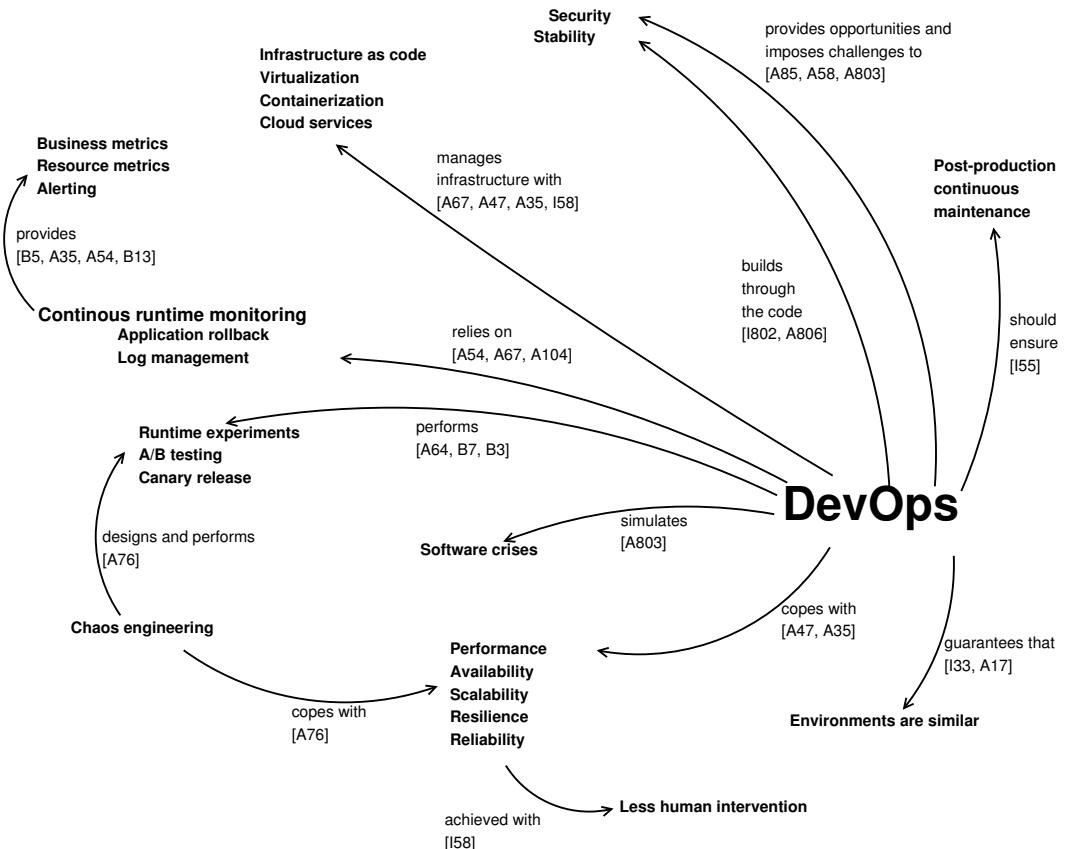


Fig. 8. Runtime category of concepts

We complement our conceptual analysis with the practical side of DevOps by investigating DevOps' tools and actors, and their relations to concepts. As the presentation of DevOps' concepts has already raised some concerns that demand further discussion, some issues will also emerge from the discussion on DevOps tools in the next section.

6 TOOLSET

We define “DevOps tools” as the tools pursuing one of the following goals: 1) assisting *human collaboration* across different departments; 2) enabling *continuous delivery*; or 3) maintaining software *reliability*. Our contribution is to associate tools to concepts presented in Section 5. This correlation between tools and concepts equips and empowers practitioners for well-informed decisions on tooling, and thereby enforces alignment between technical, managerial and organizational decisions. This conceptual approach becomes immensely important when considering the ever-changing nature of DevOps tools [42].

Although tools matter, caution is required to avoid turning them into the core of a DevOps strategy. Adopting new tools demands significant effort, and applying this effort without considering

the expected outcome can lead to considerable waste. For example, a DevOps adoption process must consider team structuring as more critical than the choice of specific tools.

DevOps heavily relies on automation tools. The choice of a proper toolset imposes challenges for DevOps professionals. It is not feasible for a single person, even an organization, to master all the DevOps tools, given the vast number of available tools [17, 27] and their rapid evolution [42].

We also contribute to discussing the most important categories of DevOps tools, presented in Table 4, by adding the following complementary information to each category: *i*) some of the most used tools in each category; *ii*) the intended users: whether developers or operators; *iii*) the goals: *human collaboration, continuous delivery or reliability*; and *iv*) DevOps concepts.

The complementary information, like tool examples, helps the reader to understand each category, leading to the following additional contributions: *i*) helping DevOps practitioners focus on organization requirements, while choosing the most suitable set of tools; *ii*) helping the reader to grasp a critical view about the objective of each tool and how to use it in an organization; *iii*) showing how the frontiers between developers and operators have been blurred in the DevOps context through toolset usage.

6.1 Tools for knowledge sharing

The DevOps strategy focuses on human collaboration across silos. Concretely, this entails different departments sharing knowledge, process, and practices, which requires sharing specific tools.

We cite GitLab and Rocket Chat as examples of tools in this category. Gitlab not only provides a code repository but also has a wiki system that enables developers and operators to share knowledge. Gitlab offers an issue management system, which is typically used by developers and business analysts to share knowledge through agile practices like issues comments and pull requests processes. Making operators also familiar with these issues helps them to understand the product and project contexts. Moreover, documenting production problems known by operators in the issue management system is an essential step toward handling non-functional requirements prioritization in the development flow. ChatOps (Chat and Operations) is a model that connects people, tools, process, and automation through conversation-driven interactions mediated by tools [102]. Rocket Chat is a communication tool that implements ChatOps by enabling a higher degree of automation and tool integration through the use of webhooks and chatbots.

These tools support *human collaboration* and the *people* category of concepts. However, despite the centrality of human collaboration in DevOps strategy, the “DevOps Periodic Table” [63], a well-known list of more than 100 DevOps tools, includes only eight tools for collaboration. This fact may suggest that human collaboration and *people* concepts have been eclipsed in the DevOps movement by the more technical goals and concepts.

6.2 Tools for source code management

Source code management tools usually intend to promote collaboration among developers. These tools are basic blocks to implement continuous integration and, therefore, *continuous delivery*. From this traditional perspective, one could relate source code management solely to the *delivery* category of concepts.

However, source code management can also be used by operators to store artifacts and automation scripts and to access software information that can impact operations activities. For example, a development bug may cause a memory leak that impacts operations. When storing infrastructure related artifacts, whether in the infrastructure-as-code style or just as plain text, operators provide developers better insight into how the software is executed. Therefore, source code sharing among developers and operators becomes a real point of collaboration.

The most traditional tools in this category are SVN and Git. More complete platforms, such as GitLab and GitHub, can also wrap Git, providing easier-to-use visualizations for code changes, as well as integrating with additional tools like the issue manager.

6.3 Tools for the build process

Build tools are highly developer-centered. Their goals relate to enabling *continuous delivery* and the *delivery* category of concepts. We have considered not only tools that generate deployable packages, but also called *builds*, but also tools that generate essential artifacts and feedback using the source code as input.

Each programming language has some tools to cope with the build process by supporting dependencies resolution, implementation of custom tasks, or generation of deployable packages. Examples of dependency managers, also called package managers, are Pip for Python, RubyGems for Ruby, NuGet for .NET, and Maven and Gradle that compete in the Java landscape. Gradle eases the implementation of custom tasks during the build, as also done by GNU Make, that is often used for GNU/Linux packages, or Rake for Ruby. Some of the cited tools, such as Maven, are also responsible for producing the deployable package, such as WAR files for Java environments. However, for some languages, like Python and Ruby, there is no need for producing a deployable package as a single-file artifact. It is also possible to use more generic package tools coupled to the target environment, such as Debian packages.

Each programming language also has some unit-test frameworks, which provide vital feedback for developers about the correctness of the software. Some examples are JUnit and Mockito for Java, RSpec for Ruby, and NUnit for .NET. More sophisticated testing which automates the end-user behavior for web applications is possible with browsing automation tools, such as Selenium.

Another type of feedback for developers is regarding source code quality, which is provided by source-code static analysis tools, such as SonarQube and Analizo [118]. SonarQube classifies code problems and evaluates coverage metrics as well as technical debts in several programming languages via its plugins. Analizo supports the extraction of a variety of source code metrics for C, C++, C#, and Java codes. In general, source-code analysis tools can point to issues in source-code, like non-compliance to the standard style, problems in maintainability, risk of bugs, and even vulnerability breaches. Source-code static analysis tools vary in supported programming languages and in the way they are delivered. They can be provided as-a-service, for example, Code Climate, or even within the developer environment through tools such as PMD for the Eclipse IDE.

One critical requirement for a build tool in DevOps context is automation. There are some products whose build actions can be only triggered through a Graphical User Interface (GUI), which is not acceptable in a continuous delivery process.

6.4 Tools for continuous integration

Continuous integration tools orchestrate several automated actions that, together, implement the *deployment pipeline* pattern. Among the stages orchestrated by the pipeline are: package generation, automated test execution for correctness verification, and deployment to both development and production environments. These tools are related to the *delivery* category of concepts.

The main actors responsible for defining the pipeline structure are typically developers. Operators usually collaborate on defining the deployment stages; they are also in charge of maintaining the continuous integration infrastructure running as a service to developers. For this, operators run continuous integration tools such as Jenkins or GitLab CI (about.gitlab.com/features/gitlab-ci-cd). Deployable packages can be stored on repositories like Nexus for enabling future rollback.

Since downtime of continuous-integration infrastructure results in the interruption of continuous delivery, it is common to use highly-available third-party services, such as GitLab.com and Travis.

6.5 Tools for deployment automation

Deployment automation tools are employed in the deployment stages of the pipeline to make the *continuous delivery* process viable. They enable frequent and reliable deployment processes, as well as other concepts related to the *delivery* category. Although the use of deployment automation tools is a joint effort between developers and operators, the primary mission of continuous delivery is putting the deployment schedule under business control.

Every automated deployment approach relies on the concept of “infrastructure as a code.” It requires engineers to specify servers, networking, and other infrastructure elements in files modeled after software code [101]. In this model, deployment and infrastructure definitions are shared among developers and operators, allowing effective cooperation between them.

Automated deployment can encompass not only the deployment of the package to the production environment but also the provisioning of the target environment. Such provisioning is usually performed in cloud environments [11], such as Amazon Web Services, Google Cloud, and Azure. These platforms deliver a vast amount of infrastructure services via Infrastructure as a Service (IaaS) model, like launching virtual machines, databases, queues, etc. The creation of all these resources can also be orchestrated, in Amazon’s platform, through the usage of AWS Cloud Formation. Operators and developers can share the task of using IaaS services.

It is also possible to use a Platform as a Service (PaaS), such as Heroku. In the PaaS approach, the platform is responsible for the deployment, and developers do not know the underlying virtual infrastructure. Serverless services – a new trend in cloud computing – aim to abstract servers for developers, which is close to PaaS; the main differences are the scaling transparency and the charge for computing usage [106]. These models suggest there is no need for specialized operations teams [11]; or, at least, that fewer people are required for operations [106].

If using IaaS, the environment provisioning is followed by the execution of scripts that effectively install the application in the target environment. A deployment script can be written using Shell Script, but configuration management tools such as Chef and Puppet offer advantages by leveraging operation system portability and idempotence mechanisms, which are difficult to achieve with Shell Script. An example of language construct of Chef that leads to portability is using the “package” resource, which is resolved to a concrete package manager, such as Apt, only at execution time. An example of an idempotent mechanism of Puppet in the “service” resource is declaring the desired final state of the service as “running” rather than writing a command to start the service [110].

Another alternative for deployment is containerization, primarily implemented by Docker. Docker containers resemble virtual machines. The main difference is that they are more lightweight [26], considering they share the kernel of the host. Docker and related tools, such as Docker Compose, Kubernetes, and Rancher, allow the specification of containers and their dependency relationships. These specifications generate container images in the build stage, which are later instantiated in the target environment. Docker has been used not only for deploying applications but also for deploying the underlying infrastructure [26].

Containerization could be seen as complementary to the deployment script strategy (done by Chef or Puppet). However, in practice, these strategies seem to compete. A Chef script is executed continuously on the target node, and its success depends on the previous target node state. On the other hand, in containerization the whole containerized environment is generated in the build, so the environment is destroyed and rebuilt at each new software version. When compared to the Chef strategy, Docker yields faster and more reliable deployment [26], but at the expense of bigger builds.

Complimentary usage of configuration management and containers entails the setup of Docker environment and update of the operating system software, like SSH. Another usage is to manage

configuration that varies across diverse environments (testing, staging, production). Such a configuration cannot be embedded in container images since the same image must be deployed in every environment. Nonetheless, the debate about configuration management versus containers is not fully explored in the literature. Among practitioners, although it is possible to find support for a complementary usage of both [54], there is a trend to favor containers over configuration management tools [68, 79, 85].

Zhu *et al.* compare the provisioning reliability of two approaches: using “lightly baked images” versus “heavily baked images” [50]. In the lightly baked approach, a virtual machine image is instantiated, and Chef is executed to install the application, whereas in the heavily baked approach the image already contains the entire application. This heavily baked approach is similar to the container strategy, but using virtual machine technology. The authors conclude that deployment using lightly baked images is less reliable since it involves more external resources during deployment, which tends to increase errors and delays.

As the application evolves, the database structure evolves as well. Traditionally, “database administrators” maintain the database structure. The adoption of emergent design [96] and the need for frequent releases have encouraged the use of database migrations tools such as Flyway, which controls the automated application of schema updates across different environments, thus enabling developers to manage database structure themselves. In this context, one must rethink the traditional role of database administrators.

Developers and operators share these tools for deployment. The main risk is the absence of clarity of responsibilities, which can cause friction in organizations during DevOps adoption [34]. Developers believe they do not control their application and feel they are doing someone else’s job. Additionally, operators may judge developers as unable to perform some tasks or operate some tools.

6.6 Tools for monitoring

Monitoring tools usually track applications’ non-functional properties, such as performance, availability, scalability, resilience, and *reliability*. Self healing, alerts, log management, and business metric follow up are example tasks performed by monitoring tools; they relate to the *runtime* category of concepts.

Example of tools for monitoring and alerting are Nagios, Zabbix, and Prometheus. Examples of log management tools are Graylog and Logstash. Cloud services also play an essential role in guarantying non-functional properties of applications since they provide elastic resources that can be allocated on demand [98]. It is also typical for cloud services to provide monitoring and alerting services.

The tendency toward cross-functional, full stack teams [2, 17], combined with the expectation that developers must be accountable for the product [18, 20] pushes the use of these tools to the development team. Therefore, once again, the responsibility for using and mastering specific tools is unclear.

6.7 Actors

The still undefined division of responsibilities in the DevOps world makes it difficult to associate a role to each DevOps tool. At the same time, the variety of DevOps tools seems to challenge the idea of a single person holding the title of a “DevOps engineer.” Even a whole cross-functional team may struggle to know all these tools. On the other hand, companies preserving the operations department may have difficulty in precisely defining which roles should use which tools.

Humble and Molesky suggest operations teams should provide IaaS to product teams [22], such as continuous integration platform, compute provisioning, and monitoring services. Traditional

Table 4. Major DevOps tools related to actors and DevOps concepts

Category	Examples	Actors	Goals	Concepts
Knowledge sharing	Rocket Chat GitLab wiki Redmine Trello	Everyone	Human collaboration	Culture of collaboration Sharing knowledge Breaking down silos Collaborate across departments
Source code management	Git SVN CVS ClearCase	Dev / Ops	Human collaboration Continuous delivery	Versioning Culture of collaboration Sharing knowledge Breaking down silos Collaborate across departments
Build process	Maven Gradle Rake JUnit Sonar	Dev	Continuous delivery	Release engineering Continuous delivery Automation Testing automation, Correctness Static analysis
Continuous Integration	Jenkins GitLab CI Travis Nexus	Dev / Ops	Continuous delivery	Frequent and reliable release process Release engineering Continuous integration Deployment pipeline Continuous delivery, Automation Artifact management
Deployment automation	Chef, Puppet Docker Heroku Open Stack AWS Cloud Formation Rancher Flyway	Dev / Ops	Continuous delivery Reliability	Frequent and reliable release process Release engineering Configuration management Continuous delivery Infrastructure as code Virtualization, Containerization Cloud services, Automation
Monitoring & Logging	Nagios Zabbix Prometheus Logstash Graylog	Ops / Dev	Reliability	You built it, you run it After-hours support for Devs Continuous runtime monitoring Performance, Availability, Scalability Resilience, Reliability, Automation Metrics, Alerting, Experiments Log management, Security

product teams become responsible for using these tools, whereas operations teams are responsible for set-up and maintenance, ensuring their performance and availability, and consulting in their usage. In this scenario, the operations team becomes itself a product team [22].

This pattern bears that of adopting outsourced development infrastructures, such as GitHub or GitLab. However, outsourcing the infrastructure has additional benefits, like freeing the team to experiment with different tools [20], and not only the ones supported by the operations staff. Most of the time, reputable infrastructure providers have better availability than in-house solutions. We discuss further the possible DevOps adoption approaches and their consequences in Section 8.2.

7 IMPLICATIONS FOR ENGINEERS, MANAGERS, AND RESEARCHERS

DevOps principles, practices, and tools are changing the software industry. However, many industry practitioners, both engineers and managers, are still not aware of how their daily work can be

affected by such principles, practices, and tools. By surveying the DevOps literature, we found several implications of DevOps adoption for industry practitioners and for academic researchers. In this section, we lay out such implications as a practical guide to help professionals to adapt to the significant impacts of DevOps in their fields.

Our contribution is to discuss the DevOps implications for each perspective individually: engineers, managers, and researchers. In these implications, readers will find issues they will likely confront, some core concerns they should have, and potential solutions already adopted by the community. We also outline DevOps-related topics that the academic community could exploit in the future. Finally, presenting these implications raises relevant considerations, preparing our discussion of unresolved DevOps challenges in Section 8.

7.1 Implications for engineers

Based on the reviewed literature, we list DevOps implications affecting how engineers must architect systems, interact with their peers, and even how to adopt processes, such as incident handling.

Microservices: adopting microservices architecture is recommended in conjunction with adopting continuous delivery [2, 42]. Loosely coupled and well-encapsulated microservices architecture lead to good system testability and deployability [21]. Since microservices do not come without new challenges, we further discuss this topic in Section 8.1.

Cloud services: architecture patterns involving cloud services can assist in the deployment and operation of applications [11], diminishing the need for dedicated operations teams [106]. It is particularly useful in the context of cross-functional teams.

Rolling back: several authors state that DevOps engineers should be able to roll back applications in case of problems after deployment [6, 10]. However, in scenarios with complex integration or database evolution [6], rolling back can be a tricky task [86]. Alternatives such as feature toggle [78] or handling the root causes of problematic releases can avoid rolling back [61, 86].

Embedded systems and IoT devices: deployment on embedded systems can be hard, especially when only the customer owns and controls the hardware platform [31]. Therefore, engineers must customize the updating mechanisms for embedded systems and IoT devices.

Inhibitors for high-frequency delivery: engineers must be aware of scenarios that could impose barriers to high-frequency delivery. Updating the software of automation control systems can require a factory to stop its production [29], which can be costly. Traditional techniques to avoid downtime on deployment, such as Blue Green Deployment [73] and Canary Release [109], may not be suitable for critical applications, such as for factories, and medical equipment. Another less critical scenario, such as mobile apps, can wait for a release cycle to publish a new update [29].

Testing: although companies recognize the importance of automated testing, they still struggle to implement it [35] fully. It is especially true for user interface tests automation [29]. Other factors that make automated testing complex are hardware availability for load testing and user experiment assessment [29]. Parallelizing test suites can be necessary to reduce testing time [33]. Tests that seem to fail at random called “flaky tests,” are not acceptable [33].

Quality assurance team: quality assurance skills are necessary for locating specific error scenarios and corner cases. However, preserving the quality assurance team separate from the development team is questionable. DevOps and agile practices require a change in the role of quality assurance teams, or even its elimination [4].

Legacy systems: although it takes a great deal of work to achieve continuous delivery on mainframe platforms, there are successful reports of it [21]. Some legacy architectures might not be designed to run automated tests [29]. Nonetheless, teams must be aware that cultural factors, such as managers who say “This is the way we have always done it” [21], can limit the adoption of continuous delivery more than technical factors.

Communication: improving communication among organizational silos is key to DevOps adoption [22]. However, practitioners must be aware that effective interdepartmental communication is still a challenge, especially within remote teams [15].

Learning: software professionals must be prepared to learn new tools, focusing on automation [63]. They must also cope with the ever-changing nature of these tools, which implies maintaining heterogeneous environments and migrating technologies [42].

Building the deployment pipeline: the benefits delivered by a deployment pipeline are many. However, engineers must be aware that setting up the infrastructure for continuous deployment can demand a considerable effort [29, 33]. Breaking down the system into microservices also requires building multiple pipelines. Engineers should not try to build all the continuous delivery ecosystem in a single step: an approach based on continuous improvement is preferable [33].

Pipeline maintenance: pipeline execution generates a lot of artifacts, such as build and logs. Artifacts such as production logs, bug tracks, parameters configuration, and temporary files, must be appropriately archived and removed at some point [36]. Despite its importance to pipeline sustainability, organizations often overlook this maintenance process [36].

On-boarding: the automation built for carrying continuous delivery also promotes faster on-boarding of new members to the team [33].

Incident handling: software developers must be educated in software security and must cope with incident handling [24], bug tracking, systems failure, or take over primary responsibility for this activity.

Coding for stability and security: although software stability and security are traditional operational concerns, in a DevOps context such non-functional requirements must be leveraged by software implementation, including the code for deployment automation and a consideration of the possibility of failures and delays in the underlying infrastructure [30, 38, 50].

7.2 Implications for managers

Based on the reviewed literature, we list implications related to how managers must face the DevOps phenomenon: required management and cultural paradigms, training people, structuring and assessing the DevOps-adoption process, as well the expected outcomes from this process.

Adoption of lean principles: since DevOps is based on lean principles [10], organizations eager for DevOps adoption should take a step back and learn them [104]. In particular, Kim recommends [88]: 1) mapping the value stream for optimizing global system performance, rather than for local optimization; 2) amplifying continuous feedback loops to support necessary corrections; and 3) improving daily work through a culture promoting frequent experimentation, risk-taking, learning from mistakes, and knowing that practice and repetition are prerequisites to mastery.

DevOps adoption: how to deploy DevOps in an organization is a critical question for managers. However, the lack of a consensual definition of DevOps is a reason for making it a hard decision. Many studies explore this subject, organizations still struggle with it, and we discuss it in more depth in Section 8.2.

Assessment: an organization should be able to measure the success of a DevOps adoption process. Nonetheless, any top-down imposition of a metric-based evaluation must be used with care. If personal evaluation depends on such metrics, engineers can focus on producing good numbers rather than improving the software process [61]. We further discuss how to assess the quality of DevOps practices in Section 8.3.

Training: DevOps demands additional technical skills from software professionals. Developers must acquire skills from operators and vice versa. It is necessary to conduct training in using not only tools but also in using DevOps concepts to keep pace with tooling's rapid pace evolution [42]. Top management sponsorship for training is often necessary for many contexts.

Job titles: Defining job titles impacts hiring and training. Although the DevOps movement emerged to approximate developers and operators, nowadays, the industry adopts the role of DevOps engineer, which executes tasks mostly linked to scripting automation and CI/CD practices [23]. However, this role blurs with other ones, such as release engineer and build engineer. There is no consensus in industry and academia when defining these roles. By analyzing hiring ads, Hussain *et al.* found that DevOps positions usually do not impose build and release management as attributions [23], whereas Kerzazi and Adams found that these three roles share common activities [87]. The challenge for managers to define job titles is also related to how to structure development and operations teams, which we discuss in Section 8.2.

Culture: Humble stated that, typically, the obstacle to continuous delivery adoption is not the skill level of individual employees, but failures at the management and leadership level [21]. High-performing organizations strive, at all levels, towards continuous improvement, rather than treating workers as fungible “resources” that should merely execute tasks as efficiently as possible [21]. Top and low-level management are responsible for creating an environment in which failure is allowed, and people seek continuous improvement.

Increase in delivery throughput: Neely and Stolt reported that the delivery throughput per developer in their company increased with the adoption of continuous delivery and defect rate became more predictable [33]. Siqueira *et al.* also reported how, after investments in DevOps and continuous delivery, the number of releases per semester remained the same even after a considerable decrease in the number of team members [44].

Building trust: when an organization builds software for other large organization, especially the government, it is common for the relationship between the contractor and contracted to be dominated by mistrust, which leads to cumbersome development processes. However, Siqueira *et al.* advocate, based on their experience, that continuous delivery leverages a trust relationship among the contractor and contracted in software development, even in governmental projects [44].

Building for the government: although building software for government involves more bureaucratic processes, requirements and prioritization can often change due to political reasons [44], which leads to the need to shorten the release cycle by adopting agile and DevOps practices in the governmental scenario.

7.3 Implications for researchers

Based on the reviewed literature, we list DevOps-related open topics that the academic community could exploit in future research.

Software Architecture: some authors explore software design in the context of DevOps, continuous delivery, and continuous deployment [5, 42, 48]. However, engineers may still struggle with this in practice, since achieving the desired architecture can be infeasible in a single first DevOps

project. Researchers should investigate transitioning strategies to adopt practices and architectural changes.

Education: DevOps adoption requires more skilled software engineers. A broader investigation is needed to explore how to teach operations skills to developers and vice-versa, as well as to introduce software operation topics in software engineering courses [8, 9]. We discuss this in more detail in Section 8.4.

Embedded systems and IoT: continuously delivering new software versions in embedded systems is still a hard question [31, 35]. How to effectively adopt DevOps in an IoT context is an open research question to be investigated.

Compliance: DevOps provides both benefits and obstacles to regulatory compliance [21, 22, 28]. Research should be done to detect scenarios in which a high-frequency delivery is indeed not welcome or how one can demonstrate that automated deployment can help organizations comply with regulations, as stated by Humble [21]. Moreover, some practitioners advocate that engineers must have unrestricted access to production data [52, 84], which can be controversial in specific environments, such as financial systems.

Security: high-frequency of continuous delivery poses questions about the security in a DevOps context [25, 49]. However, DevOps can also bring benefits for security. This dual relationship generates a dichotomy that should be further investigated.

Testing large-scale distributed systems: some errors in large-scale distributed systems can be hard to reproduce in testing environments [20]. The chaos engineering approach advocates running experiments in production [3]. This is an incipient topic in the research literature.

Quantitative assessment metrics: few works use adequate quantitative metrics for assessing DevOps as is the case of the Puppet State of DevOps Reports [120]. The investigation and use of such quantitative metrics should be intensified in related research.

Deployment approach: two automated deployment approaches are the container-based one [26] and the deployment scripts written in domain-specific languages such as Chef. Investigating how complementary or exclusive these approaches are can help engineers choose the best toolset.

Improving interdepartmental communication: how to effectively enable interdepartmental communication is a challenge, especially for distributed organizations [15]. Better communication does not merely mean more communication since too much communication, and an excess of meetings can negatively impact productivity.

Adoption strategies: there are still many open questions about how organizations should adopt DevOps. It is stated that DevOps adoption requires top-management support [10, 35]. Sometimes it does not happen in the first moment, and a “guerrilla” strategy can take place (i.e., acting outside the company standard procedures). Moreover, arguments to encourage DevOps adoption can differ from engineers to managers [33]. In such context, researchers can support DevOps adoption by providing a further investigation into balancing top-management support and guerrilla strategies; studying arguments to convince managers; providing guidelines about which DevOps-adoption strategy to choose based on organizational characteristics.

Investigation involving multiple organizations: since most empirical research is conducted with data from a single organization [10, 15, 34, 34], any conclusion from these works deserves further investigation. Surveying more companies, as done by Leppanen *et al.* [29], can strengthen or weaken previous findings.

Other research topics: Claps *et al.* list other technical and social challenges for continuous deployment adoption, such as product marketing, team coordination, customer adoption, feature discovery, plugin management, cross-product dependencies, and scaling CI tools [10]. Olsson *et al.* list among the challenges in continuous delivery adoption: coordination of supplier integration, business models, and difficulty in overviewing projects status [35].

8 UNRESOLVED CHALLENGES

Throughout this survey, we have shown that overviewing DevOps concepts and tools raises some issues to debate. The previous section also listed some not-so-straightforward implications for practitioners and researchers. Therefore, based on the surveyed literature and additional sources of knowledge, we further discuss in this section some of the DevOps challenges faced by managers, engineers, and researchers that are not thoroughly handled by the current state-of-the-art.

8.1 How to re-design systems toward continuous delivery

Highly coupled monolithic architectures are obstacles to effective continuous delivery. Complex dependency management of software components and teams is imposed to the deployment pipeline [42]. An essential principle for successfully adopting and implementing continuous delivery is an architecture composed of small and independently deployable units, also called *microservices* [42]. In such architectural style, services interact through the network and are built around business capabilities [94].

Conceiving of an application architecture with loosely coupled and well-encapsulated microservices guarantees two architectural attributes required by continuous delivery: *testability* and *deployability* [21]. If each microservice has its test suite, this reduces the blocking of deployment due to long-running tests [29].

Whereas some authors say microservices facilitate effective implementation of DevOps [2], others say microservices require DevOps [17], since deployment automation minimizes the overhead to manage a significant number of microservices. However, adopting microservices comes with several challenges. First, there is heterogeneity in non-functional patterns such as “startup scripts, configuration files, administration endpoints, and logging locations” [6]. Technological heterogeneity can be a productivity barrier for newcomers in the team. Second, microservices must be deployed to production with the same set of versions used for integration tests [5]. These challenges can be overcome by adopting a minimum set of microservices standards across the organization [6].

Common microservice management patterns associated with DevOps are: one deployment pipeline per microservice; log aggregator; service registry; correlation ID’s [5]; and segregation of source code, configuration, and environment specification [2]. Other complimentary patterns are strangler application [72][5, 21], load balancer, consumer-driven contracts [2], circuit breaker [103], backwards compatibility, and versioned APIs [21]. However, the last one is controversial, and some communities usually do not recommend microservice versioning [2].

In highly regulated environments, the proper use of microservices can constrain unfriendly regulations to specific system modules and people [21]. Additionally, the usage of a platform-as-a-service (PaaS) can automate much of the compliance checking, a model used, for example, by the U.S. federal government [21]. PaaS is also recommended for its operational simplicity, like other cloud services, such as storage services, asynchronous processing with queues, email delivery, and real-time user monitoring [11].

Finally, although reusability is a historical goal of software engineering [97], Shahin *et al.* recommend engineers not to focus too much on it [42]. It brings coupling, which is a huge bottleneck

to continuous delivery [42]. Therefore, each team should discuss the trade-off between reusability and independence from other components, services, and teams.

8.2 How to deploy DevOps in an organization

A hard question not yet fully answered by the literature is how – perhaps whether – an organization should be restructured to adopt DevOps. We believe the literature is incomplete and even contradictory regarding this subject. We discuss a few studies that have evaluated this matter.

The seminal paper of Humble and Molesky about DevOps presents three scenarios. First, preserving the structures of development and operations departments, DevOps is led by human collaboration among developers and operators, with operators attending agile ceremonies and developers contributing to incident solving [22, 24]. Alternatively, product teams, also called cross-functional teams, effectively incorporate operators. Finally, in a third scenario, one product team is composed of only operators offering support services (continuous integration, monitoring services) to the entire organization.

Similarly, Nybom *et al.* also present three distinct scenarios to DevOps adoption [34]: *i*) collaboration among development and operations departments; *ii*) cross-functional teams; and *iii*) creation of “DevOps teams.” These approaches are summarized in the Table 5 and discussed in detail in the following paragraphs.

Table 5. DevOps adoption approaches

Collaborating departments
Development and operations departments collaborate closely.
It implies overlapping of developers and operators responsibilities.
<i>Downside:</i> new responsibilities can be unclear for employees.
Cross-functional team
The product team is responsible for deploying and operating (<i>You built it, you run it</i>).
Recommended by Amazon and Facebook.
<i>Downside:</i> requires more skilled engineers.
DevOps team
Acts as a bridge between developers and operators.
It is better accepted when it is a temporary strategy for cultural transformation.
<i>Downside:</i> risk of creating a third silo.

In the first approach, the responsibilities of developers overlap with operators [10, 15, 34, 42]. High risk for adopting this strategy occurs when new responsibilities do not become evident in the organization [10, 34], which could lead to frictions among developers and operators [7]. This is especially true when delivery automation is not prioritized [34].

Cross-functional teams resemble the “whole team” practice from XP [57], and seems to prevail in literature [2, 11, 18, 20]. This structure appeals to “T-shaped” professionals, who have expertise in few fields and basic skills in multiple correlated areas [14]. In this model, at least one team member must master operations skills. From this perspective, the so-called “DevOps engineer” [23], also called the “full stack engineer” [23], is known as a developer with operations skills.

One could wonder whether it makes sense to talk about developers and operators collaboration in cross-functional teams context. Adopting cross-functional teams may just be a matter of moving operators to product teams. However, this is not trivial, considering that large organizations usually have only a small pool of operators for several development teams [34]. Therefore, companies must hire people with both development and operations skills or train some of their developers in operations skills. Nevertheless, pressuring developers to learn operations can be ineffective, as

they are already overwhelmed with other skills they need to learn. Transforming development teams into product teams has still another major cultural shift, as developers become responsible for 24/7 service support [14, 42]. Even when services should be designed to remain available without human interaction [77]. An extra challenge for cross-functional teams is guaranteeing that specialists interact with their peer groups to share novelties in the field [14]. Spotify, for example, achieves this through inter-team “guilds,” which are communities of common interest within the organization [91].

Assigning a “DevOps team” as a bridge between developers and operators [34] has become a trend [120]. However, it is not clear what precisely “DevOps teams” are, and how they differ from regular operations teams [34][120]. This approach is criticized by Humble, who considers that potentially creating a new silo is not the best policy to handle the DevOps principle of breaking down walls between silos [81]. Skelton and Pais present some variations, called “DevOps topologies,” and also DevOps anti-patterns [113]. They argue that, although the DevOps team silo is an anti-pattern, it is acceptable when it is temporary and focused on sharing development practices to operations staff and operations concerns to developers. Siqueira *et al.* report how a DevOps team with senior developers and rotating roles among members was a key decision to construct and maintain a continuous delivery process [44].

There is yet the strategy adopted by Google, called Site Reliability Engineering (SRE) [58], which involves the evolution of the operations engineer role. In addition to product teams, Google has SRE teams responsible for product reliability engineering, what includes performance, availability, monitoring, and emergency response. SRE teams have an upper limit of using 50 percent of their time on operational tasks because they must allocate at least 50 percent of their time increasing product reliability through engineering and development activities.

One should also question the impact of deployment automation on operators’ roles. Chen reported that, after continuous delivery adoption, operations engineers only needed to click a button to release new versions [7]. One should question the role of an engineer in such a context. Moreover, if software updates are adequately broken down into small increments, and delivery automation turns deployment in a “non-event” [22], practices such as “development and operations teams should celebrate successful releases together” [22] can be perceived as contradictory.

Therefore, if operations provide support services (continuous integration, monitoring) [22] and product reliability is appropriately designed, product teams can easily deploy and operate their services. Thus, one can even replace operations with third-party cloud services, as done by Cukier [11], and achieve an organizational structure known as “NoOps”. Some blog posts promote NoOps concepts, but the literature has not yet covered this subject. One possible definition is: “NoOps (no operations) is the concept that an IT environment can become so automated and abstracted from the underlying infrastructure that there is no need for a dedicated team to manage software in-house” [108]. Cross-functional teams can embrace NoOps, with a particular focus on employing cloud technology that automates much of the operational work, rather than reallocating operators or teaching developers all the operational work.

8.3 How to assess the quality of DevOps practices in organizations

Some researchers have proposed maturity models for DevOps adoption and continuous delivery adoption [29, 51, 56]. Feijter *et al.*, for example, provide maturity DevOps models to help organizations measure their current maturity level and to determine areas that require additional investment [13]. However, none of these maturity models are widely adopted by industry. Indeed, most organizations have almost no visibility or reliable measurement of their software-delivery practices [19].

Forsgren and Kersten claim that both survey data and system data [19] should measure DevOps transformations. System data can provide a continuous flow of information, although setting up the aggregation of metrics from different sources can be challenging. Survey data provides a holistic view of out-of-the-system issues, such as culture, job satisfaction, and burnout, and can even point to problems on system-data collection.

Whether survey or system data, Forsgren and Kersten still make clear that if results are used to punish teams, data collection will be unreliable [19], as also reinforced by Brown from Microsoft [61]. Kua cautions that inappropriate use of metrics can lead to undesired behavior and even divert the organization from its goals [93]. Kua also compiles guidelines for better usage of metrics by: explicitly linking metrics to goals, favoring trends over absolute numbers, tracking shorter periods, and changing metrics when they do not drive change anymore.

Some examples of metrics based on survey data used on the State of DevOps Reports [53, 120] are: IT performance as a function of time from commit to deployment, frequency of deployment, and recovery time. Time spent on unplanned work and rework. Typology of organizational culture (pathological, bureaucratic or generative) based on climate for learning, job satisfaction, developers and operators having win-win relationships, usage of version control, automated testing. Employee engagement, based on the Net Promoter Score, indicating how likely employees are to recommend the company products and services.

Snyder and Curtis use metrics to evaluate DevOps in a specific company [45]. Metrics such as productivity, defect ratio, dollars spent, cycle time release, and build count were aggregated in a “total quality index”. More importantly, the aggregation of data from different silos was used to align organizational efforts. The authors observed a higher increase in the total quality index of agile and DevOps teams, which helped executives justify investments in the agile-DevOps transformation.

There is a severe lack of industry consensus on how to measure software delivery [19], and designing a good survey requires some expertise. Thus, researchers should provide standards for collecting and analyzing evaluation metrics for the software delivery process, as done by Feijter *et al.* [13]. In the health sector, this model is mature, wherein researchers instrument therapists with standard protocols for diagnoses purposes. Nonetheless, Forsgren and Kersten provide some general guidelines for survey application: it should be limited to 20 to 25 minutes and applied every four to six months [19]. Forsgren and Kersten also advise avoiding surveying management and executives, since they tend to overestimate the maturity of their organizations.

Researchers can also run the surveys across multiple organizations as done in the State of DevOps Reports. The results of these questionnaires can help practitioners in orienting their careers and help organizations to self-evaluate their performance by comparing themselves to their peers.

8.4 How to qualify engineers for DevOps practice

More than one-hundred DevOps tools are available [63], and they are continuously evolving [42]. Professionals must be able to choose which tools to study in depth, which tools to have some familiarity with, which tools to test, and which ones to ignore. Such choices must be based on a conceptual analysis of personal and organizational demands and perspectives. Our conceptual maps (Section 5) and our table of major DevOps tools (Table 4) may guide engineers to prioritize the improvement of their professional skills.

Care is needed when managers assume that employees lack competence. When questioned “Where do you get Netflix’s amazing employees from?”, a Netflix architect replied: “I get them from you!” [21]. The lesson here is that a culture of autonomy and responsibility [18, 20] is an important motivating factor for employees continuous learning.

In software engineering courses, operations skills, often neglected in college education, become mandatory. However, elaborate effective DevOps programs are challenging. Exercises that demand

students build distributed systems with adequate availability, scalability, and performance require not only some infrastructure, but also efficient correction of such exercises with automated checks in submissions [8]. Although there are stable platforms for auto-grading coding assignments (autolab.github.io) [100], evaluating infrastructure code and non-functional concerns are harder to automate. Christensen used Docker in the classroom to develop a small set of scripts to automate some tasks in executing and assessing submissions [8]. However, he did not achieve automation of exercise correction due to the nontrivial task of setting up a multi-server environment to detect errors in students' submissions. Bai *et al.* automate the generation of assessment reports based on patterns of tasks, commits, branches, tests, and the source code itself [1].

Knowledge in software engineering is always evolving, and any software engineer must also have self-learning skills to overcome daily basis challenges. Social interactions on Internet forums are a vital source of information in the daily activities of engineers. Believing that it can be enhanced, Magoutis *et al.* built a social network that helps DevOps engineers analyze results of past deployment executions as well as communicate and exchange ideas to better understand trade-offs in the space of deployment solutions [32]. The work of Magoutis *et al.* show evidence that more can be done to support engineers' self-learning in their daily activities.

9 LIMITATIONS OF THIS STUDY

Given the vast amount of work on DevOps, it was not feasible to conduct an exhaustive search in all available sources. We have not considered academic workshops, and we mentioned only a few books in the field. It would not be practical to try to cover every existing work on DevOps and to condense it all in this single survey. However, by focusing on journal and conference papers, and by applying the snowballing process, we aimed to cover the primary literature in the field. We decided not to expand the query string to avoid artificially favoring some of the DevOps concepts over others. However, the appearance of the keyword "DevOps" was not mandatory in the snowballing process so that we could select vital work tackling highly related subjects such as continuous delivery.

The choice of core and relevant papers may suffer from subjective bias. To reduce this bias, at least two authors conducted the selection process and the classification of papers. Senior researchers also oversaw the whole process. Our study also suffers the publication bias, which is the propensity of researchers and practitioners to report (and for referees to accept) more about positive results than negative results. We did not find, for example, a paper reporting a case of failed DevOps adoption. We did not address this issue in this study.

10 CONCLUSIONS

In this survey, we have discussed DevOps concepts and challenges presented in the literature. By associating these concepts with tools, we contributed to supporting practitioners in choosing a proper toolset. This paper also aides IT professionals by presenting in a systematic way the most relevant concepts, tools, and implications, associated with the professional perspectives of researchers, managers, and engineers – including developers and operators. We hope our reader can now better understand the impact of DevOps on daily activities based on each profile.

Two pillars of DevOps are 1) human collaboration across departments and 2) automation. We found that technical issues regarding delivery automation are more consolidated, with only a few minor controversies in the literature. On the other hand, there is no consensus on how to effectively empower collaboration among departments, and there are only a few tools for tackling this issue.

A technical topic highly related to DevOps revealed by our sources was microservice architecture. Some technical conflicting points are: there is no consensus on whether microservices should be versioned; some practitioners see configuration management tools and containerization as

complementary approaches, but others understand them as competitors; maximizing reusability of software components and services may not be the better strategy; rolling back is usually seen as a DevOps practice, but some practitioners tend to favor feature toggles to avoid these rollbacks.

Structuring an organization to introduce DevOps is now a major concern. There are three alternative structures: 1) preserving collaborating departments, 2) building cross-functional teams, and 3) having DevOps teams. Each one of these strategies has its challenges and, by following the current literature, it is not possible to define the best strategy for a given scenario. Therefore, exploring these and other structures in real organizations and how these organizations handle the “DevOps role” are promising topics for future research. However, whatever the structure an organization adopts, it is clear that the DevOps movement has irreversibly blurred the frontier between developers and operators, even for organizations that have not yet fully embraced DevOps.

ACKNOWLEDGEMENTS

We thank the Brazilian Ministry of Citizenship, via the TAIS project, and the Brazilian Federal Data Processing Service (Serpro) for the support. This research is also part of the INCT of the Future Internet funded by CNPq proc. 465446/2014-0, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, FAPESP proc. 14/50937-1, and FAPESP proc. 15/24485-9.

LIST OF CORE PAPERS

- [1] X. Bai, M. Li, D. Pei, S. Li, and D. Ye. 2018. Continuous Delivery of Personalized Assessment and Feedback in Agile Software Engineering Projects. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET '18)*. 58–67. Code: I818.
- [2] Armin Balalaie, Abbas Heydarnoori, and Pooyan Jamshidi. 2016. Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture. *IEEE Software* 33, 3 (2016), 42–52. Code: A2.
- [3] A. Basiri, N. Behnam, R. de Rooij, L. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal. 2016. Chaos Engineering. *IEEE Software* 33, 3 (2016), 35–41. Code: A76.
- [4] Len Bass. 2018. The Software Architect and DevOps. *IEEE Software* 35, 1 (2018), 8–10. Code: I33.
- [5] Kyle Brown and Bobby Woolf. 2016. Implementation Patterns for Microservices Architectures. In *Proceedings of the 23rd Conference on Pattern Languages of Programs (PLoP '16)*. The Hillside Group, Article 7, 7:1–7:35 pages. Code: A104.
- [6] Matt Callanan and Alexandra Spillane. 2016. DevOps: Making It Easy to Do the Right Thing. *IEEE Software* 33, 3 (2016), 53–59. Code: A67.
- [7] Lianping Chen. 2015. Continuous delivery: Huge benefits, but challenges too. *IEEE Software* 32, 2 (2015), 50–54. Code: B15.
- [8] Henrik Bærbak Christensen. 2016. Teaching DevOps and Cloud Computing Using a Cognitive Apprenticeship and Story-Telling Approach. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '16)*. ACM, 174–179. Code: A47.
- [9] Sam Chung and Soon Bang. 2016. Identifying Knowledge, Skills, and Abilities (KSA) for Devops-aware Server Side Web Application with the Grounded Theory. *J. Comput. Sci. Coll.* 32, 1 (2016), 110–116. Code: A16.
- [10] Gerry Gerard Claps, Richard Berntsson Svensson, and Aybüke Aurum. 2015. On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software Technology* 57 (2015), 21–31. Code: B13.
- [11] Daniel Cukier. 2013. DevOps Patterns to Scale Web Applications Using Cloud Services. In *Proceedings of the 2013 Companion Publication for Conference on Systems, Programming, & Applications: Software for Humanity (SPLASH '13)*. ACM, 143–152. Code: A35.
- [12] Maximilien de Bayser, Leonardo G. Azevedo, and Renato Cerqueira. 2015. ResearchOps: The case for DevOps in scientific applications. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. 1398–1404. Code: I40.
- [13] Rico de Feijter, Sietse Overbeek, Rob van Vliet, Erik Jagroep, and Sjaak Brinkkemper. 2018. DevOps Competences and Maturity for Software Producing Organizations. In *Enterprise, Business-Process and Information Systems Modeling*. Springer, 244–259. Code: S805.
- [14] Patrick Debois. 2011. Devops: A software revolution in the making. *Cutter IT Journal* 24, 8 (2011), 3–5. Code: B4.
- [15] Elisa Diel, Sabrina Marczak, and Daniela S. Cruzes. 2016. Communication Challenges and Strategies in Distributed DevOps. In *11th IEEE International Conference on Global Software Engineering (ICGSE)*. 24–28. Code: I19.

- [16] Andrej Dyck, Ralf Penners, and Horst Lichter. 2015. Towards Definitions for Release Engineering and DevOps. In *2015 IEEE/ACM 3rd International Workshop on Release Engineering*. 3–3. Code: B26.
- [17] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano. 2016. DevOps. *IEEE Software* 33, 3 (2016), 94–100. Code: A54.
- [18] Dror G Feitelson, Eitan Frachtenberg, and Kent L Beck. 2013. Development and deployment at Facebook. *IEEE Internet Computing* 17, 4 (2013), 8–17. Code: B7.
- [19] Nicole Forsgren and Mik Kersten. 2018. DevOps Metrics. *Commun. ACM* 61, 4 (2018), 44–48. Code: B21.
- [20] Jim Gray. 2006. A conversation with Werner Vogels. *ACM Queue* 4, 4 (2006), 14–22. Code: B3.
- [21] Jez Humble. 2017. Continuous Delivery Sounds Great, but Will It Work Here? *Queue* 15, 6 (2017), 57–76. Code: B22.
- [22] Jez Humble and Joanne Molesky. 2011. Why enterprises must adopt DevOps to enable continuous delivery. *Cutter IT Journal* 24, 8 (2011), 6. Code: B5.
- [23] Waqar Hussain, Tony Clear, and Stephen MacDonell. 2017. Emerging Trends for Global DevOps: A New Zealand Perspective. In *Proceedings of the 12th International Conference on Global Software Engineering (ICGSE '17)*. IEEE Press, 21–30. Code: A25.
- [24] Martin Gilje Jaatun. 2018. Software Security Activities that Support Incident Management in Secure DevOps. In *Proceedings of the 13th International Conference on Availability, Reliability and Security (ARES 2018)*. ACM, 8:1–8:6. Code: A803.
- [25] Martin Gilje Jaatun, Daniela S. Cruzes, and Jesus Luna. 2017. DevOps for Better Software Security in the Cloud. In *Proceedings of the 12th International Conference on Availability, Reliability and Security (ARES '17)*. ACM, Article 69, 69:1–69:6 pages. Code: A85.
- [26] Hui Kang, Michael Le, and Shu Tao. 2016. Container and Microservice Driven Design for Cloud Infrastructure DevOps. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*. 202–211. Code: I58.
- [27] Mik Kersten. 2018. A Cambrian Explosion of DevOps Tools. *IEEE Software* 35, 2 (2018), 14–17. Code: I808.
- [28] Teemu Laukkarinen, Kati Kuusinen, and Tommi Mikkonen. 2017. DevOps in Regulated Software Development: Case Medical Devices. In *Proceedings of the 39th International Conference on Software Engineering: New Ideas and Emerging Results Track (ICSE-NIER '17)*. IEEE Press, 15–18. Code: A26.
- [29] M. Leppanen, S. Mäkinen, M. Pagels, V. Eloranta, J. Itkonen, M. V. Mantyla, and T. Mannisto. 2015. The highways and country roads to continuous deployment. *IEEE Software* 32, 2 (2015), 64–72. Code: B14.
- [30] Z. Li, Q. Lu, L. Zhu, X. Xu, Y. Liu, and W. Zhang. 2018. An Empirical Study of Cloud API Issues. *IEEE Cloud Computing* 5, 2 (2018), 58–72. Code: I802.
- [31] Lucy Ellen Lwakatare, Teemu Karvonen, Tanja Sauvola, Pasi Kuvaja, Helena Holmström Olsson, Jan Bosch, and Markku Oivo. 2016. Towards DevOps in the embedded systems domain: Why is it so hard?. In *49th Hawaii International Conference on System Sciences (HICSS)*. IEEE, 5437–5446. Code: A42.
- [32] Kostas Magoutis, Christos Papoulias, Antonis Papaioannou, Flora Karniavoura, Dimitrios-Georgios Akestoridis, Nikos Parotsidis, Maria Korozi, Asterios Leonidis, Stavroula Ntoa, and Constantine Stephanidis. 2015. Design and implementation of a social networking platform for cloud deployment specialists. *Journal of Internet Services and Applications* 6, 1 (2015). Code: S3.
- [33] Steve Neely and Steve Stolt. 2013. Continuous Delivery? Easy! Just Change Everything (Well, Maybe It Is Not That Easy). In *2013 Agile Conference*. 121–128. Code: B23.
- [34] Kristian Nybom, Jens Smeds, and Ivan Porres. 2016. On the Impact of Mixing Responsibilities Between Devs and Ops. In *International Conference on Agile Software Development (XP 2016)*. Springer International Publishing, 131–143. Code: S18.
- [35] Helena H. Olsson, Hiva Alahyari, and Jan Bosch. 2012. Climbing the "Stairway to Heaven" – A Multiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software. In *38th Euromicro Conference on Software Engineering and Advanced Applications*. 392–399. Code: B17.
- [36] Candy Pang and Abram Hindle. 2016. Continuous Maintenance. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 458–462. Code: I55.
- [37] Rahul Punjabi and Ruhi Bajaj. 2016. User stories to user reality: A DevOps approach for the cloud. In *2016 IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*. 658–662. Code: I17.
- [38] Akond Rahman. 2018. Characteristics of Defective Infrastructure As Code Scripts in DevOps. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. ACM, 476–479. Code: A806.
- [39] M. Rajkumar, A. K. Pole, V. S. Adige, and P. Mahanta. 2016. DevOps culture and its impact on cloud delivery and software development. In *2016 International Conference on Advances in Computing, Communication, Automation (ICACCA)*. 1–6. Code: I48.
- [40] James Roche. 2013. Adopting DevOps Practices in Quality Assurance. *Commun. ACM* 56, 11 (2013), 38–43. Code: A74.
- [41] S. Van Rossem, W. Tavernier, D. Colle, M. Pickavet, and P. Demeester. 2018. Introducing Development Features for Virtualized Network Services. *IEEE Communications Magazine* 56, 8 (2018), 184–192. Code: I77.

- [42] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. 2016. The Intersection of Continuous Deployment and Architecting Process: Practitioners' Perspectives. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '16)*. ACM, 44:1–44:10. Code: A64.
- [43] Alan Sill. 2014. Cloud Standards and the Spectrum of Development. *IEEE Cloud Computing* 1, 3 (2014), 15–19. Code: I67.
- [44] Rodrigo Siqueira, Diego Camarinha, Melissa Wen, Paulo Meirelles, and Fabio Kon. 2018. Continuous Delivery: Building Trust in a Large-Scale, Complex Government Organization. *IEEE Software* 35, 2 (2018), 38–43. Code: B29.
- [45] Barry Snyder and Bill Curtis. 2018. Using Analytics to Guide Improvement During an Agile/DevOps Transformation. *IEEE Software* 35, 1 (2018), 78–83. Code: I7.
- [46] Johannes Wettinger, Vasilios Andrikopoulos, and Frank Leymann. 2015. Automated Capturing and Systematic Usage of DevOps Knowledge for Cloud Applications. In *2015 IEEE International Conference on Cloud Engineering*. IEEE, 60–65. Code: A61.
- [47] Johannes Wettinger, Vasilios Andrikopoulos, and Frank Leymann. 2015. Enabling DevOps Collaboration and Continuous Delivery Using Diverse Application Environments. In *On the Move to Meaningful Internet Systems (OTM 2015 Conferences)*. Springer International Publishing, 348–358. Code: A17.
- [48] Eoin Woods. 2016. Operational: The Forgotten Architectural View. *IEEE Software* 33, 3 (2016), 20–23. Code: A82.
- [49] Hasan Yasar and Kiriakos Kontostathis. 2016. Where to Integrate Security Practices on DevOps Platform. *International Journal of Secure Software Engineering (IJ SSE)* 7, 4 (2016), 39–50. Code: A58.
- [50] L. Zhu, D. Xu, A. B. Tran, X. Xu, L. Bass, I. Weber, and S. Dwarakanathan. 2015. Achieving Reliable High-Frequency Releases in Cloud Environments. *IEEE Software* 32, 2 (2015), 73–80. Code: B12.

REFERENCES

- [51] 2017. xMatters Atlassian DevOps Maturity Survey Report 2017. (2017). <https://www.xmatters.com/press-release/xmatters-atlassian-2017-devops-maturity-survey-report/>, accessed on Jun 2018.
- [52] 2018. How Netflix Thinks of DevOps. (2018). <https://www.youtube.com/watch?v=UTKIT6STSVM>, accessed on Jun 2018.
- [53] Nicole Forsgren Velasquez Alanna Brown and, Gene Kim, Nigel Kersten, and Jez Humble. 2016. 2016 State of DevOps Report. (2016). <https://puppet.com/resources/whitepaper/2016-state-of-devops-report>, accessed on Jul 2018.
- [54] Hrishikesh Barua. 2015. The Role of Configuration Management in a Containerized World. (2015). <https://www.infoq.com/news/2015/12/containers-vs-config-mgmt>, accessed on July 2018.
- [55] Len Bass, Ingo Weber, and Liming Zhu. 2015. *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional.
- [56] Helen Beal. 2015. Where are you on the DevOps Maturity Scale Webcast. (2015). <https://www.youtube.com/watch?v=a50ArHzVRqk>, accessed on Jul 2018.
- [57] Kent Beck and Cynthia Andres. 2004. *Extreme programming explained: embrace change*. Addison-Wesley Professional.
- [58] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. 2016. *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media.
- [59] Jonas Bonér, Dave Farley, Roland Kuhn, and Martin Thompson. 2014. The Reactive Manifesto. (2014). <https://www.reactivemanifesto.org/>, accessed on August 2018.
- [60] Rob Brigham. 2015. DevOps at Amazon: A Look at Our Tools and Processes. (2015). At AWS re:Invent 2015, <https://www.youtube.com/watch?v=esEFaY0FDKc>, accessed on Jun 2018.
- [61] Donovan Brown. 2018. Our DevOps journey - Microsoft's internal transformation story. (2018). DevOneConf 2018, <https://www.youtube.com/watch?v=cbFzojQOjyA>, accessed on Jul 2018.
- [62] David Budgen and Pearl Brereton. 2006. Performing Systematic Literature Reviews in Software Engineering. In *Proceedings of the 28th International Conference on Software Engineering (ICSE '06)*. ACM, 1051–1052.
- [63] Necco Ceresani. 2016. The Periodic Table of DevOps Tools v.2 Is Here. (2016). <https://blog.xebialabs.com/2016/06/14/periodic-table-devops-tools-v-2/>, accessed on April 2018.
- [64] Kathy Charmaz. 2008. Chapter 7: Grounded Theory as an Emergent Method. In *Handbook of Emergent Methods*. The Guilford Press.
- [65] Juliet Corbin and Anselm Strauss. 2014. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory* (4th ed.). SAGE Publications, Inc.
- [66] Breno B. Nicolau de França, Helvio Jeronimo, Junior, and Guilherme Horta Travassos. 2016. Characterizing DevOps by Hearing Multiple Voices. In *Proceedings of the 30th Brazilian Symposium on Software Engineering (SBES '16)*. ACM, 53–62.
- [67] Patrick Debois. 2008. Just Enough Documented Information. (2008). At Agile 2008 Toronto.
- [68] Phil Dougherty. 2015. Containers Vs. Config Management. (2015). <https://blog.containerization.io/containers-vs-config-management-e64ccb744a94>, accessed on July 2018.

- [69] Floris Erich, Chintan Amrit, and Maya Daneva. 2014. A Mapping Study on Cooperation between Information System Development and Operations. In *Product-Focused Software Process Improvement*, Andreas Jedlitschka, Pasi Kuvaja, Marco Kuhrmann, Tomi Männistö, Jürgen Münch, and Mikko Raatikainen (Eds.). Springer International Publishing, Cham, 277–280.
- [70] F. M. A. Erich, C. Amrit, and M. Daneva. 2017. A Qualitative Study of DevOps Usage in Practice. *Journal of Software: Evolution and Process* 29, 6 (2017), e1885.
- [71] Nicole Forsgren, Jez Humble, and Gene Kim. 2018. *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. IT Revolution Press.
- [72] Martin Fowler. 2004. StranglerApplication. (2004). <https://www.martinfowler.com/bliki/StranglerApplication.html>, accessed on Jul 2018.
- [73] Martin Fowler. 2010. Blue Green Deployment. (2010). <https://martinfowler.com/bliki/BlueGreenDeployment.html>, accessed on Jul 2018.
- [74] Georges Bou Ghantous and Asif Gill. 2017. DevOps: Concepts, Practices, Tools, Benefits and Challenges. In *21st Pacific Asia Conference on Information Systems (PACIS 2017)*. 96:1–96:12.
- [75] David Guest. 1991. The hunt is on for the Renaissance Man of computing. *The Independent* (London). (September 1991).
- [76] Peter J Hager, Howard Jeffrey Scheiber, and Nancy C Corbin. 1997. *Designing & delivering: Scientific, technical, and managerial presentations*. John Wiley & Sons.
- [77] James Hamilton. 2007. On Designing and Deploying Internet-Scale Services. In *Proceedings of the 21st Large Installation System Administration Conference (LISA '07)*. USENIX, 231–242.
- [78] Pete Hodgson. 2017. Feature Toggles (aka Feature Flags). (2017). <https://martinfowler.com/articles/feature-toggles.html>, accessed on Jul 2018.
- [79] Jonah Horowitz. 2017. Configuration Management is an Antipattern. (2017). <https://hackernoon.com/configuration-management-is-an-antipattern-e677e34be64c>, accessed on July 2018.
- [80] Jez Humble. 2010. Continuous Delivery vs Continuous Deployment. (2010). <https://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/>, accessed on April 2018.
- [81] Jez Humble. 2012. There's No Such Thing as a "Devops Team". (2012). <https://continuousdelivery.com/2012/10/theres-no-such-thing-as-a-devops-team/>, accessed on May 2018.
- [82] Jez Humble and David Farley. 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional.
- [83] Ramtin Jabbari, Nauman bin Ali, Kai Petersen, and Binish Tanveer. 2016. What is DevOps?: A Systematic Mapping Study on Definitions and Practices. In *Proceedings of the Scientific Workshop Proceedings of XP2016 (XP '16 Workshops)*. ACM, 12:1–12:11.
- [84] Adam Jacob. 2015. Chef Style DevOps Kungfu. (2015). At ChefConf 2015, https://www.youtube.com/watch?v=_DET0XsgPc, accessed on Jun 2018.
- [85] Dan Kelly. 2016. Configuration Management And Containers: Which Is Better? (2016). <https://blog.containerization.io/configuration-management-and-containers-which-is-better>, accessed on July 2018.
- [86] N. Kerzazi and B. Adams. 2016. Botched Releases: Do We Need to Roll Back? Empirical Study on a Commercial Web App. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 574–583.
- [87] N. Kerzazi and B. Adams. 2016. Who Needs Release and DevOps Engineers, and Why?. In *2016 IEEE/ACM International Workshop on Continuous Software Evolution and Delivery (CSED)*. 77–83.
- [88] Gene Kim. 2012. The Three Ways: The Principles Underpinning DevOps. (2012). <http://itrevolution.com/the-three-ways-principles-underpinning-devops/>, accessed on Jul 2018.
- [89] Gene Kim, Kevin Behr, and Kim Spafford. 2014. *The phoenix project: A novel about IT, DevOps, and helping your business win*. IT Revolution.
- [90] Gene Kim, Jez Humble, Patrick Debois, and John Willis. 2016. *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. IT Revolution Press.
- [91] Henrik Kniberg. 2014. Spotify engineering culture (part 1). (2014). <https://labs.spotify.com/2014/03/27/spotify-engineering-culture-part-1>, accessed on Sep 2018.
- [92] Per Kroll and Philippe Kruchten. 2003. *The rational unified process made easy: a practitioner's guide to the RUP*. Addison-Wesley Professional.
- [93] Patrick Kua. 2013. An Appropriate Use of Metrics. (2013). <https://martinfowler.com/articles/useOfMetrics.html>, accessed on Jul 2018.
- [94] James Lewis and Martin Fowler. 2014. Microservices. (2014). <https://www.martinfowler.com/articles/microservices.html>, accessed on Jul 2018.

- [95] Lucy Ellen Lwakatare, Pasi Kuvaja, and Markku Oivo. 2015. Dimensions of DevOps. In *Agile Processes in Software Engineering and Extreme Programming*. Springer International Publishing, 212–217.
- [96] Robert C. Martin. 2008. Chapter 12: Emergence. In *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.
- [97] M. Douglas McIlroy, J. M. Buxton, Peter Naur, and Brian Randell. 1968. Mass-produced software components. In *Software Engineering Concepts and Techniques, 1968 NATO Conference on Software Engineering*. 88–98.
- [98] Peter Mell and Timothy Grance. 2011. The NIST definition of cloud computing. (2011). <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>, accessed on May 2018.
- [99] Matthew B. Miles and A. Michael Huberman. 1994. Chapter 2: Focusing and Bounding the Collection of Data - the Substantive Start. In *Qualitative Data Analysis: An Expanded Sourcebook* (6th ed.). SAGE Publications.
- [100] Dejan Milojevic. 2011. Autograding in the Cloud: Interview with David O'Hallaron. *IEEE Internet Computing* 15, 1 (2011), 9–12.
- [101] Kief Morris. 2016. *Infrastructure as Code: Managing Servers in the Cloud*. O'Reilly Media.
- [102] Eueung Mulyana, Rifqy Hakimi, and Hendrawan. 2018. Bringing Automation to the Classroom: A ChatOps-Based Approach. In *2018 4th International Conference on Wireless and Telematics (ICWT)*. 1–6.
- [103] Michael T. Nygard. 2009. *Release It! Design and Deploy Production-Ready Software*. Pragmatic Bookshelf.
- [104] Mary Poppendieck and Tom Poppendieck. 2006. *Implementing Lean Software Development: From Concept to Cash*. Addison-Wesley Professional.
- [105] Roger S Pressman. 2005. *Software engineering: a practitioner's approach* (6th ed.). Palgrave Macmillan.
- [106] Mike Roberts. 2018. Serverless Architectures. (2018). <https://martinfowler.com/articles/serverless.html>, accessed on Oct 2018.
- [107] Kevin Roebuck. 2011. *DevOps: High-impact Strategies - What You Need to Know: Definitions, Adoptions, Impact, Benefits, Maturity, Vendors*. Tebbo.
- [108] Margaret Rouse. 2015. What is NoOps? - Definition from WhatIs.com. (2015). <https://searchcloudapplications.techtarget.com/definition/noops>, accessed on May 2018.
- [109] Danilo Sato. 2014. CanaryRelease. (2014). <https://martinfowler.com/bliki/CanaryRelease.html>, accessed on Jul 2018.
- [110] Danilo Sato. 2014. Chapter 12: Infrastructure as Code. In *Devops in Practice: Reliable and automated software delivery*. Casa do Código.
- [111] Alexandra Sbaraini, Stacy M Carter, R Wendell Evans, and Anthony Blinkhorn. 2011. How to do a grounded theory study: a worked example of a study of dental practices. *BMC medical research methodology* 11, 128 (2011), 1–20.
- [112] Julia Silge. 2017. How Much Do Developers Earn? Find Out with the Stack Overflow Salary Calculator. (2017). <https://stackoverflow.blog/2017/09/19/much-developers-earn-find-stack-overflow-salary-calculator/>, accessed on April 2018.
- [113] Matthew Skelton and Manuel Pais. 2013. DevOps Topologies. (2013). <https://web.devopstopologies.com/>, accessed on Jul 2018.
- [114] Jens Smeds, Kristian Nybom, and Ivan Porres. 2015. DevOps: A Definition and Perceived Adoption Impediments. In *Agile Processes in Software Engineering and Extreme Programming*. Springer International Publishing, 166–177.
- [115] Ian Sommerville. 2011. *Software engineering* (9th ed.). Addison-Wesley.
- [116] Daniel Stahl, Torvald Martensson, and Jan Bosch. 2017. Continuous practices and devops: beyond the buzz, what does it all mean?. In *43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2017)*. 440–448.
- [117] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. 2016. Grounded Theory in Software Engineering Research: A Critical Review and Guidelines. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE '16)*. 120–131.
- [118] Antonio Terceiro, Joenio Costa, João Miranda, Paulo Meirelles, Luiz Romário Rios, Lucianna Almeida, Christina Chavez, and Fabio Kon. 2010. Analizo: an extensible multi-language source code analysis and visualization toolkit. In *Brazilian Conference on Software: Theory and Practice (Tools Session) (CBSOFT)*, Vol. 29.
- [119] JC van Winkel. 2017. Life of an SRE at Google. (2017). At Codemotion Rome 2017, <https://www.youtube.com/watch?v=7Oe8mYPBZmw>, accessed on Jun 2018.
- [120] Nicole Forsgren Velasquez, Gene Kim, Nigel Kersten, and Jez Humble. 2014. 2014 State of DevOps Report. (2014). <https://puppet.com/resources/whitepaper/2014-state-devops-report>, accessed on May 2018.
- [121] Adam Wiggins. 2011. The Twelve-Factor App. (2011). <https://12factor.net/>, accessed on August 2018.
- [122] Claes Wohlin. 2014. Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE '14)*. ACM, 38:1–38:10.

MeasureSoftGram: A Future Vision of Software Product Quality

Hilmer Rodrigues Neri
UnB/FGA and COPPE/UFRJ
Brazil
hilmer@unb.br

Guilherme Horta Travassos
COPPE/UFRJ
Brazil
ght@cos.ufrj.br

ABSTRACT

Background: Software product quality assurance affects the acceptance of releases. The one dimensional observational perspective of current software product quality (SPQ) models constrains their use in continuous software engineering environments. **Aims:** To investigate multidimensional relationships between software product characteristics and build an evidence-based infrastructure to observe SPQ continuously. **Method:** To mine and manipulate datasets regarding software development and use. Next, to perform multidimensional analytical SPQ interpretations to observe quality. **Results:** There is empirical evidence on the multidimensionality linkage of quality characteristics throughout the software life cycle. **Conclusions:** The one-dimensional quality perspective is not enough to observe the SPQ in continuous environments. Alternative mathematical abstractions should be investigated.

CCS CONCEPTS

Software and its engineering → Software creation and management; Quality Assurance

KEYWORDS

Software Product Quality, Software Analytics, Release acceptance, Continuous Software Engineering, Continuous Experimentation, Empirical Software Engineering.

ACM Reference format:

Neri H.R.; and Travassos G.H.; 2018. In *Proceedings of ACM International Symposium on Empirical Software Engineering and Measurement, Oulu, Finland, October 11-12, 2018 (ESEM'18)*, four pages.

DOI: <https://doi.org/10.1145/3239235.3267438>

1 INTRODUCTION

Software engineers have studied software product quality (SPQ) for nearly four decades. The reference models usually categorize SPQ factors as a hierarchy of quality characteristics and sub-characteristics. Also, some studies are concerned with their

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
ESEM'18, October 11-12, 2018, Oulu, Finland FIN

© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5823-1/18/10. \$15.00
DOI: <https://doi.org/10.1145/3239235.3267438>

associated measurements and relationships between them [1-2-3-4]. Empirical evidence emphasizes the need to observe the relationships among the quality characteristics when analyzing SPQ [3-7-8]. In open source development, other quality characteristics and sub-characteristics were defined [2]. ISO 9126 and its update ISO 25000 have been the most studied SPQ models in the literature [27]. A recent study demonstrated the relationships between ISO 25010 quality characteristics [4].

It is possible to observe in the previous standard SPQ models [1-6] a lack of experimental evidence regarding the relationships among factors, concentrating most of their findings in opinions obtained in surveys of the industrial sector. Besides, they make difficult to aggregate studies due to semantic differences and the use of a one-dimensional perspective (observing a quality characteristic per time without considering side effects or influences of other characteristics) to observe software quality, constraining its observation. Also, empirical studies on measurement and software quality measures commonly report a lack of rigor in the measurement process and definition of measures [23]; problems in scales between measures [19]; the difficulty of establishing reference values for software measures [24]; and finally, the characteristics of the context that make difficult, or even impossible, the comparison of measures between different products or projects [25]. All of these may jeopardize to observe the quality of software products in operation; the software behavior in use, and consequently, the organization's business strategy. In some sense, these issues have been addressed in recently proposed ISO-based SPQ models such as SQuale, Quamoco and particularly in QATCH. [28-29-30]. These models provide a good advance in the operationalization of the missing connections between abstract descriptions of software quality characteristics and specific software analysis approaches. It turns out such models also observe SPQ using a one-dimensional perspective, focusing on internal quality. We argue that relationships between SPQ characteristics must be explicitly exposed and analyzed in order to effectively capture the whole spectrum of quality.

Also, software engineers' perception on how to develop software changed. It has been increasingly perceived more holistically. According to this vision, activities and tasks from the development process are organized and executed to establish a continuous cadence of workflows. It contributes to characterize the "continuity" in software engineering environments. Usually, such environments offer an entwined set of development and deployment activities and tasks organized and executed continuously and no longer offered as a set of phases, activities, and tasks, organized and executed discretely [5]. Therefore, they

allow the continuous collection of development, experimentation, and software usage data.

Due to strong engineering principles in such development environments, continuous and systematic SPQ assurance is mandatory in this new development reality. However, the limitations of current SPQ models restrict their direct use in such environments. For instance, the one-dimensional perspective and a *posteriori* measurement do not support a holistic quality observation *on-the-fly*. This problem directly affects the decision making on the releases acceptance regarding what should be deployed and delivered for use at a particular level of quality.

The acceptance of releases represents a challenge for software organizations. Notably, in our case, it is even more critical when dealing with public software development contracts in Brazil. Usually, the releases are accepted *ad-hoc*, without the capacity of observing characteristics that can affect internal, external, and usage product quality altogether. Therefore, we believe a multidimensional and continuous perspective to address and observe quality can represent a vision of the future and contribute to reducing the acceptance risks in our organizations.

We realized it because the multiple relations between quality characteristics and sub-characteristics reveal multiple interactions and side effects among measures. The side effects of these relations will only be observable when the characteristics and sub-characteristics are analyzed together; hence evidence on their adherence to the phenomena and relationships must be explicitly known and used in the analysis. To guide this ongoing research, we formulate the following question:

How to perform multidimensional SPQ observations in continuous engineering environments to improve the decision making on release acceptance?

As far as we could observe, multiple relationships among quality characteristics and sub-characteristics must be captured and represented in the n-dimensional space, considering the differences in the measurement scales and their need for aggregation and comparison. Therefore, SPQ characteristics and their sub-characteristics should be represented by a multidimensional and interrelated structure in the space of quality. A mathematical concept allowing a multidimensional representation in the space is the tensor [18]. In this sense, the notion of tensors can be understood as a generalization of vectors and scalars concepts. Comparable tensors in space can represent different variables and scales, including their mutual influence. In our case, each of the quality characteristics and its sub-characteristics can be represented through tensors, called from now on tensors of quality.

The following sections depict our vision on how to use a multidimensional perspective to observe SPQ in contemporary software engineering environments continuously. Besides this introduction, Section 2 describes basic concepts regarding our research. Next, our proposal is presented, including the different pieces of technology that could make it available. Section 4 presents some conclusions for this vision paper.

2 BACKGROUND

2.1 Continuous Software Engineering

Continuous engineering principles and practices have been progressively adopted in the industry [9-5-10]. It recognizes that the software development process activities must be managed holistically. Continuous engineering principles mean that activities carried out by expert teams (with clear divisions of roles and following a rigid and defined workflow) make it challenging to adapt software changes besides to prevent the autonomy of the development teams. It compromises the cadence of software delivery and damages organizations' business as it increases waste of effort.

Besides the continuous development process cadence, one of the most notable changes is concerned with the high level of automation in delivering and deployment software releases. Such automated practices and technologies have been allowing different software organizations to take advantage of better balancing the time-to-delivery pressures, by instrumenting the DevOps pipeline [11] and thus breaking the limitation of automation technologies as observed in the 70's [6].

2.2 Continuous Experimentation

Over the last ten years, experimental strategies have been proposed in continuous environments. Continuous experimentation (CE) allows to base decisions on releases acceptance of hypothesis testing. According to statistical testing results, a particular version of a product is chosen and deployed [12-26]. However, it usually focuses on usability and usage characteristics of web applications using simple experimental arrangements, such as A/B testing. In this type of experiment, half of the sample (users) is exposed to the treatment (new version), and later it is compared to the version in operation (control). At the moment of comparison, when the hypothesis test result is taken into consideration, the chosen version is deployed to the whole population of users. Typically, the selection and randomization of the sample are done from active user sessions, therefore during software operation.

Recent works proposed generic models for adopting CE practices. Fabijan *et al.* report a consolidated model acquired from over ten years working at Microsoft [14]. Fagerholm *et al.* propose a model based on observing CE use in startups [13]. Schermann *et al.* present an overview of CE techniques, which helps to understand this practice's state and suggest that organizations need foundational support to move from an experience-driven culture to experiment-driven culture in the decision-making on release acceptance [15].

2.3 Software Analytics

Mining techniques in software repositories are currently remarkable as well as the high availability of projects' data. Automation of mining, mathematical and statistical analysis have boosted research in the area of software analytics. Although there is no standard definition of software analytics, there is a common understanding that it refers to the process by which a computational solution examines data extracted from development

and operational environments, making use of mathematical methods with the purpose of finding patterns and relations to obtain insights that drive the technical-managerial decision-making [16-17].

2.4 Basics of Tensors and their Analysis

An array of components represent a tensor. Components are coordinates functions of space. Therefore, a vector is less general than a tensor. The order of a tensor defines its dimension. Therefore, a vector is a particular case of a tensor of order 0 [18]. One of the interests of multilinear algebra is the n -dimensional vector space with p -linear applications. It is similar on what we can observe with the quality characteristics and their relationships. There are multilinear algebra foundations, such as canonical bilinear isomorphism, commutative and associative properties that can support numerical transformations in the scalars (in this research, the quality measures). The tensor product enables us to do mathematical operations of sum and product of tensors and it is thus represented: $U \otimes V$, let U and V are vector spaces of m e n dimensions. The linear case can be extended to p -linear, generalizing its application. Then, there is a canonical bilinear application expressed by: $\phi: V_s^r \otimes V_{s'}^{r'} \rightarrow V_{s+s'}^{r+r'}$. This application is called tensor multiplication. It can be used, for instance, to combine different quality characteristics. Another important operation with tensors is the contraction used to reduce its dimensionality. Through contraction we can reduce a tensor to a linear application. It generalizes the canonical bilinear form $U \otimes V^* \rightarrow \mathbb{R}$ being thus defined: $c_j^i: V_s^r \rightarrow V_{s-1}^{r-1}$.

By doing so, the mathematical properties of the tensor product are supposed to enable the treatment of software quality in multidimensional and multivariate vector spaces homogeneously.

3 OUR PROPOSAL: The MeasureSoftGram

One of the challenges regarding a multidimensional observation of quality regards the different perspectives and measurement scales used to represent the quality characteristics, sub-characteristics, and their relationships. Scale transformations and quality characteristics aggregation become a complicated problem whether following the one-dimensional perspective precluded by SPQ models. Therefore, we believe that tensors algebraic application should mitigate the problems related to the measurement scales that usually invalidate experimental studies and making the aggregations of behaviors hard [19-20]. So, of tensors of quality will model the quality measures after statistical analyzes and treatments.

In our case, the challenge is to observe the quality of releases and decide about their deployment and delivering in continuous environments. Due to the dynamicity of the development process, the expectation of quality evolves in each interaction and use of the released product, turning the continuous observation of quality an explicit requirement for software engineers.

In this development and releasing scenario, each release (R_i) has a defined duration (Δ_{Ti}). Throughout R_i planning activity, the product quality measurement objectives should be planned based

on different stakeholders' perspectives [21-22]. It is assumed that R_i is developed according to a continuous delivering flow and after Δ_{Ti} a decision-making on the release acceptance happens. So, one or more quality characteristics (and sub-characteristics) will represent the quality measurement objectives. The quality measures, mined and extracted in static or dynamic way, in turn, will support the observation of quality characteristics including their side effects (captured through the relationships among them). Thus, a product quality baseline must be defined at each $\Delta_{Ti(start)}$ and at $\Delta_{Ti(finish)}$. At R_i acceptance activity, the previously defined baseline will be compared with the realized measures and, next, evolved and becomes the baseline for the next release. Each baseline is called a product quality configuration - $SPQC_i$. Therefore, for all R_i there will be a $SPQC_i(\text{planned})$ at $\Delta_{Ti(start)}$ and $SPQC_i(\text{realized})$ at $\Delta_{Ti(finish)}$. Although it sounds common place, these quality settings will be spatial modeled as tensors, representing n -dimensional evidence-based quality characteristics arrays, as defined in some certain SPQ models, as well as its $T_{QCi_s}^r$ relationships. Thus, we understand that the multiplication of these tensors of quality provides us with a vector measure, on a ratio scale, allowing the comparison between different product quality configurations, as following:

$$SPQC_i = T_{QCi_s}^r \otimes T_{QCi_{s'}}^{r'} \otimes T_{QCi_{s''}}^{r''}, \dots, \otimes T_{QCi_n}^{r_n} \quad (1)$$

An overview of our proposal is presented in Fig. 1. The next paragraphs depict it. Due to the sake of space, we will only describe the main components of our approach, marked in the figure as the steps (I) and (VI).

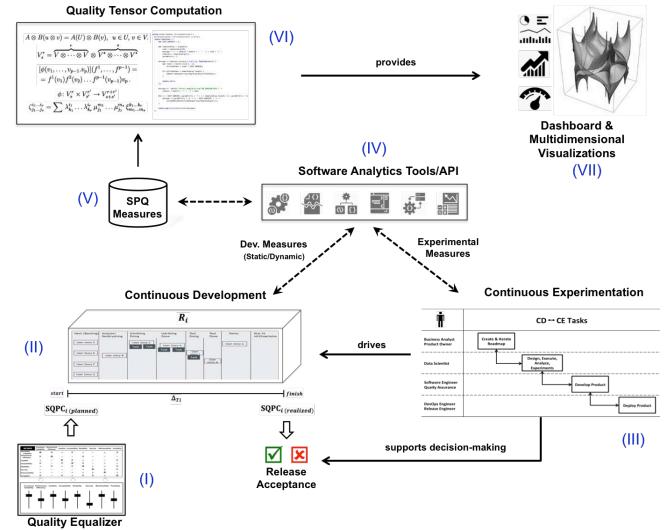


Figure 1: Continuous and Multidimensional Quality Observation in MeasureSoftGram.

The Quality Equalizer (QE) – step I - supports the tuning of a software product quality configuration in the MeasureSoftGram, as shown in Fig. 2. The stakeholders involved in the release acceptance decision making can use the QE to define their quality expectations for R_i , which will be represented by

the $SPQC_i$ (*planned*). When R_i is evaluated at $\Delta_{Ti(finish)}$, $SPQC_i$ (*realized*) will be measured in the developed product version, and compared to $SPQC_i$ (*planned*). Consequently, the assessment of $SPQC_i$ will take place through evidence obtained from data, which will guide the decision-making regarding R_i acceptance. At the starting point (R_1) or when no data is yet available, the effects of ISO 25010 relationships, as identified by Lavazza and Morasca [25], are going to be used to build $T_{Qc1_s}^r$. Therefore, each $T_{Qc1_s}^r$ represents a multidimensional set of measures of a given quality characteristic (and sub-characteristics) with their relationships. The relationships between the characteristics are captured by the tensor indices, which by definition are the covariant and contravariant tensors.

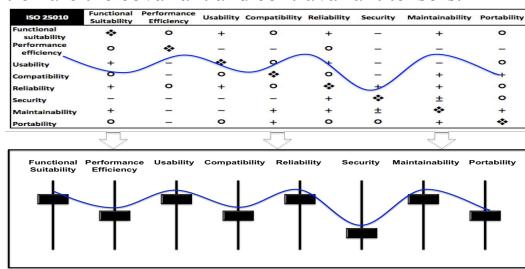


Figure 2: MeasureSoftGram Quality Equalizer.

The quality measures mined and extracted from the continuous environments (II and III) will be statistically treated and processed by software analytics tools (IV), then providing measures to build the $T_{Qc1_s}^r$ and support further comparison between them, systematizing the decision making support.

These data will be stored in a repository of quality measures (V), so preserving static and dynamic measures; plans and experiments results, beyond the results of decisions. The DevOps practices can ensure the complete and automated reproduction of operational environments, analysis, and decisions that were mutually agreed. We believe all of these will allow us to look at and compare continuously $SPQC_i$ using a multidimensional quality perspective when performing SPQ.

4 CONCLUSIONS

This paper described our vision of the future of supporting the continuous and multidimensional observation of software product quality in continuous software environments. It has been called MeasureSoftGram. Our motivation to deal with this problem resides in the context of Brazilian Public Software Development Contracting, which operates a considerable amount of public resources in different software projects annually. To improve the decision making on the acceptance of software product releases is vital to protect the public resources and guarantee that high-quality software products (under the perspective of stakeholders) are available to the Brazilian society.

We are currently instrumenting a computational infrastructure in the Experimental Software Engineering Lab at COPPE/UFRJ to support all of the MeasureSoftGram features. It will allow

evaluating the feasibility of our visionary proposal. Later it is intended to observe its use in different development contexts, such as startups and innovative software companies, besides Brazilian public organizations.

ACKNOWLEDGMENTS

UnB, CAPES, and MinC support this work. Prof Travassos is a CNPq researcher (grant 305929/2014-3). Thank designers Putu K., Abhishek R., Rafael G. M., Icon I., Hea P. L., to share images (icons) under Creative Commons license from Noun Project.

REFERENCES

- [1] McCall, J.A. & Richards P.K. & Walters G.F., Factors in Software Quality, Volumes I, II, and III. US Rome Air Development Center Reports, US Department of Commerce, USA, 1977.
- [2] Miguel J.P. & Mauricio D. & Rodrigues G., A review of software quality models for the evaluation of software products. IJSEA, 2014
- [3] Henningsson K. & Wohlin C., Understanding the relations between software quality attributes - a survey approach. In Proceedings of 12th ICSQ, 2002
- [4] Haoues M. & Sellami A. & Ben-Abdallah H. & Cheikhi L., A guideline for software architecture selection based on ISO 25010 quality-related characteristics, IISAEM, 2017.
- [5] Fitzgerald B. & K.J. Stol., Continuous software engineering: A roadmap and agenda, JSS, 2017.
- [6] Boehm B. W., Characteristics of Software Quality, Volume 1, TRW Series, North-Holand, 1978.
- [7] Aldaajeh S. & Asghar T. & Abed A.K. & Zakaullah M., Communing Different Views on Quality Attributes Relationships' Nature. EJSR, 2012.
- [8] Svahnberg M. & Henningsson K., Consolidating different views of quality attribute relationships, WoSQ@ICSE, 2009.
- [9] Dybå T. & Dingsøyr T., Agile Project Management: From Self-Managing Teams to Large-Scale Development, ICSE, 2015.
- [10] Poppendieck M. & Poppendieck T., Implementing Lean Software Development: From Concept to Cash, (The Addison-Wesley Signature Series), 2006.
- [11] Debois P., DevOps: A software revolution in the making, JITM, 2011.
- [12] Kohavi R. & Longbotham R. & Walker T., Online Experiments: Practical Lessons, IEEE Computer, 2010.
- [13] Fagerholm F. & Sanchez G. A. & Mäenpää H. & Münch, J, The RIGHT model for Continuous Experimentation. JSS, 2017.
- [14] Fabijan A. & Dmitrie P. & Olsson H. & Bosch J., The Evolution of Continuous Experimentation in Software Product Development, ICSE, 2017.
- [15] Schermann G. & Cito J. & Leitner P. & Zdun U. & Gall H. C. We're Doing It Live: A Multi-Method Empirical Study on Continuous Experimentation, IST, 2018.
- [16] Buse R.P.L. & Zimmermann T., Analytics for software development, FoSER, 2010.
- [17] Menzies T. & Zimmermann T. Software Analytics: So What?, IEEE Software, 2013.
- [18] Lima E.L., Tensorial Calculus, Vol. 32 de Math Notes, IMPA, 1965
- [19] Juristo N. and Moreno A.M., Basics of Software Engineering Experimentation, Springer Publishing, 2010.
- [20] Wohlin C. & Runeson P. & Host M. & Ohlsson M.C., Regnell B. & Wesslén A., Experimentation in Software Engineering, Springer Publishing, 2012.
- [21] Buse R.P.L. & Zimmermann T., Information needs for software development analytics, ICSE, 2012.
- [22] Fenton N. & Bieman J., Software Metrics-A Rigorous and Practical Approach, 3th, CRC Press, 2014.
- [23] Kitchenham B., What's up with software metrics? – A preliminary mapping study, JSS, 2010.
- [24] Lavazza L. & Morasca S., An Empirical Evaluation of Distribution-based Thresholds for Internal Software Measures, PROMISE, 2016.
- [25] Dybå T. & Sjøberg D. & Cruzes D., What works for whom, where, when, and why? On the role of context in empirical software engineering, ESEM, 2012.
- [26] Bosch J., Building products as innovation experiment systems, LNBP, 2013
- [27] Ouhbi, S. et al., Evaluating Software Product Quality: A Systematic Mapping Study, IWSM-MENSURA , 2014
- [28] Mordal-Manet K. et al., The squale model – A Practice-Based Industrial Quality Model, ICSM, 2009
- [29] Wagner S. et al., The quamoco product quality modeling and assessment approach, ICSE, 2012
- [30] Siavvas M. G. et al., QATCH - An adaptive framework for software product quality assessment, ESWA Journal, 2017

Dinheiro Público

Código Público



Modernizando a Infraestrutura Pública
com Software Livre



Publicado pela Free Software Foundation Europe (FSFE)

Berlim, janeiro de 2019

Identificador de registro de transparência da União Europeia: 33882407107-76

www.fsfe.org

Responsável perante a Lei Europeia de Imprensa:

Matthias Kirschner / FSFE e.V.

Schönhauser Allee 6/7

10119 Berlim

Alemanha

Contribuições de: Erik Alberts, Alexandra Busch, Matthias Kirschner, Max Mehl, Katharina Nocum, George Brooke-Smith, George Brooke Smith

Os artigos com contribuições de Francesca Bria (pág. 6) e Constanze Kurz (pág. 21) foram extraídos de entrevistas mais longas. Você pode lê-las integralmente em fsfe.org.

Editado por: Carol McGuigam, Jennifer Neal

Projetado por: Markus Meier

Versão em português brasileiro traduzida por: Nitai Bezerra, Anna e Só, Nina Gaul, Guilherme Kawakami, Ricardo Poppi, Francisco José Alves, Fábio Tramasoli, Eduardo Paiva e Augusto Herrmann.

O conteúdo deste relatório pode ser citado ou reproduzido desde que a fonte da informação seja mencionada. Todos os conteúdos são, exceto se explicitamente dito o contrário, licenciados sob a licença Creative Commons 4.0, cláusulas BY-SA (Atribuição-CompartilhaIgual).

Créditos fotográficos:

págs. 1, 11, 23: Vídeo da campanha Dinheiro Público, Código Público. CC BY 4.0 de www.fsfe.org e motionensemble.de.

pág. 7: Cúpula Global 2018 de Inteligência Artificial para o Bem (AI for Good Global Summit). CC BY 2.0 por ITU Pictures.

pág. 15: Retrato de Cedric Thomas. Todos os direitos reservados.

pág. 15: Crepúsculo em “Lá Défense” – Paris. CC BY 2.0 por Gael Varoquaux.

pág. 21: Retrospectiva de fim de ano do Chaos Computer Club no 30º Congresso Chaos Communication em Hamburgo, 2013. CC BY-SA 4.0 por Wikipédia/Tobias Klenze.

Informações de licença:

https://creativecommons.org/licenses/by/2.0/deed.pt_BR

https://creativecommons.org/licenses/by/4.0/deed.pt_BR

https://creativecommons.org/licenses/by-sa/4.0/deed.pt_BR

Agradecimentos:

Gostaríamos de agradecer aos membros da equipe de campanha da FSFE pelas discussões e comentários inspiradores nos rascunhos dos textos. Agradecemos a Cedric Thomas, Prof. Dr. Simon Schulari, Dr. Matthias Stürmer, Basanta Thapa, Fernanda G. Weiden e Lori Roussey pelas suas maravilhosas contribuições. Somos gratos a Alexander Lehman e Lena Schall por produzirem o vídeo da campanha Dinheiro Público Código Público e fornecerem os elementos gráficos para esta publicação. Agradecemos a Ura Design pelo seu apoio em visualizar os fatos mais importantes sobre o Software Livre. Agradecemos a nossos doadores, apoiadores e especialmente ao Digital Rights Fund pelo apoio financeiro a esta publicação.



Matthias Kirschner

Presidente da Free Software Foundation Europe

Caro leitor,

Nos dias atuais as tecnologias digitais são um elemento crucial na infraestrutura de Estados modernos. Isso tem desafiado as administrações públicas e levanta novas questões referentes a controle, segurança, eficiência, distribuição de poder, e transparéncia das instituições.

A Free Software Foundation Europe (FSFE) tem trabalhado como uma instituição de caridade desde 2001 ajudando os usuários a controlarem a tecnologia. Acreditamos que a sociedade precisa de tecnologias que empoderem os usuários ao invés de restringirem suas liberdades. O Software Livre dá a todos – indivíduos, empresas, organizações e administração pública – os direitos de usar, estudar, compartilhar e melhorar o software. Para a administração pública, Software Livre significa mais sustentabilidade, devido ao reúso de código de softwares existentes e aos benefícios do compartilhamento de código e de custos com outras instituições. Para empresas, sociedade civil e cidadãos, políticas inovadoras de licenciamento significam mais opções de escolha, transparéncia, competição e eficiência de custos.

O Software Livre na administração pública não é uma tendência de curto prazo. Os últimos anos têm demonstrado mudanças significativas de opinião de administrações públicas sobre aquisições de TI, favorecendo cada vez mais uma abordagem estratégica e de longo prazo. Mais e mais gestores públicos estão preocupados com os custos e riscos a longo prazo em virtude do aprisionamento tecnológico (*Vendor Lock-In*). Estratégias bem-sucedidas contra esse fenômeno, com comprovado funcionamento na prática, sustentam-se em grande parte em pa-

drões abertos e licenças de Software Livre. Novas políticas de licitação ajudam a minimizar dependências e a baixar custos através de ofertas competitivas de Softwares Livres. Um crescente número de países tem implementado planos ou legislações que viabilizam o uso de licenças de Software Livre no setor público. Hoje, até mesmo projetos governamentais de TI de grande escala são publicados regularmente sob licenças de Software Livre.

A publicação desta brochura é uma resposta ao crescente número de solicitações do setor público enviadas a nós da FSFE. Essa coleção de artigos, entrevistas e informações básicas responde às questões mais comuns quanto à implementação de Software Livre no setor público. As páginas seguintes contêm casos de uso, informações de contexto e conselhos especializados relevantes para a modernização da infraestrutura pública. Como um cientista formado em administração pública, eu espero que este material contribua para a modernização da infraestrutura de TI na administração pública, e consequentemente proporcione melhores serviços para os cidadãos.

Atenciosamente,

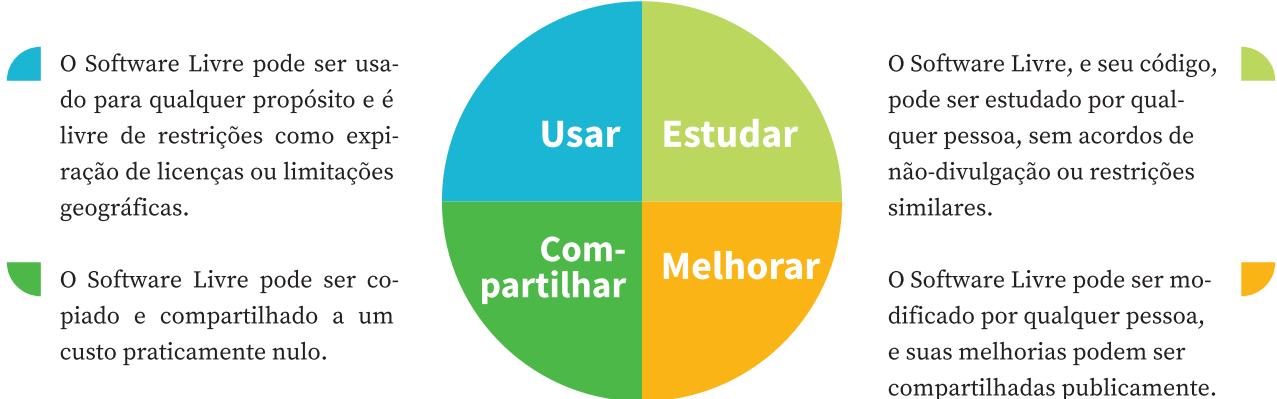
Matthias Kirschner

O que é Software Livre?

Os princípios do Software Livre são simples mas as formas de licenciamento e seus sinônimos acabam gerando confusões. Explicaremos os seus fundamentos.

O termo Software Livre foi criado em 1986 por Richard M. Stallman. Software Livre se refere à liberdade não ao preço. Ele garante aos seus usuários as 4 liberdades essenciais. A ausência de pelo menos uma dessas liberdades significa que um aplicativo é proprietário, portanto software não-livre.

As quatro liberdades



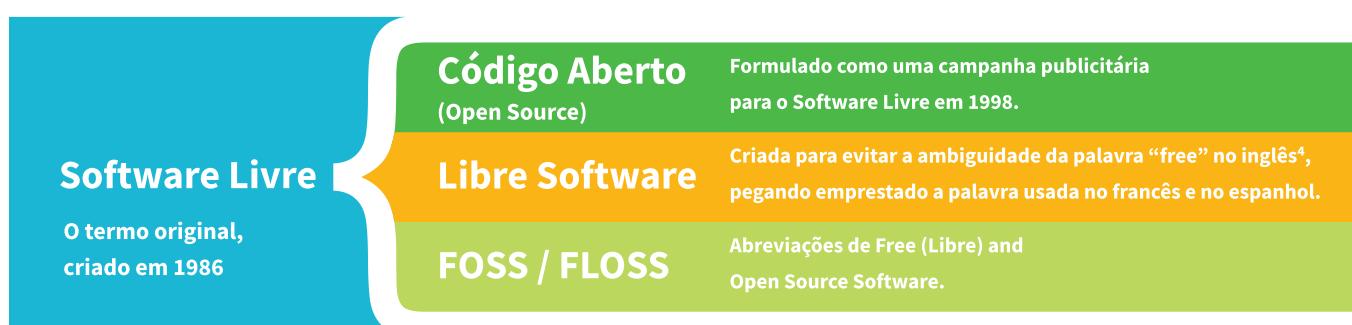
Licenças

As quatro liberdades são garantidas através de uma licença de software. A Free Software Foundation¹ e a Open Source Initiative² mantêm listas de licenças revisadas e aprovadas. Em geral, um aplicativo não deve ser considerado Software Livre se sua licença não aparecer nestas listas.

Há uma miríade de licenças com diferentes pontos focais. A escolha na verdade é uma questão estratégica e é aconselhado que se opte por uma das licenças mais amplamente utilizadas.

Sinônimos

Ao longo do tempo, pessoas criaram rótulos adicionais para o Software Livre.³ Geralmente, as motivações por trás destes termos são de destacarem diferentes características e de evitarem confusões.



O grau de liberdade que um software específico oferece é sempre determinado pela sua licença, e não pelo seu rótulo. Ou seja, não confunda termos diferentes para as mesmas características.

¹ Mais informações sobre os diferentes termos e categorias de licenças: <https://fsfe.org/freesoftware/basics/comparison>

² <https://www.gnu.org/licenses/license-list.html>

³ <https://opensource.org/licenses/category>

⁴ A palavra “free” no inglês pode significar tanto “grátis” quanto “livre” em vários contextos.

Índice

6



Como digitalizar a administração pública sem perder o controle?

10



O setor público tem permissão para liberar os seus próprios códigos publicamente?

18



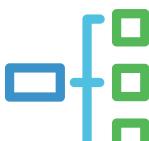
Como a sustentabilidade digital funciona na prática?

22



Como a abertura melhora a segurança em TI?

29



Como podemos modernizar as licitações públicas?

Editorial 3
por Matthias Kirschner, Presidente da FSFE

O que é Software Livre? 4

Utilizando Software Livre para democratizar cidades inteligentes 6
Entrevista com Francesca Bria, CTIO da Câmara Municipal de Barcelona

Os custos do aprisionamento tecnológico 8

Campeões ocultos 9

O impacto do Software Livre na concorrência 10
por Prof. Simon Schlauri, especialista em leis da concorrência

10 Mitos sobre Software Livre 12

Criando negócios e sentido econômico com Software Livre 14
por Cedric Thomar, CEO da OW2

Infográfico: Modernize sua TI 16

Lições da abertura de software na Suíça 18
por Dr. Matthias Stürmer, Centro de Pesquisa para a Sustentabilidade Digital

Diferentes opções para publicar Software Livre 20
por Dr. Matthias Stürmer, Centro de Pesquisa para a Sustentabilidade Digital

A caixa-preta dos softwares eleitorais 21
Entrevista com Constanze Kurz, porta-voz do CCC

Uma nova abordagem para segurança em TI 22
por Lori Roussey e Fernanda G. Weiden, especialistas em cybersegurança

Cooperação internacional através do Software Livre 24

Projetos da União Europeia e políticas de apoio ao uso de Software Livre 26

Reprogramando a Lei de Licitações 28

Como licitar Software Livre 29
por Basanta E.P. Thapa, Centro de Competências para TI Pública, Instituto Fraunhofer

Primeiros passos para apoiar Software Livre 30

Utilizando Software Livre para democratizar cidades inteligentes

O Software Livre tem se tornado um elemento central nas agendas sobre digitalização e cidades inteligentes em Barcelona. Nesta entrevista, Francesca Bria, chefe do Escritório de Tecnologia e Inovação Digital da Câmara Municipal de Barcelona, explica como o Software Livre está estimulando a inovação.

Em seu trabalho, você frequentemente menciona soberania digital e padrões digitais éticos. Poderia explicar brevemente o que é soberania digital, e qual é o papel do Software Livre em relação a isso?

Fui nomeada CTIO de Barcelona para repensar a agenda digital e tecnológica da cidade, em particular a agenda sobre cidades inteligentes. A minha missão é democratizar dados e tecnologia, e repensar as suas respectivas governanças numa maneira que sirva à população.

Faz diferença se uma plataforma como a Decidim é Software Livre ou não?

O Software Livre faz toda a diferença. Em primeiro lugar, o governo está investindo dinheiro público e é por isso que os cidadãos devem controlar o software, e a plataforma deve permanecer em domínio público. Como um dos maiores projetos de Software Livre da cidade, nós estamos aprendendo muito com o Decidim Barcelona. Tivemos até mesmo que alterar as regras de licitação para garantir que a legislação permitisse que a plataforma fosse gerenciada e mantida por uma comunidade.

Para nós, consciência sobre privacidade, soberania dos dados, tecnologia distribuída e Software Livre são componentes-chave da infraestrutura digital de uma cidade. Devido a outro projeto de Software Livre chamado Decode, estamos adicionando um módulo ao Decidim que dá aos cidadãos o controle sobre seus dados. Asseguramos que os dados estejam seguros e

anônimos, e que as pessoas possam decidir quais dados desejam manter privados e quais dados desejam doar à cidade e em quais condições.

Qual é a principal vantagem do Software Livre nesse sentido?

A maior vantagem está na possibilidade de ler e inspecionar o seu código-fonte, aprender a partir dele, e reusá-lo. Isso é muito importante porque você pode minimizar os custos e concentrar-se em investir em recursos humanos e capacitações ao invés de investir em licenciamento.

Outro motivo importante é a soberania tecnológica, que significa estar livre do aprisionamento tecnológico e da dependência de grandes corporações globais, estar livre para mudar de fornecedores, trabalhar com empreendedores locais que respeitam os direitos e liberdades do usuário, além de reter o controle sobre nossos dados. Com software proprietário, tudo era delegado a fornecedores e profissionais externos que trabalham com tecnologias específicas. Nós não queremos continuar perdendo esse tipo de conhecimento interno.

“... há muita colaboração em andamento. Sem o Software Livre isso não seria possível.”

O Software Livre nos permite trabalhar com comunidades, usar o talento de desenvolvedores de Software Livre e colaborar com outras cidades em projetos conjuntos. A longo prazo, você pode ser mais autônomo, você pode ser mais independente, e você pode ser mais transparente. Além disso, publicar o código-fonte é uma forma de devolver à sociedade o dinheiro de pagamento de impostos.



Decidim é um framework para democracias participativas que permite que cidadãos debatam, participem de eventos e criem propostas com objetivo de melhorar a vida na cidade. O código-fonte da plataforma está disponível publicamente, possibilitando o uso e adaptação do software em outras cidades. Ele é baseado em um projeto de Software Livre similar da Câmara Municipal de Madri chamado Consul.

Francesca Bria

Bria tem doutorado em Economia da Inovação pelo Colégio Imperial de Londres e mestrado em Economia Digital por Birbeck (Universidade de Londres). Ela é pesquisadora sênior e conselheira da Comissão Europeia sobre a Internet do Futuro e Políticas de Inovação.

Por último mas não menos importante, essa é uma decisão ética e política. Barcelona tem um guia específico sobre soberania de dados e padrões em ética digital – regulamentações que estabelecem que as informações e infraestrutura digitais que usamos devem ser um bem público, pertencente aos cidadãos.

Em cinco anos, como você acha que estará a situação?

Barcelona está constantemente desenvolvendo ferramentas e aplicativos de software. Quando começamos do zero, demos preferência ao uso de Software Livre e de Código Aberto. Adicionalmente, o Plano de Transformação Digital de Barcelona se comprometeu a investir 70% do orçamento anual com desenvolvimento de Software Livre e de Código Aberto.

Estamos gradualmente executando um plano de transição incluindo um projeto piloto de migração das estações de trabalho para um sistema operacional completamente livre, mas isso não se limita apenas a elas, toda a infraestrutura informacional está sendo convertida na direção de padrões abertos, tecnologias abertas e interoperabilidade.

Além disso, é importante que essas decisões não dependam de uma pessoa ou da orientação política de um governo. Acredito que o caminho certo para realizar uma transição grande como essa está em empoderar os funcionários, investir em treinamento, e construir processos de compartilhamento de conhecimento dentro das organizações.

“Atualmente temos 3.000 empresas que trabalham conosco ... mais de 60% delas são pequenas e médias.”



Sentilo é mantido por um consórcio, e já foi reusado em Dubai, nos EUA, na Itália e em outras partes da Europa. Decidim é usado por muitas cidades atualmente e temos pretensões de aprimorá-lo. Temos também outros projetos de software como o ID Digital, que compartilhamos localmente com prefeituras menores na Catalunha.

Estamos também fazendo entrevistas e pesquisas para verificar que projetos outras cidades têm desenvolvido e publicado como Software Livre. Por exemplo, Helsinski tem desenvolvido um aplicativo muito bom para compartilhamento de transporte e eles têm outro aplicativo de cidadania parecido com o nosso. Estamos cooperando com Amsterdã e Turim, então há muita colaboração em andamento. Sem o Software Livre, isso não seria possível.

Você mencionou que investe 70% do orçamento de novos desenvolvimentos na implementação de Software Livre. Que efeito isso tem na economia local?

Isso cria Softwares Livres locais e abre um ecossistema tecnológico que pode fortalecer a economia de inovação colaborativa. Licitações públicas podem criar novos mercados e estimular a indústria local.

Atualmente temos 3.000 empresas que trabalham conosco através de licitações públicas e mais de 60% delas são

Sentilo

Sentilo é uma plataforma de sensores e atuadores projetada para municípios ou organizações que processam grandes quantidades de informação recebidas do terreno. Ela gera informações geradas por vários dispositivos, como sensores de ruído e de contaminação do ar ou congestionamento de trânsito. Ela é usada e apoiada por uma comunidade ativa e diversa de cidades e empresas.

pequenas ou médias empresas. Esses contratos nos beneficiam por não nos aprisionarem ou nos restrigirem tecnologicamente. Ou seja, quem tiver capacidade pode ganhar estes contratos. Essa é uma grande mudança para uma administração municipal. Queremos empoderar o movimento local de Software Livre e de Código Aberto e prover uma plataforma para sustentá-lo e desenvolvê-lo.

Já existem outras cidades que enxergam vantagens no Software Livre, mas há também algumas administrações que ainda têm preocupações. Como você os convenceria? Qual seria o seu principal argumento?

Primeiro, o dinheiro que você investe entra no seu ecossistema de empresas locais, da indústrias local e

de empreendedores locais; segundo, a possibilidade de colaborar com outras cidades em projetos conjuntos e ajudar pequenas cidades a se beneficiarem desses projetos; terceiro, garantir a soberania tecnológica da infraestrutura e de serviços críticos. Isso é muito importante para construir uma sociedade mais democrática, inclusiva e sustentável.

Por Erik Albers.

Editado por Alexandra Busch.

Os custos do aprisionamento tecnológico

Funcionalidades convincentes, bom serviço e confiança no fornecedor atual são motivos positivos pelos quais instituições devem evitar fazer mudanças em sua infraestrutura digital. O mais importante e geralmente subestimado motivo negativo para a renovação de um contrato com um fornecedor específico é o aprisionamento tecnológico (em inglês, *Vendor Lock In*).

O aprisionamento tecnológico torna os clientes dependentes de um único fornecedor. Isso cria obstáculos artificiais ao aumentar significativamente os custos e o esforço envolvido numa migração para outro fornecedor. Este aprisionamento pode ser causado por obstáculos legais como cláusulas contratuais, dependência indispensável de outro software, licenças proprietárias, bem como padrões fechados ou obscuros que causam incompatibilidades.¹

Nas administrações públicas existem muitos casos de aprisionamento tecnológico. Por exemplo, formatos de arquivos de documentos que são legíveis apenas por um produto específico, conteúdos de banco de dados não conversíveis para o formato de um fornecedor concorrente, ou a obrigação da aquisição de uma atualização superfaturada de software para que seja possível acessar um arquivo e para receber atualizações de correção. Ao longo dos anos, muitas instituições têm gastado grandes quantias de recursos financeiros em sistemas desatualizados apenas para evitar os custos de migração causados pelo aprisionamento tecnológico.

Com a terceirização da prestação de serviços e de armazenamento através de provedores de nuvem se tornando cada vez mais proeminentes, o problema do aprisionamento tecnológico está crescendo. O controle e o conhecimento das tecnologias implantadas está sen-

do reduzido² enquanto que os custos podem facilmente explodir devido ao conhecimento limitado. Quanto mais profunda for a integração de um departamento a um ambiente tecnológico específico, mais difícil será a migração para soluções oferecidas por outros fornecedores.

Obviamente, clientes não escolhem conscientemente se tornarem presos a um determinado fornecedor, e geralmente não estão atentos a essa ameaça. Mas existem algumas formas de se evitar essa situação:

- Observe o mercado antes de adquirir um produto e leve em conta tanto os custos de entrada quanto os de saída.
- Certifique-se de que os dados poderão ser migrados para fornecedores alternativos sem custos imprevisíveis.
- Use produtos que apoiam padrões abertos³ independentes e interoperáveis com softwares alternativos.
- Use Softwares Livres que permitam a contratação de terceiros para aprimorar e corrigir o software.

Produtos de Software Livre que utilizam Padrões Abertos ajudam a prevenir a migrações de custos ao permitir melhorias incrementais e suporte independente. Eles proporcionam flexibilidade em um mundo digital em constante mudança.

¹ Mackintosh S. 2018, An Open Digital Approach for the NHS.

² McKendrick J. 2011, Cloud Computing's Vendor Lock-In Problem: Why the Industry is Taking a Step Backward.

<https://www.forbes.com/sites/joemckendrick/2011/11/20/cloud-computings-vendor-lock-in-problem-why-the-industry-is-taking-a-step-backwards>

³ <https://fsfe.org/activities/os>

Campeões ocultos

Ao pensar no bem público, a maioria das pessoas pensa em ruas, escolas ou hospitais. Progressivamente, mais e mais administrações públicas agora também pensam em software e, sendo mais preciso, em Software Livre.

A lista de projetos de software financiados por dinheiro público tornando os seus códigos-fonte disponíveis publicamente e compartilhando-os com outras instituições vem crescendo bastante dia após dia. Por exemplo, a Rede Nacional de Bibliotecas Alemãs (GBV) provê uma solução em Software Livre que é usada amplamente por bibliotecas na Alemanha.¹ O Estado de Luxemburgo oferece um registro eletrônico de dados de saúde utilizado por muitos médicos e clínicas.² O Ministério do Interior holandês publica boa parte do código-fonte do software do banco de dados de registro civil (BRP).³ O Ministério das Finanças checo disponibiliza uma visualização em tempo real dos planos orçamentários das instituições.⁴ Algumas soluções em Software Livre têm sido inclusive utilizadas internacionalmente. O serviço Nacional de Levantamento

Topográfico finlandês desenvolveu o Oskari, um software para visualizar e analisar dados espaciais e estatísticos. O Oskari inclui funcionalidades que canalizam o feedback dos cidadãos quanto aos novos projetos de infraestrutura, fornece serviços de informação em tempo real e mostra áreas apropriadas para pesca.⁵ O tem convencido o portal Geográfico Nacional Islândês e o Serviço Nacional de Levantamento Topográfico da Moldávia a também utilizarem o Oskari.

Instituições que não consideram a possibilidade de publicar seus softwares estão deixando passar oportunidades importantes. Se outros agentes governamentais não sabem da existência de um código potencialmente reutilizável, isso pode resultar em projetos de software redundantes e consequentemente maiores custos para as instituições e os contribuintes. Respaldadas por experiências positivas, as administrações públicas estão percebendo que compartilhar o código-fonte de seus projetos está totalmente de acordo com os seus próprios interesses.

Centenas de agentes governamentais têm uma conta no GitHub⁶, uma plataforma privada para compartilhamento de código, e alguns países até mantêm os seus próprios repositórios públicos de código.

Centenas de agentes governamentais têm uma conta no GitHub

A mudança atualmente em curso no setor público não é meramente uma questão de quantidade. É também uma iniciativa que propõe melhor governança e mais transparéncia para os serviços governamentais. Tem sido comprovado que a transparéncia de software

cria confiança na infraestrutura digital do governo, especialmente em áreas sensíveis. As políticas de Software Livre permitem verificações de segurança por agentes independentes. O aplicativo de mensagens instantâneas criptografadas oferecido pela Agência de Cybersegurança Nacional francesa é baseado em dois projetos de Software

Livre: Matrix e Riot. O projeto de Software Livre OSiP (Online-Sicherheitsüberprüfung, ou Verificação de Segurança On-line) é utilizado nas checagens de segurança em aeroportos alemães.⁷ O código-fonte do ProZorro, software ucraniano vencedor do prêmio de transparéncia em processos de licitação, pode ser verificado *on-line*.⁸ O capítulo ucraniano da ONG Transparéncia Internacional apoiou esta decisão.⁹ Ainda mais importante que os benefícios econômicos do Software Livre está a conquista do bem mais valioso numa democracia: a confiança dos cidadãos na infraestrutura estatal. Quanto mais as infraestruturas dos Estados modernos dependerem da TI, mais críticos se tornarão esses argumentos.



OGPtoolbox

O software “Open Government Toolbox” (Kit de Ferramentas de Governo Aberto) agrupa mais de 1400 ferramentas (Software Livre em sua maioria) desenvolvidas por mais de 590 organizações. De visualização de dados até ferramentas voltadas à participação cidadã, e aplicações para iniciativas urbanas locais – o espectro dessa impressionante coleção mostra o potencial do Software Livre em conjunto com o uso de dados abertos. <http://ogptoolbox.org>

¹ <https://github.com/gbv>

² <https://joinup.ec.europa.eu/community/osor/news/luxembourg-open-source-health-records-system-gains-foothold>

³ <https://github.com/MinBZK>

⁴ <https://github.com/otevrena-data-mfcr>

⁵ <http://www.oskari.org>

⁶ <https://government.github.com/community>

⁷ <https://www.wirtschaft.nrw/online-sicherheitspruefung-osip>

⁸ <https://openprocurement.io>

⁹ <https://ti-ukraine.org/en/news/prozorro-sale-wins-global-anti-corruption-challenge>

O impacto do Software Livre na concorrência

As administrações públicas distorcem o mercado ao publicar Software Livre? O prof. Dr. Simon Schlauri publicou um relatório detalhado para o Cantão de Bern, na Suíça, sobre os argumentos legais e econômicos resultantes desse questionamento.

O Software Livre é bem estabelecido no mundo da TI. Um número significante de empresas, desde as pequenas até as corporações globais, investem consideráveis somas na personalização e no uso de Software Livre. As administrações públicas agora também usam Software Livre regularmente. Por exemplo, nas administrações municipais, nos bancos de dados dos tribunais, ou no fornecimento de dados georreferenciados na internet.¹ As razões para utilização de Software Livre em empresas e nas administrações públicas são diversas, por exemplo, a abertura dos padrões utilizados, a independência de provedores e de produtos, o intercâmbio com a comunidade de usuários e desenvolvedores, segurança, bem como estabilidade e possíveis economias de custo.

A ampla disponibilidade de Software Livre cria um ecossistema no qual desenvolvedores, fornecedores de serviços complementares (tais como manutenção e suporte) e usuários estão igualmente envolvidos. Outra importante vantagem do modelo de Software Livre é o desenvolvimento de software acelerado pelos usuários e desenvolvedores, uma vez que o código-fonte do software se torna disponível a terceiros.

É discutível se a liberação de software sob uma licença de Software Livre poderia violar a neutralidade de concorrência (ou seja, a obrigação do Estado em tratar todos os competidores igualmente). Em alguns países, a obrigação de assegurar a neutralidade de concorrência em ações do Estado é um princípio constitucional. Isso também pode derivar da legisla-

ção da União Europeia, a exemplo da lei da política de mercado único, ou de licitação pública ou de assistência estatal.

Se o Estado por si só entra num mercado em busca de interesses econômicos no processo, isso geralmente não é problemático da perspectiva da neutralidade da concorrência. Por outro lado, na maioria dos casos a consideração de outros motivos (interesse público) leva a uma distorção dos mercados e por consequência viola a neutralidade da concorrência. Em casos extremos, a atuação do setor privado é

completamente deslocada  virtude do provimento por parte do governo já que a comunidade subsidia estes serviços com o uso de fundos estatais do orçamento geral. Vice-versa, pode-se argumentar que quanto mais o Estado age como um competidor privado racional, menor é o risco de distorção da concorrência.

Um foco exclusivo em
software de código
fechado poderia
violar o princípio de
neutralidade da
concorrência

Permissibilidade para tornar Software Livre disponível pelo Estado, do ponto de vista de um participante do setor privado, depende de se a liberação de Software Livre seria um modelo de negócios viável, isto é, se um participante do setor privado, em uma situação similar como a da comunidade, também decidiria liberar o código-fonte sob licença de Software Livre.²

Adicionalmente, recorrer a leis de subsídio pode ser útil. Subsídios incluem benefício a fundo perdido (não reembolsáveis), condições preferenciais para empréstimos, garantias, serviços gratuitos ou cobrados com desconto, e benefícios em espécie. De acor-



do com o caso da lei da UE, tais benefícios existem se um investidor privado, comparado a uma administração pública relevante, não tomou a mesma providência em uma situação comparável.³

Dessa maneira, quando existem motivos para participantes do setor privado liberarem os seus próprios códigos livres de custo sob uma licença de Software Livre, a liberação por parte do Estado geralmente não gera problemas numa perspectiva de neutralidade da concorrência.

Ademais, um foco exclusivo da administração pública em softwares de código-fonte fechado poderia também constituir discriminação contra as empresas envolvidas no ecossistemas de Software Livre mencionado acima e assim também violando o princípio da neutralidade na concorrência.

Além disso, em termos das regras de licitação, a questão que surge é em que condições a cooperação entre duas ou mais instituições públicas contratantes seria possível em um projeto compartilhado de Software Livre. Esse é o caso coberto pela Diretiva de Licitação Pública da União Europeia. Se há uma base contratual entre as instituições públicas em que objetivos comuns são buscados, onde a cooperação é exclusivamente do interesse público, e onde as instituições contratantes envolvidas realizam menos de 20% das ações cobertas pelo mercado como um todo, especialmente no caso de softwares específicos para administrações públicas, dificilmente isso causa algum problema.⁴

Devemos notar, entretanto, que a disponibilização de Software Livre por uma administração pública pode ser problemática a partir de uma perspectiva de comércio justo ou das normas administrativas, se

a disponibilização daquele software exceder as atribuições legais da administração pública. Sendo assim, por exemplo, a disponibilização de um software genérico de escritório por um ente público não deveria ser permitida.

Não obstante, na maioria dos casos uma estratégia governamental para publicar Software Livre permanece não problemática, visto que há muitas razões válidas para fazê-lo que também se aplicariam aos participantes privados do mercado.

¹ OSS study 2018, <https://www.oss-studie.ch/assets/pdfs/OSS-Studie2018.pdf>; Thomas Poledna / Simon Schlauri / Samuel Schweizer, Gutachten zu den rechtlichen Voraussetzungen der Nutzung von Open Source Software in der öffentlichen Verwaltung, Berlin 2017, <http://carlgrossmann.com/?ddownload=11748>, p. 23 ss.

² Poledna/Schlauri/Schweizer, p. 101 ss., 108.

³ Poledna/Schlauri/Schweizer, P. 107 s.

⁴ Poledna/Schlauri/Schweizer, p. 123 ss.

⁵ Poledna/Schlauri/Schweizer, p. 85, 158.



**Prof. Dr.
Simon Schlauri**

Prof. Dr. Simon Schlauri é um advogado e, desde 2012, sócio do escritório de advocacia suíço Ronzani Schlauri Attorneys, especializado em leis de tecnologia e de informação. De 2009 a 2012 ele trabalhou como conselheiro no setor da indústria de telecomunicações e de TI. Simon Schlauri é doutor no tema assinatura eletrônica e habilitado em neutralidade da rede (leis de telecomunicações). Ele regularmente escreve artigos em tópicos sobre leis de TI e faz consultorias a clientes sobre questões legais de TI, especialmente nas áreas de software de código aberto e conteúdo aberto.

10 mitos sobre Software Livre

Apesar do Software Livre ter se tornado cada vez mais popular, a sua percepção ainda é dominada por mitos bem persistentes. É hora de esclarecer os equívocos mais comuns.

01

“É impossível fazer negócios com Software Livre.”

As licenças de Software Livre guiam inovação e negócios mundo afora. Várias grandes empresas se baseiam fortemente nelas. Montadoras de automóveis aplicam Software Livre em computadores de bordo para gerenciar chamadas de emergência automatizadas. A plataforma de negócios da Bolsa de Valores de Londres é baseada em Software Livre. Além disso, os principais servidores em muitas empresas globais se baseiam em Software Livre. Algumas das maiores empresas de tecnologia não existiriam hoje sem o Software Livre.

02

“Software Livre é desenvolvido por amadores.”

Apesar de haver um grande número de projetos de Software Livre iniciados por voluntários, é pre-judicial ao Estado que apenas programadores por “hobby” contribuam com o código. Muitos entusiastas do Software Livre são profissionais de TI altamente qualificados. Grandes empresas investem milhões de euros em projetos de Software Livre ao destacarem seus funcionários para trabalharem na melhoria de código. Hoje, estima-se que 90% das contribuições ao *kernel* do Linux, a base dos sistemas operacionais GNU/Linux, veio de desenvolvedores profissionais. Apesar do *kernel* do Linux ter sido iniciado por um estudante de Ciência da Computação, hoje ele é parte da infraestrutura crítica de TI de praticamente todas as corporações globais.

03

“Não há suporte profissional para produtos de Software Livre.”

Muitas empresas de Software Livre especializam-se em serviços de suporte a seus clientes tais como treinamento, documentação, desenvolvimento e implementação de melhorias, ou soluções sob medida. Clientes que procuram por pacotes de suporte profissional podem escolher entre um amplo número de fornecedores. Software Livre não é mais um nicho. É um mito que as empresas de tecnologia não são capazes de ganhar dinheiro com Software Livre. Associações empresariais tais como OW2, OpenForum Europa (OFE) e a Aliança de Negócios em Fonte Aberta (OSBA na sigla inglesa) representam centenas de pequenas e médias empresas europeias especializadas em serviços em Software Livre.

04

“Tornar o código-fonte publicamente disponível traz riscos de segurança.”

O Software Livre que pode ser acessado publicamente pode ser verificado por grupos independentes na busca de falhas de segurança e *backdoors* propositalmente instalados. A publicação do código serve como uma medida de ganho de confiança. O conceito de segurança através da obscuridade é avaliado por especialistas como ineficaz por esconder os problemas de segurança ao invés de ajudar a resolvê-los. Em alguns contextos, isso pode até mesmo gerar um risco à segurança. Licenças fechadas [] mantêm distante a possibilidade de ajuda útil, além de falhar em desarmar agentes mal intencionados.

05

“Software Livre reduz os custos dos serviços de TI a zero.”

É verdade que o reuso de código de Software Livre pode ser gratuito, mas isso não significa que uma instituição que use somente softwares livres não tenha custos com TI. O desenvolvimento e implementação de melhorias sob medida, bem como serviços de suporte, custarão dinheiro. As licenças de Software Livre são, na maioria dos casos, escolhidas não apenas por causa dos incentivos monetários a curto prazo, mas também como parte de uma estratégia de autodeterminação da soberania em TI, prevenindo o aprisionamento tecnológico.

06

“Software Livre é geralmente menos amigável ao usuário.”

A época em que as alternativas em Software Livre careciam de interface de usuário apropriada já acabou. O sistema operacional mais popular em smartphones (Android) é baseado em Software Livre. A maioria dos aparelhos de TV atuais funcionam com Software Livre. A Wikipédia, um dos sites mais populares atualmente, é baseada completamente em Software Livre. Alguns dos mais usados Sistemas de Gerenciamento de Conteúdo (CMS) para sites como WordPress, Drupal e Typo3 também são Software Livre.

07

“Software Livre não é compatível com software proprietário.”

O Software Livre pode ser complementar ao software proprietário. Muitas organizações usam software proprietário e livre. Exemplos proeminentes de projetos que rodam em vários sistemas operacionais são o navegador Firefox, o LibreOffice e o leitor de mídias VLC. Assim como esses, também existem aplicativos proprietários que são compatíveis com os sistemas operacionais livres. Enquanto nos projetos proprietários é o dono do software quem decide sobre sua compatibilidade desejada, as licenças livres permitem que usuários corporativos e privados os modifiquem livremente de acordo com suas necessidades.

08

“Software Livre é software sem licença.”

Existem muitas licenças de Software Livre que têm termos específicos para cópia e modificação do código. O “livre” do Software Livre se refere às “quatro liberdades”: os direitos de usar, estudar, compartilhar e melhorar o software. Para um trecho de código ser qualificado como Software Livre, não é suficiente apenas publicá-lo. Para garantir que o software garanta essas liberdades a outros, é necessário uma licença apropriada.

09

“Usar Software Livre impõe riscos legais.”

Decisões judiciais têm reforçado que você não é obrigado a fornecer nenhuma garantia para o Software Livre se não houver nenhuma evidência adicional que afirme o contrário. Entretanto, como qualquer outra licença, há certas regras que vêm com uma licença de Software Livre que requerem concordância. Por exemplo, você não tem o direito de inibir aos outros usuários as quatro liberdades de usar, estudar, compartilhar e melhorar o software.

10

“Software Livre é uma tendência que não durará muito tempo.”

Software Livre não é uma tendência recente – é, na verdade, uma história de sucesso há um bom tempo. A primeira licença explícita de Software Livre foi publicada no ano de 1980. Desde então, o número de indivíduos, negócios e instituições que utilizam Software Livre e contribuem com código cresce constantemente. Mais e mais governos encorajam suas administrações públicas a usarem Software Livre e a prover acesso a códigos financiados publicamente sob uma licença de Software Livre. Alguns países, como a Bulgária e a Itália, têm implementado leis que estabelecem que novos projetos financiados com dinheiro público deverão resultar em código público.

Criando negócios e sentido econômico com Software Livre

Um número crescente de empresas seguem modelos de negócio que são baseados em políticas de licenciamento de Software Livre.

Cedric Thomas, CEO da OW2, explica como esse desenvolvimento tem mudado o panorama do setor de TI europeu.

O software pode ser gratuito, mas custo zero não significa valor nenhum! Uma pesquisa recente mostrou que 80% a 90%¹ de um aplicativo é tipicamente composto de componentes reusados dos quais a maioria é código aberto. Embora o valor econômico gerado por desenvolvedores de Software Livre por código reusado, redução de esforços e economias em custos de manutenção geralmente não são contabilizados, ele é estimado em mais de 300 bilhões de euros² para a economia europeia. Além disso, empresas fazendo negócios com Software Livre, como fornecedores de software, consultores e integradores de sistemas, geram uma visibilidade para o mercado europeu com valor estimado em 20 bilhões de euros³, crescendo duas vezes mais que outros mercados de tecnologias.



“Como as empresas podem ganhar dinheiro com Software Livre?” Essa é provavelmente a questão mais comumente perguntada por aqueles que não são familiarizados com Software Livre. De uma perspectiva econômica e mercantil tradicional, a pergunta faz sentido. Entretanto, em muitos setores, produtos e serviços oferecidos livres de custo são geralmente mantidos por um modelo de negócio nem sempre aparente para consumidor. Por exemplo, apesar de não cobrar pelos seus programas, as estações de rádio fazem dinheiro vendendo anúncios. Da mesma forma, o Software Livre pode ser monetizado pela venda de serviços e produtos relacionados. Empresas que utilizam Software Livre frequentemente optam por terceirizar serviços como os de integração de sistemas, manutenção, suporte ao usuário, etc. e por adquirir produtos adicionais que não pretendem desenvolver

elas mesmas, mesmo que tenham pleno acesso ao código-fonte. Isso é apenas um bom gerenciamento.



Orientada a serviços, e com demanda crescente, a indústria de Software Livre representa cerca de 200.000 empregos na Europa. O Software Livre prospera em todos os setores da indústria e seus desenvolvedores estão por toda parte, mesmo em empresas que não se identificam com isso. A maioria dos empregos ligados ao Software Livre dependem de habilidades tecnológicas avançadas e engajamento do consumidor, sendo mais difíceis de se deslocarem para outras regiões e mais fáceis de se manterem localmente. Empregos em Software Livre são altamente qualificados e melhor pagos, com maior poder aquisitivo que a média. A maioria estão em pequenas e médias empresas, e isso contribui para uma economia mais



saudável. Além disso, contribuindo diretamente para o crescimento econômico por ajudar a cortar custos de desenvolvimento e o tempo de entrega do produto ao mercado, o Software Livre acelera a inovação enquanto entregam soluções mais efetivas.



O Software Livre tem se tornado um veículo para inovação colaborativa. As ondas atuais de inovação com computação em nuvem, “Big Data”, tecnologias em rede, inteligência artificial, “Deep learning”, “Blockchain” e, em um contexto maior, a Internet das Coisas, são todas impulsionadas por Software Livre. Inovações que não são controladas por uma única empresa, graças à sua situação de código-fonte aberto e sua abordagem colaborativa, damente entregando resultados significativos em termos tanto de resul-



Cedric Thomas

Cedric Thomas é o CEO da OW2, uma organização independente e sem fins lucrativos aberta a empresas, órgãos públicos, academia e indivíduos que compartilham o objetivo de promover o Software Livre para sistemas de informação corporativos. Ele tem mais de 30 anos de experiência em consultoria estratégica e de mercado para a indústria de Tecnologias da Informação e Comunicação (TIC). Antes de lançar a OW2, Cedric fundou a empresa de consultoria FronTier Associates e contribuiu para o surgimento de várias pequenas startups de tecnologia. Ele ajudou a estabelecer uma incubadora de startups em Paris e montou empresas tecnológicas em Boston e na Área da Baía de São Francisco.

tados técnicos quanto de penetração no mercado. A razão não é ideológica, mas organizacional: o Software Livre ajuda a combinar múltiplas tecnologias e conhecimento de fornecedores independentes, torna fluida a cooperação mais complexa por aumentar a confiança e reduzir esforços adicionais de coordenação, e também reduz barreiras econômicas ou legais. A inovação moderna é complexa, colaborativa e de código aberto.

Graças ao Software Livre, tecnologias de ponta estão prontamente disponíveis a pequenas e médias empresas, permitindo a elas competirem contra empresas grandes ao entregar soluções a preços competitivos com tecnologias de estado-da-arte. Construir produtos de software proprietário para entregar soluções similares chega a custar até dez vezes mais e incorre em significativos esforços adicionais em negociações contratuais. O mercado de software proprietário é um modelo de desenvolvimento de produto dominado por poderosos fornecedores monopolistas, enquanto o mercado de Software Livre é um mercado integrador de soluções guiado pelas necessidades do usuário, proximidade com o consumidor e mão de obra especializada. O Software Livre facilita o acesso a tecnologias de estado-da-arte, protege pequenas e médias empresas de processos caros e rígidos orientados a produto e permite-as prosperar em um processo sem atritos e orientado a serviços.

De uma perspectiva econômica, o Software Livre continua enfrentando desafios significativos na Europa. Da mesma forma que a indústria mainstream de software, da qual é um avatar, o Software Livre é dominado por fornecedores de software norte-americanos. Na América do Norte, o Software Livre é percebido como uma estratégia racional da indústria: líderes

globais de TI a combinam com enormes investimentos no desenvolvimento de produto e na publicidade para ganhar participação de mercado. A situação na Europa é diferente porque os líderes de TI são provedores de soluções ao invés de vendedores de produtos. Eles são fortes em consultoria e integração de sistemas, porém mais fracos em marketing. Como resultado, o Software Livre na Europa é percebido principalmente como um processo colaborativo, um método eficiente para desenvolver software, compartilhar propriedade intelectual e reduzir custos. Ainda existe quem veja o Software Livre como algo guiado por caçadores de liberdades e programadores individualistas. Clientes e formuladores de políticas públicas ainda estão céticos quanto ao seu valor estratégico. Eles falham em perceber que existe também um vibrante ecossistema de atividade comercial de Software Livre e que deveriam se interessar em apoá-lo.

¹ Fonte: Sonatype, DevSecOps Community Survey, 2018.

² Baseado em Estimating the Economic Contribution of Open Source Software to the European Economy, Carlo Daffara, the First OpenForum Academy Conference Proceedings, Shane Coughlan Ed. 2012.

³ Baseado em Impact du logiciel libre/Open Source en France en 2017-2020, PAC-CXP, Unpublished Survey, December 2017.



Modernize a sua TI

Falar sobre Software Livre é falar sobre liberdade. Mais precisamente, a liberdade de usar, estudar, compartilhar e melhorar software. E ainda há mais motivos para apoiar licenças de Software Livre.



1 Inovação

Uma licença de Software Livre estimula a inovação para o seu software.

Competição

Software Livre evita monopólios e fortalece a concorrência.



3 Autonomia

Software Livre encoraja o desenvolvimento e a manutenção de softwares sob medida que se adequam às suas necessidades, e não apenas às do modelo de negócio do fornecedor.

4 Sem Lock-in

As licenças de Software Livre reforçam a independência de fornecedores e oferecem mais opções de provedores de serviços.



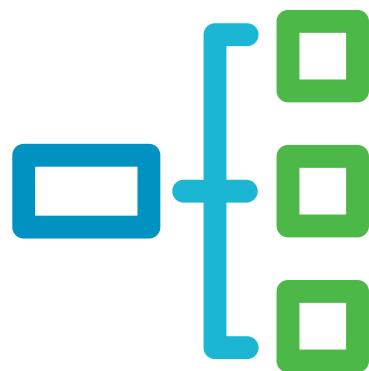


Colaboração 5

Software Livre pode ser compartilhado e utilizado de forma não-exclusiva por todos – servindo ao bem comum.

Segurança 6

Software Livre favorece verificações de segurança independentes que ajudam a corrigir falhas de segurança mais rapidamente.



Compartilhe & Copie 7

Uma licença de software livre permite que um número ilimitado de instalações sejam executadas, sem pagamento extra.

Reutilize código 8

Software Livre proporciona a liberdade de reúso de código a outros projetos.

Lições da abertura de software na Suíça

Como os governos se beneficiam ao publicar Software Livre?

Dr. Matthias Stürmer, chefe do Centro de Pesquisas para Sustentabilidade Digital em Berlim, explica por que as administrações públicas deveriam atualizar suas definições sobre o que é um “bem público”.



Parldigi

O Grupo Parlamentar sobre Sustentabilidade Digital (Parldigi) foi fundado em 2009. Sua função é dedicada a apoiar Software Livre, dados abertos e padrões abertos no setor público por meio de iniciativas parlamentares, audiências públicas e comunicados à imprensa. O Parldigi inclui mais de 50 conselheiros nacionais e estaduais dos Partidos SP, FDP, SVP, CVP, Greens, GLP, BDP, and EPP.

Na Suíça, muitas instituições e empresas públicas, como a Ferroviária Federal da Suíça, concordam que dados não-críticos devem ser liberados como Dados Governamentais Abertos (OGD). O Conselho Federal chegou a editar uma estratégia nacional para OGDs. Dentro dela, o governo argumenta que OGDs beneficiam a inovação, permitem transparência e participação, além de aumentarem a eficiência na administração. Essas são exatamente as mesmas motivações que estimulam a disponibilização de aplicativos governamentais sob uma licença de Software Livre. Por que, então, o fato de órgãos públicos publicarem seus softwares se tornou motivo de disputa na Suíça? A base deste debate e alguns eventos recentes são descritos neste artigo.

Questões de política regulatória

Em 2011, a Corte Federal Suíça ofertou o seu sistema de gerenciamento de casos OpenJustitia que foi desenvolvido internamente como Software Livre.¹ A Corte pretendia permitir a colaboração com outros tribunais federais e estaduais, poupando assim custos de desenvolvimento ao longo do tempo. Essa decisão não foi, entretanto, bem vista por todos. Uma pequena empresa de software da cidade de Bern chamada Weblaw opôs-se à publicação, uma vez que vendiam seu próprio sistema de gerenciamento de casos para a Corte Federal e a outros tribunais suíços. Eles argumentaram que a Corte Federal estava distorcendo o mercado de software ao usar dinheiro dos contribuintes.² Isso iniciou um debate público no qual um político local tomou as dores desta empresa e propôs uma política regulatória proibindo agências governamentais, e em particular a Corte Federal, de publicarem os seus aplicativos como Software Livre.³

Em reação a este debate, o grupo parlamentar para sustentabilidade digital (Parldigi) advogou pela publicação de Software Livre por governos.^{4, 5} Dessa forma, a administração federal requisitou a eles uma opinião legal sobre a possibilidade de governos poderem produzir e publicar Software Livre e, caso permitido, de que forma. Infelizmente, os professores comissionados sobre a legislação não eram familiarizados com desenvolvimento de Software Livre e em 2014 recomendaram, em uma publicação de 36 páginas, que o governo não fosse autorizado a publicar Software Livre a não ser que uma lei fosse aprovada especificamente para autorizá-lo.⁶ Essa decisão foi bastante criticada pelos políticos do Parldigi.⁷

Ao mesmo tempo, em 2014, o parlamento do cantão de Bern, que é a segunda maior região da Suíça, aprovou uma decisão unânime (com 130 votos) afirmando que a administração pública deve usar sinergias com outros governos ao colaborar no desenvolvimento e publicação de software sob uma licença de Software Livre.⁸ Além disso, uma segunda opinião legal foi requisitada e financiada pelo cantão de Bern, e publicada em 2016.⁹ A conclusão dessa segunda opinião legal indicou que, na verdade, não há necessidade de uma lei separada que permita que agências governamentais publiquem Software Livre. Isso acontece pelo fato de que código-fonte puro não é um recurso completamente monetizável, o que requereria uma regulação especial. Por isso, a publicação de Software Livre por uma organização governamental não pode ser classificada como uma interferência mercadológica.

O uso de um software livre complexo requer muito mais esforços que apenas executar o código. Um sistema de TI precisa de planejamento, integração, perso-

Dr. Matthias Stürmer



Dr. Matthias Stürmer é o diretor do Centro de Pesquisa de Sustentabilidade Digital na Universidade de Bern. Ele pesquisa, ensina e faz consultorias sobre Software Livre, dados abertos, dados ligados, governos abertos, blockchain, cidades inteligentes, licitações públicas e sustentabilidade digital. Até 2013, ele trabalhou como gerente na EY (Ernst & Young) e como líder de projetos na Liip AG, um fornecedor de código aberto suíço. Em 2009 Matthias terminou a sua dissertação de doutorado na ETH Zürich focado em comunidades de código aberto e envolvimento de firmas. Ele é o secretário do Grupo Parlamentar sobre Sustentabilidade Digital e desde 2011 é membro do parlamento municipal de Bern.

nalização, migração de dados, capacitação, suporte, etc. Nenhuma dessas tarefas são executadas por um governo ao publicar o software, mas por empresas que fornecem serviços profissionais que permitem o uso de software publicamente disponível. Portanto, a liberação de Software Livre não obstrui ou compete com o setor privado, pelo contrário, cria novas oportunidades e demandas por serviços comerciais em torno do Software Livre.

Exemplos de órgãos governamentais suíços liberando Software Livre

Em 2018, o cantão de Bern oficialmente iniciou as suas atividades de publicação de Software Livre. Primeiro, toda regulação existente foi melhorada, afirmando explicitamente que publicar o próprio código-fonte sob uma licença de Software Livre é permitido.¹⁰ Em seguida, o departamento de TI do cantão desenvolveu uma diretriz sobre como exatamente a publicação de Software Livre deve funcionar de uma perspectiva legal, técnica e organizacional.¹¹ Por último, o cantão planeja publicar o seu código de Software Livre em uma plataforma (possivelmente o GitHub).

Enquanto isso, a cidade de Bern, capital da Suíça, começou a publicar os seus primeiros aplicativos de Software Livre em 2018: um software para gerenciar recursos de assistência infantil^{12, 13} e uma segunda solução de TI em larga escala para licitações públicas chamada Submiss, a ser publicada em breve.

E, apesar do debate político, agências governamentais em nível nacional têm publicado código-fonte há vários anos: o Escritório Federal Suíço de Topografia (Swisstopo) publica e mantém o código

do geoportal no GitHub para colaborar com outros agentes públicos.¹⁴ Adicionalmente, a agência Meteorológica Suíça tem vastas quantidades de código publicados sob uma licença de Software Livre¹⁵ e o Seguro Governamental de Desemprego recentemente lançou uma grande plataforma web cujo código-fonte pode ser encontrado no GitHub.¹⁶

Esses exemplos indicam um forte comprometimento dos agentes públicos suíços em publicar código sob uma licença de Software Livre, apesar do conflito inicial com a Corte Federal. As reviravoltas mostram o efeito a longo prazo de uma defesa política feita com sucesso combinada com prestação de apoio operacional a profissionais, ocasionando em um amplo suporte à publicação de Software Livre.¹⁷

¹ <https://www.inside-it.ch/articles/26217>

² https://www.plaedyoyer.ch/document/?no_cache=1&m=Artikel&rid=1088723&attr=zusatz

³ <https://www.parlament.ch/de/ratsbetrieb/suche-curia-vista/geschaeft?AffairId=20124273>

⁴ <https://www.parlament.ch/de/ratsbetrieb/suche-curia-vista/geschaeft?AffairId=20113379>

⁵ <https://www.parlament.ch/de/ratsbetrieb/suche-curia-vista/geschaeft?AffairId=20124247>

⁶ <http://www.news.admin.ch/NSBSubscriber/message/attachments/37015.pdf>

⁷ <https://www.blick.ch/news/politik/gutachten-gegen-sparen-bund-darf-keine-gratis-software-weitergeben-id3241215.html>

⁸ <https://www.gr.be.ch/gr/de/index/geschaefte/geschaefte/suche/geschaefte.gid-df80389c50524a03aed5bbe9f4d0309c.html>

⁹ <https://www.digitale-nachhaltigkeit.ch/de/2016/08/gutachten-oss-freigabe>

¹⁰ <https://www.digitalenachhaltigkeit.ch/de/2018/04/oeffentliche-gelder-fuer-offene-software-kanton-bern-passt-seine-gesetzgebung-an>

¹¹ OSS Studie 2018, articles by Rolf Aegler and Thomas Joos, <https://www.oss-studie.ch/assets/pdfs/OSS-Studie2018.pdf>

¹² <https://github.com/StadtBern/Ki-Tax>

¹³ <https://joinup.ec.europa.eu/news/manage-childcare-funds>

¹⁴ <https://github.com/geoadmin/mf-geoadmin3>

¹⁵ <https://github.com/MeteoSwiss/easyVerification>

¹⁶ <https://github.com/alv-ch/jobroom-api>

¹⁷ <https://www.derbund.ch/bern/Eigennuetzige-Software-Geschenke/story/16408835>

Diferentes opções para publicar Software Livre

Contribuições do setor público para o Software Livre vêm em diferentes tamanhos e formatos. Dr. Matthias Stürmer, chefe do Centro de Pesquisa para Sustentabilidade Digital na Universidade de Bern, fornece argumentos de por que mesmo pequenas contribuições podem ter um grande impacto.

1.

Resolução de bugs e aprimoramentos de funcionalidades

Se uma agência está utilizando Software Livre como o MariaDB (um banco de dados) ou Angular (um *framework* da linguagem JavaScript), é essencial que os seus engenheiros de software liberem algum software de vez em quando. Desenvolvedores que usam Software Livre podem consertar um *bug* ou adicionar novas funcionalidades. Se eles mantiverem os *bugs* resolvidos e os código das funcionalidades apenas para si, o *bug* aparecerá novamente na próxima versão e a nova funcionalidade não poderá ser melhorada. É, portanto, de grande interesse da organização pública contribuir de volta com melhorias, por menores que sejam, ao repositório principal de desenvolvimento da solução de Software Livre. Se a melhoria for aceita, a nova versão já incluirá a resolução do *bug* e a funcionalidade nova, levando a uma maior velocidade de desenvolvimento e menos esforços redundantes.

2.

Financiamento coletivo do desenvolvimento dos principais Softwares Livres

Em várias ocasiões, agências governamentais financiaram coletivamente o desenvolvimento de determinadas extensões de soluções de Software Livre já existentes. Por exemplo, Swisstopo ajudou no financiamento do desenvolvimento da versão 3 do OpenLayers (um *framework* de mapas para web) junto com outros escritórios de topografia europeus.¹ Juntar dinheiro e então contratar fornecedores de serviços de Software Livre para melhorar aplicativos de Software Livre já existentes, ao invés de começar novos projetos, pode melhorar a qualidade do código e diminuir os gastos através de custeamentos compartilhados.

3.

Lançar novos projetos de Software Livre

Começar um novo projeto em Software Livre (tal como o OpenJustitia da Corte Federal suíça, ou o portal geográfico da Swisstopo) através da liberação do código-fonte completo do produto de software é um investimento a longo prazo. São necessários recursos para preparar e liberar o código-fonte, a coordenação com a comunidade e possivelmente fundar uma associação sem fins lucrativos para controlar o código-fonte. Entretanto, se a criação da comunidade for bem sucedida, o software também será melhorado por outras agências, levando a uma solução mais ampla e reduzindo custos de desenvolvimento a longo prazo. Adicionalmente, através da criação de uma base de usuários ampla, o mercado de fornecedores de serviços em Software Livre cresce, diminuindo a dependência de fornecedores externos.

>

Esses três casos representam as diferentes formas que governos podem publicar Software Livre. O código-fonte resultante portanto se torna um bem público²: por definição, é não-exclusivo e não-concorrente. Publicar software financiado por dinheiro público portanto faz sentido, visto que agências governamentais devem investir em bens públicos para maximizar o seu benefício para a sociedade, como acontece por exemplo no apoio à pesquisa básica ou na promoção de proteções ambientais.

¹ <http://www.ossdirectory.com/che/oss-top-news/single/article/institutionelles-crowdfunding-fuer-open-source-entwicklung-von-swisstopo>

² <https://link.springer.com/article/10.1007/s11625-016-0412-2>

A caixa-preta dos softwares eleitorais

Antes da eleição federal alemã, o Chaos Computer Club (CCC) encontrou graves falhas de segurança em um software de contagem de votos (PCWahl). Isso levou a um debate público sobre segurança de TI nas eleições. Nós conversamos a respeito deste assunto com Constanze Kurz, porta-voz do CCC.

Seria possível manipular os resultados das eleições?

Eu considero isso um perigo teórico, adicional aos perigos já existentes. Foi importante para nós não simplesmente dizer “estas são as brechas de segurança”, mas ao invés disso, “existem problemas estruturais que devemos resolver”. Além disso, mesmo que não sejam a mesma coisa, a discussão sobre possível manipulação na eleição dos EUA nos mostra que talvez iremos lidar com grandes ameaças.

Como se chegou ao ponto de um sistema como o PCWahl ter sido utilizado?

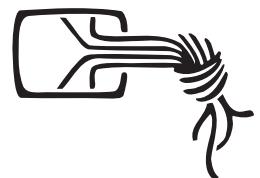
Em um nível estrutural, o problema existe em virtude dos produtores do software **manterem o desenvolvimento no mesmo nível de confiança** que temos no sistema eleitoral.  simplesmente assumimos que os resultados são legítimos. Entretanto, nós somos da opinião de que é preciso criar confiança através de uma nova forma de transparência, e também através do acesso ao código-fonte das ferramentas de apoio eleitoral. Uma eleição não pode ser uma caixa-preta.

Frequentemente se argumenta que não se pode publicar código-fonte de projetos de TI específicos por motivos de segurança...

Se você programou um sistema organizadamente, então você também consegue provar que desenvolveu a segurança que, por exemplo, você havia garantido ao seu cliente. Você pode também provar como você fez isso. Eu penso que esse debate já está ultrapassado. Se você escreve software e o publica, você está assumindo o risco de que alguém produza cópias dele. Todavia isso não deveria ser uma desculpa. E certamente não em atividades estatais, onde pagamos nossos impostos para sistemas de software, ou onde o software é instalado em áreas democraticamente críticas. A importância de termos a possibilidade de conduzir homologações independentes em tais softwares se sobrepõe a essas preocupações.

Por Katharina Nocun.

Editado por George Brooke-Smith.



Chaos Computer Club (CCC)

O Chaos Computer Club (CCC) é a maior organização hacker europeia, com mais de 9.000 membros. Membros da organização regularmente participam de audiências parlamentares como especialistas e aconselham o Tribunal Federal Alemão em decisões sobre questões tecnológicas.



Linus Neumann, Constanze Kurz, and Frank Rieger, porta-vozes do CCC (da esquerda para direita).

Uma abordagem aberta para a segurança em TI

Como pode um software ser seguro quando seu código-fonte está publicado abertamente? Existem bons motivos pelos quais muitas empresas e instituições governamentais confiam no Software Livre.



Engenharia reversa

Se o código-fonte de um aplicativo não está disponível, a engenharia reversa pode ajudar a revelar como ele funciona.

Isso é, em geral, muito trabalhoso e às vezes até ilegal.

Enquanto que a tecnologia assume cada vez mais um papel central em como as organizações e indivíduos se auto-organizam e interagem uns com os outros, a segurança se torna uma área vital de preocupação no governo e nas empresas. A segurança também está se tornando cada vez mais complexa à medida em que se aumenta a sofisticação e a complexidade com a qual nós usamos e contratamos tecnologia. Isso se aplica não apenas aos dispositivos, mas também à proteção de um volume crescente de informações pessoais e sensíveis armazenadas em ambientes de nuvem pública. Os governos têm a responsabilidade de manterem apropriadamente os dados pessoais que armazenam. Diretrizes de privacidade e segurança precisam ser cumpridas. Particularmente de um ponto de vista de privacidade, normas como a Regulamentação Geral de Proteção de Dados (GDPR) evidenciam o aumento da preocupação sobre como os dados dos usuários são manipulados por grandes empresas de Software como Serviço (SaaS).

Quando falamos de segurança, entretanto, as coisas tornam-se ainda mais complexas. Se você considerar que os governos devem ser responsabilizados nos mesmos padrões das grandes empresas privadas, é importante que sejam exigidos os mais altos padrões de segurança e higiene. Segurança e higiene se apresentam por meios e formas distintas. Uma delas assegura que você mantenha seu conjunto de softwares atualizado, e assim que você seja capaz de auditar o que está acontecendo na sua solução. Quanto a isso, o Software Livre exerce um papel importante.

Softwares Livres podem ser criados colaborativamente por uma ampla comunidade de especialistas

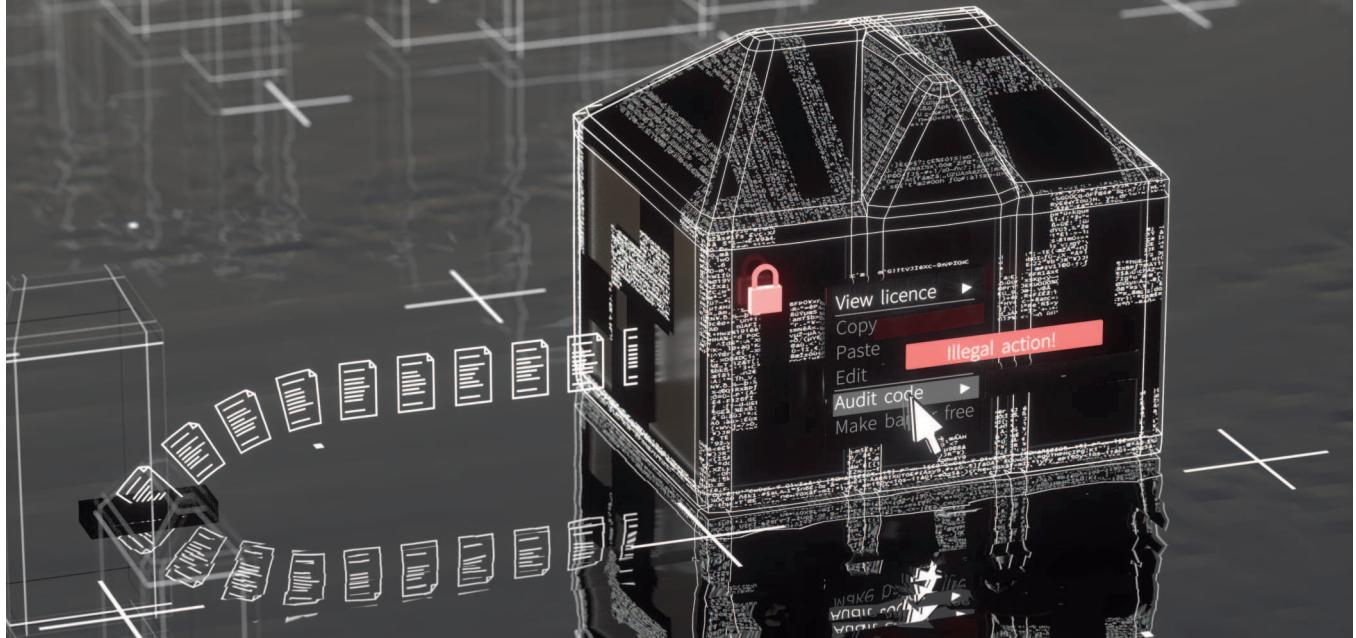
em software aliados a especialistas em segurança. À medida em que os trechos de código são escrutinados, garantimos que uma série de outras questões venham à tona e com mais rapidez.

De uma perspectiva de engenharia de software, é muito mais fácil inspecionar código do que fazer engenharia reversa a fim de aprender como um trecho de software funciona. Com Software Livre, qualquer empresa ou órgão público pode realizar sua própria auditoria do trecho de código ou de qualquer parte do aplicativo no qual tiver interesse. Empresas de software proprietário geralmente contratam seus próprios auditores, e os clientes devem confiar na palavra deles em relação às funcionalidades de segurança do software que vendem.

Órgãos governamentais podem realizar a sua própria auditoria do código-fonte ou de qualquer parte do aplicativo no qual eles têm interesse

Isso está relacionado a uma outra questão em termos da gestão das vulnerabilidades. Se a segurança de um sistema é tratada apenas a portas fechadas, pactos de acessos clandestinos (*backdoors*) que forneçam a terceiros acesso a dados, são mais propensos a serem criados – geralmente após uma auditoria. Isso tem consequências políticas e de segurança para empresas e administrações públicas, o que pode consideravelmente manchar suas reputações aos olhos da população.

À medida em que softwares publicamente editáveis são frequentemente aprimorados por uma quantidade relativamente grande de usuários, produtos baseados em Software Livre garantem que questões de segurança são desvendadas rapidamente. Uma vez descoberta, qualquer pessoa ou empresa pode ler o código, entender a questão, e submeter uma corre-



ção que a resolverá. Pelo mesmo raciocínio, sem uma comunidade por trás, código disponível abertamente não é mais seguro do que software de fonte fechada.

No modelo de software proprietário, apenas uma empresa tem acesso ao código-fonte. Mais importante, as prioridades são alinhadas à lucratividade das funcionalidades desenvolvidas. Uma vez que uma melhoria é levantada, ela será analisada e comparada com outras requisições, e pode ser aquela melhoria de segurança (mesmo que seja importante para você) que pode não ser tão prioritária para a empresa da qual você depende para consertar o problema. Você deverá aguardar pelo seu lugar na fila e conviver com aquela vulnerabilidade enquanto não existir uma solução. Em alguns casos, você também dependerá da agilidade do fornecedor para aplicarem as correções. Se você suspender o contrato, pode ser que eles não tenham mais obrigação alguma de consertar. A queda de servidores de hospitais britânicos causada pelo WannaCry é um exemplo perfeito.¹ No modelo de Software Livre, se o seu fornecedor não é solícito para consertar seus problemas, você pode contratar qualquer outro para fazê-lo: você e qualquer outra pessoa pode ter acesso ao código-fonte. Você possui seus próprios compromissos e prioridades e pode conseguir uma correção tão rápida quanto queira, e assegurar que ela será aplicada ao seu sistema tão rápida quanto você precisar que seja.

Os padrões mais populares para segurança surgem com a liberdade de utilizar, estudar, compartilhar e modificar. Desde a forma como nós protegemos nossos sites, protegemos a comunicação com e-mail encriptado ou na segurança de nossa rede, existem padrões abertos documentados. Garantir que os

melhores padrões sejam implementados de maneira sólida e colocados ao serviço das pessoas para proteger suas informações é dever dos governos. Os governos devem buscar urgentemente manter-se independentes das agendas ou fidelidades dos fornecedores em suas missões críticas. E para isso, o Software Livre é a única solução lógica.

¹ Veja Townsend M. e Doward J., “Cyber-attack sparks bitter political row over NHS spending” the Guardian (Londres, 14 de Maio de 2017) disponível em <https://www.theguardian.com/technology/2017/may/13/cyber-attack-on-nhs-sparks-bitter-election-battle> acessado em 11 de Agosto de 2018.



Fernanda G.
Weiden

Fernanda G. Weiden é membro da Assembleia Geral da FSFE e atuou como sua vice-presidente entre 2009 e 2011. Ela atualmente trabalha como Diretora de Engenharia de Produção no Facebook. Fernanda é originalmente do Brasil, e está no Facebook desde 2012. Antes disso, Fernanda trabalhou na filial do Google em Zurique, na IBM e em pequenas empresas no Brasil.

Lori Roussey

Lori Roussey é uma advogada especializada na Lei Europeia de Proteção de Dados e em leis de Cybersegurança. Foi anteriormente membro da organização da sociedade civil francesa The Exégètes, o time de litígio por trás de muitos casos proeminentes contra leis de vigilância da França.

Cooperação internacional através do Software Livre

As soluções em Software Livre estão ajudando governos a superarem diferentes desafios, desde governança democrática até prevenção de desastres naturais. Alguns projetos não são apenas disponibilizados, mas como também desenvolvidos internacionalmente. Projetos populares, tais como Consul, GNU Health, X-Road e CKAN ressaltam o potencial das licenças de Software Livre para cooperação através das fronteiras.

A cooperação entre nações através do Software Livre ajuda a estimular a inovação, elevar o desenvolvimento econômico, e assegurar valores de autonomia e sustentabilidade. Reusar e compartilhar software existente além das fronteiras economiza tempo e recursos valiosos, estimula a colaboração, e simplifica a integração de dados entre organizações, administrações públicas e instituições. O Ministério Federal da Cooperação e Desenvolvimento Alemão, por exemplo, encoraja o uso de padrões abertos e licenças de Software Livre em projetos que recebem seu financiamento pois isso pode abrir portas para cooperações futuras.

“Configurar cinco plataformas diferentes de participação cidadã em cada país parece contraproducente. Ainda assim, é comum casos em que diferentes ONGs e organizações que desenvolvem de forma cooperativa produzam plataformas similares porém competitivas. Para evitar esse tipo de duplicidade, descubra quais iniciativas tem sido produzidas por atores locais, ou outras organizações, e faça contato com eles.”¹ – Ministério Federal da Cooperação e Desenvolvimento Alemão

Projetos de Software Livre iniciados e financiados por administrações públicas já oferecem funcionalidades variadas. Eles ilustram um alto nível de cooperação transnacional com soluções significativas disponibilizadas por cidadãos do mundo inteiro.

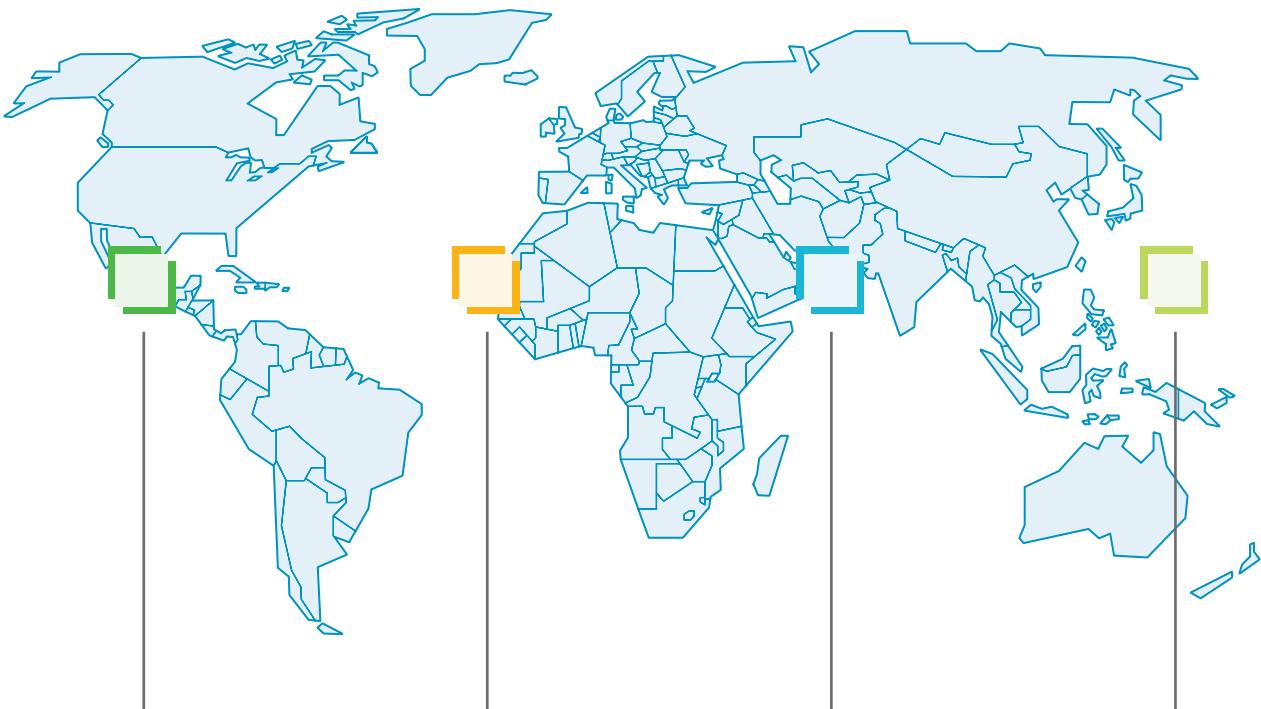
¹ Ministério Federal da Cooperação e Desenvolvimento Alemão,
“Toolkit – Digitalisation in Development. Cooperation and International Cooperation in Education, Culture and Media”, 2016, pg. 91

X-Road

O X-Road fornece um meio para corporações do setor público e do privado conectarem sistemas de bancos de dados internacionalmente. A flexibilidade desse macro-sistema é um benefício para cidadãos e funcionários públicos. A troca direta de dados dentro do X-Road permite às instituições economizarem tempo, recursos e custos, enquanto que sua estrutura distribuída assegura disponibilidade, integridade e confidencialidade para as informações trocadas. O X-Road foi iniciado pelo governo da Estônia e está em contínuo funcionamento por 15 anos. Em 2017, por exemplo, o X-Road conectou uma infinidade de instituições, bases de dados e serviços, atendendo 563,3 milhões de requisições e economizando cerca de 800 anos de tempo de trabalho.

CKAN

O CKAN (Rede Compreensiva de Arquivos de Conhecimento) fornece ferramentas para agilizar o processo de publicação, compartilhamento e busca de dados. Através de uma interface de gerenciamento de conteúdo, o serviço melhora a acessibilidade e a utilidade dos dados. Os usuários podem melhorar a facilidade de busca de seus dados e organizar catálogos com etiquetas especializadas. Por esses motivos que governos de 31 países adotaram o CKAN em suas políticas de dados abertos, permitindo aos cidadãos explorarem numerosos bancos de dados de todo o país com facilidade. Além desses países individualmente, o Portal de Dados Abertos da União Europeia também utiliza este software. Nele, qualquer um pode acessar os resultados de pesquisas sobre países do bloco. O CKAN é um projeto da Open Knowledge Foundation, que mantém seu código-fonte principal.



O X-Road é usado na Estônia, Finlândia, no Azerbaijão, nas Ilhas Faroé, na Argentina e em El Salvador.



O CKAN é usado por governos de 31 países, incluindo a Alemanha, o Reino Unido, a Holanda, a Austrália, o Brasil e os Estados Unidos.



O GNU Health é usado no Brasil, na Espanha, Alemanha, Áustria, Argentina, no México, Peru, na Guatemala, em Honduras, Camarões, Jamaica e na República Dominicana.



Consul é usado na Espanha, França, Albânia, em Malta, na Eslovênia, no Brasil, Chile, na Bolívia, Costa Rica, no Peru, Equador, na Colômbia, Guatemala, no México e na Coreia do Sul.

GNU Health

O GNU Health oferece uma solução informacional para infraestruturas de sistemas públicos de saúde e para administração de centros médicos sociais. Em 2008, o projeto passou a apoiar a prevenção de doenças em áreas rurais e desde então evoluiu para um sistema de informação em larga escala de dados sobre saúde, contando com um time internacional de contribuidores. Ele tem sido adotado pela Universidade das Nações Unidas além de outras instituições ao redor do mundo. O GNU Health utiliza uma abordagem modular, com diferentes funcionalidades que podem ser adicionadas para atender às necessidades específicas de centros de saúde. A boa escalabilidade permite que ele seja utilizado em diferentes cenários por indivíduos e por organizações públicas de saúde.

Consul

O Consul permite que os cidadãos participem como tomadores de decisão dos governos de suas cidades, participando de votações eletrônicas, apoiando projetos e criando petições baseadas em demandas. É um projeto de software criado especialmente para administrações municipais. Ele é usado e desenvolvido por mais de 90 governos nacionais e locais do mundo todo. Inicialmente desenvolvido pela Câmara Municipal de Madri, ele fornece uma plataforma de debates e propostas em formato de fórum, incluindo orçamentos e páginas cusotmizáveis para processos sobre legislação. O Consul é um Software Livre, portanto pode ser instalado por qualquer instituição governamental. Desenvolvedores podem se juntar ao projeto. Através da construção e da utilização do código, o software torna-se parte da comunidade.

Projetos e políticas da União Europeia que apoiam o uso Software Livre

O Software Livre garante controle sobre a tecnologia que está sendo usada e possibilita acesso público aos softwares financiados com recursos públicos. Estas vantagens explicam por que o interesse das administrações públicas por Software Livre está crescendo constantemente. A União Europeia apoia o Software Livre e padrões abertos com diversas políticas e projetos.

ISA²

O programa ISA² (Soluções de Interoperabilidade para Administrações Públicas, Empresas e Cidadãos) é dirigido pela Comissão Europeia para apoiar o desenvolvimento de soluções digitais para administrações públicas, empresas e indivíduos. Estes serviços vão desde o compartilhamento de dados até financiamento de serviços públicos transfronteiriços e intersetoriais. O ISA² também promove o Concurso de Premiação para Compartilhamento e Reúso, que fomenta a consciência sobre as vantagens em compartilhar e reutilizar soluções de TI. Ele também evidencia as organizações do setor público que tem se beneficiado com isso. Várias organizações públicas ao redor do mundo possuem tarefas e atividades semelhantes, o que as tornam propensas a reutilizarem mais soluções desenvolvidas. Em 2017 o concurso distribuiu prêmios num total de EUR 100.000 para administrações públicas dentro e ao redor da Europa.

> <https://ec.europa.eu/isa2>

JOINUP

O Joinup é uma plataforma *on-line* de compartilhamento de conteúdos criada pela Comissão Europeia e financiada pela União Europeia no âmbito do programa ISA². Esta plataforma não apenas apoia o governo eletrônico em geral, mas também serve como uma comunidade para troca de informações e experiências, bem como para aumentar a reutilização de software nas administrações públicas. O Joinup fornece um meio para encontrar softwares disponibilizados por outros, para resolver problemas relacionados ao desenvolvimento e para compartilhar soluções autorais. Em termos de interoperabilidade, ele também aumenta a conscientização sobre os melhores projetos de Software Livre na Europa, e sobre eventos e desenvolvimentos relacionados a FOSS (Software Livre e de Código Aberto, tradução da sigla em inglês).

> <https://joinup.ec.europa.eu>

EU-FOSSA 2

O EU-FOSSA 2 (Comunidade de Auditoria de Software de Código Aberto e Livre da UE) é um projeto lançado pela Comissão Europeia para as instituições da UE com o objetivo de garantir a segurança e a integridade dos softwares críticos amplamente utilizados. A iniciativa identifica vulnerabilidades de segurança e corrige erros através da colaboração com a comunidade de Software Livre. Isso inclui conferências de desenvolvedores dentre outros eventos. Como parte deste projeto, recompensas especiais são oferecidas para descobertas de brechas e vulnerabilidades em software, com prêmios no valor total de 2,6 milhões de euros. Parte da colaboração na comunidade tem por objetivo atrair técnicas inovadoras para aprimorar a segurança dos softwares e descobrir as ferramentas necessárias para isso. Como monitorar e melhorar a segurança é uma questão importante, a FOSSA pode se tornar uma atividade permanente necessária.

> <https://joinup.ec.europa.eu/collection/eu-fossa-2>

FREEWAT

Financiado pela UE, o projeto FREEWAT (Ferramentas de software livre e de código aberto para gestão de recursos hídricos, da tradução livre do inglês) faz parte do programa de pesquisa Horizon 2020. Esta plataforma em Software Livre foi desenvolvida com o objetivo de monitorar a quantidade e a qualidade dos recursos hídricos. O software combina diferentes módulos integrados e ferramentas para lidar com questões de gerenciamento de água. A natureza aberta da solução permite que qualquer pessoa interessada possa contribuir com a evolução da plataforma. Exemplos do projeto foram desenvolvidos em 10 estados membros da UE, como França, Romênia e Grécia, e em países não-UE, como Suíça, Ucrânia e Turquia.

> <http://www.freewat.eu>

DECODE

Decode é outro projeto financiado pelo programa de pesquisa Horizon 2020 da União Europeia. Nele participa um consórcio de 14 parceiros europeus, incluindo membros da Espanha, Holanda, Itália, Suécia, França e Reino Unido. O Decode desenvolve ferramentas práticas para gerenciar a coleta e o armazenamento de dados on-line e visa criar uma plataforma descentralizada, garantindo a segurança e a privacidade dos dados dos cidadãos. Os módulos do Decode possuem uma arquitetura distribuída e aberta, permitindo que as pessoas assumam o controle de seus dados pessoais, incluindo as permissões de acesso a informações privadas. Projetos-pilotos estão sendo realizados em Amsterdã e Barcelona entre 2018 e 2019, e todos os residentes elegíveis podem participar.

> <https://decodeproject.eu>

Horizon 2020

O Horizon 2020 é o programa de investigação e inovação da UE para 2014 a 2020, com cerca de 80 bilhões de euros de financiamento disponíveis ao longo de 7 anos. Embora proporcione algum benefício, este programa oferece apoio limitado a Software Livre e padrões abertos.

Reprogramando a Lei de Licitação

Por que o Estado deveria financiar uma ampla gama de projetos de software que fornecem serviços semelhantes, uma vez que é mais eficiente se concentrar em um único projeto e depois compartilhar os custos e o código entre instituições?



Dinheiro Público
Código Público

*Com a iniciativa
Dinheiro Público
Código Público, a FSFE
ajuda as
administrações
públicas a tornarem o
Software Livre como o
padrão para softwares
financiados com
recursos públicos. Mais
de 19.000 indivíduos e
mais de 150 ONGs
apoiam a carta aberta
em publiccode.eu
solicitando aos seus
governos que utilizem
licenças de Software
Livre por padrão.*

A maioria dos estados membros da UE publicou guias para o uso de licenças de Software Livre na administração pública para incentivar a colaboração e a reutilização de software. A estratégia de governo eletrônico da Polônia, publicada em 2016, recomenda que o software financiado pelo governo utilize uma arquitetura aberta e considere a publicação do código sob uma licença de Software Livre. Em seu Plano Digital de 2017, o governo austriaco encorajou o uso de Software Livre. Guias oficiais focados na promoção dessas metas atualmente estão disponíveis em todos os países europeus.

Alguns governos inclusive iniciaram medidas legislativas. A diretriz italiana de 2004 para licitações de software afirmou que, nas aquisições de softwares, as administrações públicas devem incluir considerações sobre Software Livre. Além disso, as instituições devem avaliar as ofertas de cada software de acordo com sua transferibilidade, interoperabilidade, dependência de fornecedor e acessibilidade do código-fonte para fins de verificações imparciais de segurança. Em 2016, o parlamento búlgaro aprovou a Lei de Governança Eletrônica, exigindo que todo software criado para o governo fosse publicado sob uma licença de Software Livre e que fosse desenvolvido em um repositório público. Em 2016, a câmara baixa do parlamento holandês aprovou uma lei tornando obrigatório o uso de padrões abertos para as administrações públicas. Alguns países inclusive estão estabelecendo marcos para os próximos anos. Claro, todas essas leis permitem exceções. No entanto, elas indicam que os tempos estão mudando. As Licenças de Software Livre podem um dia se tornar a regra na administração pública. Em novembro de 2016, o governo húngaro estabeleceu uma meta para reduzir o uso de software proprietário no governo eletrônico em 60% até 2020.

O principal estímulo para essa legislação é um número cada vez maior de experiências positivas em nível local. O compartilhamento e reutilização de código de software tornou-se comum em alguns municípios de pequeno e médio porte e até mesmo em cidades maiores. Em 2015, a administração municipal de Helsinque adotou uma nova estratégia de TI enfatizando a preferência pelo Software Livre, especialmente ao desenvolver novas soluções de software. A cidade de Barcelona anunciou em 2017 que as licenças de Software Livre deveriam se tornar padrão para software financiado com recursos públicos.

A ajuda adicional vem da União Europeia. Em 6 de outubro de 2017, 32 países da UE e do Acordo Europeu de Livre Comércio (EFTA) assinaram a Declaração de Tallinn sobre governo eletrônico. Através dessa declaração, os ministros da UE solicitam à Comissão Europeia para fortalecer a utilização de soluções de Software Livre e padrões abertos – principalmente quando as soluções de software desenvolvidas forem financiadas pela UE. Significativamente, essa decisão não se baseou apenas em argumentos econômicos. A declaração de Tallinn visa principalmente estimular o desenvolvimento de governos digitais centrados no usuário que respeitem os direitos e liberdades dos seus cidadãos, como a liberdade de expressão e a privacidade e o direito à proteção de dados pessoais. Praticamente, as licenças de Software Livre são uma solução perfeita para essas preocupações.

A FSFE mantém um compêndio abrangente de legislações referentes a Software Livre:
<https://fsfe.org/fs-policies>

Como licitar Software Livre

Muitos governos da UE possuem políticas vigentes para promover o uso de Software Livre no governo, em alguns casos desde os anos 2000. No entanto, a adoção de Software Livre na administração pública até agora tem sido limitada. Uma razão para isso é que a maioria das licitações públicas para soluções de software não são adequadas a Softwares Livres. Aqui estão as quatro dicas mais importantes para tornar a sua licitação propícia ao Software Livre:

1.

Licite soluções, não licenças

Escreva suas licitações de forma neutra em relação a diferentes tecnologias e formas de entrega. Se você restringir para um produto de software comercial específico, o Software Livre estará fora do escopo. Se sua proposta exigir licenças de software, o Software Livre será excluído, pois suas licenças não poderão ser compradas. Por outro lado, incluindo serviços como customização ou suporte ao cliente qualifica a licitação para o campo de atuação das empresas de Software Livre. Isso também permite uma comparação competitiva com base no custo total de propriedade. Com o Software Livre, todos os custos concentram-se nos serviços, enquanto o software proprietário geralmente inclui serviços básicos no preço da licença, distorcendo a comparação.

2.

Familiarize-se com as ramificações legais

Na maioria dos estados membros da UE, o Software Livre não se encaixa facilmente nas regras de licitação existentes. Você adquire produtos (licenças de software) ou serviços (personalização e procedimentos separados), que são procedimentos geralmente separados? Como você pode licitar algo que é livre? Muitos países fornecem guias especiais ou pelo menos orientações legais sobre como abrir licitações públicas para Software Livre. Consulte estes documentos e / ou fale com associações de Software Livre ou outras organizações públicas em seu país que tem licitado Software Livre com sucesso para entender como preparar propostas compatíveis com a legislação.

3.

Diminua os requisitos da licitação

Participar de uma licitação pública geralmente exige que você realize uma série de etapas além de apenas descrever seu produto e seu preço. Alguns países trabalham com listas de fornecedores para as quais as empresas devem se registrar antes que possam participar em licitações. Em outros casos, as empresas interessadas devem preencher amplos questionários de pré-qualificação ou fornecer vários certificados sobre padrões de segurança, de fornecimento ou de produção. O Software Livre geralmente é fornecido por pequenas e médias empresas. Portanto, reduzir tais requisitos ao mínimo e auxiliar na conformidade pode reduzir as barreiras para a participação de empresas de Software Livre em licitações.

4.

Inclua os pontos fortes do Software Livre em sua proposta

Dê ênfase na licitação aos principais pontos fortes do Software Livre: interoperabilidade, independência estratégica de fornecedores exclusivos de software, evitando situações de aprisionamento. Além disso, as licenças e serviços de Software Livre são benéficos para customização e autodesenvolvimento. Incluir tais aspectos na convocação de licitações permite que o Software Livre desempenhe seus pontos fortes.

Basanta E. P. Thapa



Basanta E. P. Thapa trabalha com digitalização no setor público no Competence Centre for Public IT (ÖFIT) situado no Fraunhofer Institute for Open Communication Systems e está cursando doutorado na DFG Research

Training Group ‘Wicked Problems, Contested Administrations’ da Universidade de Potsdam. Ele estudou pesquisa em administração pública, ciência política e economia em Münster e Potsdam, e trabalhou como pesquisador na Hertie School of Governance, no European Research Centre for Information Systems e na Technical University Tallinn.

Primeiros passos para Apoiar o Software Livre

Quando o ponto é como modernizar sua infraestrutura de TI, até mesmo pequenos passos podem ter um grande impacto. A rede de especialistas da FSFE coletou dicas preciosas para decisões em política e administração pública que querem fortalecer o papel do Software Livre em suas organizações.



Organizacional

- > Utilize padrões e formatos de arquivos que sejam abertos. Isso reduz os custos de migração e o aprisionamento tecnológico.
- > Reduza os obstáculos para que sua equipe publique software sob uma licença livre fornecendo diretrizes claras.
- > Certifique-se que o repositório público de seu projeto em Software Livre esteja atualizado regularmente.



Projetos

- > Caso não tenha experiência, comece com projetos menores.
- > Certifique-se que o código de seu software esteja bem documentado se você quer que outros contribuam, e forneça recursos para documentação.
- > Não invente novas licenças, mas utilize as que são comuns.¹ Isso facilita que outros reutilizem seu código.



Cooperação

- > Verifique primeiro se já existe um projeto de Software Livre que resolve seu problema antes de lançar novos projetos.
- > Beneficie-se do ganho de escala. Procure por aliados que estejam em busca da mesma solução de software antes de iniciar grandes projetos sozinho.
- > Colabore com projetos e comunidades existentes e se beneficie do conhecimento e lições aprendidas que eles detêm.



Educação

- > Faça divulgação quando você publicar algum código para encorajar colaborações.
- > Forneça treinamento sobre o uso de licenças de Software Livre.
- > Peça opinião de especialistas da comunidade de Software Livre e se beneficie de seus talentos.



Jurídico

- > Incentive o uso de licenças de Software Livre criando quotas ou fornecendo subsídios.
- > Altere o processo de licitação tornando a licença de Software Livre como a opção padrão.
- > Exija que as administrações públicas justifiquem formalmente a compra de software proprietário quando houver alternativa em Software Livre.

¹ Lista de licenças comuns de Software Livre: <https://www.gnu.org/licenses/license-list.html>



A Free Software Foundation Europe (FSFE) é uma instituição de caridade que empodera usuários a controlarem a tecnologia. O software está profundamente envolvido em todos os aspectos de nossas vidas; e é importante que esta tecnologia nos fortaleça em vez de nos restringir. O Software Livre dá a todos os direitos de usar, entender, adaptar e compartilhar software. Estes direitos ajudam a apoiar outras liberdades fundamentais, como liberdade de expressão, de imprensa e a privacidade.

A FSFE foi fundada em 2001 como uma organização não governamental sem fins lucrativos e como uma rede parte de uma comunidade global de pessoas com valores e visões comuns. A FSFE é apoiada por seus membros de toda a Europa e possui capítulos em onze países. O componente central do trabalho da FSFE é garantir a base legal, política e social para um Software Livre forte, seguro e livre de interesses privados.



Em 2017, a FSFE iniciou a iniciativa Dinheiro Público Código Público para tornar licenças de Software Livre o padrão para software desenvolvidos com recursos públicos. Uma carta aberta publicada no outono de 2017 foi assinada por mais de 170 organizações e mais de 19.000 pessoas. A campanha chamou atenção para a importância de decisões sobre licenças e ajuda a administração pública a ter controle total sobre sua infraestrutura digital a fim de criar sistemas confiáveis.

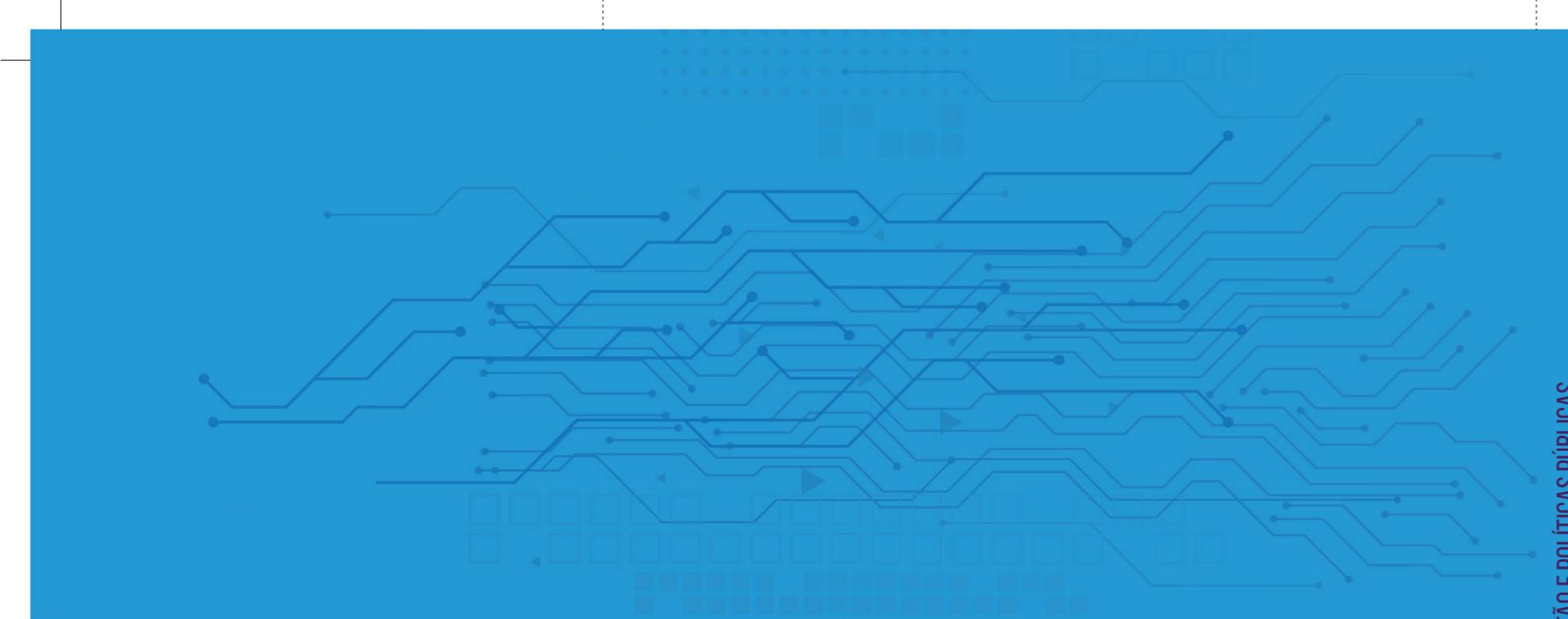
Apoie o trabalho da FSFE através de uma doação:

<https://fsfe.org/donate>



Dinheiro Público

Código Público



SOFTWARE E CULTURA NO BRASIL

PRODUÇÃO, GESTÃO E POLÍTICAS PÚBLICAS

SOFTWARE E CULTURA NO BRASIL: PRODUÇÃO, GESTÃO E POLÍTICAS PÚBLICAS

“Software e Cultura no Brasil: produção, gestão e políticas públicas” é resultado das atividades desenvolvidas por meio da parceria que envolveu pesquisadores do Laboratório de Tecnologias Livres da Universidade Federal do ABC (LabLivre/UFABC), servidores do antigo Ministério da Cultura (atual Secretaria Especial da Cultura do Ministério da Cidadania) e da comunidade de desenvolvedores de software livre no Brasil. A obra reúne importantes resultados alcançados por meio das pesquisas realizadas pelo LabLivre entre 2016 e 2018, e espera contribuir em dois campos: na consolidação dos estudos sobre a importância dos softwares na produção e gestão de cultura e de políticas culturais e, no fortalecimento do formato de parcerias de pesquisa entre governos, universidades e sociedade civil na promoção de inovação, cidadania e do uso de tecnologias livres voltadas para a democratização da gestão pública e da cultura.



SECRETARIA ESPECIAL DA
CULTURA
MINISTÉRIO DA
CIDADANIA

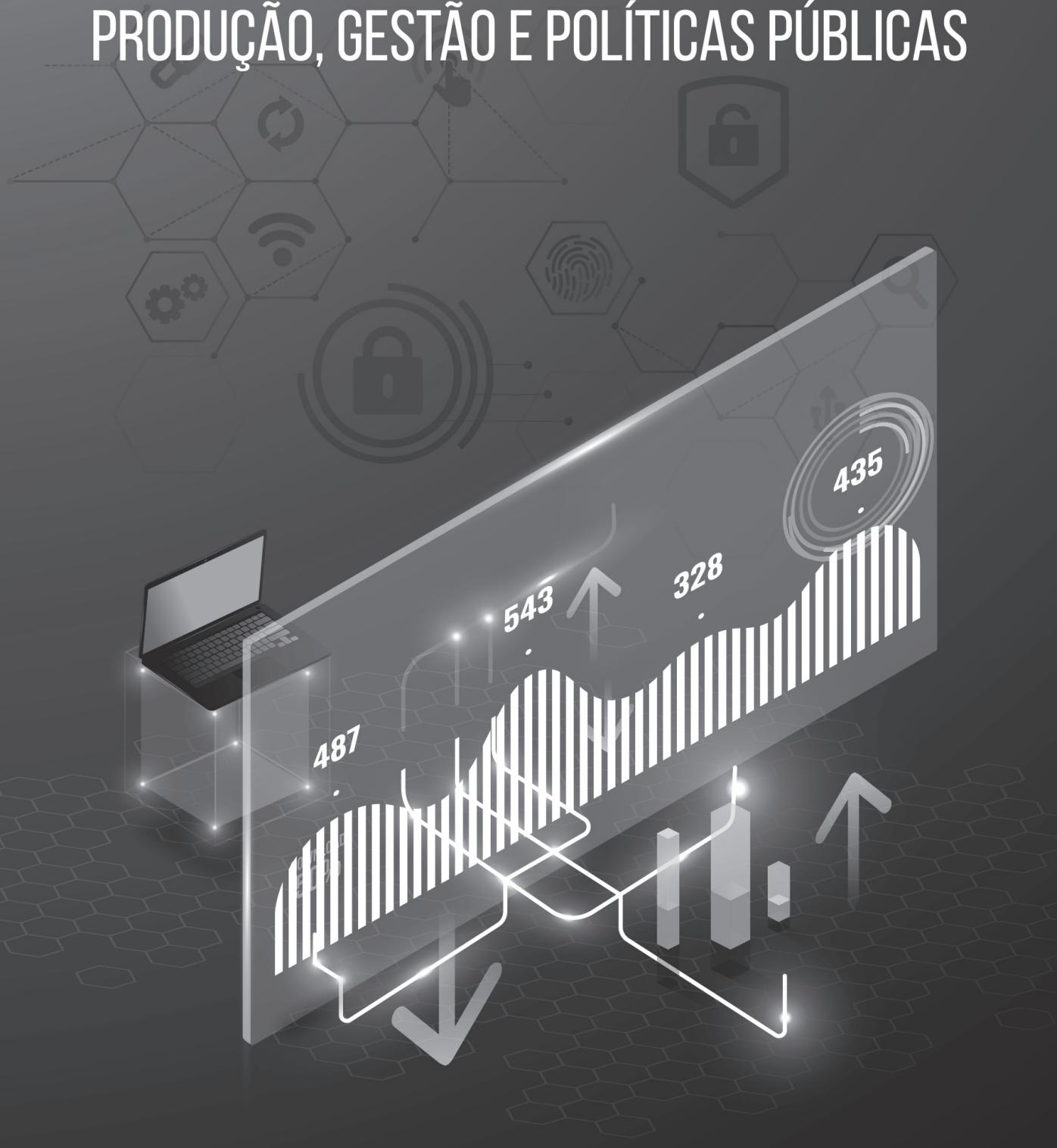


LABLIVRE
UFABC



SOFTWARE E CULTURA NO BRASIL

PRODUÇÃO, GESTÃO E POLÍTICAS PÚBLICAS





ORGANIZADORES

Jerônimo Pellegrini
Cláudio Penteado
Paulo Souza
Luana Homma

SOFTWARE E CULTURA NO BRASIL PRODUÇÃO, GESTÃO E POLÍTICAS PÚBLICAS

APOIADORES



SECRETARIA ESPECIAL DA
CULTURA
MINISTÉRIO DA
CIDADANIA



São Bernardo do Campo
2019

Organizadores

Jerônimo Pellegrini
Cláudio Penteado
Paulo Souza
Luana Homma

Projeto gráfico, capa e diagramação

Kleber de Andrade

Impressão

Kma Soluções Gráficas

Revisão

Thiene Pelosi Cassiavillani

S681

Software e cultura no Brasil: produção, gestão e políticas públicas /
Organizadores: Jerônimo Pellegrini ... [et al.] . - São Bernardo do Campo, SP:
KMA: 2019.

204p. -

ISBN: 978-85-92728-11-3

Outros autores: Cláudio Penteado, Paulo Souza, Luana Homma.

Apoiadores: Secretaria Especial da Cultura (Ministério da Cidadania),
Wikilab e LabLivre da UFABC

1. Software – Tecnologia da informação 2. Sistema de recuperação da
informação 3. Cultura e tecnologia 4. Tecnologia – Aspectos sociais
I. Penteado, Cláudio II. Souza, Paulo III. Homma, Luana IV. Título

CDD: 303.483

CDU: 004.05

SUMÁRIO

Software é cultura!	07
Equipe LabLivre / UFABC	
O que está por trás da criação do LabLivre? Softwares culturais na gestão pública sob a ótica da propriedade intelectual	30
Daniel Astone	
O uso dos softwares pelos agentes de cultura no Brasil	50
Claudio Luis de Camargo Penteado Paulo Roberto Souza Luana Hanaê Gabriel Homma Giuliana Fiacadori Marcus Vinícius da Cunha Casasco Ligia Machiavelli de Lima	
Políticas públicas de cultura no Brasil: revisão e análise da produção acadêmica de 2010 a 2017	76
Lúcio Nagib Bittencourt Paulo Roberto Souza	
Cultura, TICs e gestão pública: percepções e perspectivas dos gestores de estados e municípios brasileiros	104
Luana Hanaê Gabriel Homma Lucca Amaral Tori Jana Tiemi Gabriel Homma	
Tecnologias digitais: necessidades e dificuldades na implementação nas políticas públicas culturais	128
Debora Machado Joyce Souza	
Política cultural e tecnológica nos governos locais: uma análise da presença online de municípios brasileiros	141
Sergio Amadeu da Silveira Murilo Bansi Machado	

**Colaboração aberta e sua relação com a contratação
de software na administração pública.....** 161

Carla Rocha
Ricardo Augusto Poppi Martins

Repensando o desenvolvimento de softwares no Estado 182

Jerônimo Cordoni Pellegrini
Murilo Bansi Machado

Colaboração aberta e sua relação com a contratação de software na administração pública

Carla Rocha¹
Ricardo Augusto Poppi Martins²

Introdução

O objetivo deste texto é apresentar sucintamente o fenômeno geral da colaboração aberta e o mais específico da codificação social, que diz respeito à construção de códigos em plataformas de colaboração. A partir daí, pretende-se estabelecer uma relação sobre os efeitos desses fenômenos no setor público e na relação da sociedade com o Estado.

Uma vez delineado esse campo, vamos tratar de como esses novos paradigmas que estão surgindo afetam os mecanismos de contratação de software pelo Estado, tendo como base duas experiências atualmente em voga na administração pública federal, os Termos de Execução Descentralizada com universidades públicas federais (TEDs) e as licitações de Ateliê de Software.

Este texto não busca ser exaustivo na análise nem dos novos paradigmas, nem dos instrumentos mencionados, buscando, na verdade, apontar algumas de suas características gerais e levantar hipóteses para a pesquisa e a reflexão pela administração pública, por pesquisadores e pela sociedade em geral.

O fenômeno da colaboração aberta e da codificação social

Em respeito ao estudo da inovação, já faz alguns anos que a literatura tem apontado para o fenômeno da colaboração aberta ou inovação colaborativa (SØRENSEN & TORFING, 2011; FORTE & LAMPE, 2013; MERGEL, 2015). A inovação colaborativa se dá por meio de uma plataforma tecnologicamente mediada com baixas barreiras de entrada e saída; participa

¹ Professora de Engenharia de Software/Universidade de Brasília, Coordenadora do Laboratório Avançado de Produção, Pesquisa e Inovação em Software - Lappis.

² Mestrando no programa de pós-graduação em Ciência Política/Universidade de Brasília e Pesquisador Sênior do Laboratório Avançado de Produção, Pesquisa e Inovação em Software - Lappis.

quem quer, quando quer, como quiser. Seria, também, uma espécie de inovação produzida pela colaboração baseada em rede, diferente daquela produzida pelas hierarquias do setor público ou pela competição do mercado.

Dentro desse fenômeno mais amplo, é possível localizar um tipo específico voltado à construção e inovação em software. Chamado na literatura de “social coding” (ou *codificação social*, numa tradução livre) (MERGEL, 2015), esse tipo de colaboração se manifesta na construção de softwares de código aberto por meio de plataformas de colaboração onde redes de desenvolvedores e usuários colaboram aos milhares. Essas plataformas proporcionam ambientes colaborativos onde os desenvolvedores podem receber relatos organizados dos usuários e outros membros das comunidades de colaboração (as chamadas “issues”) podem se abrir para colaboração externa e colaborar numa miríade de projetos distintos (através dos chamados “pull requests”) e podem estudar o código de qualquer projeto através dos recursos de transparência da própria plataforma, que permite navegar pelas discussões que levaram à produção dos códigos ali presentes.

No que diz respeito ao setor público, ainda que seja incalculável o potencial de colaboração da sociedade nos softwares adotados pelo Estado através das plataformas de codificação social, a literatura demonstra que o principal uso desse fenômeno tem sido o compartilhamento de código entre uma mesma organização estatal ou entre organizações distintas (MERGEL, 2015; CRIADO, 2016). Segundo Criado,

O caso do GitHub³ demonstra o potencial para a colaboração aberta entre servidores públicos dentro de uma plataforma aberta na qual podem interagir com outros profissionais do setor para desenvolver projetos conjuntamente, tanto de software quanto de outro tipo (CRIADO, 2016, p. 264, tradução nossa).

Para que a colaboração aberta de software (“social coding”) seja viável, é fundamental que o acesso ao código seja viável por qualquer pessoa na

³ O Github é a maior plataforma de colaboração aberta em código no mundo. Mantida por uma empresa norte-americana, a plataforma permite que os desenvolvedores e organizações publiquem seus códigos e colaborem nos códigos publicados pelos outros desenvolvedores. Embora seja uma plataforma de compartilhamento e colaboração em código, ela possui funções típicas de mídias sociais, como curtidas e menções, facilitando a gestão colaborativa das propostas de melhorias e de tarefas. Link de acesso: <https://github.com/>

internet. Uma das principais definições da colaboração aberta diz respeito às baixas barreiras para participar (ou deixar de participar). Se o acesso ao código for dificultado, a colaboração não floresce. Desde algumas décadas atrás, a partir do desenvolvimento dos instrumentos de licenciamento de software, o código-fonte tem sido tratado como propriedade intelectual, um ativo comercial a se proteger. Esse cercamento moderno do conhecimento embutido nos códigos-fonte dos softwares teve uma resposta, na década de 80, pelo movimento do software livre, que entende que a liberdade de modificar e estudar o código do software que se utiliza deve ser inalienável. Para isso, esse movimento formulou uma lista de 4 liberdades que nenhum ser humano pode ser privado no contato com qualquer software, chamadas de “as 4 liberdades básicas do software livre” (executar, estudar, distribuir e modificar)⁴. Para garantir isso, esse movimento utilizou o mecanismo da propriedade intelectual para criar um tipo de licença de software que exige a distribuição do código-fonte, incluindo qualquer modificação feita no software por qualquer pessoa, mesmo que posteriormente. A licença criada pelo movimento software livre é a GPL (“General Public Licence”), que está, atualmente, na sua terceira versão (AGPLv3).

Atualmente, a disponibilidade de licenças que viabilizam a colaboração aberta em software é maior, graças a movimentos dissidentes que surgiram a partir do movimento software livre, como a turma do “open source”, ampliando a possibilidade de abertura de código. As licenças ligadas ao movimento “open source” desobrigam os futuros contribuidores a licenciar suas modificações em formato aberto, atraindo outros modelos de negócio para o ecossistema da colaboração aberta em software. Kon et al (2012) propuseram uma tipologia que organiza essa grande variedade de licenças em três tipos: (1) Licenças permissivas – que são aquelas onde “não é feita nenhuma restrição ao licenciamento de trabalhos derivados, podendo estes, inclusive, ser distribuídos sob uma licença restrita” (Kon et al, 2012) ; (2) Licenças recíprocas totais – que “determinam que qualquer trabalho derivado precisa ser distribuído sob os mesmos termos da licença original” (Kon et al, 2012) ; e, finalmente, as (3) Recíprocas parciais – que, embora exijam os mesmos termos para os trabalhos derivados, “quando o trabalho é utilizado apenas como um componente de outro projeto, esse projeto não precisa estar sob a mesma licença” (Kon et al, 2012).

⁴ (1) A liberdade de executar o programa, para qualquer propósito; (2) A liberdade de estudar o programa, e adaptá-lo para as suas necessidades; (3) A liberdade de redistribuir cópias do programa de modo que você possa ajudar ao seu próximo; (4) A liberdade de modificar (aperfeiçoar) o programa e distribuir estas modificações, de modo que toda a comunidade se beneficie.

O nosso objetivo aqui não é analisar as vantagens e desvantagens de cada abordagem, mas sim apontar que todas permitem que o fenômeno da colaboração aberta em software aconteça. Para as licenças ligadas ao movimento do software livre, a obrigação de manter abertos os códigos vem do próprio instrumento da propriedade intelectual (o texto da própria licença), enquanto para as ligadas ao movimento “open source”, a manutenção da abertura não é uma obrigação, mas algo que é incentivado pelo nível de atividade do próprio projeto. Quando um projeto de software “open source” é muito ativo, o custo em não publicar uma modificação está em se distanciar da base de código principal do projeto (árvore principal) e perder as próprias modificações no longo prazo, já que sempre será muito grande o incentivo para obter as outras atualizações feitas pelos demais desenvolvedores do projeto. Para fins de definição, sempre que nesse texto nos referirmos ao código construído nas plataformas de codificação social usaremos o termo FLOSS, que é uma junção das iniciais de “free”, “libre” e de “*open source software*”⁵. O termo FLOSS será utilizado aqui sempre para a referência ao software livre ou ao software de código aberto (“open source”).

Efeitos no setor público e na relação Sociedade/Estado

Para além dos efeitos na eficiência estatal pelo compartilhamento de códigos entre servidores e órgãos, a literatura de governança digital aponta para um ganho de participação e co-construção de políticas por meio das plataformas de colaboração aberta. Vaz (2017) alerta que existe um processo global de superação do modo “broadcasting” da governança eletrônica, que adquire um caráter cada vez mais relacional. Segundo o autor,

O quadro de transformações tecnológicas e a sua interação com os processos sociais permitem que se considere a emergência de uma segunda geração da governança eletrônica. Supera-se o modo broadcasting de governança eletrônica: quebra-se o monopólio do Estado sobre as decisões e iniciativas de transparéncia e participação nas políticas públicas. Surgem outras formas de promover a participação, a transparéncia e o controle social das políticas públicas. As práticas de desenvolvimento compar-

⁵ Para uma explicação (em inglês) do conceito FLOSS ver: <https://www.gnu.org/philosophy/floss-and-foss.html>

tilhado e os dados governamentais abertos permitem a coprodução e a produção descentralizadas de aplicações e serviços de base tecnológica [...]. Isso significa que podem emergir, desvinculadamente dos governos, formas de participação e intervenção nas decisões das políticas públicas baseadas na tecnologia (VAZ, 2017, p. 91).

A visão aguçada sobre o potencial da colaboração aberta na construção de softwares no setor público já se manifesta há alguns anos nas experiências brasileiras. Uma das iniciativas precursoras para o aproveitamento desse potencial foi a do Portal do Software Público Brasileiro (SPB), lançado em 2007, que, em apenas três anos de vida, já “disponibilizava à sociedade trinta e seis soluções – ou softwares públicos. Tais soluções eram utilizadas por 66 mil usuários cadastrados à época” (FREITAS & MEFFE, 2010, p. 535). O portal era utilizado por vários atores diferentes, como pessoas físicas, empresas e órgãos das diversas esferas de governo, que cooperavam numa mesma lógica de “disponibilização de bens e serviços públicos em uma rede virtual interorganizacional de produção e difusão de conhecimento tecnológico” (FREITAS, 2012, p. 111). Pelo portal, que ainda segue ativo, qualquer órgão público poderia aderir a qualquer uma das soluções disponibilizadas, se beneficiando dos seus arranjos de desenvolvimento, fossem eles realizados pelos próprios órgãos, ou abrindo procedimentos licitatórios para que as empresas envolvidas no ecossistema (o portal chamava de “comunidades”) pudessem prestar serviços relacionados ao software adotado.

A iniciativa do Portal do Software Público Brasileiro atuou em consonância com a recomendação de Sørensen & Torfing (2011) quando dizem que “o papel dos gestores públicos não é o de produzir a inovação pública, mas o de criar, institucionalizar e gerir arenas abertas e flexíveis para a interação colaborativa com outros atores relevantes ou afetados” (p. 857). Por outro lado, o SPB sempre enfrentou dificuldades para se manter como uma iniciativa vibrante. A baixa capacidade de produzir impacto na administração pública pode estar relacionada com sua excessiva institucionalização, o que acabou criando uma barreira burocrática para a adesão e participação no portal. Isso teria causado o efeito de distanciar as práticas do portal das práticas mais comuns das comunidades de FLOSS nas demais plataformas de colaboração aberta, como GitHub etc. Segundo O’Maley (2013), esse diagnóstico é confirmado por um grande número de desenvolvedores brasileiros de FLOSS “que acreditam faltar à iniciativa tanto uma comunidade de programadores vibrante e inovadora quanto a oferta de software de ponta” (p. 5).

A adoção de softwares pelo Estado é relevante porque impacta diretamente a oferta dos serviços públicos aos cidadãos. A estimativa é que existam, atualmente, aproximadamente 2 mil serviços⁶ públicos só na administração pública federal. Diante do fato de que esses serviços estão em processo de digitalização, diversos tipos de softwares estão sendo adotados para viabilizar a sua oferta e, consequentemente, dar suporte às políticas públicas. Segundo Germani (2016),

Serviços digitais se concretizam na forma de software que, acessíveis a partir de dispositivos conectados à rede, podem atender milhares – ou milhões – de pessoas simultaneamente. Softwares, por sua vez, são conjuntos de instruções, escritos em forma de texto, que indicam para o computador o que ele deve fazer. Logo, serviços digitais são, também, produtos de produção intelectual – e, portanto, conhecimento – assim como textos, leis, partituras etc. Este entendimento é importante, pois, ao analisar as novas práticas e metodologias de desenvolvimento de serviços digitais, analisaremos, sobretudo, sob a perspectiva das mudanças estruturais trazidas pelos meios digitais de comunicação, em especial a Internet, sobre a forma como a humanidade produz conhecimento (p. 5).

Entre os exemplos de softwares que suportam os serviços públicos digitais está o Sistema de Apoio às Leis de Incentivo à Cultura (SALIC)⁷. Esse sistema suporta a política nacional de incentivo à cultura, amparada pela popular Lei Rouanet. A maneira com que os diversos atores do setor cultural têm acesso a essa política é através desse serviço digital. Dessa forma, uma importante política pública no Brasil tem nesse serviço digital um de seus principais componentes. Muitas das decisões ligadas ao desenvolvimento dos softwares que viabilizam a oferta de um serviço desse tipo passam por decisões políticas. Decisões sobre essas tecnologias acabam, portanto, refletindo ou condicionando as decisões sobre as políticas públicas que esses serviços suportam.

⁶ Reportagem: <http://www.convergenciadigital.com.br/cgi/cgilua.exe/sys/start.htm?UserActiveTemplate=site&infoid=46522&csid=154>

⁷ Link de acesso ao SALIC, mantido pelo Ministério da Cultura: <http://salic.cultura.gov.br>

Arranjo de comunidade

O politólogo Steven Weber (2004) explorou o tema das comunidades de desenvolvimento colaborativo de software de código aberto. Segundo Weber (2004), o processo do código aberto é uma organização política que opera a gestão de conflitos, poder, interesses, regras, normas comportamentais, procedimentos de tomada de decisão e mecanismos de sanção, porém, de uma maneira diferente da lógica da economia política da era industrial (*ibidem*, p. 3). Não é um processo “caótico e despregrado no qual todos têm igual poder e influência” (*ibidem*, p. 3), nem uma comunidade idílica de amigos com a mesma opinião, onde reinam o consenso e os acordos fáceis. De fato, o conflito é inerente ao processo do FLOSS (*ibidem*, p. 3). Ainda segundo Weber, o processo do FLOSS demonstra a “viabilidade de um sistema de inovação massivamente distribuído que estende as fronteiras das noções convencionais sobre os limites da divisão de trabalho” (*ibidem*, p. 14). Essas características ficam claras quando o autor diz que as comunidades de código aberto “descrevem uma estrutura nascente de cultura e comunidade, [incluindo critérios] para entrar (e sair), papéis de liderança, relações de poder, questões distributivistas, formas de educação e socialização” (*ibidem*, p. 15).

O diagnóstico político de Weber é corroborado por estudos dentro da Computação e Engenharia de Software a respeito do desenvolvimento colaborativo. Esses estudos analisam a atuação dos desenvolvedores nas plataformas de colaboração aberta. A partir da análise em ambientes de codificação social, Dabbish & Stuart (2012) demonstram que os mecanismos de transparência presentes na plataforma do GitHub viabilizam o trabalho cooperativo, diminuindo a necessidade de encontros sincronizados. Segundo as autoras, seus achados trazem informações relevantes sobre o “desenho desse tipo de mídia social para colaboração em larga escala e implicam uma variedade de maneiras pelas quais a transparência pode apoiar inovação, compartilhamento de conhecimento e construção de comunidade” (DABBISH & STUART, 2012, p. 1286). McDonald & Goggins (2013) também perceberam o valor da transparência e das ferramentas sociais de colaboração no dia a dia dos desenvolvedores que operam nesses ambientes. Segundo alguns resultados de suas pesquisas, também com base no GitHub, os desenvolvedores associam a experiência de uso da plataforma com o desenvolvimento de práticas mais democráticas e transparentes de trabalho. O ambiente de discussão em torno das contribuições permite que haja um diálogo público atrelado ao processo decisório sobre novas incorporações de código, permitindo que quem

proponha a alteração tenha acesso e exponha todos os argumentos e todos os interessados naquela mudança possam se manifestar (MCDONALD & GOGGINS, 2013, p. 142). O uso do mecanismo das menções (@usuário), típico dos ambientes de mídias sociais, também é utilizado como instrumento de trabalho para aumentar a agilidade na resolução dos problemas e melhorar a comunicação entre os desenvolvedores e usuários (ZHANG, WANG & YIN, 2015, p. 4). O histórico público desse processo (transparência) também permite que ele seja posteriormente conhecido por qualquer desenvolvedor ou usuário da comunidade.

O Estado e as comunidades

Podemos dizer que um arranjo FLOSS é uma instituição social regida pela inovação distribuída, ágil, transparente e com altos níveis de qualidade técnica, que já tem funcionado com a participação de agentes estatais. O livre acesso aos códigos e à documentação da comunidade permite que diferentes atores possam investir em desenvolvimento de forma complementar e articulada com as outras partes, potencializando a entrega de resultados e, a partir da transparência, aumentando a confiança entre os membros do arranjo. Esse tipo de arranjo também fortalece a autonomia da sociedade na construção de tecnologias que dão suporte às políticas públicas ao mesmo tempo que preserva a soberania estatal na execução das políticas, eliminando a dependência das instituições estatais de um único agente privado. Porém, efetivar a adesão do Estado a esses arranjos não é uma operação simples, exigindo outros tipos de capacidades de gestão, além de instrumentos de contratação e financiamento de desenvolvimento de softwares que possam ser operados nesses novos paradigmas.

Quando se trata de instituições estatais, o principal apelo deve estar relacionado ao seu caráter público e responsável à participação da cidadania. Segundo Germani (2016), os processos de desenvolvimento de software baseados em códigos abertos, para além do licenciamento do código-fonte, promovem a abertura do processo de desenvolvimento para os cidadãos, “que serão futuros usuários do serviço, para participar da sua construção, e permitindo com que outros atores, de dentro e de fora do governo, possam acompanhar, opinar e colaborar com o processo” (GERMANI, 2016). Segundo Meireles (2015), o FLOSS permite que os “softwares sejam modificados pelas próprias pessoas, com base em suas necessidades funcionais, diferenças culturais, personalidade e estilo” (MEIRELES, 2015). Além disso, ele torna “o desenvolvimento mais

ágil, como também promove ciclos de inovação aberta, ao contar com a participação de comunidades interessadas” (MEIRELES, 2015).

Além desses argumentos, vale mencionar o trabalho realizado pelo Grupo de Trabalho que investigou a possibilidade de migrações para FLOSS na Universidade Federal do ABC (UFABC) (RATCOV, et al., 2018), que produziu um relatório que elenca os seguintes argumentos pró-software livre:

1. Economicidade: possuem custos mais baixos, principalmente quando operados em escala, mesmo considerando os investimentos necessários para a saúde do arranjo de comunidade;
2. Segurança: software auditável é sempre mais seguro;
3. Autonomia tecnológica: permite que cidadãos brasileiros e arranjos locais tenham completo conhecimento dos códigos e tecnologias, afastando dependências externas;
4. Independência de fornecedores: os arranjos de comunidade são diversos, de forma que o órgão público não fica refém de um único fornecedor;
5. Argumento democrático: já explicitado acima, sua construção é naturalmente participativa;
6. Estimula um mercado interno de desenvolvimento: permite que diversos empreendimentos locais se capacitem nas tecnologias devido à baixa barreira de entrada;
7. Promove inclusão digital: diminuindo barreiras de custo e acesso a conhecimento;
8. Softwares livres naturalmente suportam formatos abertos, pois são construídos de forma a atender uma ampla gama de iniciativas e organizações.

Quanto melhor uma tecnologia é desenhada, tendo em vista os objetivos e sujeitos que farão uso dela, mais sucesso e impacto essa tecnologia irá gerar. Pequenos detalhes, muitas vezes, são responsáveis por diferenças expressivas nos resultados. Dessa forma, optar por uma metodologia participativa na construção de tecnologias que suportam os serviços públicos digitais parece ser uma decisão acertada. Além do aumento de eficiência e qualidade na construção dos códigos informáticos, as decisões de desenho tomadas em conjunto

com o público sujeito e interessado levarão à prestação de melhores serviços públicos. Atualmente, esse processo é incentivado por duas das maiores economias mundiais, Estados Unidos⁸ e Reino Unido⁹, além de diversas cidades referência no mundo, como o caso de Barcelona¹⁰. Como já falamos anteriormente, o Brasil também tem uma iniciativa consolidada nessa mesma linha, o portal do Software Público Brasileiro, tendo obtido resultados importantes na última década¹¹.

Refletindo sobre instrumentos de contratação

Para essa reflexão, vamos considerar dois instrumentos pelos quais um órgão estatal pode aderir a um arranjo FLOSS. A ideia de adesão ao arranjo FLOSS é definida aqui como a utilização, por parte de uma instituição estatal, de instrumentos para adotar e financiar o desenvolvimento de infraestruturas e aplicações digitais FLOSS, com vistas a implementar um determinado serviço e política pública.

Os exemplos foram trabalhados considerando dois instrumentos distintos, o TED com universidades públicas federais e o Ateliê de Software via licitação com empresas privadas. Serão analisados alguns aspectos transversais à gestão de Tecnologia da Informação, como Efetividade, Sustentabilidade e Inovação, além da adequação de cada instrumento para operar a adesão do Estado aos arranjos de comunidades FLOSS.

O instrumento do TED

O instrumento do termo de Execução Descentralizada é definido pelo Decreto nº 8.180, de 30 de dezembro de 2013, como “instrumento por meio do qual é ajustada a descentralização de crédito entre órgãos e/ou entidades integrantes dos Orçamentos Fiscal e da Seguridade Social da União, para execução de ações de interesse da unidade orçamentária descentralizadora e consecução do objeto previsto no programa de trabalho, respeitada fielmente a classificação funcional programática”¹². Esse instrumento é utilizado para a

⁸ <https://code.gov/#/>

⁹ <https://www.gov.uk/guidance/be-open-and-use-open-source>

¹⁰ <http://ajuntament.barcelona.cat/digital/es/transformacion-digital/tecnologia-para-un-gobierno-mejor/transformacion-con-metodologia-agile>

¹¹ <https://softwarepublico.gov.br/social/>

¹² http://www.planalto.gov.br/ccivil_03/_ato2011-2014/2013/decreto/D8180.htm

realização de parceria entre órgãos vinculados ao Orçamento Geral da União (OGU), podendo parte do orçamento ser transferido para ser empenhado e executado por outros órgãos, desde que cumprindo a função original para a qual aquele orçamento foi destinado. É um instrumento amplamente utilizado para que órgãos da administração direta possam financiar pesquisa e desenvolvimento nas universidades públicas e assim auxiliar e qualificar a entrega dos objetivos pretendidos pelo órgão.

Alguns órgãos públicos têm usado o instrumento do TED com universidades públicas federais como uma alternativa ao modelo das fábricas de software, que entregam soluções de baixa qualidade a custos altos. A parceria com as universidades públicas permite aos órgãos operar a inovação em tecnologia fora da métrica por ponto de função. Essa métrica não considera o software como um produto de criatividade intelectual que busca solucionar um problema finalístico, buscando quantificar o seu desenvolvimento pelos seus próprios componentes tecnológicos. No modelo da fábrica de software, a empresa tem o incentivo a desenhar uma solução mais difícil de usar, desde que tenha mais componentes pelos quais possa fazer cobranças maiores. No final das contas, a relação entre a necessidade do órgão de contratar software de qualidade ao menor custo possível e a necessidade da empresa de faturar mais com menos trabalho empenhado fica desequilibrada para o lado do órgão, que acaba recebendo um software ruim e pagando caro por isso.

Já o modelo de TED com universidades públicas federais tem na inovação e no software de qualidade uma das premissas da parceria. Isso porque a parceria, além de envolver a entrega da solução de software, contempla, também, a realização de pesquisa e a entrega de outros produtos e atividades ligadas à produção de conhecimento, como formação de novos pesquisadores, publicação de artigos acadêmicos e participação/promoção de seminários. A parceria é planejada através de um plano de trabalho acordado entre as partes que prevê as diversas fases do projeto e um cronograma de desembolso que considere as necessidades reais do projeto em cada etapa. A partir da realização desse planejamento e firmado o instrumento, as partes podem focar a energia no que realmente traz valor para ambas: a pesquisa e desenvolvimento da solução. Nesse modelo, a forma de aferição de resultados é mais flexível por se tratar de dois órgãos federais. A Universidade está sujeita à legislação de licitações e presta contas aos órgãos de controle da mesma forma que o órgão descentralizador, com responsabilidade dividida.

Por outro lado, essa flexibilidade também pode ser uma fragilidade do instrumento. A falta de acompanhamento da parceria pelo órgão descentralizador pode gerar um desequilíbrio em relação aos interesses das partes, fazendo com que a universidade se concentre mais nos resultados acadêmicos de formação e publicação de artigos e menos na entrega da solução para o órgão. Nesse caso, a inovação até seria documentada, mas não seria aplicada no caso concreto, gerando prejuízos para os objetivos que foram estabelecidos para a ação orçamentária que originou a parceria. Alguns gestores têm dúvidas sobre como garantir o cumprimento dos objetivos no contexto de uma parceria em que o órgão não pode impor sanções à universidade. A parceria ficaria muito dependente dos aspectos macros, relacionados à capacidade que a universidade tem de cumprir os objetivos da parceria, mas faltariam instrumentos concretos para garantir qualidade e alinhamento das entregas específicas com as necessidades do órgão.

O arcabouço do processo de desenvolvimento das comunidades FLOSS está disponível para ser aplicado nesse tipo de parceria, elevando a qualidade do trabalho para outros patamares, principalmente o uso de metodologia ágil, comunicação instantânea entre servidores do órgão e pesquisadores da universidade, além das práticas simplificadas de documentação em tempo real (através de ferramenta de wiki). Há relatos de alguns aperfeiçoamentos recentes que foram realizados no acompanhamento desse tipo de instrumento de parceria, como a exigência à universidade de confecção de relatórios trimestrais com indicadores, além do acompanhamento constante dos repositórios de código e demais produtos da parceria (comunicação instantânea e wikis).

Se, do ponto de vista da inovação, são inegáveis as vantagens trazidas pelo instrumento TED com universidades públicas federais, vale fazer a mesma reflexão em relação aos aspectos de sua efetividade e sustentabilidade. Quanto mais consolidado o tipo de infraestrutura de TI, menos esse instrumento pode agregar a ela. Por exemplo, para o órgão manter infraestruturas básicas como link de internet ou impressoras, esse instrumento não é adequado, pois a conexão entre a inovação e a entrega de valor é mais fraca. Para esse tipo de infraestrutura, os contratos são o instrumento mais adequado. Por outro lado, para a entrega de serviços digitais com agilidade e qualidade, como sites e serviços, aos cidadãos, o instrumento do TED passa a fazer mais sentido. O legado deixado pelos TEDs nos órgãos tende a se integrar de forma permanente nas equipes, contribuindo para aumentar a efetividade do órgão na entrega de serviços digitais para as áreas finalísticas e, consequentemente,

para os cidadãos. Siqueira et al (2018) demonstrou como a capacidade de pesquisa e adaptação do arranjo de universidade foi fundamental para o sucesso de uma parceria de dois anos, de 2014 a 2016, entre a Universidade de Brasília e o Ministério do Planejamento. Os autores que atuaram nessa parceria

(...) acreditavam que a entrega constante de software era melhor para o sucesso do projeto. Em oposição a isso, o Ministério do Planejamento estava habituado à ideia de uma única entrega ao final do projeto, e nem sua estrutura burocrática, nem sua experiência técnica estavam alinhadas a esse estilo de trabalho. Isso estava prejudicando os benefícios da ferramenta (desenvolvida) e impedindo o time da UnB de mostrar os frutos do projeto para aqueles responsáveis por avaliarem o trabalho (SIQUEIRA, et al., 2018).

Dessa forma, a capacidade do time do projeto de pesquisar e implementar a técnica da engenharia de software conhecida como entrega contínua¹³ foi decisiva na transformação cultural necessária para o órgão absorver o trabalho que estava sendo realizado no contexto do projeto. Segundo os autores,

(...) a entrega contínua ajudou a fortalecer a confiança entre desenvolvedores e a equipe do ministério. Antes de utilizar a entrega contínua, eles só conseguiam validar as novas funcionalidades desenvolvidas apenas no final do ciclo de release, normalmente a cada quatro meses. Com a implementação da entrega contínua, versões intermediárias ficavam disponíveis permitindo que eles realizassem pequenas validações ao longo do tempo. O monitoramento constante do trabalho de desenvolvimento trouxe maior segurança aos líderes do ministério e melhorou as interações com nossa equipe (SIQUEIRA, et al., 2018).

Em suma, a construção e utilização da entrega contínua no TED descrito pelo artigo foi responsável por estabelecer três novas dinâmicas, fundamentais para o sucesso da parceria e para a garantia da entrega de valor para o órgão: (1) Demonstrar resultados concretos ao invés de apenas reportá-los; (2) Tornar o gerenciamento do projeto transparente e colaborativo para a equipe do ministério; e (3) Ganhar a confiança da equipe do governo (SIQUEIRA et al., 2018).

¹³ A entrega contínua é uma técnica da engenharia de software que permite que atualizações do software sejam rapidamente disponibilizadas para os usuários.

Do ponto de vista da sustentabilidade, as vantagens desse instrumento dependem do cultivo de um ambiente propício no próprio órgão. Via de regra, os servidores dos órgãos de TI se envolvem pouco com as atividades mais centrais das áreas de tecnologia da informação como desenvolvimento e gestão de projetos, que ficam mais a cargo da gestão de contratos. Essa característica da gestão das equipes prejudica o processo de apropriação e sustentabilidade, ficando todo o conhecimento apenas nos fornecedores passageiros. Por outro lado, um ambiente onde os servidores se envolvem nas discussões e decisões sobre os sistemas e soluções de software do órgão é o mais propício para aproveitar os aportes que as parcerias com as universidades públicas podem trazer. O uso do arcabouço de trabalho das comunidades FLOSS já mencionadas aqui, aliado ao comprometimento verdadeiro dos servidores com os códigos e comunidades das soluções adotadas e desenvolvidas pelo órgão, gera um ambiente seguro e duradouro, cuja sustentabilidade das soluções evita a interrupção na oferta de serviços e políticas digitais pelos órgãos estatais.

O instrumento do Ateliê de Software

O instrumento da licitação para ateliê de software usa o mesmo instrumento legal de contratação das atuais fábricas de software (que é a lei de licitações) mas com mudanças substanciais nas exigências e nas métricas utilizadas. A base filosófica do instrumento do ateliê é o manifesto do “kraftsmanship” ou manifesto para a artesania de software¹⁴, que parte da premissa de que a construção de software está mais próxima de um processo de criação artística do que do processo fabril, como é tratado atualmente pelas fábricas. Isso significa que o trabalho de criação de software deve estar orientado a criar valor em suas soluções e valorizar o trabalho das pessoas envolvidas, recompensando esforço e qualidade. Assim como o TED, esse instrumento se viabiliza a partir de uma crítica frontal à metodologia de trabalho e gestão e à métrica mais utilizada hoje nos processos governamentais de contratação de software, a contagem por pontos de função¹⁵. A métrica de pontos de função gera incentivos para que as empresas construam softwares piores e mais complicados para os usuários, mas que tenham remunerações mais atraentes para

¹⁴ <http://manifesto.softwarecraftsmanship.org/#/pt-br>

¹⁵ Análise de Pontos de Função (APF) é uma técnica para a medição de projetos de desenvolvimento de software, visando projetar uma medida de tamanho com base nas funções realizadas pelo software. É a métrica mais utilizada nas licitações para desenvolvimento de software do governo federal, e muito pouco usada fora desse contexto.

os seus negócios. A busca por soluções inovadoras e o esforço na construção de entregas com mais qualidade não são captados pela métrica. Esse incentivo implícito na métrica gera um desequilíbrio entre a qualidade do produto e o uso eficiente do dinheiro público, em favor do lucro das empresas.

Outra mudança que a lógica do ateliê de software promove em relação às fábricas de software é a mudança da metodologia. No conceito fabril, os processos são mais importantes que as pessoas e toda falha é analisada como uma falta de aderência a eles. Na lógica do ateliê as pessoas estão no centro do método, cuja preocupação começa desde as exigências da pirâmide invertida na montagem da equipe (a quantidade de juniores não pode ser maior que a de desenvolvedores plenos que não pode ser maior que a de desenvolvedores seniores) e a obrigatoriedade de manter a equipe presente no órgão. Com uma equipe formada majoritariamente de profissionais experientes e “*in-loco*”, o gestor passa a ter incidência sobre o clima organizacional que será criado e a metodologia de trabalho que será utilizada. A métrica utilizada pelo ateliê é a UST (Unidade de Serviço Técnico) que estima o nível de esforço necessário para produzir os principais componentes e etapas de um processo de desenvolvimento. Essa estimativa é feita a partir da criação de um repertório acordado entre o órgão e a empresa, que passa por revisões periódicas de comum acordo entre as partes. O fato de o repertório poder passar por revisões periódicas permite que ele seja capaz de reequilibrar constantemente a relação, a partir dos requisitos de qualidade exigidos pelo órgão. Parece complexo, mas não é: enquanto o manual de contagem de pontos de função tem aproximadamente 80 páginas, o repertório utilizado pelo Ministério das Relações Exteriores na contagem dos pontos de função tem, no máximo, 5 páginas¹⁶. Essa simplificação também permite que a gestão seja feita com mais qualidade. Um ponto importante para o funcionamento correto da métrica é que nada deve ser pago fora do previsto. Toda entrega é quantificada em UST antes de o trabalho começar e acordada entre as partes. Quando concluído, o valor cobrado deverá ser aquele que foi acordado anteriormente. Qualquer adaptação no repertório só poderá acontecer após a entrega do produto.

Do ponto de vista da inovação, como o repertório de contagem é gerido pelo próprio órgão, ele tem liberdade de remunerar trabalhos que contribuam para a inovação, como estudos e pesquisas. Por outro lado, uma gestão que usa como base o manifesto da artesania de software percebe a inovação

¹⁶ Dado obtido em entrevista realizada no ano de 2018, com técnicos do setor.

como um vetor para melhorar a qualidade do trabalho das pessoas, resolvendo problemas da organização. Não se trata, portanto, de inovar por inovar. Em relação à sustentabilidade, é evidente que o instrumento do ateliê, associado ao uso de metodologia ágil, é fundamental para que os projetos sejam entregues com qualidade e atendam às necessidades dos usuários, contribuindo para a continuidade das soluções. Por fim, em relação à efetividade, o instrumento do ateliê contribui com entregas mais rápidas e melhores, aprimorando a capacidade que a TI tem de atender às necessidades dos usuários e das áreas da organização que dependem dela para desempenhar o seu trabalho.

Adequação dos instrumentos em relação à arranjos de comunidades FLOSS

O diagnóstico dos dois instrumentos (TED e Ateliê de Software) revela que, apesar de suas especificidades, eles estão mais capacitados para lidar com os desafios de Efetividade, Sustentabilidade e Inovação inerentes à TI das organizações públicas do que o modelo atual baseado em fábricas de software regidas pela métrica dos pontos de função. Por outro lado, ambos os arranjos se diferem em relação a seus pontos fortes e suas fraquezas, tornando necessária uma avaliação minuciosa do cenário de TI do órgão antes da decisão pela adoção de um ou outro instrumento. Abaixo, apresentamos uma sistematização básica desses achados em formato de tabela (Tabela 1).

Tabela 1 . Sistematização do diagnóstico realizado nos dois instrumentos

Eixo/Aspecto	Termo de Descentralização com Universidades Públicas Federais	Licitação de Ateliê de Software
Inovação	É premissa do acordo. Com uma boa prospecção de universidades com histórico de pesquisa nas áreas de desafio pretendidas pelo órgão, é uma forma eficiente de utilizar a pesquisa aplicada nos desafios e problemas do órgão. Requer acompanhamento próximo da gestão para que a inovação não seja aplicada apenas nos produtos acadêmicos da parceria.	Depende da gestão de TI do órgão. A flexibilidade dos repertórios permite à gestão remunerar inovação. Isso depende da gestão, pois o incentivo dificilmente virá da empresa. Uma gestão orientada para que as soluções de TI entreguem valor concreto para as áreas finalísticas poderá utilizar bem esse instrumento.

Efetividade	Aumenta a efetividade na oferta de serviços complexos. O legado deixado pelas universidades instrumentaliza o órgão a fazer entregas melhores e mais efetivas. Para isso, é necessário que a gestão tenha sucesso em envolver a equipe permanente (sejam servidores, sejam terceirizados) nas dinâmicas de trabalho ao longo do tempo de execução do TED.	A flexibilidade do instrumento associado a equipes mais experientes trabalhando em metodologia ágil tende a aumentar a efetividade de todas as entregas.
Sustentabilidade	O instrumento ajuda se o ambiente de trabalho compartilhado for criado. O modo de trabalho das comunidades FLOSS ajuda no processo de criação de capacidades na equipe de servidores e terceirizados.	O instrumento ajuda, mas o peso está na gestão. Entregas melhores fazem com que os projetos durem mais. O clima organizacional faz com que o conhecimento se dissemine na equipe. Por outro lado, é papel da gestão proteger a área de TI de projetos inúteis.
Pontos fortes	Não depende, para funcionar, do conhecimento prévio da gestão em relação à metodologia de trabalho.	Tem mecanismos de sanção que podem ser utilizados para melhorar a qualidade dos produtos entregues.
Fragilidades	Sem acompanhamento próximo da gestão, o foco da universidade tende a ser em produtos acadêmicos ou experimentos sem conexão com os problemas da organização.	Depende mais do conhecimento prévio da gestão em relação à metodologia de trabalho e à cultura organizacional.

Para o recorte específico que nos propusemos a desenvolver neste texto, relacionado à capacidade do Estado em aderir a arranjos de colaboração aberta e codificação social no formato de comunidades FLOSS, percebemos que o instrumento do Ateliê de Software possui inúmeras lacunas e fragilidades nesse sentido. Isso se deve ao fato de que esse instrumento depende de uma cultura organizacional pró-software livre nas áreas de TI dos órgãos para que a adesão ao FLOSS aconteça. A licitação de Ateliê de Software não possui nenhum mecanismo incentivador ou facilitador para isso. Regido por um contrato privado, um arranjo desse tipo não garante as vantagens difusas oriundas da adesão à FLOSS, como autonomia tecnológica e desenvolvimento de mercado local, entre outras.

Por outro lado, o instrumento do TED com universidades públicas se mostrou mais adequado. Isso acontece pelo fato de o TED ser operado como um instrumento baseado em uma parceria de pesquisa aplicada e não nos moldes de um contrato cliente/fornecedor. Como a universidade não pode – nem deve – operar no mesmo ritmo do mercado, ela precisa aliar a entrega de valor para o parceiro com a construção de um ambiente saudável de pesquisa e formação. O objetivo institucional da universidade de atuar na formação de pesquisadores e profissionais requer a construção de pontes amplas de colaboração, sob o risco de oferecer um ambiente medíocre e pouco desafiador para os alunos e bolsistas. Essas especificidades fazem com que os laboratórios de universidades, de certa forma, dependam da articulação com o conhecimento disponível nas comunidades para efetuar a entrega de valor ao órgão parceiro. Seria hipoteticamente possível a existência de TEDs para construir soluções fechadas e autorais, porém, com o alto custo da universidade não cumprir o seu papel institucional, afetando a qualidade e a agilidade das entregas em médio e longo prazos. Além disso, soluções fechadas possuem níveis muito mais baixos de transparência se comparadas com a transparência inerente aos arranjos de colaboração aberta, prejudicando os trabalhos de monitoramento e controle da parceria realizados tanto pelos órgãos parceiros quanto pelos órgãos de controle. Explicando melhor, a articulação com as comunidades potencializa a entrega de inovação e valor da solução ao parceiro, ao passo que a transparência das práticas de colaboração aberta mitiga a fragilidade de controle do instrumento que foi detectada no nosso diagnóstico. Em suma, o instrumento do TED acaba sendo afetado por uma força centrípeta na direção do arranjo de comunidade, tanto pelas necessidades intrínsecas da pesquisa aplicada (inovação com entrega de valor e requisitos de transparência) quanto pelo caráter aberto da cultura da boa ciência como um empreendimento coletivo e colaborativo.

Considerações Finais

Passamos por uma visão geral do fenômeno da colaboração aberta, a partir de sua vertente mais específica da codificação social. A codificação social tem sido pesquisada como um fenômeno em crescimento e cada vez mais adotado pelas instituições estatais. As potencialidades da codificação social são viabilizadas pelo modelo de licenciamento dos softwares, geridos como um bem comum. Do software livre ao “open source”, os diversos modelos permitem a realização da codificação social, ainda que com visões de mundo e incentivos diferentes.

De forma geral, os arranjos de comunidade FLOSS baseados nas plataformas de codificação social atuam com altos níveis de transparência, utilizando-se de instrumentos específicos dessas plataformas para fomentar a comunicação e construir práticas de colaboração. Do ponto de vista do Estado, a adoção de softwares produzidos dentro desses paradigmas pode aumentar a permeabilidade e a transparência na construção dos serviços digitais disponibilizados para a população, aprimorando a participação social, além de outros benefícios mais difusos para a sociedade.

Por fim, analisamos dois instrumentos de contratação e investimento público em softwares: o TED com universidades públicas e a licitação de Ateliê de Software com empresas privadas. Ambos os instrumentos têm a vantagem de superar a mediocridade da métrica de pontos por função, aumentando a entrega de valor para os órgãos e a efetividade no gasto do recurso público. Por outro lado, o TED é mais recomendado para contextos de inovação, além de ser mais adequado para a adesão do Estado a arranjos de comunidades FLOSS. O instrumento do TED com as universidades públicas, se bem manejado, traz enormes benefícios relacionados à inovação e à sustentabilidade, além dos demais benefícios difusos trazidos pela adesão a esse tipo de arranjo.

Referências Bibliográficas

CRIADO, J. Ignacio. Las administraciones publicas en la era del gobierno abierto. Gobernanza inteligente para un cambio de paradigma en la gestión pública. **Revista de Estudios Políticos**, n. 173, 2016. Disponível em: <http://www.cepc.gob.es/Publicaciones/Revistas/revistaselectronicas?IDR=3&ID-N=1361&IDA=37798> Acesso em: 25 out. 2018.

DABBISH, Laura; STUART, Colleen; TSAY, Jason; et al. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. In: **Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work**. New York, NY, USA: ACM, 2012, p. 1277–1286. (CSCW’12). Disponível em: <<http://doi.acm.org/10.1145/2145204.2145396>>. Acesso em: 24 out. 2018.

FORTE, Andrea; LAMPE, Cliff, Defining, Understanding, and Supporting Open Collaboration: Lessons From the Literature, **American Behavioral Scientist**, v. 57, n. 5, p. 535–547, 2013.

FREITAS, Christiana Soares de; MEFFE, Corinto. Redes de produção de conhecimento tecnológico: um projeto governamental brasileiro. **Estudos de Sociologia**, v. 15, n. 29, 2010. Disponível em: <<http://piwik.seer.fclar.unesp.br/estudos/article/view/2978>>. Acesso em: 1 nov. 2018.

FREITAS, Christiana Soares de. O Software Público Brasileiro: novos modelos de cooperação econômica entre Estado e Sociedade Civil. **Informação & Sociedade**: Estudos, v. 22, n. 2, 2012. Disponível em: <<http://periodicos.ufpb.br/ojs2/index.php/ies/article/view/12231>>. Acesso em: 1 nov. 2018.

GERMANI, Leonardo Barbosa. **Desafios para o desenvolvimento de serviços digitais pelo governo federal brasileiro**. 2016. Disponível em: tede2.pucsp.br/tede/handle/handle/18772. Acesso em: 1 nov. 2018.

KON, Fabio et al. **Software Livre e Propriedade Intelectual**: Aspectos Jurídicos, Licenças e Modelos de Negócio. <http://ccsl.ime.usp.br/files/slpi.pdf> Acesso em 05/nov/2018, v. 2, p. 12, 2012.

MCDONALD, Nora; GOGGINS, Sean. Performance and Participation in Open Source Software on GitHub. In: **CHI '13 Extended Abstracts on Human Factors in Computing Systems**. New York, NY, USA: ACM, 2013, p. 139–144. (CHI EA '13). Disponível em: <<http://doi.acm.org/10.1145/2468356.2468382>>. Acesso em: 25 out. 2018.

MEIRELES, Adriana Veloso. **Democracia 3.0**: interação entre governo e cidadãos mediada por tecnologias digitais. 2015. Disponível em: <<http://repositorio.unb.br/handle/10482/19044>>. Acesso em: 26 nov. 2017.

MERGEL, Ines. **Open collaboration in the public sector**: The case of social coding on GitHub. *Government Information Quarterly*, v. 32, n. 4, p. 464–472, 2015.

O'MALEY, Daniel. Software Público Brasileiro (SPB): The State in the Commons. In: Workshop Sobre Software Livre. **Anais do WSL2013**. Porto Alegre-RS: SBC. [s.l.: s.n.], 2013, p. 1–10.

RATCOV, David; PELLEGRINI, Jerônimo Cordoni; VIEIRA, Miguel Said; da SILVA, Silas Justiniano. **Relatório - GT Software Livre**. Santo André, 2018. Disponível em: <http://nti.ufabc.edu.br/cetic-2/relatorios-de-grupos-de-trabalho>

SIQUEIRA, Rodrigo. et al, Continuous Delivery: Building Trust in a Large-Scale, Complex Government Organization, **IEEE Software**, v. 35, n. 2, p. 38–43, 2018. Disponível: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8255783>

SØRENSEN, Eva; TORFING, Jacob. Enhancing Collaborative Innovation in the Public Sector. **Administration & Society**, v. 43, n. 8, p. 842–868, 2011.

VAZ, José Carlos. Transformações tecnológicas e perspectivas para a gestão democrática das políticas culturais. **Cadernos Gestão Pública e Cidadania**, v. 22, n. 71, 2017.

WEBER, Steven. **The Success of Open Source**. Harvard University Press, 2004.

ZHANG, Yang; WANG, Huaimin; YIN, Gang; et al. Exploring the Use of @-mention to Assist Software Development in GitHub. In: **Proceedings of the 7th Asia-Pacific Symposium on Internetware**. New York, NY, USA: ACM, 2015, p. 83–92. (Internetware '15). Disponível em: <<http://doi.acm.org/10.1145/2875913.2875914>>. Acesso em: 25 out. 2018.

Only you can see this message



This story's distribution setting is on. [Learn more](#)

Framework de Assistente Virtual do Laboratório Lappis



LAPPIS

Apr 22 · 10 min read

O assistente virtual

O projeto de assistente virtual do Lappis é um chatbot voltado a orientar cidadãos a respeito das políticas públicas desenvolvidas pelo Estado Brasileiro. O primeiro produto completo desenvolvido pelo laboratório é a assistente virtual Tais (Tecnologia de Aprendizado Interativo do Salic), que tem como objetivo ajudar cidadãs e cidadãos a tirar dúvidas sobre a lei de Incentivo à Cultura (Lei Rouanet) e sobre o processo de incentivo de projetos culturais, no contexto da Secretaria Especial de Cultura do Ministério da Cidadania.



26

O assistente virtual utiliza um conjunto de ferramentas e abordagens cuja gestão é realizada por pesquisadores de diversos backgrounds, incluindo engenheiros de software, designers de experiência e roteiristas. A esse conjunto de

ferramentas e abordagens gerido por um time multidisciplinar damos o nome de “Framework de Assistente Virtual do Laboratório Lappis”. Esse Framework tem duas características que tornam ele especialmente interessante para o uso no relacionamento dos cidadãos com as políticas e serviços públicos do Estado. (i) A primeira é que sua concepção é baseada no uso de ferramentas open source que permitem à organização ter total autonomia e independência em relação a guarda dessas informações. Todo o conteúdo planejado, o histórico de conversas e os dados gerados pelo monitoramento ficam armazenados nos servidores da organização (“on premises”), sem a necessidade de cedê-las ou fazê-las circular ou por servidores de terceiros (atendendo a regulação de dados GDPR). (ii) A segunda característica é que o assistente virtual desenvolvido pelo laboratório Lappis não trabalha com o conceito de árvore de decisão, mas sim com redes neurais, que conseguem interpretar a mensagem de fato, inferir o contexto do diálogo e aproximar a conversa com o chatbot de uma conversa natural.

Dessa maneira, de toda demanda recebida pelo assistente são inferidas as intenções do usuário (chamadas de “intents”) para as quais o assistente busca a melhor resposta possível a partir do contexto da conversa. O uso de algoritmos de aprendizado profundo (deep learning) faz com seja possível definir sinônimos dentro de um contexto específico. Um dos exemplos, no caso a assistente virtual Taís, diz respeito à palavra “proponente” que, no contexto da lei de incentivo, é sinônimo de produtor cultural. Isso também é possível por meio de

inferência de contextos de diálogos, ou seja, o chatbot não só usa a pergunta atual do usuário, mas também o histórico da conversa (identificando assim o contexto da conversa), para conseguir dar a resposta mais apropriada. Logo, a inteligência do assistente se aproxima da capacidade que um ser humano tem de entender e interpretar perguntas, em vez de simplesmente navegar em uma lista de opções sequenciais, como um roteiro de telemarketing da organização. Essa característica é especialmente importante por evitar expor ao cidadão a complexidade da administração pública, com todas as suas caixinhas, nem exigir que o cidadão conheça previamente a estrutura do órgão que faz a gestão da política pública para a qual busca algum atendimento. Esse fato que tende a ocorrer em outros canais, como websites institucionais e URAs.

É igualmente importante que o planejamento e gestão de um assistente virtual no órgão sejam feitos de forma integrada à gestão dos demais processos de relacionamento com os cidadãos, sob o risco de fragmentar e complexificar ainda mais a busca por informações sobre as políticas. O assistente não pode ser tratado como uma ferramenta “ad hoc”, isolada das outras estratégias de relacionamento já executadas pelo órgão, levando ao risco de atender apenas a demandas pontuais de grupos específicos, diminuindo a eficiência do gasto público e o impacto na missão global do orgão. Tratar o assistente virtual de forma complementar aos meios disponíveis para atendimento dos cidadãos permitirá ao órgão realizar integrações de canais (como por exemplo a assistente virtual

respondendo perguntas por comandos de voz) além de viabilizar uma leitura e interpretação mais holística das estatísticas de monitoramento. Isso permitirá aferir o quanto a ativação do assistente virtual melhorou de fato a qualidade do atendimento e a resolução de problemas do público alvo das políticas, sem falar na redução de custos operacionais. Dessa forma, a nossa visão estratégica sobre a ativação de um assistente virtual é menos a da instalação de uma nova ferramenta e mais a da melhoria da eficiência do gasto público e da agilidade e qualidade do relacionamento dos cidadãos com as políticas públicas.

Vamos indicar quais são as tecnologias utilizadas e os perfis mínimos necessários para o funcionamento desse framework de assistente virtual criado pelo laboratório Lappis. Dentro da visão estratégica exposta aqui, alguns desses perfis provavelmente já existirão na organização e devem ser aproveitados para a gestão do assistente virtual, favorecendo a integração e transversalidade dos canais de atendimento do órgão.

Discutimos algumas dessas questões durante nossa Participação no Gnpapo sobre Chatbots (evento da Escola Nacional de Administração Pública, Brasília). Veja abaixo o evento na íntegra. Nossas participações: Rodrigo Maia (min 40:35), Carla Rocha (min 51:30), Ricardo Poppi (min 1h38:50).



Participação no Gnpapo sobre chatbots organizado pela Enap (11/04/2019)

O framework

A arquitetura geral do framework desenvolvido pelo Laboratório é mostrado na figura abaixo. Foi desenvolvido com a Licença AGPL3.





Arquitetura do framework de chatbot desenvolvido pelo LAPPIS.

Ferramentas que compõem o framework

Rasa NLU

O Rasa NLU é uma ferramenta de processamento de linguagem natural utilizado para a classificação de intenções e extração de entidades aplicadas ao Bot, ou seja, o NLU é responsável por entender o que os usuários escreveram. Isso é feito por uma comparação da mensagem do usuário com os modelos de *Machine Learning*, procurando inferir a intenção mais próxima da base de conhecimento. A partir daí, a saída do Rasa NLU será a intenção do usuário que enviou a mensagem, que pode ser, por exemplo, cumprimentar, despedir ou uma dúvida específica.

Rasa Core

O Rasa Core é o centro do *chatbot*, o que é um diferencial do Framework LAPPIS. Ele é responsável por escolher a melhor ação a ser tomada a partir do histórico da conversa e da atual intenção do usuário. Seja essa ela uma resposta dentro do

contexto atual da conversa ou uma ação realizada em um novo contexto. Isso garante que o usuário esteja sempre no controle da conversa, ao invés de ser conduzido por uma árvore de decisões. Assim como o Rasa NLU, o Rasa Core utiliza *Machine Learning* para responder as mensagens. Para que isso seja possível, é necessário a criação das **stories**, que são os exemplos de intenção e resposta, ou seja, o *chatbot* vai escolher a melhor resposta para interagir com os usuários a partir desses exemplos de conversas implementados.

Jupyter Notebooks

Uma das maiores dificuldades ao projetar o conteúdo e diálogos de um chatbot é a construção adequada do conjunto de dados de treinamento para as intents e as stories. Jupyter Notebooks é uma aplicação web que ajuda a entender e visualizar dados e resultados de análises, facilitando a experimentação, colaboração e publicação online. O Jupyter foi a ferramenta escolhida para automatizar a análise do conteúdo do chatbot e gerar recomendações de ajustes nas intents e stories. O que nos permite antecipar os problemas de interação com o chatbot antes de colocar o conteúdo em produção.

Elasticsearch/Kibana

É impossível fazer gestão de atendimento sem entender como os cidadãos usam e se apropriam dos canais oferecidos. Conhecer as estatísticas de uso, perfis dos usuários e tendências das demandas é crucial para o sucesso permanente da

estratégia. O Elasticsearch é a fundação do módulo de análise de uso do *chatbot*, pois nele são gerenciadas as informações coletadas das conversas. Para interpretação de todo o conteúdo coletado pelo Elasticsearch, o Kibana é a camada *visual* responsável por disponibilizar todos os dados por meio de *dashboards, gráficos e imagens*. Apresenta esses dados de forma a facilitar a interpretação.

RocketChat

O RocketChat é um software criado por uma startup brasileira que tem o objetivo de fornecer uma infraestrutura de salas de conversa que podem ser utilizadas para diversos fins. Tem recursos de conversação, repositório de documentos e API para bots. Sua interface se assemelha com a do WhatsApp ou Telegram. Seu código é livre e licenciado em MIT. No framework do assistente virtual, o RocketChat carrega a janela de conversação onde os cidadãos interagem de fato com o assistente.

Perfis que compõem o framework

Líder de produto

O líder de produto é um perfil fundamental, responsável por orientar o time em relação à aplicação prática do assistente no caso concreto. É o líder de produto que vai estar mais próximo ao órgão para compreender como o assistente virtual se encaixa

na estratégia de atendimento da organização. A partir disso, define o escopo da base de conhecimento do bot e prioriza as principais demandas de evolução de conteúdo e novas funcionalidades.

Engenheiro de software

O time de Engenharia de Software tem como principal objetivo manter e evoluir o framework. Garantir a integração entre as partes e a estabilidade da solução tecnológica. Sempre que uma das partes evolui nas suas comunidades de origem (ex: Rasa, RocketChat, etc) é papel desse time avaliar os impactos da atualização dos componentes no framework Lappis. Também é o time que garante a entrega contínua, ou seja, o processo automatizado de atualizações dos ambientes de homologação e produção.

Cientista de dados

O cientista de dados tem o papel de analisar os dados relativos ao conteúdo de interação do assistente virtual e as estatísticas de monitoramento. Seu papel é importante para identificar gargalos de pontos de melhoria nos outros módulos. Esse tipo de análise é capaz de identificar quando uma determinada intenção do usuário não está sendo adequadamente interpretada pelo aprendizado de máquina e levantar esse alerta para o time de conteúdo, que pode produzir textos mais claros e para o time de engenharia que pode trabalhar em uma melhor calibragem do aprendizado de máquina.

Complementarmente, a análise dos dados de acesso e interação pode produzir novas demandas de cruzamento e visualização de dados para serem implementadas nas ferramentas de visualização dos dados, como o Kibana. Esse tipo de análise é de especial importância para que a ferramenta produza dados que serão agregados nos indicadores de melhoria de atendimento do órgão.

Roteirista de chatbot

O roteirista de bot tem a responsabilidade de adaptar os conteúdos das políticas fornecidos pelo orgão para que se adequem ao formato conversacional do bot. Para isso ele é responsável por definir a persona do bot a partir de características básicas de personalidade (como empatia, objetividade, nível do vocabulário etc). Uma vez definidas e pactuadas com o órgão, essas características servem de base para a formulação dos textos das respostas disponíveis no banco de conhecimento da assistente virtual. Também é papel do roteirista interpretar os dados de interação e fazer melhorias nos conteúdos que eventualmente não estiverem sendo compreendidos pelos usuários.

Especialista em UX

A pessoa responsável pela experiência de usuário (UX) trabalha lado-a-lado com a equipe de roteiristas. Especialistas de UX atuam para otimizar a experiência de conversa e reforçar a ideia de uma interação que flui naturalmente. Garantem que

sequências de pergunta-resposta se desenvolvam de forma harmônica tanto dentro do mesmo tópico, quanto de forma global na base de conhecimento.

Importância do Parceiro

Órgãos da administração pública federal, governos estaduais e prefeituras, além de órgãos públicos ligados aos outros poderes são os parceiros preferenciais na adoção do framework de assistente virtual do Lappis. Na linha de que o framework de assistente virtual do Laboratório Lappis é um processo sustentado por tecnologia que busca a melhoria dos indicadores de atendimento do órgão, ele deve ser implementado num arranjo de parceria do laboratório com o próprio órgão. É um trabalho a ser realizado a quatro mãos em que o laboratório entra com a **pesquisa, facilitação de aprendizado, desenvolvimento e costura das tecnologias necessárias** para a operação do framework e o órgão entra com a definição das **prioridades, metas e indicadores (gestão)**, o **esforço de homologação e disponibilização em produção (tecnologia da informação)** além da **definição da equipe** que fará a sustentação e gestão do assistente em caráter permanente. O papel do Laboratório Lappis na parceria será o de customizar o framework para o contexto do parceiro, colaborar no processo de implantação e facilitar o aprendizado da equipe do órgão. Do lado do parceiro, será necessária a indicação de responsáveis para ocupar os papéis de **Líder de projeto, Tecnologia da Informação e Comunicação**. A indicação desses responsáveis é especialmente importante pois

eles cuidarão do serviço dentro do órgão de forma permanente. Nesse arranjo, tanto o Lappis quanto o órgão serão parceiros do resultado final da iniciativa, que é o de melhorar a eficiência do gasto público e do atendimento à população.

O futuro do framework

O Laboratório Lappis já iniciou alguns trabalhos para simplificar a adoção do framework por organizações com interesse em desfrutar as vantagens de nossa abordagem “on premises” e baseada em redes neurais. A primeira delas foi a publicação de um modelo de chatbot baseado em FAQ já customizado para português brasileiro. Esse modelo, também conhecido como “Rasa boilerplate(*)” é uma arquitetura pronta de chatbot baseado em Rasa com todos os componentes necessários para qualquer organização poder criar o seu assistente virtual. Além do nosso boilerplate já ter sido objeto de um webinário (ver a lista completa abaixo) já começou a ser utilizado por pelo menos duas outras organizações. Mas não queremos parar aí. Nossa visão de futuro para o framework de assistente virtual é desenvolver mecanismos automatizados para a criação e gestão de conteúdo, além de aprimorar os componentes de monitoramento, para num futuro um pouco mais distante viabilizarmos a oferta do framework em modelo SaaS (Software as a Service) impulsionando o campo de serviços e negócios baseados nessa tecnologia livre.

(*) Boilerplate é um termo da computação utilizado para indicar pedaços de código que podem ser reutilizados facilmente. O termo

vem do comunicação impressa, aquela das prensas metálicas, onde os boilerplates eram as matrizes de texto que não eram alteradas entre as edições impressas dos jornais.

Conteúdos abertos disponibilizados pelo Laboratório Lappis

Repositório com o trabalho de desenvolvimento

Todo trabalho de desenvolvimento realizado no framework de assistente virtual do Lappis é público e aberto. Algumas contribuições são feitas diretamente nas comunidades das ferramentas utilizadas. As contribuições específicas do assistente virtual estão organizadas no seguinte repositório público: <https://github.com/lappis-unb/tais>

A comunidade Rasa Brasil, fundada pelo Laboratório Lappis também mantém um grupo público no Telegram destinado a troca de experiências entre pessoas e organizações que estejam utilizando o Rasa no Brasil: <https://t.me/RasaBrasil>

Webinários com tutoriais sobre as tecnologias utilizadas no assistente virtual





13/dez/2018 — Fazendo seu chatbot inteligente com RASA e Rocket.Chat

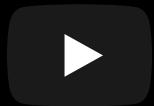


23/jan/2019 — Boilerplate de FAQ Chatbot em português usando RASA — Customizando seu chatbot





27/mar/2019 — Chatbot Rasa conectado com API do Google



10/04/2019 — Webinar Rasa Bot no Telegram com Slots e Entities



17/04/2019 — Webinar relâmpago: Rasa Stack, explicando a fusão entre o Rasa Core e o Rasa NLU

Chatbots

Governo

Rasa

Assistente Virtual

Unb

Discover Medium

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. [Watch](#)

Make Medium yours

Follow all the topics you care about, and we'll deliver the best stories for you to your homepage and inbox. [Explore](#)

Become a member

Get unlimited access to the best stories on Medium — and support writers while you're at it. Just \$5/month. [Upgrade](#)

Medium

About

Help

Legal

Only you can see this message

×

This story's distribution setting is off. [Learn more](#)

What I wish I knew before starting my first Chatbot project



LAPPIS

Jan 14 · 4 min read

For the past 14 months, we have been working on a FLOSS FAQ chatbot project in collaboration with the Ministry of Culture in Brazil. We have come a long way testing, evaluating chatbot frameworks, NLP techniques, and user experience with chatbots, doing applied research while developing [TAIS](#) (chatbot in Brazilian Portuguese), our chatbot for the Law of incentive to the culture, known as “LEI Rouanet”, and its service, “SALIC”. Chatbot presence is growing in the private sectors, but it is still rare in government agencies.

Even if chatbots are trending, the documentation to guide its project and decision making is most of the times not very clear and is dispersed. There is a quite broad range of option to chatbot projects, from services such as Dialog Flow, Facebook, Watson, LUIS, to open sources solutions, such as Rasa, Botkit, Parlai, among others.

We have made many mistakes along the way. Mainly due to the lack of mature documentation and community still emerging. Here we list what we wish we have known before starting our chatbot project.

Chatbot is an ecosystem: a chatbot platform is the integration of several components. Some solutions provide only a subset of these components and it makes the comparison between platforms challenging. The components of a complete chatbot platform is despitely in Fig. 1.

VOICE ASSISTANTS

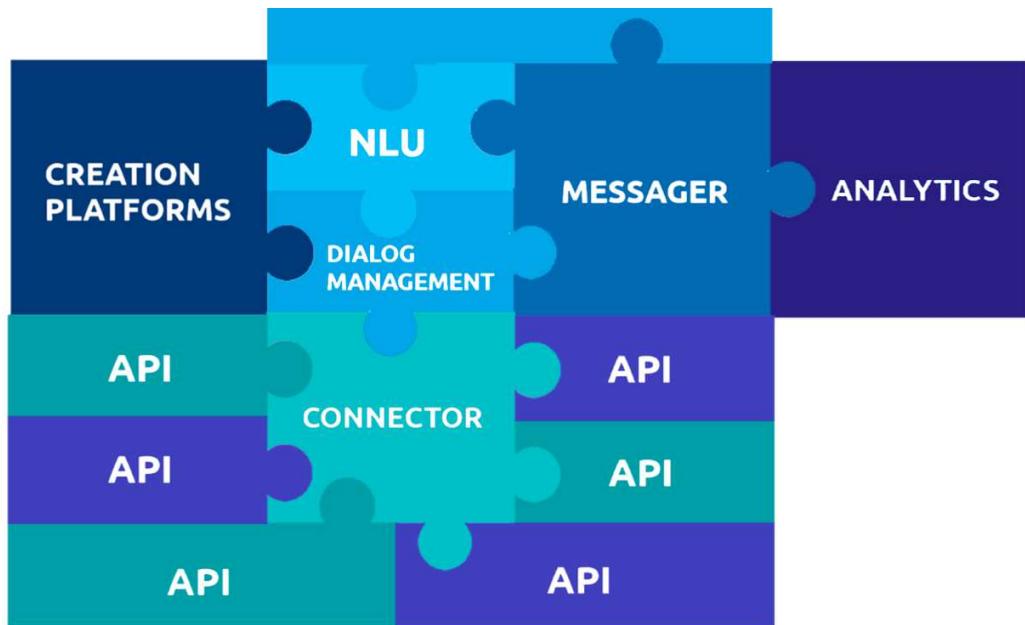


Fig1. Chatbot Platform: components that, integrated, form a chatbot platform.

- **Fullstack team:** developing a chatbot demands a big set of abilities, differently from most of the software projects. It requires software engineers, DevOps, dialog designer or chatbot scriptwriter, data scientist, business strategist, and so forth. They need to work closely and make decisions together. Figure 1 shows an ideal chatbot team. The absence of one of these abilities in the team jeopardizes the chatbot quality (We have made this mistake).

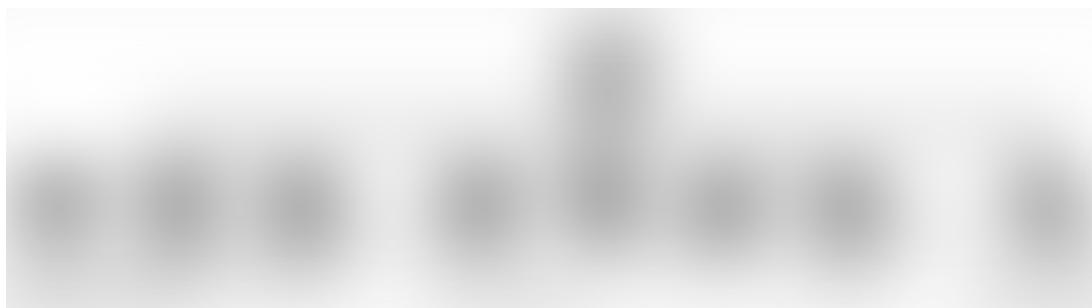


Fig.2 — Ideal Fullstack chatbot team

- **Algorithms impact a lot the user experience:** how the Natural Language Processing is done, the techniques used, impact the ability of the chatbot to understand user intentions. The strategy/policy chosen to deal with dialogs (menus, guided flows, learning from example conversations) impacts enormously the user experience (UX). Each chatbot platform available has one of several strategies to process the conversation. Make sure the Chabot solution you choose it meets your project requirements.

- **Compliance with data regulations:** Pay attention to how your organization deals with data regulations, such as the General Data Protection Regulation (GDPR), not all solutions comply with these regulations.
- **Chatbots take time to mature:** you don't want your chatbot to become a lousy meme! So, understand that a chatbot project is iterative and incremental. It takes time to have a cool chatbot ;P Make sure all stakeholders understand that! Controlled chatbot presentations make clients dazzle, and they tend to pressure the team to put it as soon as possible in production. Take time to test and validate your chatbot! The beta test is the way to go (a more controlled ambient and engaged community with mostly constructive feedbacks).
- **Corpus in other Languages than English** — it is a big problem! So, try to find communities in your country working with chatbots, and share training datasets :P
- **Analytics** —how users interact with chatbots tends to be very different from how you thought they would behave! User interaction data is the most precious asset of a chatbot. It gives you the real interests/doubts of your customer, and you can have data-driven business strategies! Additionally, it tells your team where the chatbot is making mistakes and where to focus improvements.
- **Continuous Delivery** — enabling Continuous delivery allows to instantiate a chatbot, and test it very often. It is important for users to perceive a better and more accurate interaction with the chatbot continuously , increase of its knowledge; Think about Continuous deploy, DevOps, use of dockers.

We have chosen to have a FLOSS chatbot, completely on-premise. We have chosen Rasa Rasa-Core and Rasa-NLU as our chatbot creation service and Rocket.chat as our distribution service (how to interact with users). We also used elasticsearch + kibana as our analytics. Our complete architecture is depicted in Figure 2. It is under the open-source license GPL3 and all project source code, documentation is available in <https://github.com/lappis-unb/tais>.



Fig.3 — Our Chatbot architecture available in <https://github.com/lappis-unb/rasa-ptbr-boilerplate>

We will detail some of our lesson learned and some results, technical issues in future posts.

Chatbots

Rasa

Rocket Chat

Egovernment

Machine Learning

Discover Medium

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. [Watch](#)

Make Medium yours

Follow all the topics you care about, and we'll deliver the best stories for you to your homepage and inbox. [Explore](#)

Become a member

Get unlimited access to the best stories on Medium — and support writers while you're at it. Just \$5/month. [Upgrade](#)

Medium

About

Help

Legal

Only you can see this message



This story's distribution setting is off. [Learn more](#)

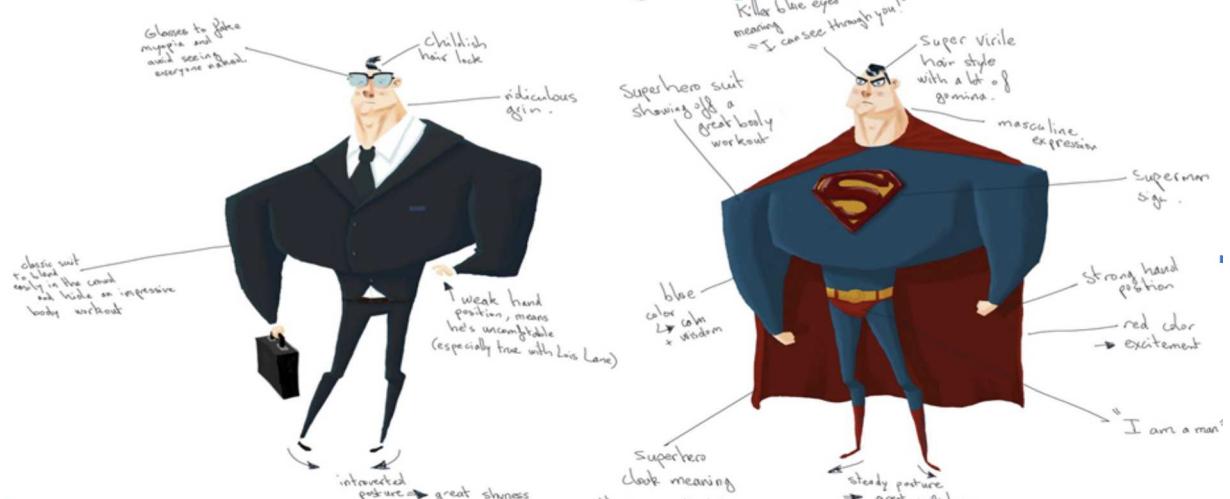
Encantando pessoas: poder da personalidade em Chatbots



LAPPIS

Jan 30 · 4 min read

Persona? Quê que é?



"POWER IS POWER !!!" [Cersei]

Persona — Power is Power

Persona quê que é?

Na teoria de Jung, chamada psicologia analítica, têm-se a ideia de que o indivíduo modifica sua personalidade de acordo com a

função que exerce para atender aos requisitos da sociedade. A persona se moldaria então de acordo com o ambiente, sendo assim uma pessoa possui várias faces segundo o papel que desempenha. Como exemplo disso podemos observar o Superman, ele tem duas personas. Ele desempenha o papel de Superman, sendo imbatível e salvador do Planeta Terra. Sai por aí com seu collant, voando de prédio em prédio, é totalmente confiante e não sente dificuldades em ter os holofotes virados para o seu show pessoal. Em contrapartida, ele também é o Clark Kent que tem a personalidade totalmente introvertida, mal consegue conversar com o amor da sua vida, Lois Lane. Muitas pessoas fazem piada sobre as pessoas não identificarem os dois apenas por causa dos óculos, mas na verdade, a grande sacada se encontra na personalidade. É isso mesmo!! Os dois são tão diferentes no quesito comportamento que é impossível associar o fragilizado e tímido repórter ao super-herói destemido.

Uma persona depende de outra persona

Para construir a personalidade do seu chatbot, você precisará entender também outro conceito de persona. Persona também é uma estratégia de Marketing. É quando montamos as características de um público alvo de um determinado produto:

“Criação da persona que interagirá com o seu Chatbot: Essa será o perfil do seu potencial cliente, ou seja, preverá situações onde seu Chatbot será exposto. Como falávamos acima.”

Ou seja, é o usuário que vai definir a personalidade do seu bot, pois não adianta nada criar um chatbot que te agrade e não agrade o seu público alvo.

O usuário manda no seu projeto!

O seu bot precisa ser muito encantador, ele deve fazer o usuário sentir vontade de continuar a conversa, ele deve de alguma forma instigar o usuário, e é por este motivo que ele precisa de uma personalidade hipnotizante. Pois como disse a odiada ou querida Cersei “Power is Power!”.

Encantando Pessoas: o poder da personalidade

Okay, agora que já compreendemos a importância da personalidade, e como ela pode te ajudar a criar um chatbot funcional e eficiente, temos que aprender a criar uma. Existem várias técnicas para se criar uma personalidade para um chatbot, é como se você estivesse criando um personagem para uma história. Deste modo, você pode se inspirar em personagens que já existem, em pessoas, ou criar uma personalidade baseando-se em uma teoria psicológica. Um exemplo desses tipos de teoria é o MBTI. A famosa tipologia de Myers-Briggs, em que divide a personalidade em 4 instâncias: analistas, diplomatas, sentinelas e exploradores. Cada instância possui 4 personalidades. Foi assim que nos baseamos para criar a personalidade da Tais. Ela é uma junção entre Cônslul e Protagonista.

A nossa escolha foi baseada na função que ela vai desenvolver, ela como assistente do ministério da cultura precisa ser extrovertida e muito atenciosa com os usuários, sempre buscando sanar as dúvidas dos usuários com muito carisma. Além disso, aos poucos estamos implantando mais características e comportamentos que estão moldando e salientando a personalidade dela através do diálogo.

É preciso também definir um objetivo para a persona, assim como suas dores e desafios. Outra curiosidade é que quando nós seres humanos, falamos com alguém ao telefone automaticamente imaginamos como seria essa pessoa do outro lado, ao ouvir a voz, ou até mesmo ler uma mensagem de texto, nosso cérebro cria automaticamente uma ideia de como aquela pessoa é. E isso não é diferente com o contato com uma assistente virtual, por isso é interessante definir uma história, um gênero, às vezes uma idade, uma profissão. Todas as características que possam dar vida ao seu bot. Encha-o com as peculiaridades mais extraordinárias que conseguir.

Sobre as características da Tais (pode acessá-la em <http://rouanet.cultura.gov.br>):

Gênero : Feminino

Cargo/Ocupação: Assistente Virtual do MinC

Meio de comunicação: Chatbot

Objetivos da persona: Realizar os **projetos** dos proponentes priorizando o cumprimento da lei.

Desafios da persona:

Aprender as variedades sintáticas e dialetais dos usuários para melhoramento da comunicação, estabelecer uma interação mais natural e confortável para com o usuário compreendendo as intenções/dúvidas do usuário. Sanar dúvidas de forma rápida e eficiente.

Como fazer a personalidade do seu bot aparecer?

Ao construir o seu chatbot e definir a personalidade dele, deve-se criar estratégias para fazer a personalidade dele se tornar visível para o usuário. Deste modo, a personalidade de um chatbot textual vai aparecer pela sua linguagem, ou seja, o modo como ele vai se comunicar com o usuário. Na construção da Tais, nós optamos por fazer dela uma assistente gentil e sempre pronta para ajudar o usuário, que é uma característica latente da personalidade Cônsul. Ao mesmo tempo em que ela é prestativa e atenciosa, costuma tentar engajar o usuário, animando-o para que ele continue confortável para fazer perguntas que é uma característica mais comum da personalidade Protagonista, criando assim uma conversa guiada, mas que ao mesmo tempo é fluida e harmoniosa.

REFERÊNCIAS

Creating a persona: what does your product sound like?

A guide to developing bot personalities

Myers-Briggs — 16 Personalities

How to Give Your Chatbot a Personality

Dicas Matadoras para Criar Chatbots Bons de Conversa

Chatbot

Personalidade

Rasa

Lei Rouanet

Dialogue

Discover Medium

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. [Watch](#)

Make Medium yours

Follow all the topics you care about, and we'll deliver the best stories for you to your homepage and inbox. [Explore](#)

Become a member

Get unlimited access to the best stories on Medium — and support writers while you're at it. Just \$5/month. [Upgrade](#)

Medium

[About](#)

[Help](#)

[Legal](#)