Brazilian Public Software Portal: an integrated platform for collaborative development

Paulo Meirelles
Faculty Gama (FGA)
University of Brasília
Gama, Brazil
paulormm@unb.br

Melissa Wen
FLOSS Competence Center
University of São Paulo
São Paulo, Brazil
melissa.srw@gmail.com

Antonio Terceiro
Colivre
Salvador, Brazil
terceiro@colivre.coop.br

Rodrigo Siqueira
FLOSS Competence Center
University of São Paulo
São Paulo, Brazil
siqueira@ime.usp.br

Lucas Kanashiro
FLOSS Competence Center
University of São Paulo
São Paulo, Brazil
lkd@ime.usp.br

Hilmer Neri
Faculty Gama (FGA)
University of Brasília
Gama, Brazil
hilmer@unb.br

ABSTRACT

The Brazilian Public Software (SPB) is a program promoted by the Brazilian Federal Government to foster sharing and collaboration on Free/Libre/Open Source Software (FLOSS) solutions for the public administration. In this context, a public software is considered a public good and the Federal Government assumes some responsibilities related to its use. Once its devolpment principles is the same of the FLOSS projects, we have designed the SPB Portal, a platform based on the integration and evolution of existing FLOSS tools. It provides several modern features for software collaborative development, helping the Brazilian public administration in sharing its solutions. In this paper, we present this integrated software development platform that was developed for the SPB program by a heterogeneous team composed by professors, master students and undergraduate students, as well as by professionals from FLOSS communities. The development of this platform used several FLOSS projects, providing a non-trivial integration among them. This effort has also produced several new features that were contributed back to these projects. Alongside the architectural challenges, we also discuss in this paper our work process, based on agile and free software development practices, and the lessons learned during 30 months of work on the SPB project.

ACM Classification Keywords

K.6.1 Project and People Management: Systems development

Author Keywords

Free Software, Software Integration, Management Team.

OpenSym '17, August 23–25, 2017, Galway, Ireland

© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-5187-4/17/08...$15.00

DOI: https://doi.org/10.1145/3125433.3125471

INTRODUCTION

In the year 2000, the Brazilian Government released the Eletronic Government program (eGov) aiming at democratizing information access and improving the quality of public provision of service and information. In 2003, the Federal Government created a committee for implementation of Free/Libre/Open Source Software (FLOSS)[1] and thereafter a circular-letter was sent to all Ministries in which the recommendation to adopt FLOSS became a public policy. In 2007, the Brazilian Public Software Portal (*Portal do Software Público Brasileiro*, in Portuguese) was released with the goal of sharing FLOSS projects developed by, or for, the Brazilian Government. Additionally, the Brazilian legal instrument on software contracting (known as IN 04/2012) mandates that public agents must prioritize solutions available on the SPB Portal. The acquisition of a proprietary solution must be explicitly justified by demonstrating that there is no suitable alternative on the SPB Portal. In 2013, the Brazilian Federal Court issued a ruling (*Acórdão 2314/2013*) about contracts between the public administration and suppliers using agile methodologies in software development.

Despite of these legal advancements, in practice, Brazilian government agents still do not practice, or even understand, collaborative and empirical software development methods, such as FLOSS or agile methodologies. Thus, hierarchical and traditional processes and the lack of expertise of public agents in real-world software development produce inefficiency in software development contracts, besides unjustifiable expending of taxpayers money. For instance, since 2009 the SPB Portal has had several technical issues. The original codebase development has stopped. The system was a modified version of an existing FLOSS platform called OpenACS[2], and the old SPB Portal did not receive updates from OpenACS releases. In this scenario, the portal maintenance has become increasingly difficult.

[1]Free Software, Open Source, or Free/Open Source Software.
[2]http://openacs.org

Approximately five years later from the first problems, a new platform for the SPB Portal was planned and funded. Between January 2014 and June 2016, the University of Brasília (UnB) and the University of São Paulo (USP) in a partnership with the Brazilian Ministry of Planning, Budget, and Management designed an integrated platform for collaborative software development [1], including social networking, mailing lists, version control system, and source code quality monitoring. To coordinate and develop this project during 30 months, UnB was funded by a grant of 2,619,965.00 BRL (about 750,000.00 USD in June 2016) from the Federal Government.

The project was developed by a team of three professors, two masters students, about fifty undergraduate students (not all of them at the same time, since the team changed along the time), two professional designers, and six senior developers from FLOSS communities. Professors and undergraduate students were from UnB and master students were from USP. Regarding the designers and senior developers, seven of them were living outside Brasília, the UnB location. In other words, we had a distributed team working in a collaborative virtual environment. This diversity of actors and the relationships between industry, academy, and government also made the project a valued opportunity to explore the benefits and challenges of using FLOSS [11, 3, 6, 5] and Agile [14, 8] practices for Software Engineering education.

All the code was developed as free software. The changes we needed in the FLOSS tools were implemented by ourselves and contributed back to their respective communities. Our process was based on agile practices and FLOSS communities interaction. We incrementally released five versions of the new SPB Portal. The first release (beta) was in September 2014, only 9 months from the beginning of the project. The old portal was shut down in September 2015. Finally, the last version was released in June 2016.

In this paper, we present an overview of the new SPB Portal. We share the methodology employed to develop this project. This methodology has the goals of satisfying Government requirements and adhering as much as possible to FLOSS and agile practices [13, 15]. Moreover, we discuss lessons learned in providing a distributed and collaborative virtual environment involving a large team of undergraduate students and remote senior developers. In short, we released an innovative platform for helping the Brazilian government to apply empirical software development methods. This case can help other projects to overcome similar software engineering challenges in the future, as well as to illustrate how universities can improve the real-world experience of their students.

BACKGROUND

Since the beginning of computing the majority of developers worked in the way that we now identify as free software, that is, sharing code openly. This openness makes the code available for inspection, modification, and use by any person or organization [9, 11].

The elements that distinguish FLOSS from other types of software are the reasoning about the development process, the economic context, the relationship between developers and users, as well as the ethical and legal characteristics that relate to the software. In the context of FLOSS, user freedom is promoted and its development is based on open collaboration and development practices [11].

From the economic point of view, unlike what happens with proprietary software, FLOSS promotes the establishment of several suppliers that can compete with each other based on the same software. This stronger competition among suppliers brings benefits to users because it gives better assurances regarding the evolution of the system and induces a reduction in prices [11]. These freedoms and assurances on software are guaranteed in Brazil by Law 9610/98 (copyright law). Most of the time, this protection from the law complies with the terms conferred by a contract related to certain software. This contract is called "license". A software license determines a list of rights that are given to, and duties that are imposed on a user of the software. In particular, what differentiates FLOSS from proprietary software is just the way they are licensed [11]. The FLOSS licenses guarantee the right to execute, study, adapt, and improve the software. Example of common FLOSS licenses are the *GPL (GNU General Public License)*, the Apache license, the MIT license, and the BSD license.

The original incarnation of SPB portal has been designed in 2005 and released in 2007. From a practical point of view, it is a web system that has consolidated itself as an environment for sharing software projects [7]. It also provides a space (community) for each software. Therefore, it was designed to include tools that promote collaboration and interaction in communities (by managers, users, and developers) of the projects, according to the practices used in FLOSS communities. This includes mailing lists, discussion forums, issue trackers, version control systems, and social networking environments.

Initially, the purpose of the portal was only to share the software developed in the Brazilian government to reduce the costs of hiring software. However, it was observed that when a software was released, a community was formed around it, with several people collaborating and sharing the results obtained through the use of those solutions, as commonly occurs in FLOSS [4]. In this way, some software development cooperatives and private companies have shown an interest in making their software available on the SPB Portal.

The concept of Brazilian Public Software goes beyond FLOSS [7]. In addition to being licensed under a FLOSS license, this software needs to have explicit guarantees that it is a public good, and its project must be available on the SPB portal. Being a true public good assumes requirements that can not be met solely by means of FLOSS licensing. For example, there must be a relaxed trademark usage policy by the original vendor that does not stop eventual competitors from advertising services for that same software. Inclusion in the SPB Portal also has extra requirements, such as having a public version control system, installation manual, and hardware requirements specification.

RELATED PROJECTS

The new SPB platform is a fully integrated environment, very advanced in comparison with related initiatives. For instance, the USA government has a platform designed to improve access to federal government software, called Code.gov[3]. It is an interface to organize USA government projects, making it easy for users and developers to obtain information and to access their source code repositories at GitHub. However, it does not provide social networking and CMS (Content Manager System) features, as well as, other communication resources.

The European Commission (EC) assists the Open Source Observatory (OSOR)[4] that intends to exchange information, experiences, and best practices around the use of FLOSS in the public administration. It supports the discovering of FLOSS projects made available by public agencies, providing information about these projects, such as news, events, studies, and solutions. It also offers discussion forums and community mailing lists. But it does not have an integrated source code repository manager, so for each project there is a link to its own external repository. Previously, from 2007 to 2011, the EC promoted the QualiPSo project that aimed to support FLOSS users, developers, and consumers with resources and expertise on FLOSS quality. The QualiPSo project also had planned to develop a platform called QualiPSo Factory, but it was not fully completed.

In Latin American there is an initiative called “Software Público Regional”[5] that provides just a customized Gitlab instance to share source code and documentation of projects developed by the involved countries. Moreover, Chile has also its own portal named “Repositório Software Público”[6]. In the communities of the portal, users can create content such as news, documents, and wiki pages, but source code repositories are available at the Bitbucket platform[7].

The Brazilian government needed to evolve the SPB platform that existed since 2007. When we started this project, the SPB Portal had about 200 thousand registered users. We could not just do something like contacting these users and asking them to register an account at Github. Moreover, after the Edward Snowden’s case, the Brazilian government issued a decree (8.135/2013) requiring public agencies to host their information systems by themselves, avoiding the usage of private platforms, especially the ones provided by foreign companies. Therefore, we needed to develop our own solution to cover all the requirements, producing a complete governmental integrated platform for collaborative software development.

OPEN QUESTIONS

In this paper, we share our experience in designing and developing the new SPB Portal by reporting the technical efforts carried out, our empirical work process, and the lessons learned. The new SPB Portal project presented three main challenges, related to the open questions described below.

Q1. *Which strategy could be used to integrate several existing FLOSS tools to promote a collaborative software development?* Based on an extensive list of functional requirements defined by the Brazilian Federal Government, we selected some FLOSS systems to compose our solution, engineering a nontrivial integration among them. We looked for the systems set realizing the largest possible subset of the requirements list. However, we were fully aware that we would need to improve those systems in order to satisfy the remaining requirements. We were also convinced that it would be impossible to satisfy all of those requirements with a single tool.

Q2. *How to involve students in real-world projects interacting with real customers?* Our team was mainly composed of software engineering undergraduate students, who had the opportunity to interact with the UnB managers, senior developers, designers, and even with technicians and managers from the Brazilian Government. For the majority of the students, this was a first professional experience. Even though, our development process defined a central role on students participation.

Q3. *How to introduce typical FLOSS collaborative and agile practices in the governmental development process?* The software development in Brazilian government is based on a very traditional way, frequently focusing documentation deliveries. We had to convince them to accept the idea of open scope and empirical development. They had certain expectations about the project development according to the Rational Unified Process (RUP) and the Project Management Body of Knowledge (PMBOK) approaches, which mismatched our work style based on agile and FLOSS practices. So we created strategies to conciliate these different organizational cultures within the project.

REQUIREMENTS

By preparing the SPB Portal evolution, the Brazilian Government has executed three steps to collect the requirements. The first step was the collection of proposals using an online tool called Pligg[8] and the open sharing of them on the Internet. In this step, the citizens have written and voted on proposals they were more interested in. At the end, the Brazilian Government collected about 100 proposals and its initial perspective was to give to the most voted ones the priority of implementation on the new SPB Portal.

The second step was two face-to-face meetings that aimed to discuss ideas (not necessarily based on the previous collected proposals) to improve the SPB Portal and its environments. On the first day, the participants were divided in two groups to discuss (i) features and technologies as well as (ii) user experience and general ideas regarding the SPB Portal. Each group has generated a “mind map” to summarize and to correlate its ideas. During the second day, the participants were allocated in three groups to discuss features related to (i) the process of software evaluation and acceptance in the SPB Portal, (ii) approaches to share the SPB projects, and (iii) ways to attract universities and students to collaborate to SPB projects.

[3] https://code.gov

[4] https://joinup.ec.europa.eu/community/osor

[5] http://softwarepublicoregionalbeta.net

[6] http://softwarepublico.gob.cl

[7] https://bitbucket.org/softwarepublico

[8] https://pligg.com

The last step was a workshop with some IT representatives of the Federal Government and public organizations and, again, it was focused on collecting new proposals to evolve the SPB Portal.

After these unconnected three steps, the Brazilian Government has generated a list of 145 requirements. To provide a cohesive initial list of requirements, we have proposed to release the first stable version of the new platform to replace the old SPB Portal, prioritizing the following features:

1. An organized public software catalog;
2. Social network environment (profiles for users, software pages, and community pages);
3. CMS features;
4. Web-based Git repository manager with Wiki and issue tracking features;
5. Mailing lists and discussion forums.

Moreover, the new SPB Portal would only work properly if there was a unique authentication to use the provided features. Additionally, a unified interface was an important non-functional requirement to provide better user experience on the new platform.

Other requirements were in the wishlist such as an integrated search engine and a web-based source code static analysis monitor. By analyzing all of these requirements, we have designed the SPB evolution project based on existing FLOSS tools.

ARCHITECTURE

From the architeture point of view, the integration of several features (such as centralized authentication, unified interface, search engine as well as other back-end features) of systems with different programming languages and frameworks would require a non-trivial amount of work. In this context, the most important architetural requirements for the new platform were:

1. *Integrating existing FLOSS systems* with minimal differences from their original versions;
2. *Providing consistent user interface* across different systems as well as centralized authentication.

The adoption of existing FLOSS systems and the minimization of their local changes had the purpose to lower the effort of upgrading the software packages that compose the platform to newer version of their original software. With this facility, the platform benefits from maintenance and improvements made by communities. The development of a consistent user interface aims to provide to platform's users a smooth transition between different systems. Without it, the necessity of adaptation and learning for each tool could get users confused and fatigued. For the first requirement, we have identified four main systems which would have specialized teams for work in the integration process. Team members have learned how to write code to their assigned systems and how to contribute to the original communities to align the used version with the original one.

In the end of the project, the SPB portal has combined more than ten systems, such as Colab, Noosfero, Gitlab, and Mezuro.

Colab[9] is a systems integration platform for web applications. One of its goals is allowing different applications to be combined in such a way that an user does not notice the change between applications. For that, Colab provides facilities for: (i) Centralized authentication, (ii) Visual consistency, (iii) Relaying of events between applications, and (iv) Integrated search engine. Colab implements this integration by working as a reverse proxy for the applications, i.e., all external requests pass through Colab before reaching them. Initially, Colab had support for a small set of applications (Trac, GNU Mailman, and Apache Lucene) hard-coded in its core. Our team have helped Colab upstream to redesign its whole architecture, enabling the development of plugins to integrate new tools. We also added a feature that allowed Colab to run asynchronous tasks, which was a major improvement for us since we were developing a complex system. We have also migrated Django(web framework used by Colab) to the latest version and worked on RevProxy (the more important dependency of Colab) to put it in a good shape, fixing many bugs.

Noosfero[10] is a software for building social and collaboration networks. Besides the classical social networking features, it also provides publication features such as blogs and a general-purpose CMS. Most of the user interactions with SPB is through Noosfero: user registration, project home pages and documentation, and contact forms. Noosfero was the tool that contemplated several functional requirements, therefore we have made a large number of contributions to upstream. We have also helped it to migrate to the latest Rails version (web framework used by Noosfero), to enable the federation implementation (federation with other social networks) and to decouple the interface and the back-end.

Gitlab[11] is a web-based Git repository manager with wiki pages and issue tracking features. It is a FLOSS platform and focuses on delivering a holistic solution that will see developers from idea to production seamlessly and on a single platform. Gitlab has several unique features, such as built-in continuous integration and continuous deployment, flexible permissions, tracking of Work-in-Progress work, moving issues between projects, group-level milestones, creating new branches from issues, issues board, and time tracking. We have contributed to Gitlab upstream with some improvements related to configuration files and with the development of a new plugin that enables user authentication in Gitlab through the REMOTE_USER HTTP header. This plugin was needed because Colab uses this mechanism to manage the authentication.

Mezuro[12] is a platform to collect source code metrics to monitor the internal quality of software written in C, C++, Java, Python, Ruby, and PHP. In general, source code metrics tools

[9] https://github.com/colab
[10] http://noosfero.org
[11] http://gitlab.com
[12] http://mezuro.org/

do not present a friendly way to interpret their results and, even more, do not follow a standardization between them. Mezuro collects and presents these results to the end user, specially, by analyzing source code metric history during its life cycle. The Mezuro platform provides a single interface grouping available tools, allows selection and composition of metrics in a flexible manner, stores the metrics evolution history, presents results in a friendly way, as well as, allows users to customize the given interpretation accordingly to their own context. During the project, we helped to modularize the Mezuro project in several independent services to minimize the amount of code to maintain it, helping to test it and grant its code quality. Currently, its computation and visualization modules use Kalibro and Prezento, respectively. They were developed into the Mezuro project and evolved during its integration to the new SPB Portal.

System unification and User eXperience evolution

The conceptual architecture of the platform is presented in Figure 1. Colab initially handles all user interaction, directing requests to one of the integrated applications. It post-processes responses from the applications to apply a consistent visual appearance, manages authentication, and provides a unified search functionality: instead of using the redundant restricted search functionality of each application, a search on the SPB portal might return content from any of the applications, be it web pages, mailing list posts, or source code.

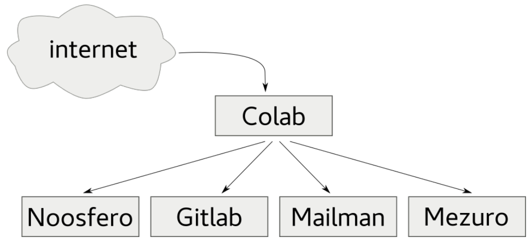

Figure 1. SPB architecture overview.

However, integration of collaborative environments goes beyond functional aspects. To reduce the citizens perception of system complexity and to encourage them to use the software, a platform should offer a unified experience across its environments. Thus, the SPB Portal information architecture was redesigned to provide a transparent navigation and to reach users with different profiles. A process of harmonization has been employed on the interaction models of each tool to reduce the learning curve. At the same time, a new visual style was created to unify the navigation experience and to comply with the guidelines of the digital communication identity standard established by the Federal Government.

With the increase in system features and the addition of new tools, the visual style has steadily evolved to keep the navigation unified. Moreover, tools from different backgrounds, which in many cases provide similar functionality, prompted the development of a unified interface. Some features, such as search and user profile editing were eliminated from the individual applications, and implemented centrally to ensure a consistent look and feel.

Another challenge was responsive web design. The integrated applications had varying degrees of support for responsiveness, and the common interface had to adapt for each individual scenario. In particular, Noosfero did not yet have a responsive design; we also engaged in its development and contributed towards that goal.

Deploy

The SPB platform was deployed in 7 virtual machines with different functions, as we can see in Figure 2. The *reverseproxy* handles the HTTP requests and redirects them to the *integration*, the *email* sends and receives e-mails on behalf of the platform and the *monitor* keeps the entire environment tracked. These three mentioned virtual machines - *reverseproxy*, *email* and *monitor* - are accessible via Internet and the other ones are only available in the local network created between them.

Integration works as a second layer of proxy beneath *reverseproxy*, any request to the platform will be handled by it. The Colab service provides interface, authentication and search engine integration among all the services. When a request is received to a specific service, Colab authenticates the user in the target tool, sends the request and makes a visual transformation in the HTML page which is the content of the response. Another user-oriented feature is the integrated search engine, when the user want to find something in the platform Colab will perform the search in the whole databases. Colab itself provides a web interface for GNU Mailman and we have two others integrated tools in *integration*: Gitlab and Prezento. Gitlab provides web interface for Git repositories and issues tracker, and Prezento is a front-end for source code static analysis.

The source code static analysis is performed by *mezuro*. It runs some static analysis tools on source code stored in a repository and provides this data to Prezento. A social network and CMS is provided by Noosfero in *social*, and the databases of all tools with a cache service are in *database*.

FEATURES

The new generation of the SPB Portal combines features adapted from existing collaborative software and features developed by us. The new functions (newly developed or partially modified) were contributed back to the official repositories.

As a result, we have a platform that integrates and harmonizes different features such as social networking, mailing list, version control system, content management, and source code quality monitoring. Our aim was to develop functionalities by reusing functionality of collaborative software already integrated to the platform. In addition, we tried to keep this integration transparent to end users. We can discuss on the main features according to the three groups below.

Software Project and Software Community. In the new SPB Portal, each software has a standard set of pages and tools. Besides accessing support pages (such as FAQ and installation guide) within the platform, users will be able to download different versions of the software and find several mechanisms of software development management.

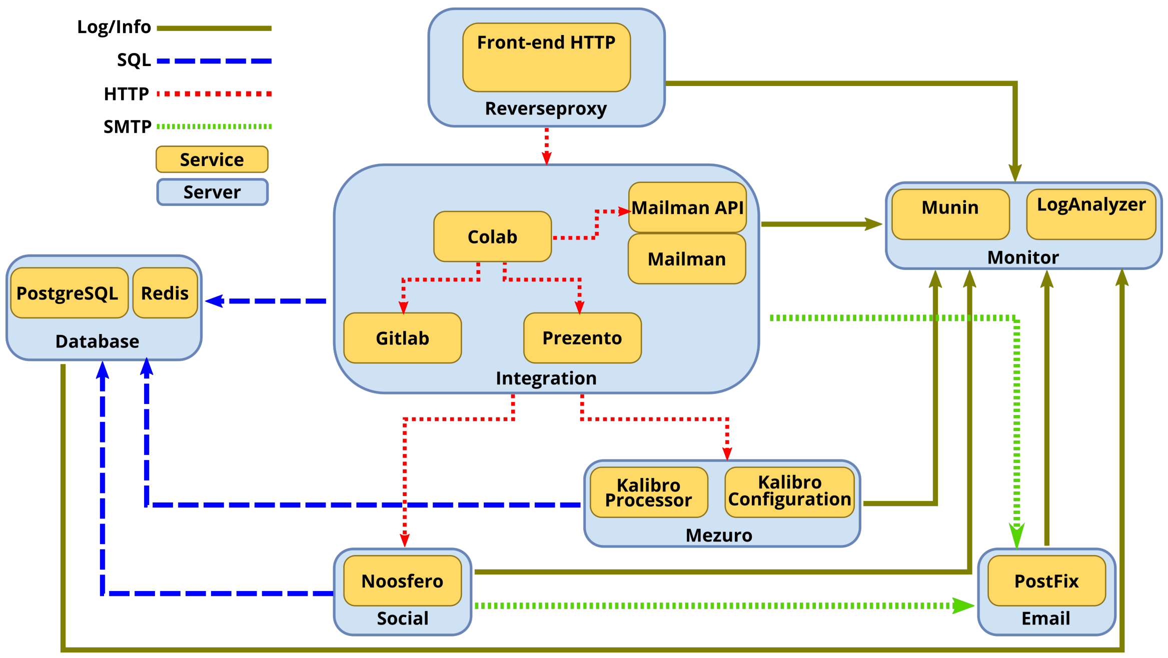

Figure 2. Instanciation view of the SPB architecture.

Focusing on the collaborative development, Mailman was integrated to the platform to allow the dialogue and communication between developers, users and enthusiasts of a determined software. The software has its own mailing list whose privacy can be configured by the software community administrators.

The software has a social interface area ("software community") where users can find other users, blogs, summary of recent activities, or any other relevant community-produced content. Users logged to the platform can request membership to different software communities and each community member can access and edit restricted content. For this purpose, many Noosfero features related to social networking and content management were integrated to the portal.

To assist decision-making, the new SPB Portal has acquired assessment and statistical tools. Now, users will be able to rate the software and make comments and all information will be avaiable to other users. Moreover, the software has a section containing its statistical data, where values are calculated based on data provided by users and the system.

The role of the administrator is present in the software project and in its community. The administrator is responsible for moderating content, memberships and user comments. The administrator is also the one who can make changes in the software homepage content.

Software Catalog and Global Search. The platform also provides a search tool called Software Catalog, which allows users to find softwares available in the portal. In this catalog, we developed some search options to make the navigation easier, such as filters (by type of software or category), sorting and score.

To expand the searching scope and cover more types of content, we developed the global search tool. This tool unifies search mechanisms provided by the different collaborative software used on the SPB Portal. Any user can find a public content in the context of social networking, mailing list, and software repository.

Software Development Tools. Usually, Collaborative Development Environments (CDE) demand a version control system, trackers, build tools, knowledge centers, and communication tools [12]. The new SPB Portal also provides tools to encourage developers to keep the source code and its development activity within the platform. Any created software has, by default, an associated Git repository with Wiki pages and issue tracking. These tools are supplied by the integration of Gitlab into the platform.

Developers can also evaluate the software source code to measure software quality. With Mezuro, they can schedule the analysis of the source code and follow its metric results evolution over time. Results of each metric analysis are public, which allows greater transparency between the developer and the community that uses the software. Thereby, the maintainers can decide if the given solution meets the source code quality requirements.

Thus, the SPB Portal became a platform to stimulate the openness of the source code; dialogue between users and the development team; and also maintenance and evolution of the software, which provide more transparency in Government investments regarding to software development.

DEVELOPMENT ORGANIZATION AND PROCESS

The SPB team was composed of a variety of professionals with different levels and skills, where most of them were undergraduate students of software engineering. Since students could not dedicate many hours per week to the project, they had the flexibility to negotiate their work schedule during the semester in order to not harm their classes and coursework. Their work routine in the project included programming and DevOps tasks.

The project required a vast experience and background that usually undergraduate students do not have. For this reason, a few senior developers have joined the project to help with the more difficult issues and to transfer knowledge to the students. Their main task was to provide solutions for complex problems, working as developers. As these professionals are very skillful and the project could not fund full-time work for all of them, they worked part-time on the project. In addition, they lived in either different Brazilian states or other countries, which led much of the communication to be online.

In short, our work process was based on open and collaborative software development practices. The development process was based on the adaptation of different agile and FLOSS communities practices, with a high degree of automation resulting from DevOps practices. Thus, the work process was executed in a cadenced and continuous way.

Finally, the last group of actors of this project was composed of employees of the Brazilian Ministry of Planning, Development, and Management. All the project decisions, validations, and scope definitions were made by them. In this way, we incrementally developed a software product with releases aligned to strategic business objectives. As one can see, the project had a wide range of different stakeholders that had to be organized and synchronized.

Team Organization

Approximately 70% of the development team was composed of software engineering undergraduate students from UnB and they worked physically in the same laboratory. Each student had their own schedule based on her classes, what complicates the implementation of pair programming. The senior developers tried to synchronize their schedule with students schedules. To cope with this scenario, we had a few basic rules guiding the project organization:

1. Classes have high priority for undergraduate students;
2. Pairing whenever possible (locally or remotely);
3. We had one morning or afternoon per week when *everyone*, but the remote members, should be together physically in the laboratory;
4. Every 2 to 3 months the senior developers would travel to work alongside the students for a few days.

With the aforementioned rules, we divided all the project into four different teams: Colab, Noosfero, Design, and DevOps. One student of each team was the coach, responsible for reducing the communication problem with other teams and helping the members to organize themselves in the best way for everyone (always respecting their work time). The coach had also the extra duty of registering the current tasks developed in the sprint. One important thing to notice is the mutability of the team and the coach. During the project many students changed their teams to try different areas.

One characteristic of the teams was the presence of (at least) one senior per team. This was essential, because hard decisions and complex problems were usually referred to them. Thus, it was not the coach role to deal with complicated technical decisions, what encouraged students to be coaches. Lastly, the senior developers worked directly with the students, and this was important to give to students the opportunity to interact with a savvy professional in their areas and to keep the knowledge flowing in the project.

Finally, we had to add two more elements to the team organization that were essential for the project harmony: the meta-coach and professors. The former was a software engineer recently graduated that wanted to keep working on the project. The latter were professors that orchestrated all the interactions between all members of the project. The meta-coach usually worked in one specific team and had the extra task of knowing the current status of all teams. Professors and the meta-coach worked together to reduce the communication problem among teams. Lastly, all the paperwork tasks, such as reporting on the project progress to the Brazilian Government, was handled by professors.

Communication and Management

Our team had many people working together, and most of the seniors worked in different cities remotely. Also, we tried to keep our work completely clear to the Brazilian Government and citizens interested in following the project. To handle these cases, we used a set of communication and management tools.

For communication between members in different places we used: video conferencing with shared terminal tools, IRC, and mailing lists. For example, when one student had to work in pair with a senior, normally, they used video conferencing tool for talking and shared a terminal session (both typing and seeing each other screen in real time). For questions and fast discussion, we used IRC. For general notification, we used the mailing lists.

For managing the project, we used the SPB Portal itself; first to validate it by ourselves, and also because it had all the required tools. We basically created one Wiki page per release in the SPB Gitlab instance with a mapping between strategical, tactical, and operational views. We had one milestone per user history (feature) and one or more issues for addressing each feature. With this approach we achieved two important goals: keeping all the management as close as possible to the source code and tracking every feature developed during the project. Initially, our decision to use the Wiki was empirical, but later such decision was reinforced by a research conducted by Joseph Chao showing the advantage of using Wikis [2, 10].

High-level Project Management and Reporting

The Brazilian Government used to work with software development in a very traditional way. They would frequently focus on documents and not on what was, in our opinion, what really matters: working software. This dissonance caused us a communication noise with the Government, because they would often question our work style. It was especially hard to convince them to accept the idea of open scope and agile development, but after months of labor and showing results they stopped resisting.

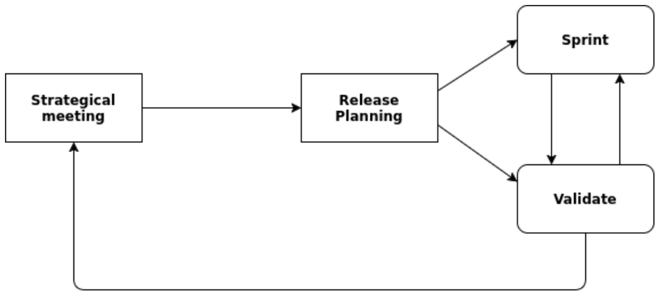

Figure 3. Meetings cycles.

We defined some level of meeting granularity to avoid generating too much overhead to the developers. We had a strategical and a validating meeting with the Brazilian Government (the former once in month and the latter biweekly), a release planning with the entire team (one per month), and finally a sprint planning (biweekly). Figure 3 is a diagram that represents our meeting organization.

In the strategical meeting we usually defined the priorities and new features with the Brazilian Government. Normally the professors, the coach of each team, the meta-coach, and some employees of the Federal Government would participate in this meeting. We usually discussed what the team already produced since our last meeting, and established the new features for the next release. Notice that just part of the team would join this meeting to avoid generating unnecessary overhead to developers, but all the students interested to participate were allowed to join, since many students wanted this experience during the project.

After the strategical meeting with Brazilian Government agents, we had a planning phase with all teams together. In this part, each team worked together to convert the Government wishes into smaller parts which were represented by the epics of the release. Each coach was responsible for conducting the planning and recording it on the project Wiki. With this epic, biweekly the team have documented their sprint schedule (with small achievements mapped to issues).

To keep the Brazilian Government always updated, we invited them to work with us to validate the new features in progress. Normally we had a meeting biweekly. Basically, this was our work flow. We always kept everything extremely open to the Government (our way of working, and the one often used by FLOSS projects) and to the team.

To keep the track of all of these things we used the SPB Portal itself, especially Gitlab. Basically, we had:

1. Project repository: we have one organization with many repositories;
2. Wiki: each release has one Wiki page with the compilation of the strategical meeting;
3. Milestones: each milestone was used to register a user story (feature);
4. Issues: each sprint planning generated issues, which we associated to the related milestone (feature as user story) and registered on the related Wiki page. Finally, each developer assigned the issue to herself.

Notice that this workflow gave us and the Brazilian Government agents full traceability from a high level view of each feature to the lowest level (source code). It is important to highlight that we converged to this workflow after many experiments. For example, we used a tool named Redmine for organizing our tasks during some sprints. However, this tool revealed to be inefficient for our case since the government agents lost part of the project traceability. We realized that centralizing all the work in the SPB Portal was the best option for our case.

CONCLUSION

In this work, we presented and discussed issues experienced during a government-funded project, in partnership with the University of Brasilia and the University of São Paulo, to evolve the Brazilian Public Software Portal. Our contributions are twofold. First, we present the strategy used to develop and to deliver an unprecedented platform to Brazilian government. Second, based on the results of the SPB Portal project, we point out that it is possible to mitigate conflicts experienced in the development environment and to conciliate governmental and academy cultures. To summarize our main contributions, we answered in this section the three open questions those guided this paper.

Which strategy could be used to integrate several existing FLOSS tools to promote a collaborative software development? The SPB portal integrates more than ten FLOSS tools and provides several features, such as social network, mailing list, version control, content management and source code quality monitoring. Concerned with the platform sustainability and maintainability, the aforementioned FLOSS tools were integrated with minimum differences from their official versions and the new developed features were sent upstream to ensure an alignment between the portal systems and their respective official versions. In the integration process, the main software were identified, specific teams were formed to work with each one of them and each team was composed of students with different levels of skills and at least one senior professional.

How to involve students in real-world projects interacting with real customers? In terms of mitigating conflicts, we tried to show that, as long as the university can provide a healthy and challenging environment to its students, one may conciliate studies and professional training in universities. The students interacted with professionals of diverse fields of expertise, and they were able to participate in all levels of the software development process. This contributed to a great learning opportunity. In our work process, based on open and collaborative software development practices, students could negotiate their work schedule as well as count on IT professionals to solve development issues. Among the students, we have defined coaches for each team and a meta-coach (coach of whole project). All coaches, together with professors, have intermediated the communication between client (the Brazilian Government) and the rest of the group. After the end of the project, some students successfully embraced opportunities in public and private sectors, within national borders and abroad. Some other students went further and started their own companies.

How to introduce typical FLOSS collaborative and agile practices in the governmental development process? With some adaptations, it is feasible to conciliate agile methodologies and FLOSS practices to develop software to governmental organizations with functional hierarchical structures that use traditional development paradigm. Aiming at reducing client questions about workconclusion, a DevOps team was created to automate all deploy process and also to work in continuous delivery. The Government was brought to our work environment and interacted with our management and communication tools. For the project success, we focused on providing a friendly working environment as well as on showing to governmental agents another way to interact with the FLOSS community and the university.

Lessons Learned

From the answers of our initial open questions, we can also highlight six lessons learned to better share our experience during the development of the new SPB Portal.

The participation of experienced professionals is crucial to the success of the project. One important factor for the students was the composition of the teams with the participation of experienced professionals. On the technical side, the senior developers and designers would handle the more difficult technical decisions, creating a work environment where the students could develop their skills in a didactic way without pressure. On the management side, the active participation of professors – who are, in the end, the ones responsible for the project – is crucial to make sure students participation is conducted in a healthy way, and it is an instance of leading by example.

A balanced relationship between academia and industry. The experience of the SPB project led UnB to develop a work style which proved to be appropriate for an educational environment that brings academia and industry together. The highest priority from the university's point of view is the students. Considering this, the activities of the project were never prioritized to the detriment of classes and other pedagogical activities. In summary, we had students working at different times, part time, remotely or locally, always respecting their individual conditions, but doing the work in a collective, collaborative and open way. And even under a potentially adverse environment, the project delivered the desired solution with success.

Managing different organizational cultures. In the beginning of the project, the Brazilian Government stakeholders had certain expectations about the project development that, let's say, did not exactly match our work style based on agile and FLOSS practices. We had to develop strategies that would support these different organizational cultures. Therefore, we have adapted the process between our team and the Government managers who managed the project on their side, assuming a traditional waterfall process.

Managing higher-level and lower-level goals separately. During the initial phase of the project, the Brazilian Government team has brought strategic discussions to technical/operational meetings that were supposed to be about practical technical decisions. This produced a highly complex communication and management environment, overloading the professors, who were supposed to maintain the Government strategy synchronized with the implementation plans of the development team. This was hard, especially because the aforementioned cultural mismatch. Mixing both concerns in the same discussions caused confusion on both sides. From the middle of the project we were able to keep those concerns separated, what eased the work of everyone involved.

Living with ill-advised political decisions. At the initial phases of the project, by political and personal motivation, the main stakeholders from the Brazilian Government imposed the use of Colab to guide the development of the new SPB platform. Our team was totally against the idea because we already knew that Colab was a very experimental project and its adoption could dramatically increase the project complexity. Even though, we provided technical reasons to not utilize Colab, the Government was adamant and we had to manage this problem. We did massive changes to Colab, and by the end of the project we had completely rewritten it to make it stable. It is important to notice that the Government compelled us to accept a technical decision based only on political interests, without considering all the resources that would be spent due to that decision. At the end of the project, we verified that Colab indeed consumed a vast amount of the budget and increased the project complexity. After our analysis on the decision made by the Government, we understand that some Brazilian Government managers are capable of ignoring technical reasons in favor of political decisions.

Consider sustainability from the beginning. In the process of deploying the SPB platform in the Brazilian Government infrastructure we had to interact with the Government technicians. We did several workshops, training and a meticulous documentation describing all the required procedures to update the platform, however, we realized that the technicians would constantly avoid the responsibility. After noticing the aforementioned situation, we organized a DevOps team that completely automated all the deployment procedure. We simplified all the platform deployment to a few steps: (i) initial configurations (just ssh configuration) and (ii) the execution of simple commands to completely update the platform. By the end of the project, we observed that the Government technicians invariably still depended on our support to update the platform even with all the automation provided by us. We were sadly left with a feeling of uncertainty about the future of the platform after the project ended. In hindsight, we realize that the Brazilian Government dedicated system analysts and managers to the project, but not operations technicians. The later should have been more involved with the process so they could at least be comfortable in managing the platform infrastructure.

Final Remarks and Future Work

The SPB portal is in production[13] and its full documentation, including detailed architecture and operation manuals, is also available[14]. All the integrated tools are FLOSS and our contributions were published in open repositories, available on the SPB Portal itself. We also contributed these features back to the respective communities, which benefits both those communities and us, since we can share future development and maintenance effort with other organizations that participate in these projects.

Future work should use data produced by the project to validate and evaluate how the used FLOSS and Agile practices have impacted the students and the governmental development process. For this, we will conduce a *postmortem* analysis using the project open data and a survey targeting the involved stakeholders.

REFERENCES

1. Grady Booch and Alan W. Brown. 2003. Collaborative Development Environments. *Advances in Computers* 59 (2003), 1–27. DOI: http://dx.doi.org/10.1016/S0065-2458(03)59001-5
2. Joseph Chao. 2007. Student project collaboration using Wikis. In *Software Engineering Education & Training, 2007. CSEET'07. 20th Conference on*. IEEE Computer Society, 255–261. http://dblp.uni-trier.de/db/conf/csee/csee2007.html
3. Gregory DeKoenigsberg. 2008. How Successful Open Source Projects Work, and How and Why to Introduce Students to the Open Source World.. In *CSEET*, Hossein Saiedian and Laurie A. Williams (Eds.). IEEE Computer Society, 274–276. http://dblp.uni-trier.de/db/conf/csee/csee2008.html
4. Nicolas Ducheneaut. 2005. Socialization in an Open Source Software Community: A Socio-Technical Analysis. *Computer Supported Cooperative Work* 14, 4 (2005), 323–368. DOI: http://dx.doi.org/10.1007/s10606-005-9000-1
5. Fabian Fagerholm, Alejandro S. Guinea, Jürgen Münch, and Jay Borenstein. 2014. The Role of Mentoring and Project Characteristics for Onboarding in Open Source Software Projects. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '14)*. ACM, New York, NY, USA, Article 55, 10 pages. DOI: http://dx.doi.org/10.1145/2652524.2652540
6. Fabian Fagerholm, Patrik Johnson, Alejandro Sánchez Guinea, Jay Borenstein, and Jürgen Münch. 2013. Onboarding in Open Source Software Projects: A Preliminary Analysis. *CoRR* abs/1311.1334 (2013). http://dblp.uni-trier.de/db/journals/corr/corr1311.html
7. C.S. Freitas and C. Meffe. 2008. *FLOSS in an Open World: best practices from Brazil*. Roadmap white paper. Paris, France. 69–73 pages. https://www.pilotsystems.net/actus/2020-floss-roadmap.pdf In: 2020 FLOSS Roadmap.
8. Annemarie Harzl. 2017. Can FOSS projects benefit from integrating Kanban: a case study. *Journal of Internet Services and Applications* 8, 1 (06 Jun 2017), 7. DOI: http://dx.doi.org/10.1186/s13174-017-0058-z
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 (mar 2003), 209–223. DOI: http://dx.doi.org/10.1287/orsc.14.2.209.14992
10. Eirini Kalliamvakou, Daniela E. Damian, Kelly Blincoe, Leif Singer, and Daniel M. Germán. 2015. Open Source-Style Collaborative Development Practices in Commercial Projects Using GitHub. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*. 574–585. DOI: http://dx.doi.org/10.1109/ICSE.2015.74
11. Fabio Kon, Paulo Meirelles, Nelson Lago, Antonio Terceiro, Christina Chavez, and Manoel Mendonça. 2011. Free and Open Source Software Development and Research: Opportunities for Software Engineering.. In *SBES*. IEEE Computer Society, 82–91. http://dblp.uni-trier.de/db/conf/sbes/sbes2011.html
12. Filippo Lanubile, Christof Ebert, Rafael Prikladnicki, and Aurora Vizcaíno. 2010. Collaboration Tools for Global Software Engineering. *IEEE Software* 27, 2 (2010), 52–55. DOI: http://dx.doi.org/10.1109/MS.2010.39
13. Audris Mockus, Roy T. Fielding, and James D. Herbsleb. 2002. Two case studies of open source software development: Apache and Mozilla. *ACM Trans. Softw. Eng. Methodol.* 11 (July 2002), 309–346. Issue 3. DOI: http://dx.doi.org/10.1145/567793.567795
14. Jan-Philipp Steghöfer, Eric Knauss, Emil Alégroth, Imed Hammouda, Håkan Burden, and Morgan Ericsson. 2016. Teaching Agile: Addressing the Conflict Between Project Delivery and Application of Agile Methods. In *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE '16)*. ACM, New York, NY, USA, 303–312. DOI: http://dx.doi.org/10.1145/2889160.2889181
15. Davide Tosi, Luigi Lavazza, Sandro Morasca, and Marco Chiappa. 2015. Surveying the Adoption of FLOSS by Public Administration Local Organizations.. In *OSS (IFIP Advances in Information and Communication Technology)*, Ernesto Damiani, Fulvio Frati, Dirk Riehle, and Anthony I. Wasserman (Eds.), Vol. 451. Springer, 114–123. http://dblp.uni-trier.de/db/conf/oss/oss2015.html

[13]https://softwarepublico.gov.br
[14]https://softwarepublico.gov.br/doc/