Continuous Delivery:
Building Trust in a Large-scale, Complex Government Organization

Authors

Rodrigo Siqueira is a Computer Science MSc student at the Institute of Mathematics and Statistics of the University of São Paulo (IME-USP). His research interests include Software Engineering, Operating System, and Computer Architecture. He worked for two years with the SPB project as a coach and developer. Contact him at siqueira@ime.usp.br.

Diego Camarinha is a Computer Science MSc masters student at IME-USP. His research interests include Software Engineering, Computer Networks, and Source Code Metrics. He worked for two years with the SPB project as a senior developer. Contact him at diegoamc@ime.usp.br.

Melissa Wen is a Bachelor of Computer Science from Federal University of Bahia (UFBA). She worked on the SPB project as a senior developer and was core developer of Noosfero social networking platform at Colivre. Her areas of interest include Software Engineering and Free Software Development. Contact her at melissa.srw@gmail.com.

Paulo Meirelles is a Professor of Software Engineering at the University of Brasilia (UnB) and researcher at the Free/Libre/Open Source Software Competence Center (CCSL) at IME-USP. He was the head of the new SPB portal development. His research interests include Free Software development, Agile Software Development, and Source code metrics. Contact him at paulormm@unb.br.

Fabio Kon is a Full Professor of Computer Science at IME-USP, co-founder of the CCSL at IME-USP, and Editor-in-Chief of the SpringerOpen Journal of Internet Services and Applications. His research interests include Agile Software Development, Smart Cities, and Distributed Systems. Contact him at fabio.kon@ime.usp.br.

Abstract

For many software development teams, the first aspects that come to mind regarding Continuous Delivery (CD) are the operational challenges and the competitive benefits. In our experience, CD was much more: it was a survival technique. This article presents how and why we applied CD in a large governmental project for the development of a Collaborative Development Environment. We share the challenges we faced and the strategies we used to overcome them. The article concludes with a set of lessons learned that can be valuable for readers in a variety of situations.

Introduction

From 2014 to 2016, we worked on a thirty-month-long Brazilian government project to modernize the Brazilian Public Software (SPB) portal (www.softwarepublico.gov.br) [1]. This project was a partnership between the *Ministry of Planning, Budget, and Management (MPOG)* and two public universities: University of Brasília and University of São Paulo.

During this time, the SPB portal evolved into a Collaborative Development Environment (CDE), which brought significant benefits for the government and the society. The government could minimize bureaucracy and software development costs, by reusing the same set of applications across different government Agencies. Society could more transparently follow government expenses and contribute to software project communities.

In this article, we discuss the use of Continuous Delivery (CD) during our experience as the academic partner in this project. We show how we implemented CD in a large institution with traditional, waterfall-like values and how CD helped to build trust between the government and the university team. CD enabled us to show our progress and to earn the government's confidence that we could adequately fulfill their requests, becoming an essential aspect of our interaction. According to this experience, the use of CD as a tool to build trust relationships is yet another of its benefits [2,3].

Context

SPB is a governmental program created to foster sharing and collaboration on Open Source Software (OSS) development for the Brazilian public administration. The government managed both software requirements and server infrastructure. However, its hierarchical and traditional processes made them unfamiliar with new software development techniques, such as CD. All our requests had to pass through layers of bureaucracy before being answered; accessing their infrastructure to deploy software was not different. The difficulties were aggravated because the new SPB portal is an unprecedented platform in the Brazilian government, with a complicated deployment process [4].

The project suffered significant interference from the Board of Directors throughout time because the portal represents an interface between government and society. In light of political interests, directors continually imposed changes to the platform while ignoring our technical advice. In 2015, the Board of Directors was changed and, with it, the vision of the project. New directors had different political agendas, which affected the project's requirements previously approved.

In this context, we overcame two distinct challenges: (1) deconstructing the widespread belief among government staff that any project in partnership with a University is doomed to fail, and (2) dealing with bureaucracies involved in the government deployment process.

First, our development team was not the typical one, consisting mainly of undergraduate interns supported by senior developers and designers, all coordinated by two professors. Our unconventional team structure and organization was considered as "unprofessional" by government standards with regard to time and resource allocation, accountability, and team continuity. On the government side, the SPB portal evolution was the first software development collaboration involving universities and the MPOG staff, raising disbelief.

Second, our team approached software deployment differently from the government. We believed frequent delivery is better for the project's success. In contrast, the MPOG is used to the idea

of a single deployment at the end of the project, and neither their bureaucratic structure nor their technical expertise was conducive to this style of work. That was hampering the benefits of the tool and preventing us from showing off the fruits of the project to those responsible for evaluating it.

These challenges made our relationship with their staff strained, in particular during the first year, and alerted us to the fact that they could cancel the project at any time. The deployment limitation was the substantial technical issue we could tackle in the short term. Thus, we worked to deploy the software into our infrastructure and showed it to the government evaluators. This strategy proved them we could efficiently provide new features, fulfill their requirement delivery expectations, and incited them to demand that the latest version be deployed in their infrastructure. Our CD approach generated more pressure on the IT department responsible for the deployment routines. With each CD cycle, we gradually built a new relationship among all parties and, by the end of the project, we became active participants in the deploy operations delivering quality software very frequently.

Our Continuous Delivery Pipeline

The SPB portal is an open CDE with additional social features, having 83 software communities and 6,460 user accounts, mostly from government employees. We built a system-of-systems [5] adapting and integrating five existing OSS projects: Colab (www.github.com/colab), Noosfero (www.noosfero.org), Gitlab (www.gitlab.com), Mezuro (www.mezuro.org), and Mailman (www.list.org). Colab orchestrates these multiple systems using a plugin architecture and allowed us to smoothly provide a unified interface to final users, including single sign-on and global searches [1]. All these integrated systems involve a total of 106,253 commits and 1,347,421 lines of code.

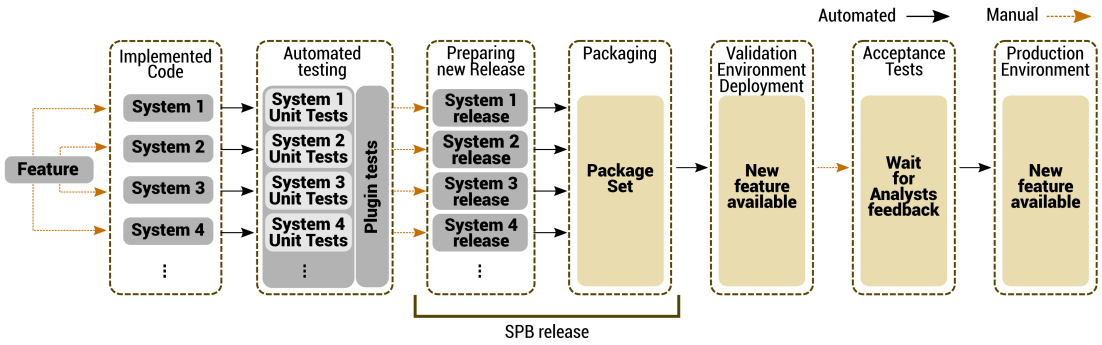

Figure 1: The SPB Deployment Pipeline.

Portal deployment follows a typical CD pipeline [6], adapted to our technical and organizational context and the use of OSS best practices. As depicted in Figure 1, it begins when a new feature is ready for deployment and ends when it reaches production.

Automated Tests

Each integrated system is a Colab plugin and had to be tested with its own test suite. In addition, we also had to test the platform as a whole. Since the plugins have their own test suites, they also assume a double role as both plugin tests and as integration tests. If any test suite failed, by either a test error or coverage reduction below a certain threshold, the process stopped. Only when all tests passed, the pipeline proceeded to the next step.

Preparing a New Release

A separate Git repository hosted each system. New software component releases were tagged referencing a specific new feature or bug fix. SPB had its own Git repository (www.softwarepublico.gov.br/gitlab/softwarepublico). An SPB portal release was an aggregate of all its systems. When a new component release passed all of the SPB integration tests, we manually created a corresponding new tag in its repository. At the end of this process, we started packaging.

Packaging

After creating a new tag, our DevOps team started the packaging process. Packaging brings portability, simplifies deployment, and enables configuration and permission control. Our approach involved building separate packages for each system, in three fully automated steps: generating scripts for the specific environment, building the package, and uploading it to a package repository. When all ran successfully, the new packages would be ready and available for our deployment scripts.

Validation Environment

The Validation Environment (VE) is a replica of the Production Environment (PE) with anonymised data, and access restricted to MPOG staff and our DevOps team. To configure this environment, we used Chef (www.chef.io) and Chake, a serverless configuration tool created by our team (www.github.com/terceiro/chake) to maintain environment consistency, simplifying the deployment process.

Acceptance Tests

After a new SPB portal VE deployment, we used the environment to verify the integrity of the entire portal. MPOG staff also checked the new features, required changes, and bug fixes. If they identified a problem, they would notify developers via comments on the SPB portal issue tracker, prompting the team to fix it and restart the pipeline. Otherwise, we could move forward.

Production Environment Deployment

After the VE check, we could finally begin the deployment in production, with the same configuration management tool, scripts, and package versions as in the VE. After the deploy was

completed, both VE and PE were identical, making new features and bug fixes available to end users.

Benefits

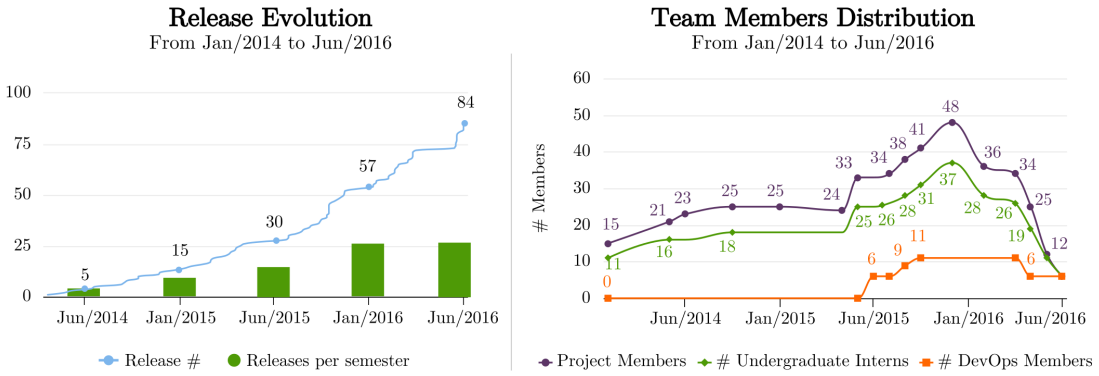

Figure 2: The evolution of SPB releases and the development team members distribution.

CD brings many advantages such as accelerated time to market, building the right product, productivity and efficiency improvements, stable releases, and better customer satisfaction [2,3]. The charts presented in Figure 2 show these benefits in our project and associates them with the creation of a DevOps team. Over 30 months, we deployed 84 versions. Over 64% of the releases happened in the last 12 months, after the creation of the DevOps team. Besides these results, working with the government we noticed the following additional benefits.

Strengthening Trust in the Relationship with the Government

CD helped strengthen trust between developers and the MPOG staff. Before using CD, they could validate features developed only at the end of the release cycle, usually every four months. With the implementation of CD, intermediate and candidate versions became available, allowing them to perform small validations over time. Constant monitoring of the development work brought greater assurance to the MPOG leaders and improved the interactions with our team.

Responsiveness to Change

Responsiveness was one of the direct benefits of adopting the CD pipeline. Political forces may change requirements and priorities at any time. The ability to react quickly to changes requested by the government was vital to the project survival for 30 months. We noticed that if we took too long to meet their demands, they could reduce financial support and even cancel the project.

CD helped us keep the PE up-to-date, even with partial versions of a feature. Therefore, we always had new results to present on meetings, easing their concerns about expected final delivery. For our team, CD made developers always confident that the project would last a little longer.

Shared Responsibility

According to the conventional MPOG process, the development team could not track what happened to the code after its delivery, since their employees were the only ones responsible for deployment. The implementation of CD made our development team feel equally responsible for what was getting into production and take ownership of the project [4].

Interestingly, the CD pipeline had the same effect on the MPOG staff. They became more engaged in the whole process, opening and discussing issues during the evolution of the platform. Additionally, developers worked to improve the CD pipeline and speed up the process of making new features available in the production environment.

Synchronization Between Government and Development

The CD pipeline performance depended on the synchronization between our development team and the MPOG staff: each party had to be prepared to take action as soon as the other concluded a given task. Initially, the MPOG staff did not contemplate this in their schedule which, combined with the bureaucracy in providing access to the PE (up to 3 days), resulted in significant delays in the validation of new features. The use of an explicit CD pipeline helped us to identify critical points of delay, and increase our productivity.

Lessons Learned

Due to the successful building of the CD pipeline, we not only overcame the challenges we described above but also improved the MPOG deployment process and kept the project alive until its successful conclusion. We now look at the lessons we learned, which can be leveraged by readers in other situations.

Build CD From Scratch

Taking on the responsibility for implementing CD impacted the whole team. Most of our team members did not have CD know-how, and we had few working hours available to build the initial version of the pipeline. The construction and maintenance of the CD process were made possible by the key decisions to:

1. *Select the most experienced senior developers and some advanced interns to work on a specific DevOps team.* These senior developers used their experience in OSS projects to craft an initial proposal for the deployment process. The solution enabled us to automate the deployment, even though the process was, initially, still rudimentary.
2. *Interchange team members and encourage teammates to migrate to the DevOps team.* The benefits were twofold: mitigating the difficulty in sharing knowledge between DevOps developers and feature developers, and evolving the process on-the-fly.

Overcoming Mistrust

Adopting an unfamiliar approach requires trust. The government staff, traditionally, expected and were prepared to validate and deploy a single deliverable. However, the steady growth of SPB

complexity made large deliveries unsustainable. The CD approach was necessary [4]. Therefore, we developed the following line of action to enable this new dynamics:

1. *Demonstrate actual results, instead of simply reporting them.* Initially, we did not have access to the government infrastructure, so we created our own validation environment. Thus, we were able to follow the CD pipeline until production deployment, when we faced two problems. First, our pace of intermediate deliveries was faster than the deployment to production by the MPOG staff. Second, specific issues of the governmental infrastructure made some validated features not work as expected in the PE. That situation gave us arguments to negotiate access to the PE.

2. *Make project management transparent and collaborative for government staff.* Allowing the MPOG staff to track our development process showed them we were fulfilling our commitments. They started to interact more actively in the generation of versions and became involved in the CD. After understanding it, the government staff helped us negotiate access to a VE with the MPOG leaders, creating an isolated replica of the PE.

3. *Gain the confidence of government staff.* With the PE replica, we were able to run the entire pipeline and win the trust of the MPOG staff involved in the process. They saw the mobilization and responsiveness of our team to generate each new version package. They also recognized the quality of our work and deployment process. In the end, the government staff realized that it would be beneficial for the project if they granted us access to the infrastructure, both VE and PE.

In summary, we encourage the use of a well-thought CD pipeline as a means to gain trust in software development projects with government and large organizations as well.

References

1. P. Meirelles, M. Wen, A. Terceiro, R. Siqueira, L. Kanashiro, and H. Neri, "Brazilian Public Software Portal: an integrated platform for collaborative development", OpenSym, 2017.
2. L. Chen, "Continuous Delivery: Huge Benefits, but Challenges Too", IEEE Software, 32 (2) 2015.
3. T. Savor, M. Douglas, M. Gentili, L. Williams, K. Beck and M. Stumm, "Continuous Deployment at Facebook and OANDA", ICSE, 2016.
4. M. Shahin, M. A. Babar, and L. Zhu. "The Intersection of Continuous Deployment and Architecting Process: Practitioners' Perspectives". ESEM, 2016.
5. C. B. Nielsen, P. G. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska, "Systems of Systems Engineering: Basic Concepts, Model-Based Techniques, and Research Directions", ACM Comput. Surv. 48, 2, 2015.
6. J. Humble and D. Farley, "Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation", Addison-Wesley, 2010.