Scaling and Internationalizing an Agile FOSS Project: Lessons Learned

Stephan Fellhofer, Annemarie Harzl, Wolfgang Slany

Scaling & Internationalizing an Agile FOSS Project: Lessons Learned

Stephan Fellhofer, Annemarie Harzl, and Wolfgang Slany

Institute for Software Technology, Graz University of Technology,
Inffeldgasse 16b/II, 8010 Graz, Austria
stephan.fellhofer@gmail.com, annemarie.harzl@ist.tugraz.at,
wolfgang.slany@tugraz.at

Abstract. This paper describes problems that arose with the scaling and internationalization of the open source project Catrobat. The problems we faced were the lack of a centralized user management, insufficient scaling of our communication channels, and the necessity to adapt agile development techniques to remote collaboration. To solve the problems we decided to use a mix of open source tools (Git, IRC, LDAP) and commercial solutions (Jira, Confluence, GitHub) because we believe that this mix best fits our needs. Other projects can benefit from the lessons we learned during the reorganization of our knowledge base and communication tools, as infrastructure changes can be very labor-intensive and time-consuming.

Keywords: agile development, Kanban, distributed software development, documentation management, communication, scaling, internationalization

1 Introduction

Scaling and internationalizing a Free and Open Source Software (FOSS) project is not an easy task, at least not in our experience. Our project grew from five contributors in 2010 to over 130 in 2014, which lead to various organizational problems. In this paper we describe our problems, approaches we devised to solve them, and lessons learned along the way. We think that other FOSS projects can profit from our experiences.

In our project Catrobat we develop a visual programming language which has been inspired by Scratch [11] from the Lifelong-Kindergarten-Group at the MIT Media Lab. Our aim is to empower children and teenagers to easily create their own programs and to express themselves creatively using their smartphones. To reach this goal we develop Integrated Development Environment (IDE) and interpreter apps natively for several mobile platforms. Our app, Pocket Code, is available for Android on Google Play[1]. Versions for iOS, Windows Phone, and HTML5 capable browsers are currently in development. Catrobat was initiated

[1] https://play.google.com/store/apps/details?id=org.catrobat.catroid

by Wolfgang Slany and a team of students from Graz University of Technology, seeking development challenges beyond those seen in class and to practice what they have learned. This eagerness to apply software development principles taught in courses heavily influenced the basic structure of the project, with all its advantages and disadvantages.

On the positive side the usage of agile development methods such as Kanban [2, 5] enables us to stay flexible and to easily adjust the scope of our project as we go. eXtreme Programming (XP) (especially pair programming [3]) facilitates knowledge transfer between developers. Test-Driven Development (TDD) [4] ensures that the code remains functional and testable while new developers join and former developers leave the project. On the negative side our contributors additionally have to learn about agile development methods, which steepens the learning curve.

A dedicated Usability and User Experience (UX) team applying the personas method [6] and other usability techniques helps us focus on the users. This is particularly important because, in contrast to most FOSS projects, Catrobat's developers (mostly university students) are not a subgroup of Catrobat's targeted users (mostly children and teenagers). The UX team helps to create an understanding of user needs in all developing teams.

Communication within our teams was initially mainly face-to-face, which is good for localized agile teams, but leaves many decisions undocumented, which is disadvantageous for a larger, distributed FOSS project. Discussions tend to get started over and over again, when nobody remembers why and based on what information a decision was made in the first place. Later we will elaborate on the communication problems, because they are at the core of our difficulties with scaling and internationalizing.

The increasing number of contributors from five in 2010 to over 130 in 2014, and the participation of international contributors in our project made organizational problems apparent. Documentation was not equally available for everyone, the entire current project status was not visible online and we lacked important communication channels. For example most of our contributors did not use Internet Relay Chat (IRC), which is often an integral infrastructure of FOSS projects and compulsory for the participation in Google Summer of Code (GSoC)[2], in which our project participates since 2011. To address these problems and to be able to integrate more and international contributors, we had to change our project infrastructure, the ways we communicate and some of our tools.

Section 2 gives a short overview of work related to the approaches we formulated. In Section 3 we will identify the problems in greater detail and describe our approaches to solve them. Section 4 contains the lessons we learned on our way from a small, localized to a larger, more international project. Subsequently we discuss possibilities for future work and provide some concluding remarks.

[2] A global program from Google to support FOSS projects by sponsoring *students* and pairing them with *mentors* to develop given tasks over summer (`https://developers.google.com/open-source/soc/`)

2 Related Work

Our main goal was to enable and facilitate contributions from project members around the world. Various studies ([7, 9, 10, 13, 14]) highlight the importance of communication in FOSS projects or in projects which use agile development techniques.

We focused on optimizing the communication for FOSS and agile software development projects. Specifically, we tried to address some of the issues and approaches highlighted in the literature: Layman et. al [9] recommended among other things that *"when face-to-face, synchronous communication is infeasible, use an email listserv"* and *"use globally-available project management tools"*. Korkala et. al [7] suggested to *"enable and support direct communication between the developers"*. Some technologies and common practices used in FOSS development such as instant messaging, IRC, news postings, how-to guides, Frequently Asked Questions (FAQ), or wikis are listed in [12, 13]. In [16] technologies such as versioning systems and TODO lists are mentioned. Difficulties for newcomers like *"selection of a suitable task"*, *"lack of up-to-date development documents"*, or *"no response from core developers for their doubts"* are mentioned in [15].

3 Optimizing Services for Distributed Participation

Our project grew faster than the supporting infrastructure, which led to organizational problems. For each part of our infrastructure we will first describe the initial situation, then problems which occurred over time and finally how we resolved them.

3.1 User Management

Initial Situation. In the beginning there was no consistent user management. Every piece of infrastructure (for example: instant messaging, source code repository) had its own built-in user management, and accounts were created manually on demand. Since some services were used through shared user accounts, there was no accountability.

Resulting Problems. The effort necessary for user management and maintenance increased tremendously with the growing number of contributors, because every user had to be created manually and every change had to be populated manually to all platforms. This resulted in missing and outdated account information. Rights management was not even a topic. Sometimes this lack of restrictions caused inexperienced contributors to inadvertently change or delete infrastructure. Shared accounts made it impossible to trace who made changes to project services. On the side of the contributors the account management was elaborate too, as for every service, contributors had to use different credentials.

Method of Resolution. Our goal was to simplify the management of user databases and to support the contributors by providing them with only one account for (almost) all our infrastructure. Most parts of our infrastructure have a built-in support for Lightweight Directory Access Protocol (LDAP), so LDAP was the most suitable solution to simplify our user management. We decided to use LDAP groups as well for various reasons:

- different experience levels need different rights
- different contributor groups need different resources and services
- no shared accounts, so there are clear responsibilities
- infrastructure administration should be left to experienced contributors

The goal was to design a user management which serves experts and beginners alike. Experts should have all the rights they need, while beginners are not overwhelmed by too many services and rights too soon.

Unfortunately, not all services support LDAP. For example, externally hosted services such as GitHub do not support foreign user directories and still need additional maintenance. Other services like Crowdin[3] support OAuth[4] as authentication method, but to take the user groups and corresponding rights needed for our project into account, the service's Application Programing Interface (API) or an additional configuration interface must be used.

3.2 Communication

Our project was and still is allowed to use a room at Graz University of Technology, where our local contributors can meet, discuss, and code. In the beginning all contributors fit in the room, and communication was mainly face-to-face. The reliance on face-to-face was very beneficial in the beginning but caused some communication and documentation problems later on as mentioned in Section 1.

Instant Messaging

Initial Situation. Other means of communication were and still are e-mail, mailing lists[5] as recommended in [9] and Instant Messaging (IM). In the beginning we used a Skype group chat for project discussions with all contributors.

Resulting Problems. Many contributors joined the Skype group chat but did not participate actively, because they preferred face-to-face communication, e-mail, or were overwhelmed by the number of messages that did not concern them. This massive number of messages was a direct result of the growing number of contributors.

Another problem with Skype was that group chats are invite-only, which made it difficult for aspiring contributors to join the discussion. They first had

[3] Tool to help non-developers to translate texts (`https://crowdin.com/`)
[4] `http://oauth.net/`
[5] `https://groups.google.com/forum/?hl=en#!forum/catrobat`

to find a project member to invite them. Yet another problem with our Skype group was the language used. Almost all messages were in German, because all of the initial team members spoke German.

In our attempt to open up our project and allow for internationalization, we created an IRC channel, which was open to everyone and where it was obligatory to use English.

This attempt failed, because project issues were, as a matter of habit, still discussed in Skype and hardly anyone used IRC. So theoretically we maintained an IRC channel, but interested contributors still got *"no response from core developers for their doubts and support request"* [15].

Method of Resolution. We decided to switch our whole synchronous communication to IRC and deleted the Skype group chat. The reasons for this decision were:

- one IM platform for all purposes
- anybody can join the channel he or she is interested in
- 'irrelevant' messages are reduced, because messages are posted only to the channels where they belong
- faster responses to questions from aspiring contributors due to increased online time of project members
- topic specific channels can be created and deleted easily, when needed

To make the communication with IRC more attractive for our contributors we provide them with an IRC bouncer which records all messages when the user is offline and replays them when the user goes online. In our attempt to minimize the amount of messages every contributor has to read, we decided to split the conversation into different channels. There exist separate channels for subteams, technical support teams (for example for our continuous integration server) and one channel for general information and announcements. Contributors may still miss important information, if they do not join all channels, but the risk of information overload should be diminished.

One disadvantage of IRC as communication platform is that the technology seems old-fashioned to our contributors and most of our contributors have never used IRC before. Another disadvantage is that it is more time consuming to configure IRC with the bouncer than it is to configure Skype. Contributors have to authenticate to freenode[6] and to the project bouncer with different credentials.

Today IRC is widely accepted by our community and the communication improved compared to Skype because contributors only have to join and read channels they are interested in and people are by now used to IRC.

3.3 Agile Development Management

Initial Situation. As already mentioned we were and still are allowed to use a room at the university for our project. There, we have whiteboards serving as

[6] IRC network where our project runs all its channels (`https://freenode.net/`)

Kanban boards, as well as story cards on paper at our disposal. Initially every subteam had its own Kanban board in the room, though this became impossible as the number of subprojects grew. Our project used and still uses GitHub[7] as source repository. Previously, we used the integrated issue tracker not just to track bugs and issues reported by users but also as a digital version of our local Kanban boards. Labels of issues were used to indicate which kind of issue it was (*bug, story, enhancement, ...*), the current working status (*to do, in development, done, ...*), the priority (*low, medium, high,* or *critical*) and which part it affected (*development environment* or *interpreter*). To enable children and teenagers to report bugs and issue requests without bothering with Github, we created a Google Group[8].

Resulting Problems. Our user stories and bug reports had to be synchronized between three different platforms, namely the local Kanban board, Github, and the Google Group. It is not hard to imagine that this had negative consequences: the local Kanban board was often outdated and the issue tracker did not cover stories, which were created locally on the Kanban board. The Github issue tracker did not support our Kanban board and lacked the functionality of hiding security relevant issues.

Method of Resolution. We decided to switch to an online distributed agile development environment. There exist various tools and we did a comparison of some, namely GitHub, Bugzilla[9], Redmine[10], and Jira[11]. We compared them based on the following criteria:

- they should support LDAP, virtual whiteboards, and different layers of visibility for security relevant issues
- the workflow should be customizable to our needs
- the tool should be expandable through add-ons
- means for reporting and analysis should be integrated
- a wiki system should be integrated because the old one needed to be replaced (for further details see Section 3.4)

We compared the basic versions of the different tools, without considering all the available add-ons, because add-ons can be more easily abandoned by their developers than complete tools, and we can not afford commercial add-ons. Table 1 shows the results of our comparison. Based on this comparison we decided to try Jira, because together with Confluence[12], it would satisfy all our criteria and both tools are free for FOSS projects.

[7] `https://github.com/Catrobat`
[8] `https://groups.google.com/forum/?fromgroups=#!forum/pocketcode`
[9] `https://www.bugzilla.org/`
[10] `http://www.redmine.org/`
[11] Planning and tracking tool for agile project management by Atlassian (`https://www.atlassian.com/software/jira`)
[12] Wiki system by Atlassian which is free for FOSS projects - `https://www.atlassian.com/software/confluence`

As a pilot test we used Jira during GSoC 2013. Every mentor/student pair received their own project with a simple Kanban board. Over this trial period we realized we had to customize the original workflow of issues to fit our developing principles (specifically review of newly created issues and code acceptance by experienced developers) and to integrate usability reviews into the workflow.

After the successful pilot test we decided to introduce Jira as a globally-available project management tool (recommended by [9]) and as replacement for the local whiteboards. This proved to be another important step towards becoming an international FOSS project.

	GitHub	Bugzilla	Redmine	Jira/Confluence
Free for FOSS	Yes	Yes	Yes	Yes
LDAP support	No	Yes	Yes	Yes
Restrict viewing rights	No	Yes	Yes	Yes
Workflow (customizable)	Manually with labels	Yes	Yes	Yes
Agile development support	3rd-party websites	Add-ons	Add-ons	Yes
Reporting	Basic graphs	Yes	No	Yes
Integrated wiki system	Yes	No	Yes	Yes
Expandable	Via API	Yes	Yes	Yes

Table 1. Comparison of bug tracker/project management software

3.4 Documentation Management

Initial Situation. Initially Google Docs, Dropbox, and a wiki system were used for distributing documents and information. The wiki system grew more or less naturally and the structure was seldomly revised. For a group of five people, which communicates mainly face-to-face, a less formal documentation management was useful, but once the number of contributors grew, we experienced severe problems.

Resulting Problems. Document owners left the project and the owner rights were not transferred. Documents were not updated, became outdated, and there was no general overview of documents and their content. File sharing tools like Google Drive and Dropbox were neither integrated in our wiki nor were they supporting our new user management with LDAP. These facts made it harder to share and find current information, documents, and their owners.

The wiki was not well maintained on all of its pages, and the organically grown structure could be an additional hurdle if one did not take the time to learn how to use the included tools. As discussed in [15], outdated development documents lead to difficulties for newcomers. Even some of our experienced project members tended to avoid using the wiki because of its partly outdated content and complex structure.

Method of Resolution. As explained in Section 3.3 we decided to use Jira as globally-available project management tool and Confluence as new knowledge management platform. The switch of systems provides us with an incentive to revise the structure and to eliminate or correct outdated information. This will make it easier to find up-to-date and useful information. The introduction of Confluence is still ongoing, because the information representation needs major rework.

Scacchi [13] introduced *Free and Open Source Software Development (FOSSD) informalisms*, which are easy to use and publicly accessible resources like: threaded discussion forums, group blogs, news postings, how-to guides, to-do lists, and FAQ. Most of these technologies are supported either by Jira or Confluence and we expect that they will support our contributors in their search for information.

4 Lessons Learned

4.1 Human Related

Introducing and switching to IRC as the new IM service was time consuming and resulted in resistance. We believe that the confusions described in Section 3.2, the out-dated and uncomfortable user interface of IRC clients, and our underestimation of the need for change management were mainly responsible for the long (and still ongoing) resistance. To reduce this opposition to change we used techniques described in [1, 8] during the introduction of Jira. We involved more developers during the configuration period of Jira and the workflow adaptation. We communicated the benefits of the new system more clearly. We believe that this communication of benefits, training, involvement of GSoC mentors and optimizing the workflow led to a smooth change of our issue tracker and the introduction of Jira as a project management tool.

4.2 Technology Related

Centralized management of team member accounts and information about them should ideally be introduced from the very beginning as it tremendously simplifies the administration of contributors and saves a lot of time later. User rights management should be well structured and at the same time adjustable to future changes. Integrated services are preferable from a maintenance perspective, because they save time and effort related to organizational tasks. This time can then be spent on project goals. Not all external services, for example GitHub support LDAP, but if an API is available for that service, and user maintenance for this service consumes a lot of time, at least some parts should be automated by scripts.

5 Future Work

The transition to the newly deployed services is not yet finished for every team. Jira is already widely applied throughout the teams, but some teams still have

to make the transition. As stated in Section 3.4 Confluence is not yet deployed since the restructuring of the old system takes time and the focus is on easily accessible and maintainable data. Confluence will contain how-to guides, FAQ, blog-like news entries, and meeting notes to be more transparent and to provide new contributors with the most relevant and up-to-date information [13]. To support the transition [1, 8] from the wiki system to Confluence we did a survey to detect main concerns and problems with the current wiki and are involving senior contributors in the content adaptation process.

6 Conclusion

As explained in Section 2 communication and documentation are essential parts of agile software development and even more important in distributed development [15]. Face-to-face communication is suitable at the beginning, but with the growth and internationalization of a project, good communication channels and project management tools have to be introduced. Every change of workflow or established tools is time consuming and needs proper change management to succeed. So before changing major aspects it should be well considered, if the changes are worth the effort. Appropriate user and rights management simplifies the administration of infrastructure and contributors. It saves time and supports contributors by giving them access to the project's services ideally with one account.

7 Acknowledgments

Thanks to all contributors of Catrobat[13] and also other supporting parties[14]. This work has been partially funded by the EC H2020Innovation Action No One Left Behind, http://www.no1leftbehind.eu/, Grant Agreement No. 645215.

References

1. Aladwani, A.M.: Change management strategies for successful erp implementation. Business Process management journal 7(3), 266–275 (2001)
2. Anderson, D.J.: Kanban: Successful Evolutionary Change for Your Technology Business. Blue Hole Press (2010)
3. Andres, C., Beck, K.: Extreme Programming Explained: Embrace Change, 2nd Edition. Addison-Wesley Professional (2004)
4. Beck, K.: Test-Driven Development: By Example. Addison-Wesley Professional (2003)
5. Hiranabe, K.: Kanban applied to software development: from agile to lean (2008), `http://www.infoq.com/articles/hiranabe-lean-agile-kanban`, [Online; accessed 15-December-2014]

[13] `http://developer.catrobat.org/credits`
[14] `http://developer.catrobat.org/special_thanks`

6. Hussain, Z., Lechner, M., Milchrahm, H., Shahzad, S., Slany, W., Umgeher, M., Vlk, T., Koeffel, C., Tscheligi, M., Wolkerstorfer, P.: Practical usability in xp software development processes. In: ACHI 2012, The Fifth International Conference on Advances in Computer-Human Interactions. pp. 208–217 (2012)
7. Korkala, M., Abrahamsson, P.: Communication in distributed agile development: A case study. In: Software Engineering and Advanced Applications, 2007. 33rd EUROMICRO Conference on. pp. 203–210. IEEE (2007)
8. Kotter, J.P., Schlesinger, L.A.: Choosing strategies for change. Harvard Business Review (1979)
9. Layman, L., Williams, L., Damian, D., Bures, H.: Essential communication practices for extreme programming in a global software development team. Information and software technology 48(9), 781–794 (2006)
10. Poole, C.J.: Distributed product development using extreme programming. In: Extreme Programming and Agile Processes in Software Engineering, pp. 60–67. Springer (2004)
11. Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., et al.: Scratch: programming for all. Communications of the ACM 52(11), 60–67 (2009)
12. Scacchi, W.: Understanding the requirements for developing open source software systems. In: Software, IEE Proceedings-. vol. 149, pp. 24–39. IET (2002)
13. Scacchi, W.: Collaboration practices and affordances in free/open source software development. In: Collaborative software engineering, pp. 307–327. Springer (2010)
14. Schmmer, T., Schmmer, J.: Support for distributed teams in extreme programming. In: Proceedings of eXtreme Programming and Flexible Processes Software Engineering - XP2000. pp. 355–377. Addison Wesley (2000)
15. Shibuya, B., Tamai, T.: Understanding the process of participating in open source communities. In: Emerging Trends in Free/Libre/Open Source Software Research and Development, 2009. FLOSS'09. ICSE Workshop on. pp. 1–6. IEEE (2009)
16. Yamauchi, Y., Yokozawa, M., Shinohara, T., Ishida, T.: Collaboration with lean media: how open-source software succeeds. In: Proceedings of the 2000 ACM conference on Computer supported cooperative work. pp. 329–338. ACM (2000)