Forging A Community – Not: Experiences On Establishing An Open Source Project

Juha Järvensivu[1] and Tommi Mikkonen[1]
1 Institute of Software Systems, Tampere University of Technology
Korkeakoulunkatu 1, FI-33720 Tampere, Finland
{juha.jarvensivu, tommi.mikkonen}@tut.fi

Abstract. Open source has recently become a practical and advocated fashion to develop, integrate, and license software. As a consequence, open source communities that commonly perform the development work are becoming important in the practice of software engineering. A community that is lively can often produce high-quality systems that continuously grow in terms of features, whereas communities that do not gain interest will inevitably perish. Despite their newly established central role, creation, organization, and management of such communities have not yet been widely studied from the viewpoint of software engineering practices. In this paper, we discuss experiences gained in the scope of Laika, an open source project established to develop an integrated software development environment for developing applications that run in Linux based mobile devices.

Keywords: Software engineering, open source community establishment

1 Introduction

Open source development has recently become a practical fashion to develop different types of software systems. This also affects commercial systems, where open source code can be used in tools, platforms, and general-purpose libraries, for instance. Also commercial systems that are largely based on open source components exist, as contrary to the common misbelief these are not contradicting concepts. A good example of such a system is Nokia 770 Internet Tablet, which is based on popular open source components.

Open source software development takes place in communities that developers participate to compose, maintain and further develop associated software. Then, the fashion such communities operate and how they are composed of becomes an important aspect for the future of the system as well as for its users.

Some the differences of open source and proprietary software have already been addressed (e.g. [6], [14]). However, there are little practical research results on several key questions related to communities as software developing entities. Such include

how to organize a community in a meaningful and long-lasting fashion,

how to identify a community that is the most promising from others developing similar systems,

how to ensure that the results that a community has produced will be maintained and remain readily available in the future, even if it seems that like software development in general, also communities can be very person-dependent [1].

Moreover, life cycles of open source systems and communities maintaining them are rarely documented explicitly, although a lot of practical knowledge definitely exists among practitioners and the history of high-profile communities is definitely widely and well known.

The first step towards understanding community driven software development practices is to document and share experiences from establishing and maintaining an open source community. In this paper, we provide an overview to project Laika, which has developed an integrated development environment (IDE) based on Eclipse for Nokia Internet Tablet type of devices (N770, N800) running Linux. In the beginning, we were counting on the interest among developers working with the platform, as facilities that readily existed in this environment were not too sophisticated. At first, things looked promising. We were able to compose a fully functional version of the system, and gained a lot of outside interest and publicity. At present, we however face difficulties in supporting the new versions of both the platform and other programs that act as components in our system. The community has simply not been attractive enough for other developers to join, something we took for granted for long time during the project.

The rest of this paper presents our experiences structured as follows. Section 2 discusses our motivation and the target environment we had in mind when we started the project. Section 3 discusses the history of the project, and describes the phases that we have been able to identify in the progress. Section 4 forms the core of the paper by describing our experiences on the creation and maintenance of an open source community. Section 5 draws some final conclusions.

2 Target Environment

The open source project addressed in this paper, Laika, aims at the creation of an integrated development environment (IDE) for developing applications for embedded Linux devices. The particular devices we had in mind, Nokia N770 and N800 Internet Tablets, are based on the Maemo platform [15], and provide Internet browsing capabilities with a relatively large high-resolution touch screen.

The goal of our project was to integrate the work of several open source development projects into a single tool that is sophisticated enough for industry scale software development. When doing so, we planned to implement the necessary glue code ourselves, but rely on the effort of other communities for everything else.

In the beginning, we could readily find several open source projects whose output was valuable for Laika. Most importantly, we decided to build our project on

Maemo [15], Scratchbox [18], and Eclipse [4], which will be addressed in more detail in the following. In addition, we have used other components as well, although their role can be considered less important than that of the above projects. All open source components that are currently included in Laika are listed in Table 1.

Table 1. Component communities of Laika

Component	Description
CDT	C/C++ development environment for Eclipse, which served as a starting point for our development effort.
Eclipse	Vendor-neutral open development platform, whose plugin Laika is.
Glade/Gazpacho	Graphical user interface builder for GTK+.
Maemo	Development platform to create applications for the Nokia 770.
PyDev	Python development environment for the Eclipse platform.
Scratchbox	A cross-compilation toolkit used by the Maemo platform.

Maemo. Maemo, which acts as the software platform for Nokia's N770 and N800 Internet Tablet, is composed of a set of popular open source software components that are widely deployed in today's leading desktop Linux distributions [15]. Maemo consists of precompiled Linux kernel, platform libraries, and Hildon user interface framework. The Hildon UI framework in turn is based on GTK+, and as a whole the platform is binary compatible with GTK+ binaries [7]. The structure of the system is illustrated in Fig. 1, and illustrated components are introduced in more detail in Table 2.

Scratchbox. Currently available mobile devices based on Maemo run on ARM processor, whereas most developers prefer using an Intel-based computer like a PC as their development environment. The creation of ARM binaries with such development setup requires cross-compilation, where a host computer is used to compile software for a target that uses a different instruction set. In our case, we have performed cross-compilation using a PC running Scratchbox, which is a toolkit designed to ease embedded Linux application development [18]. The target platform is ARM, the hardware environment of our devices.

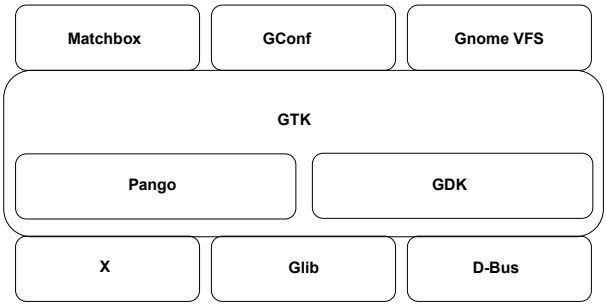

Fig. 1. Maemo's main software components

Table 2. Maemo's main software components

Component	Description
Matchbox	A light-weight window manager that is responsible for managing X11 client window geometry, stacking, and decorations in our target device.
GConf	A generic system for storing application preferences.
Gnome VFS	Gnome virtual file system, provides an abstraction layer for accessing files in both local file systems and the web.
GTK+	The Gimp toolkit, a toolkit for creating windowing applications. The kit also provides an extensive set of readily available widgets.
Pango	A library for text layout, rendering, and internationalization.
GDK	Gimp drawing kit, a graphics library that wraps routines providing low-level access to a display.
X	X Window System, a networking and display protocol for building windowing systems.
Glib	A low-level library that contains commonly needed generic routines for wrappers and runtime routines including event handling, dynamic loading, and threading.
D-Bux	Light-weight inter-process communication system.

Eclipse. The Eclipse platform is a vendor-neutral open development platform that provides tools for managing workspaces and building, debugging, and launching applications for building integrated development environments (IDEs) [4]. In essence, the Eclipse platform is a framework and toolkit that provides a foundation for running third-party development tools. The basic mechanism of extensibility in Eclipse is adding new plugins, which can also add new processing elements to plugins that already exist. In accordance to this architecture, the Laika plugin is based on the C/C++ development tools (CDT) plugin [3], which in turn is a subproject of the Eclipse community [4].

3 Short History of Project Laika

In this section, we describe the history of project Laika. The goal is to give an overview to the main events that the project has experienced, and how we have ended up in the current situation.

3.1 Becoming of Age

The origins of Laika are in meetings between Institute of Software Systems at Tampere University of Technology (TUT) and Nokia Multimedia (Nokia) during the Fall 2004, when the first Maemo based device was nearing completion. A common goal was set by both parties to aim at the development of an industry-strength integrated development environment that would encapsulate facilities necessary for Maemo development. The plan was that in the first phase, we would only implement minimal functionality, which could later be extended. The extensions would be made by either the original developers that were responsible for the first release, or by other parties that we would be able to attract to participate in the community. A related plan was that the experiment would also give the university first-hand information on how open source communities work.

As already documented in [9], the development was initiated by establishing a co-located team of students working as summer-time trainees within the premises provided by the university. Moreover, university already had existing server infrastructure that could be used for hosting the community and enabling download of Laika software. Thus, the initial investment was small, and the actual work could be started within a short period of time from the actual decision.

At the university, the project was supervised by a graduate student and a professor who had participated in meetings with Nokia. As a practical starting point, some code was provided to the community by Nokia Research Center, which had already investigated an opportunity to develop such an IDE [19]. This code then formed the baseline for further development that was carried out by the Laika community.

The first actual release of Laika was made towards the end of the summer of 2005, with the majority, if not all, the work performed by the summer trainees. At this point, the system simply comprised Eclipse, CDT, and Scratchbox, as well as some compilation and debugging capabilities that we felt were important for the project. The system is illustrated in Fig. 2.

The 1st release proved that there indeed was a need for an IDE for Maemo, and that Laika was really something that people found useful, at least based on the number of download and the feedback received from people installing and using the system. A total of 603 downloads were made during the time version 1.0 (and bug fix version 1.1 that was released a week later) was available for download. A number of downloads were from private companies that were doing industrial-scale development in Maemo environment.

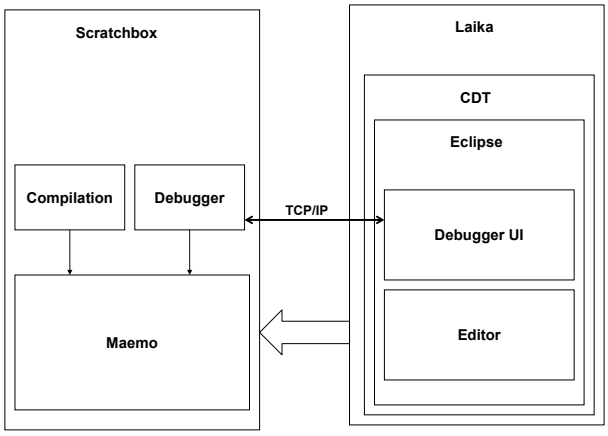

Fig. 2. Laika Release 1 components

During this development period, the main challenge of the project was to be able to create the first running version of the system, because the component systems originating from different communities were not straightforward to integrate. A part of the success is due to the technical skills of people who had already started the development work at Nokia. However, the effort invested in the development by the students should not be underestimated.

3.2 Peak of Fame

Upon completing the first public release of Laika, further interested stakeholders were identified. To begin with, a local company wanted to participate in the development of the system, and they agreed to integrate a user interface editor to the system, a task that the community happily welcomed them to perform. Secondly, project funding was provided to the community both from companies and by the university hosting the project in exchange for using the project as a guinea pig for open source related research. Funding from both sources was gladly accepted by the project, and used for further development of the system. In practice, the three students who had already worked on the system were now all hired to maintain and further develop the system by the university.

The additional funding and support described above kept the project running from September 2005 to January 2007. During this time period, a number of improvements were added to the system, including the above-mentioned UI editor, which was added by the company, as well as support for Python, a dynamic scripting language with which it is easy to compose small applications. The resulting system, illustrated in Fig. 3, required some changes in the tool architecture, as available open source components were based on different designs. In addition, a number of smaller enhancements were made, that in general eased application development.

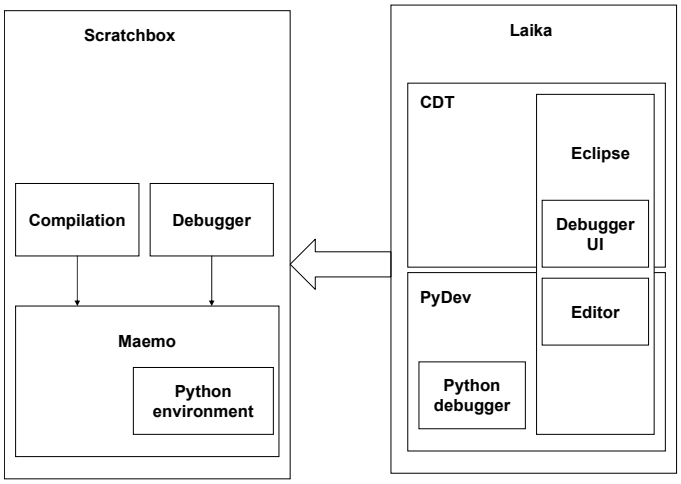

Fig. 3. Laika Release 3 components

At this point of the development, the downside of using a collection of existing components started to manifest. Creating a new configuration where some versions of component subsystems were supported often required additional coding and testing, which seemed to offer little reward for developers.

In general, a lot of external interest was paid to the community at this point, including a number of accepted papers [9][10] as well as a presentation regarding the project at one of the collocated sessions at GNOME User and Developer European Conference (GUADEC) 2006 [16].

In this phase, the main challenge of the community was to cope with ever-increasing number of updates in subsystems that comprise Laika. As already described in [10], new releases of other systems forced the community to perform regression testing of Laika repeatedly. Another issue that can be considered important is user support, which was found a difficult task to perform well. The reason is that most of the issues were related to component subsystems, not Laika as such, and we felt that to some extent this task was beyond our community's scope. In fact, in some occasions, the community ended up supporting people in installing and using Eclipse for instance in order to allow them to use Laika.

3.3 Decline

By January 2007, project Laika and the related community started running out of steam. To begin with, one of the three students that were originally participating in the development had left the project, and the other two were just about to complete their thesis about the experiences with Laika [11]. Generous funding from different sources was terminating. Moreover, also the project manager and the responsible professor had assumed other tasks and responsibilities, and it was not an option for

them to take a leading role in code-level activities in Laika. A lead developer, on the other hand, was considered a necessity for the project to remain vital.

Despite the fact that the future of the community started looking weary, a number of positive things was happening. The community still gained a lot of interest, and there have been active negotiations on receiving support from involved companies, either in the form of funding or developers. Recently, Laika's code base was moved from university's server to a new location in Maemo Garage [12], a high-visibility web site hosting Maemo related material.

The essential question was how to continue with the development and to establish a real community instead of simply maintaining the project as long as continuous funding exists, and terminate it when the funding ends. At present, there still is interest in Laika, and attracting companies or institutions to participate in development is a reasonable goal. It may even be possible to integrate our system with another similar system, like Esbox [5] for instance, which builds on the same main components as Laika, which in turn would strengthen development. Even more importantly, we need to find a suitable long-term lead for the community – preferably one of the people who have already participated in the development. Motivating the lead, however, might require external attention or in the best case secured funding over some period of time.

3.4 Epilogue

Around the same time the first complete version of this paper was composed, we decided that due to resource restrictions joining forces with Esbox would be best for the Maemo developers. As a result, a new project was established in Maemo Garage under the name Esbox, which in the end will hopefully put together the best parts of Laika and Esbox. At present, however, core developers of Laika are yet to participate in this activity, and no major role has been assumed by us.

4 Lessons Learned

Obviously, there are numerous issues that could have been implemented differently in the course of our project. However, we trust that the reasons why we were unable to attract other people onboard – the real reason for not becoming a lively community – are few. Matters related to this creation of an active developer community are addressed in the following.

Focused mission which remained undocumented. One of the particularities of open source development is the mission of the community. In our case the mission was to "implement an industry-scale integrated toolset for composing Maemo applications", which also sets the scope for the community. For instance, it overrules extending the community to systems other than Maemo. In addition, we decided to base our work to existing, readily available open source subsystems. With this as the starting point, we were able to establish the first running system relatively rapidly,

which in turn gave us instant credibility. Not documenting the mission can be considered a mistake, however. Without manifesting the mission, it has been difficult for other developers to grasp what Laika really is about.

Too much a debugging, integration, and testing project and too little a real development project. Another mistake regarding the mission is that we defined it in terms of what the community would provide, not what the community would implement in the technical sense. This led to two-fold development focus. One part was about developing support in terms of code that was composed by us, whereas another part was selecting and integrating suitable components that readily existed. To satisfy the latter, it was necessary to perform a lot of straightforward debugging and bug fixing when new versions were released. After a while, constant debugging, integration, and testing started to overrule the project. In an ideal world, we could have geared our own implementation into a direction where maintenance and building new releases would have been eased. This never happened, though.

Developers interested in SDK development, not that much in Maemo programming. Almost from the beginning of the project it became obvious that all developers were focused to developing SDK, and that developing Maemo applications was not an important issue. As a result, the development effort started from SDK and features that would be interesting from SDK point of view, not from the development of Maemo applications and easing it except indirectly. The effect of this lack of "eating our own dogfood", advocated by Spolsky in [20], to the community is hard to estimate. However, as we had another team, working in a neighboring office that was performing development using also our tools, we feel that this shortcoming is not too severe.

Relying on agile principles that are not necessarily ideal for community building. We organized our development following the spirit and principles of agile software development and extreme programming, like pair programming, collective code ownership, and a common development room [2][13]. In addition, the mindset we adopted was to go for a development mode, where code would be the main artifact, and other items would only be written when absolutely necessary. While the approach has worked well in our own development, there were some problematic issues. In particular, as we were busy developing the system ourselves, we were not too open for other potential participants. This resulted in entry barriers for new contributors, which in turn made our community closed. As the main activity of the community was focused in a single office, where all the developers and the project manager were co-located, this office quickly became the mental home of the community. Therefore, in order for others to participate, barriers existed already in the physical environment. A better alternative, at least from the viewpoint of community creation, would have been to rely on practices and processes that have already been documented explicitly in the scope of open source software [17].

Steep allocation of responsibilities but minimal staffing. The allocation of responsibilities in the community can also be considered. Led by a professor and a project manager, a number of students were performing the actual tasks. This unfortunate setting has resulted in a situation where the students had the real technical

control, while the professor and the project manager were mainly focused to obtaining funding, users, and partners. When the students had their theses completed and they moved to different positions around the same time, the project was left without a lead who singlehandedly could manage both technical aspects and external relations.

Subcontracting for money instead of community building for free. While in general considered positive – and probably a major reason for the possibility to establish a community in the first place – the generous funding we obtained also has some downsides. As our plan was to create a self-sustaining community, the development was soon driven by feature requests associated with funding, not with our own plans. This in turn strengthened an attitude that we are actually performing feature-based subcontracting, not establishing a community.

Lack of active marketing, user support, and other related supporting facilities. As pointed out in [8], open source systems require marketing, similarly to any other software whose use is being advocated. Corresponding arguments can be made for other supporting facilities, including user support for instance. This aspect of the community building remained overlooked in Laika. As a consequence, only people that we were able to relate to had the potential to become participants in the community. Moreover, even using the system built in the community required considerable technical skills.

Single client community. In many cases, open source projects can be taken as common solutions for recurring problems encountered by several organizations. In our case, however, we were only aiming at easing the development of Maemo applications, something that only associates with one other actor. While there were several companies that had interest in developing applications, in practice what we believed was the best for Maemo community was something that we in practice had to obey to a great extent.

No experience on establishing an open source project. An issue that we did not consider important when we started was experience on community work. Looking back now, it would have been essential to involve someone with experience on ramping up an open source project in the community. However, while simple on paper, achieving this in practice might have been difficult.

5 Discussion

Considering Laika now, we feel that we got the development technology right, although one can argue that for instance versioning support could be drastically improved. In addition, we were able to create pull from users of our system. They accepted our system as their development environment and in fact approached us with future development ideas. Moreover, we were able to compose a system that did satisfy – and in many cases exceeded – the requirements we initially set to it. Obviously, we owe a great deal of our success to other open source communities, as without using existing components this would have been impossible. However, we were

still unable to attract more developers to help us with our effort, which in the end has led to a decline.

During the development of Laika, we encountered a number of properties that have larger significance for community-based development. Our main observations are listed in the following:

- Community-based development worked well when developing new features. In contrast it was hard to motivate developers volunteering in the project when tasks like integration and testing was repeatedly requested. Building a toolset that would have eased the latter would have been an essential element for the continuation of the community.
- Distributing responsibilities in a fashion that resembled a small project, where managers were focusing on funding rather than coding, led to a community where no lead developer existed. As a result, there is no single actor that could (or would) singlehandedly continue maintaining the project.
- Agile practices that emphasize the importance of collocated development teams were harmful when trying to attract new developers to join the community. Instead, we should have embraced new developers by offering easy ways to practically participate in the development.
- Using the system ourselves might have helped in solving some of the problems associated with our development effort, especially when considering the priorities of different features. As a result, people using the system might have benefited more.
- Allowing supporters – especially those who provide funding – to overly control a community was fruitful in the beginning, as it gave us confidence on our mission. In the long run, however, this led to subcontracting mode, where funding was associated with individual features instead of supporting the community in general.
- It is easy to overextend the development effort, as the number of readily available open source components that can be integrated in the system is so large. However, in order to keep the system manageable in the long run, it is essential to focus on a relatively small number of key features at least in the initial phases when the community is small.

Still, to our knowledge there was no single community at the time aiming at supporting industrial IDE in Maemo environment that would be considerably stronger than our attempt. Moreover, we have received frequent requests for new features and updates, but have been unable to comply with the requests due to resource constraints in community personnel. This exhaustion of resources, more than anything else, can be taken as an indication that we have failed in forging a lively community.

Acknowledgments

The authors are grateful to the whole Laika community as well as all the partners who have participated in our work. In addition, we wish to thank also the users of Laika for their choice of a development system.

References

[1] Aaltonen, Timo, Järvensivu, J., and Mikkonen, T. OSS architecture and implications. 67-77, Empirical Insights on Open Source Software Business (Eds. Nina Helander and Maria Mäntymäki), *eBRC Research Reports 34*, Tampere, Finland, 2006.
[2] Beck, K. *Extreme Programming Explained – Embrace Change*. Addison-Wesley, 1999.
[3] CDT homepage. *Available on the Internet at http://www.eclipse.org/cdt/*.
[4] Eclipse homepage. *Available on the Internet at http://www.eclipse.org*.
[5] Esbox homepage. *Available on the Internet at http://wiki.embeddedacademy.org/index.php/ESbox_Plug-in*.
[6] Feller, J., Fitzgerald, B., Hissam, S., Lakhani, K. *Perspectives on Free and Open Source Software Development*. MIT Press, Cambridge, MA, USA. 2005.
[7] GTK+ homepage. *Available on the Internet at http://www.gtk.org*.
[8] Henttonen, K. *Stylebase for Eclipse. An Open Source Tool to Support the Modeling of Quality-Driven Software Architecture*. VTT Research Notes 2387, Espoo, Finland, 2007.
[9] Järvensivu, J., Helander, N. and Mikkonen, T. Dependencies, Networks, and Priorities in an Open Source Project. 116-125, *Handbook of Research on Open Source Software: Technological, Economic and Social Perspectives* (Eds. Kirk St.Amant and Brian Still), IGI Global, 2007.
[10] Järvensivu, J., Kosola, M., Kuusipalo, M., Reijula, P. and Mikkonen, T. Developing an Open Source Integrated Development Environment for a Mobile Device. *International Conference on Software Engineering Advances*, Tahiti, French Polynesia, Oct. 29.-Nov.3., 2006.
[11] Kuusipalo, M. and Reijula, P. *Implementing a Visual Development Environment for Maemo Compatible Devices*. MSc. thesis, Tampere University of Technology, 2007. In Finnish.
[12] Laika homepage. Available on the Internet at *http://garage.maemo.org/projects/laika.*
[13] Larman, G. *Agile and Iterative Development*. Addison-Wesley, 2003.
[14] MacCormack, A., Rusnak, J., Baldwin, C. Exploring the structure of complex software designs: An empirical study of open source and proprietary code. 1015-1030, *Management Science*. Vol. 52, No. 7, July 2006.
[15] Maemo homepage. *Available on the Internet at http://www.maemo.org*.
[16] Reijula, P. Integrating Maemo Development Environment with Eclipse. *GNOME User and Developer European Conference (Guadec) 2006 warmup weekend*, Barcelona, Spain, Jun 24, 2006.
[17] Scacchi, W., Feller, J., Fitzgerald. B., Hissam, S., and Lakhani, K. Understanding free/open source software development processes. *Software Process Improvement and Practice*, 92-105, No. 11, 2006.
[18] Scratchbox homepage. *Available on the Internet at http://www.scratchbox.org/.*

[19] Sillanpää, M. Extending Eclipse and CDT for embedded systems development. *EclipseCon 2006*, Santa Clara Convention Center, Santa Clara, CA, USA. March 20-23, 2006.

[20] Spolsky, J. *Joel on Software: And on Diverse and Occasionally Related Matters That Will Prove of Interest to Software Developers, Designers, and Managers, and to Those Who, Whether by Good Fortune or Ill Luck, Work with Them in Some Capacity*. Apress, 2004.