Open Source Engineering of Proprietary Software: the Role of Community Practices

Furqan R. Shah, Imed Hammouda, and Timo Aaltonen
Department of Software Systems
Tampere University of Technology
P.O. Box 553, FI-33101 Tampere, Finland
{syed.shah, imed.hammouda, timo.aaltonen}@tut.fi

Abstract. This paper studies the problem of opening industrial software. In this regard, a conceptual framework for open source engineering of proprietary software with respect to the role of community practices is presented. To demonstrate our approach, the framework has been applied to a proprietary software planned to be released as open source. The early experiences of the case study suggest that the approach has promise and merits further investigation.

1 Introduction

Open source software is gaining increased momentum in the business world [8]. For instance, more and more companies are releasing their proprietary software as open source. For companies, the latter setting represents a complex process as it needs deep understanding of the open source phenomenon. Furthermore, since the trend is relatively recent, there are little guidelines on how to create and maintain a sustainable open source community for proprietary software. As a remedy, existing open source communities can play a significant role in proposing best practises and practical recommendations. It has been shown that any online community has its norms and practices to perform daily activities without physical interaction [6, 9].

In this paper, we argue that proprietary industrial software needs to undergo an *open source engineering process* that typically touches many aspects of the software to be released and its environment. This process needs to be carried out in a *pre-bazaar phase* [7] before evangelizing the software to the open source community. The open source engineering process itself is driven by different kinds of *influential factors* such as software attributes, project infrastructure, development process, legality issues, marketing strategies, and community needs. In case the software is targeted for a specific existing open source community, the process may use, as *ingredients*, the practices and norms of that community. Such practices may include licensing scheme, release management, communication within community, documentation, coding style and convention, and bugs reporting and tracking.

This paper introduces a general conceptual framework for open source engineering of proprietary software with respect to community practices. The framework has been applied to an industrial software platform that was originally

developed by Sesca Mobile Oy, a Finnish IT company. Our first experiences, as reported in this paper, show that the approach has been useful and may facilitate the community building process.

We proceed as follows. In Section 2, we discuss open source engineering of proprietary software. In Section 3, we present a conceptual framework to prepare and release industrial software. In Section 4, we discuss the case study in the context of the conceptual framework and we report our first experiences. Finally, we conclude the paper with discussing the results and future work in Section 5.

2 Open Source Engineering of Proprietary Software

Releasing proprietary industrial software as open source needs additional open source engineering. The process may include, among others, activities such as source code refactoring, architecture recovery, team building, development process tuning, and infrastructure establishment. Therefore, the process can be considered as multi-facet and is driven by various kinds of factors, which can be grouped along six dimensions.

Software. Improved software quality may increase the success rate of community building. Quality can be enhanced by incorporating best practices, documentation, code cleanup, coding standards and convention. Furthermore, in order to support the community, source code may be accompanied with user manuals, API documentation, and architecture descriptions. It is vital to have the first experience with downloading, installing, deploying and using the software as easiest as possible.

Infrastructure. There are two key elements in any open source project: community and project repository. An open source engineering process should provide enabling tools and technologies to facilitate the planning, coordination, and communication between the community members. In addition, efficient mechanisms and tools are needed to facilitate the access and management of the project repository.

Process. Open source development can be regarded as an open maintenance process. A process needs to be established in order to handle decisions regarding the evolution of the software, maintenance actions, and release management. The process needs to balance between the practices of communities and the needs of the company.

Legality. The releasing company needs to select a license type (e.g. GPL versus LGPL) and a licensing scheme (e.g. single or multi licensing). In addition, source code should be legally cleared against IPR and copyright issues. Also, the availability of trademarks and names used in the software should be checked.

Marketing. Building an open source community can be regarded as a marketing challenge. Effective marketing strategies are needed to market the open source project to potential users and developers. Selecting an existing open source community as a target customer can be an important success factor.

Community. The releasing company should be ready to support the project community. For instance, community members should be provided with clear guidelines on how and what to contribute. Furthermore, company developers who are participating in the community should be trained for their new roles and should be given clear responsibilities. When the software is opened, it is vital that all information are made public and that private discussions are avoided. In addition, there should be trust among community members, zero tolerance of rudeness and no use of bad language both in the software artifacts and the communication among the members.

When starting an open source engineering process, the central challenge is how to find answers for the open questions pertaining to the various dimensions listed above. We argue that by selecting an existing open source community as a target, the practices of that community can be used to provide partial answers to those open issues. In the next section we present a framework for such purposes.

3 Conceptual Framework

Fig. 1 depicts a framework for open source engineering of proprietary software with respect to the role of community practices. As a starting point, the releasing company selects a target community (phase 1). The selection process may be based on software-related factors (i.e. programming environment, application domain, etc) or any other company motives (i.e. business, strategy, etc). It is assumed that the selected community has its norms and practices. However, it is possible that the company does not go for any existing community. In this case, the company may rely on other means to plan the open source engineering process, by for example using past experience or favoring popular known solutions.

Once a target community is selected, the company collects all relevant practices pertaining to licensing, release management, communication within community, documentation, coding style and convention, bugs reporting and tracking, and source code management (Phase 2). The proprietary software and its environment is then transformed and established according to these practices. The next step is to release the software in the open (Phase 3). If the open source engineering process has been carried out with a specific target community in mind, it is hoped that adopting the practices of that community may attract some of its members to the subject software. Together with original developers allocated by the releasing company, a *community embryo* is formed (Phase 4). Once formed, an embryo community may join the target community (if there is one) or may evolve as an independent one (Phase 5).

In practice, it may be a hard task to select a target community. Even worse, the releasing company may want to go for several communities. In this case, some of the community practices such as licensing and naming conventions may be different and even conflicting with each other. It is then part of the open source engineering process to resolve such conflicts.

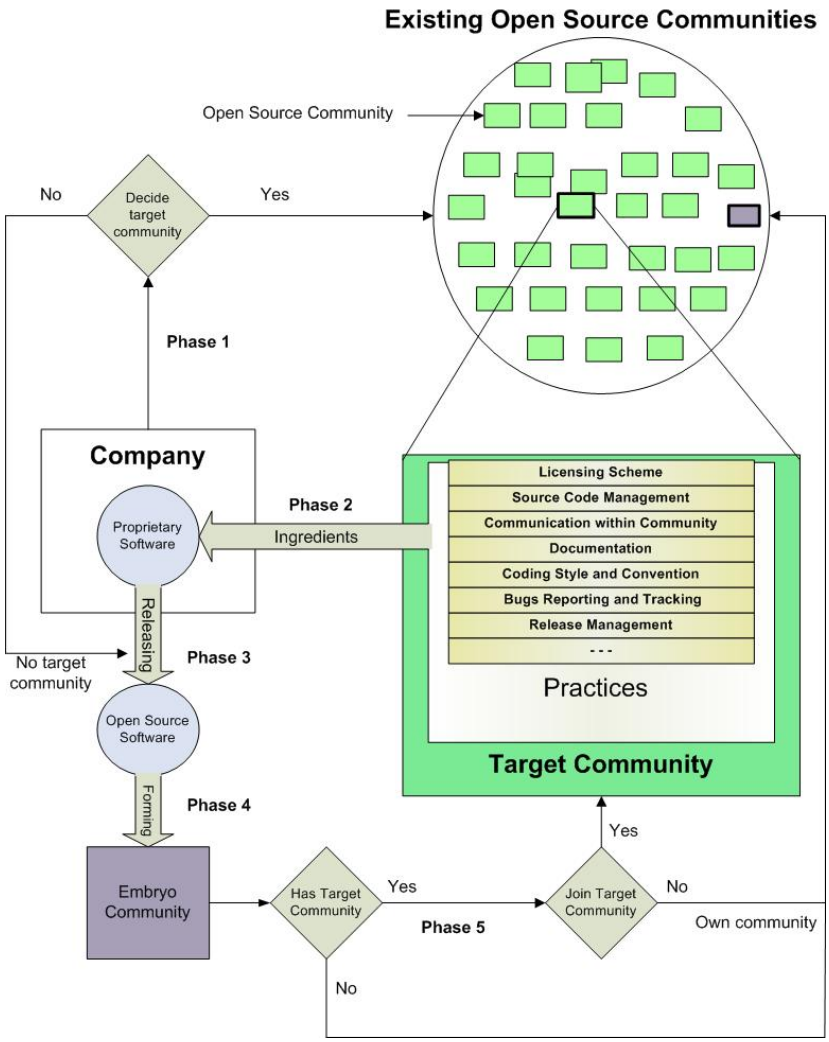

Fig.1. Conceptual Framework for Open Source Engineering.

4 Case Study: Wringer

In order to demonstrate our approach, we have applied the conceptual framework shown in Fig. 1 to the Wringer software platform. Wringer is a JavaScript binding platform for GNOME/GTK+ [4] using V8 [12] as JavaScript engine. The platform was originally developed by the Sesca Mobile Oy.

4.1 Practices of GNOME/GTK+ community

The main objective of the case study has been to build an open source community for Wringer. For this, the GNOME/GTK+ community has been selected as a target. In the following we discuss several practices of this community and their significance to Wringer.

License for Open Source Project: To start an open source software project, selecting a proper license represents an important step. Licenses give certain terms and conditions to use and evolve source code. There is a relatively large number of open source licenses (e.g., GPL and LGPL) that can be used [5].

In the case of GNOME and GTK+, LGPL is used as a license. GTK+ is licensed under the GNU LGPL 2.1 allowing development of both free and proprietary software with GTK+. Thus, Wringer version 1.0 will be released under LGPL 2.1.

Software Configuration Management: In open source software development, a version control system is used to manage code changes. Developers with commit rights make updates to the source code through such system.

The GNOME project uses CVS for software configuration management. Developers send their patches to module maintainers to incorporate their contributions [2]. In the case of Wringer, CVS can be used as a version control system. Since Wringer is relatively small, there is no need to have many module maintainers.

Communication within Community: Members of open source communities are geographically distributed and communicate with each other through various tools and channels such as mailing lists, IRC channels, and forums.

GNOME project members use the following the communication channels [2]: mailing lists, IRC, and GUADEC, which is a GNOME committee responsible for organization of conferences. Also GNOME Summaries are used to publish activities which have taken place in the last two weeks. For the Wringer community, similar communication channels could be formed. The selection could also be based on the channels used by other bindings communities for GTK+.

Documentation: In open source projects documentation is not an absolute necessity as development is code-centric. However, some documents such as APIs and tutorials can be useful.

The GTK+ community provides ample amount of documentation such as API references, tutorial, books, article and presentations [2]. All GTK+ binding communities also use similar practice. Such documentation should be provided for Wringer. Furthermore, since Wringer is a software platform, an example application using the platform, together with a tutorial, could be made available for the community.

Coding style and convention: Coding styles make source code more readable and easy to understand. Different open source communities follow different coding conventions.

The GNOME community follows Linux kernel coding style, or the GNU coding style [3]. As far as coding conventions are concerned, the GTK+ and GNOME communities follow same naming convention for libraries. For instance, function names are lowercase, with underscores to separate words. Macros and enumerations are uppercase, with underscores to separate words. Types and structure names are mixed upper and lowercase [3]. It is recommended that Wringer follow same naming convention and coding style as GNOME.

Bug reporting and tracking: Almost all software products contain bugs. These bugs should be in source code or any other artifacts. Bug reporting is considered as a valuable contribution by community members.

The GTK+ community uses Bugzilla for bug tracking. Users need an email address to report bugs with certain KEYWORD [4]. In the case of Wringer, existing bugs could be made available in a Bugzilla. This will guide community members through the contribution process.

Releasing new version: Open source development is a continuous maintenance process. To accommodate bug fixes and new features, new releases are planned.

In every release, the GNOME/GTK+ reports how many bugs have been fixed and how many new features have been added to the products. A similar release management strategy could be adopted by Wringer. Following version 1.0, the next version of Wringer could include fixes for the bugs identified.

4.2 Experiences

In this subsection, we report our experiences with open source engineering of the Wringer software platform. The process corresponds to a pre-bazaar phase where Wringer has been installed and deployed outside the company before it is released as open source.

Deploying Wringer: Initially Wringer was using SpiderMonkey [10] version as JavaScript engine. It was expected that it will take time and effort to deploy this version because it needs external libraries such as libgkt+ and libglib. However, installing Wringer has been easy. Similar experience was reported for the V8

version of Wringer. In summary, Wringer is easily deployable on Ubuntu operating system. However, there is still need for installation instruction available, for example, as a Readme file.

Architecture of Wringer: Originally no design documentation was available for Wringer. However, building a high level architecture (i.e. component diagram) of the software out of the source code has been straightforward. The architecture has been verified by the original developers of the software. Hence, we can say that Wringer is an easy to understand software and it has potential to attract community members.

Code Cleanness: The SpiderMonkey version of the Wringer contained dead code. This code has been removed from the V8 version.

Code Comments: Comments makes it easier to understand code. This includes pre-conditions, post-condition and comments on particular statements. The V8 version of Wringer is well-commented. However, there is a room for improvement. This can be left for community contribution as well.

First Experience with Wringer: After deploying Wringer, we have built an example application using the platform. We have used existing applications as guidance. Application development has been straightforward and quick. Thus we rate our first experience with the platform as very good. We have also used the platform in a white box scenario by extending the set of supported GTK+ widgets with new ones.

Bugs: For the community, bugs fixing could be a starting point for contribution. The V8 version of Wringer has some bugs related to memory leak and character rendering. These bugs should be made visible to the community.

Debugging: It has been quite hard to debug Wringer code as no debugger was available. This might be a problem when developing complex applications on top of the platform. Having a debugger in place would be very useful.

5 Discussion

This paper studied the problem of opening industrial software. As the trend is relatively recent, companies find it hard to transfer their proprietary software from the cathedral phase to the bazaar phase. In order to alleviate the challenges involved, we have presented a conceptual framework that emphasizes the role of target community practices in the open source engineering process. Such observation complements other general guidelines for starting and running open source projects [1] and other studies on open source community building [11].

We have argued that the open source engineering process needs to be carried out in a pre-bazaar phase in order to assess the readiness of the software for open source development. The pre-bazaar phase helps in getting early feedback and experiences on using and evolving the software outside its original development environment. The feedback can be used to improve certain aspects of the software and its environment before the actual release.

The framework has been applied to an industrial software platform, named Wringer, considering GNOME/GTK+ as target community. By analyzing the practices of this target community, we formulated a set of recommendations to be used in open source engineering of Wringer. The recommendations related to licensing, software configuration management, communication within community, documentation, coding style and convention, bugs reporting and tracking, and release management. In practice, these recommendations will not be taken for granted. First, they have to be assessed business-wise by the releasing company.

Incorporating these recommendations does not guarantee the success of open source community building. What is required is an effective marketing campaign that makes the software attractive to individuals and companies. Assuming that a community has been built, an evaluation framework is needed to monitor the evolution and sustainability of the community. We are developing such framework as future work.

References

[1] Fogel, K. Producing Open Source Software: How to Run a Successful Free Software Project. O'Reilly Media, Inc., October 2005.

[2] German, D. M. The GNOME project: A case study of Open Source, global software development. Journal of Software Process: Improvement and Practice 8 (4), 201-215.

[3] GNOME Programming Guidelines. http://developer.gnome.org/doc/guides/programming-guidelines/book1.html. (Last visited Feb 2009).

[4] GTK+. http://www.gtk.org/ (Last visited Feb 2009).

[5] Open Source Licenses. http://www.opensource.org/licenses (Last visited Feb 2009).

[6] Preece, J. Online Communities: Designing Usability, Supporting Sociability. Wiley, 2000.

[7] Raymond, E. S. The Cathedral and the Bazaar. O'Reilly Media. 1999.

[8] Samuel, A. A. and Wu, D. Empirical study of the effects of open source adoption on software development economics, September 2007 Journal of Systems and Software, Volume 80 Issue 9.

[9] Scacchi, W. Free/Open Source Software Development Practices in the Computer Game Community, Journal IEEE Software, Year 2004, Pages 59-67.

[10] SpiderMonkey (JavaScript - C) Engine, http://www.mozilla.org/js/spidermonkey/ (Last visited Feb 2009).

[11] Stürmer, M. Open Source Community Building, licentiate thesis, 2005.

[12] V8 JavaScript Engine. http://code.google.com/p/v8/ (Last visited Feb 2009).