Evaluating the Readiness of Proprietary Software for Open Source Development

Terhi Kilamo, Timo Aaltonen, Imed Hammouda, Teemu J. Heinimäki, Tommi Mikkonen

Evaluating the Readiness of Proprietary Software for Open Source Development

Terhi Kilamo[1], Timo Aaltonen[1], Imed Hammouda[1], Teemu J. Heinimäki[1] and Tommi Mikkonen[1]

Department of Software Systems, Tampere University of Technology
Korkeakoulunkatu 1, FI-33720 Tampere, Finland
{firstname.lastname}@tut.fi

Abstract. As more and more companies are releasing their proprietary software as open source, the need for supporting guidelines and best practices is becoming evident. This paper presents a framework called R3 (Release Readiness Rating) to evaluate the readiness of proprietary software for open source development. The framework represents a checklist for the elements required to ensure a better open source experience. The framework has been applied to an industrial proprietary software planned to be released as open source. The evaluation has been carried out by both external and internal stakeholders. The early experiences of the case study suggest that the R3 framework can help in identifying possible bottlenecks before evangelizing the software to the open source community.

1 Introduction

Companies are getting more and more interested in releasing their closed source software products to open source communities. The two large scale examples of this are Sun Microsystems' opening of its Java platform during 2006 and 2007, and Nokia's actions to open the Symbian operating system during 2009-2010 [11]. As the trend is relatively recent, the phenomenon of opening industrial software is not well understood despite of the existence of general guidelines such as in [2, 10]. In this paper we tackle the problematic of releasing industrial software.

Most often companies are not used to release the source code of their products. Their standard ways of behavior tend to be more biased to hiding than to releasing information. The processes, tools and infrastructure used by companies might turn out to be an obstacle for a successful release. The software itself might have been written so that open source developers run into troubles when trying to contribute. This suggests that there is a need for proper methodologies to evaluate the readiness of proprietary software for open source development. Such methodologies would help identifying possible bottlenecks before taking the software to the open. The bottlenecks are then resolved to in order to increase the success rate of community building around the software.

Given the above observations, the research questions we would like to explore include the following:

- What kind of evaluation criteria could be used to assess software readiness for open source development?
- How the evaluation should be planned and which stakeholders are involved?
- How to obtain data for the evaluation process?
- How to exploit the results of the evaluation process?

We argue that these issues have not been studied enough by the open source research community. The closest works to our study are the open source maturity models such as OSMM (Open Source Maturity Model™) [3], QSOS (Qualification and Selection of Open Source Software) [8], and BRR (Business Readiness Rating™) [1]. These models are typically used by companies that plan to use open source. The context of our research problem in this paper is just the opposite: taking software out from companies to open source communities.

The main contribution of the paper is two fold. First, we discuss the specificities of the problem of opening proprietary software. Second, we present a framework called R3 (*Release Readiness Rating*) to evaluate the readiness of proprietary software for open source development. In order to demonstrate our approach, we have applied the framework to an industrial proprietary software planned to be released as open source.

The rest of this paper is structured as follows. In Section 2 we discuss related work and the challenges of opening proprietary software. The details of the R3 framework and the overall evaluation process are presented in Section 3. In Section 4, we evaluate the R3 framework in the context of two industrial case studies. Future work is discussed in Section 5 and finally, we conclude in Section 6.

2 Background

2.1 Open Source Maturity Models

Several methods have been developed assessing the maturity of open source software. For instance, Open Source Maturity Model™(OSMM) enables a quick assessment of the maturity level of an open source product. Products are ranked according to OSMM scores, which are evaluated in a three-phase process: 1) assess each product element's maturity and assign maturity score; 2) define weighting for each element based on the company's requirements and 3) calculate the score.

The Qualification and Selection of Open Source Software (QSOS) maturity model is a four-step iterative process: 1) define (and organize criteria), 2) asses (against the criteria), 3) qualify (define weighted scores, new and mandatory criteria) and 4) select (asses using the weights, and select). Another evaluation framework called Business Readiness Rating™(BRR) was proposed as a

new standard model for rating open source software. The model consists of a four-phase process: 1) quick assessment filter (for quickly abandon bad candidates), 2) target usage assessment (for inputting the needs of the company), 3) data collection & processing (for collecting the actual information) and 4) data translation (which leads to one outcome: the rating).

Compared to the method we propose in this paper these maturity models take a totally different direction. Whereas our model attempts to study one software product which is going out from a company, these maturity models attempt to study a set of software products, one of which is selected to come in to the company. However, some ideas are still quite similar. For example, in both cases the architecture of the software plays an important role, and it can be evaluated similarly. On the other hand the infrastructure of an open source project might not be so important when evaluating open source software to be used, however it is a crucial element when building an open source community.

2.2 Opening Proprietary Software

Like any other online community [7], creating and maintaining a sustainable open source community for proprietary software can be considered as a multi-facet challenge. It is a complex process that is driven by various kinds of factors, which in turn 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.

In the next section, we present a framework that addresses these questions by evaluating software in a pre-bazaar phase. The pre-bazaar phase helps in getting early feedback and experiences on using and evolving the software outside its original development environment. This may need the involvement of external stakeholders.

3 The Release Readiness Rating Framework

The Release Readiness Rating (R3) framework is a tool for planning the open sourcing of a software system. The goal of the framework is to help identifying possible bottlenecks and to eliminate them in so-called pre-bazaar phase, whose goal is to prepare the software to be released and the releasing company to the continuation of the life of the system as open source.

3.1 Framework Overview

The evaluation criteria for R3 consists of four different dimensions, including software itself, intended community and its roles, legality issues, and the releasing author. The output of the evaluation can be considered as a vector that determines the relative values of these different elements. The dimensions are further decomposed as indicated in Table 1. The table also lists the relative importance of the item.

The current weights are based on our experience on previous case studies. All dimensions are equal in weights when the individual items are associated with different weights.

3.2 Evaluation Criteria

In the following, we discuss the different views that should be considered when deciding the value set for the items representing different dimensions.

Table 1. R3 dimensions, items and relative weights

Dimension	**Item**	**Weight**
Software		*0.25*
	Source code	0.5
	Architecture	0.4
	Quality attributes	0.1
Community		*0.25*
	Purpose and mission	0.4
	User community	0.4
	Partners	0.2
Legalitites		*0.25*
	Copyright	0.6
	Licensing	0.3
	Branding	0.1
Releasing authority		*0.25*
	Mindset, culture and motivation	0.5
	Process, organization and support	0.3
	Infrastructure	0.2

Software The software itself forms an important aspect for any open source project for obvious reasons. Based on our experience, at least the following issues must be taken into account.

Source code. Fundamentally, any open source project deals with source code. When releasing a new software system to open source, there are numerous properties that the system itself, manifested in its code, should contain. These in particular include *quality of code, integrity, and coding conventions* hat help other developers to participate in coding. The importance of coding conventions is highlighted, since introducing coding conventions as an afterthought can turn out to be impossible. Moreover, the source code should express *code of conduct*, which is a necessity for making the code public. Finally, *documentation* of the system is a practical necessity for attracting other developers.

Architecture. In order to make a software system easily approachable, it must be easy to understand by the developers. This in turn calls for an architecture that can be easily understood and communicated - and preferably documented. In addition to this, one should pay special attention to the design drivers of the architecture: Is the system designed as a monolithic system that solves a particular problem, or has the design taken into account extensibility, modifiability, and the use of the system as a subsystem in another system. Provided that the architecture has been designed with changes in mind, it is often easier for other developers to alter certain parts to create various types of new systems.

Quality attributes. In software, quality attributes are commonly associated with architectures. Indeed, many qualities, such as performance, scalability, and memory footprint, are often dictated by the architecture. However, when considering a completed software system, quality properties are often considered separately from the actual implementation, which makes the perceived quality of software being released as open source an important factor.

Community In order to release a software system in open source, one generally has an idea on what kinds of developers should get involved. Careful planning of the intended community participants can have an impact on what to release and how.

Purpose. As already stated in [9], good work on software commonly starts by developers scratching their own itch. Therefore, we feel that in order to become attractive for developers, the released system should be of practical importance and relevant for the developers. This in turn enables one to solve their own problems, not further developing some random software for companies who seek profit in maintenance and creativity of others. Based on the above, the goal of the community is probably the most important single issue when releasing a piece of software as open source. Provided with a mission statement welcomed by developers, companies, and other organizations, a community can obtain support from numerous sources. The goal must be practical enough to be meaningful for the developers, as well as clear enough to manifest itself in the development. Unfortunately, estimating the attractiveness of a certain purpose is difficult, and therefore it is sometimes difficult to make assumptions in this respect. Moreover, since there commonly are numerous similar ongoing projects, the adequacy of the mission is only a prerequisite for a successful launch, not an automata for succeeding in community building.

User community. In addition to partners developing software, we feel that the potential for the user community is important. Based on recent findings, it seems that a community of 100 users can support one full-time developer. In contrast, provided with an active user community, development resources can be invested in actual development, and the user community can provide support for other activities, such as peer user guidance, documentation, and testing.

Partners. The definition of partners that join in the community can be straightforward. For instance, if a releasing company has been subcontracting from another company, the latter may be automatically involved in the newly formed community. Moreover, the use of subcontracting may also imply that at least some documentation exists, which in turn simplifies introducing the system to other partners. In general, getting partners involved in a community guides the authoring company towards processes that liberate the development from company specific tools and practices. In contrast, if the company that is about to release a piece of software as open source has no partners that would share the interest in the development, there should be a clear plan to motivate others to join in the development effort.

Legalities In the context of companies, one of the most commonly considered aspects of releasing software as open source is legalities. This is a wide topic to cover, and there can be several subtle differences in different contexts. Here, we assume a straightforward view where different concerns are discussed independently.

Copyright and intellectual property rights (IPR). Most commonly, companies release software whose copyright and IPR they own. However, things can be more complex, if subcontractors or open source communities have provided pieces of the system that is being released. If the company does not own the copyright, it can sometimes be obtained via different transactions.

Licensing. Provided with the copyright of the system to be released, the company is in principle somewhat free to determine its licensing scheme. However, the choice of license (or licenses if several alternatives are offered) also has an effect on how others perceive the community. This in turn potentially affects the willingness of developers to participate in the community effort. Therefore, since licenses that have strong copyleft, such as GPL [6], can be considered safe for community building, they bear some advantage over other licenses in this respect. However, since licenses with no copyleft, such as MIT [6] and BSD [6], are favored due to their liberal flavor in some other contexts, one should also take the mission of the community into account when defining the license. Moreover, license compatibility with related systems should also be considered. Furthermore, one can even compose a list of accepted open source licenses, which can be used during the development.

Branding. Availability of brand names is an issue for any project looking for a good name that can be used in public. While issues such as trademarks may bear little significance when developing an in-house product, once the product is released, branding becomes an issue. Moreover, since an open source project can be a long lasting one, selecting suitable brand names is important. The same applies to hosting the project, since in many cases it would be practical to reflect the name of the project also in the domain.

Releasing Authority The final element we address in our framework is the releasing authority, most commonly a company in the scope of the framework, which is targeted to releasing in-house software as open source. However, also other parties can act as the releasing authority, including universities, non-profit organizations, and individuals.

Mindset, culture, and motivation. Sometimes the mindset of developers working in a company is somehow biased - either positively or negatively - towards open source development. This is particularly true when their pet project is about to be open sourced. In order to benefit from an open source community, the releasing authority should be mentally and culturally ready for dealing with developers outside the company under fair terms. The terms include equal access to code, similar guidelines and conventions, as well as mutual respect. We believe that the seeds of building such cooperative relation should somehow be sewn well before entering the pre-bazaar phase. Therefore, evaluating the

readiness of the company to go for open source is fundamentally dependent on mindset, culture, and motivation.

Process, organization, and support. In order to gain benefits from open sourcing a system, the releasing authority should have a system in place that provides support for users and developers. This requires planning of a process that is to be followed, and putting the process in practice by the support organization. Establishing such support organization is a natural step to take towards the end of the pre-bazaar phase. However, it should be in place before the actual release, since support should be available from the very beginning.

Infrastructure. In order to establish an open source project, the releasing authority sometimes must be prepared to provide infrastructure. For instance, the company that releases a piece of software may provide web servers for hosting the system, as well as maintain a build system needed for compiling the code on top of certain reference hardware. While some systems do not need such support as such - it would be perfectly reasonable solution to release a vanilla Linux program in SourceForge - companies often wish to gain visibility through offering the download opportunity. Moreover, if a company is releasing a system targeted for the development of embedded systems, it is only reasonable to assume that also tools for composing builds are offered from the very beginning in open source.

The R3 evaluation model is organized into three main levels. For each dimension there are a number of categories. Each category is then associated with a number of measures (i.e. questions). This is illustrated in Figure 1 taking the software dimension as example.

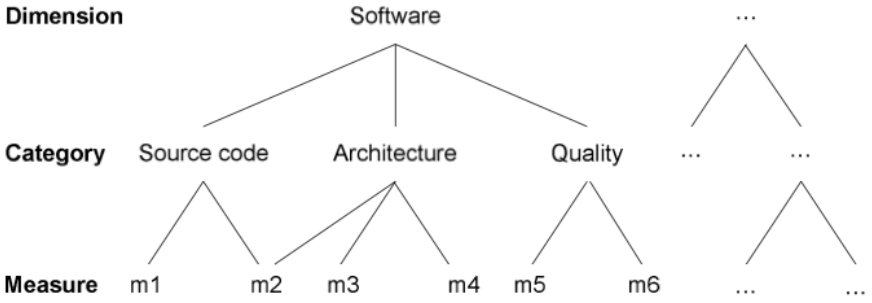

Fig. 1. The R3 framework model

3.3 Evaluation Process

The diversity of software products (and different goals of companies) makes it impossible to evaluate all software in similar fashion. The evaluation model

itself must be tuned to take into account the characteristics of the product under release. Not all criteria make sense to all cases, and some crucial criteria might be missing. The proposed R3 model should be considered as a template which has to be instantiated to each case. Instantiation R3 means going through all aspects of the model and validating that they are appropriate to the case in hand.

At the concrete level the evaluation process starts with downloading R3 Spread Sheet Template from `http://tutopen.cs.tut.fi/R3/R3_Template.xls`. Instantiating the template requires removing and adding dimensions and criteria to the spread sheet. Also the evaluation weights require attention from the evaluator.

On the Criteria The evaluation criteria form a continuum from a criterion that can stop the release process to others that can be easily fixed. Examples of the former are some legality issues, like possible copyright and IPR violations, and probably mindset of partners. Changing these is hard or even impossible. The latter group can be worked on during the releasing process: usability, and quality can be improved; infrastructure and process can be organized. An example continuum is depicted in Figure 2.

Fig. 2. The continuum of criteria

Actual Process The evaluation process consists of three phases depicted in Figure 3

Phase 1: Deciding the evaluation order. The evaluation is carried out according to continuum of criterion. This allows early no-go decisions. If, for example, the company does not own copyright of the product, it makes no sense to continue the process. However, there is no one unique and universal order of the criteria, but the order is fixed in the beginning of the process.

Phase 2: Data collection and processing. Most of the work is done in this phase. The criteria are evaluated one by one in the order fixed in the first phase. This activity is shaped by the framework model presented in Figure 1. First a dimension (e.g. software) is picked, then a specific category (e.g. source code) is selected, and finally concrete questions are answered. After each evaluation the decision of continuing the release process is made. Each measurement is filled to a standard spreadsheet with justification of the evaluation.

Measures require expertise from different fields: engineers, marketing people, legal experts and external open source experts. For example, the legal department of the company is often contacted in the beginning of the evaluation to verify the copyright and possible IPR issues of the release. Engineers take care of technically-oriented criteria. External open source experts have probably the best understanding of the whole release process, and they know how open source communities operate. The releasing process reminds much of marketing challenges, therefore, marketing people are valuable for the process.

Phase 3: Data translation. In the data translation phase the evaluation of the criteria is transformed to an array of scalars with respect to the dimensions of the framework. The final result of the process is a four-dimensional array of evaluations of each dimension: software, community, legalities and releasing authority.

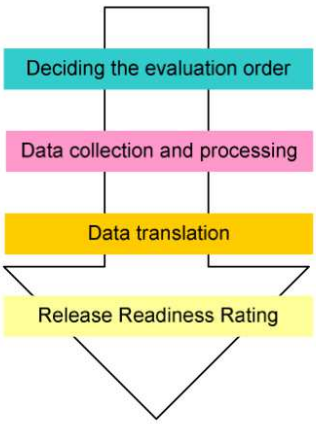

Fig. 3. Evaluation process

3.4 Open Source Engineering

R3 assessment of proprietary software is not just a pass-or-fail process. The outcome of R3 evaluation is a set of recommendations based on which the software under evaluation and its development environment undergoes an open source engineering process. This process needs to be carried out before evangelizing the software to the open source community. The aim is to eliminate the problems and shortcomings identified during the assessment process. This will increase the success rate of community building and sustaining. The open source engineering process itself is driven by different kinds of influential factors that follow the same criteria as we used in the R3 framework.

In the case of the *software* itself, any considerable rework requires an extensive investment. Therefore there are numerous restrictions on what can be accomplished during the pre-bazaar phase. Still, it is possible to clean up the code, if there are some company specific remarks in comments. Since the code may already be in use in products, special attention must be paid to determine what to do with comments that indicate faulty or incomplete features. To some extend, documentation can also be composed in pre-bazaar phase, or simply included in the comments in the code.

Adding purpose to a *community* as an afterthought can be difficult. Assuming that a system has been developed with only business interests in mind, it can be difficult to introduce attractions for an independent developer. However, a mission for a community can be defined in pre-bazaar phase, provided that the software to be released enables a number of possible uses. Unfortunately companies can be somewhat biased towards supporting their own plans regarding the released system only, which in turn sometimes hinders the outside participation in the development for reaching some other goals, especially if the missions are conflicting. For instance, the releasing authority may not be willing to incorporate a community contribution for free, if the same feature can be sold for a commercial customer by the company.

Legalities most commonly form the most straightforward category of items. There is a lot of freedom to define the other legal aspects once the copyrights have been provided. However, copyrights can be difficult to obtain in pre-bazaar phase only.

The seeds of building cooperative relation between the *releasing authority* and the actual community should in our opinion be somehow sewn at the latest when entering the pre-bazaar phase. This can already be evidenced by existing ways of working and infrastructure, but they can also be introduced later on.

4 Case Study

In order to demonstrate our approach, we have applied the R3 framework to measure the open source readiness of an industrial software platforms: Wringer and Gurux. The Wringer software is a JavaScript binding platform for GNOME/GTK+ [4] using V8 [12] as JavaScript engine. It was originally developed by Sesca Mobile Oy. The Gurux software [5] is a platform for developing device communication systems.

4.1 The Wringer case

The R3 evaluation of the Wringer platform has been carried out separately by the releasing authority as an internal stakeholder and by us as external open source experts. It was observed that we agree on most answers. However there are still a number of differences. For instance, in the software dimension, there

were few differences with respect to rating the technology used, the use of well-known design principles, rating of bugs and warnings, and scalability level. We received more pessimistic answers from the releasing authority. A partial reason for this could be that the company is assessing Wringer, which is a prototype software, relative to other high quality products developed inside the company.

On the opposite, the answers with respect to the community perspective for instance have been mostly identical. This probably shows that both parties are aware and honest about the community-related properties of the software. Also this shows that both parties are fairly aware of related user and developer communities.

Taking a numerical perspective, we noticed that the software and legality dimensions received the best scores compared to the community and releasing authority dimensions. This confirms our hypothesis that companies are generally dealing with open source from legality point of view. We could also infer that software-related properties are considered important irrespective of whether the software is supposed to be used and developed as closed source or as open source.

The low rating of the community dimension suggests that the company have to work on making the software more attractive to open source communities, involve business partners if possible, and look for potential users of the software. As for the releasing authority dimension, the low rating suggests that the company is not fully ready for open source operations. Concrete remedial actions include training internal developers, setting clear open source related processes, and building an infrastructure for the project. As mentioned earlier, these remedial actions are to be carried out in the context of an R3 evaluation post activity called the open source engineering process.

4.2 The Gurux case

Four people from the releasing authority carried out the R3 evaluation for the Gurux platform before it was released as open source in November 2009. The overall impression was positive and R3 was considered a valuable and useful tool in the pre-bazaar phase. The evaluation process acted as a good checklist for things that need to be considered when planning the release and showed well the items where most improvement is required. In addition, those items where improvement would be most beneficial were easily identified with R3, i.e. the releasing authority knew where to focus most of the effort.

Some items were not seen relevant in the case of the Gurux platform. However, this was not a significant problem for the process as irrelevant items were simply skipped by the people doing the evaluation.

Sometimes choosing the correct grading was found hard. Grades like "well" and "reasonably" may mean different things to different people as these types of grades depend on how things are seen by individuals doing the evaluation and what they value.

5 Future Work

We are in the process of improving the R3 framework based on our experiences with the case projects. This includes adjusting the weights, proposing new metrics, and covering other dimensions if found necessary. Currently the weights are chosen based on experience gained from earlier similar case studies. As the framework is applied to further cases, enough data will be gathered to enable us to better finetune the weights.

Furthermore, the case studies confirmed that it is difficult to come up with a one R3 framework template for all software projects. For instance, usability as a quality metric should be considered for end user software but was found less relevant in the case of software platforms like Wringer and Gurux. We plan to provide different templates for different kinds of software. Still, it is highly probable that the R3 framework template needs to be adapted to the needs of the subject software on a case by case basis.

6 Conclusions

Similarly to other major steps in software development, aiming at releasing a piece of in-house software as open source requires an engineering effort. Moreover, in order to estimate the outcome of the release, tools and techniques are needed for evaluating the potential of the emerging community as well as the attractiveness the system for external developers.

In this paper, we have introduced the concept of Release Readiness Rating Framework to determine how complete an in-house piece of software is for releasing in open source, and what its potential to attract external developers is - in essence evaluating how easily a cathedral could be transformed into a bazaar. We also discussed potential engineering actions that can be taken as a part of this transformation process, and provided a summary of two industrial cases.

References

1. BRR. `http://www.openbrr.org/`. Last visited March 2009.
2. Karl Fogel. *How to Run a Successful Free Software Project.* O'Reilly Media, Inc., Oct 2005.
3. Bernard Golden. *Succeeding with Open Source.* Addison-Wesley, 2004.
4. GTK+. `http://www.gtk.org/`. Last visited March 2009.
5. Gurux/open source. `http://www.gurux.fi/index.php?q=OpenSource`. Last visited December 2009.
6. Licences. `http://www.opensource.org/licenses`. Last visited February 2009.
7. Jenny Preece. *Online Communities: Designing Usability, Supporting Sociability.* Wiley, 2000.
8. QSOS. `http://www.qsos.org/`. Last visited March 2009.

9. Eric S. Raymond. *The Cathedral and the Bazaar*. O'Reilly Media, 1999.
10. Matthias Stürmer. Open source community building. licentiate thesis, 2005.
11. the Symbian Foundation. `http://www.symbian.org/`. Last visited December 2009.
12. V8 JavaScript Engine. `http://code.google.com/p/v8/`. Last visited February 2009.