An Outline of Development Process Framework for Software based on Open-source Components

Jakub Swacha, Karolina Muszyńska and Zygmunt Drążek
Institute of Information Technology in Management, University of Szczecin, Mickiewicza 64, 71-101 Szczecin, Poland

Keywords: Open-source Software, Software Adaptation, Development Methodology.

Abstract: The open-source community produces a wide range of software products every year. However, there are often significant differences between the set of functionalities and/or non-functional requirements demanded by the end-users and what the available software offers. Therefore, often an open-source system cannot simply be adopted; it has to be adapted. In this paper we propose a new process framework for adapting open-source software. We call it FEChADO which is an acronym of the six steps it consists of: Find available solutions, Evaluate solutions from the list, Choose the most appropriate solution, Adapt the solution, Develop new modules, Obtain users' feedback. The framework is a direct result of our practical experiences from developing software based on open-source components.

1 INTRODUCTION

1.1 Motivation

The open-source software (OSS) becomes increasingly popular and open-source solutions can be found for many applications. However, ready-made open-source solutions rarely fully meet the end users' requirements. Therefore, most of them require adaptation in order to make the users satisfied.

Although there is vast literature on software development process, and significant literature on selecting open-source software (to be discussed in section 2), until now little has been written on the process of developing software based on open-source components, where it is important both to choose the most appropriate existing solution and to adapt it to the requirements of the specific end users.

1.2 Problem Setting

We understand adapting OSS as tailoring it to the needs of a specific end-user or a group of them. The permission for users to modify code for private purposes is a core property of the open-source software. Such modifications are rarely useful for other users, so usually there is no reason for further redistribution of the modified software, and the relevant license requirements do not matter. Otherwise, contributing it to the community should be considered with all such requirements met.

1.3 Approach

We assume open-source software adaptation to be a repeatable process, hence a framework can be defined for its efficient execution. Such a framework could be based on theoretical inference starting with requirements, or obtained by generalization of practical experiences. We chose the latter and use observations made during our involvement in OSS adaptation projects (see, e.g., Swacha et al., 2011).

We consider six main stages that form the development process framework, describing an exemplary method of their implementation. The proposed framework can be seen as high-level and flexible as it does not enforce a specific method for any of the stages; any method is suitable for use provided it accepts the available inputs and produces the required outputs.

1.4 Contributions

Our main contribution is the development process framework that can be used in various organizations for adapting open-source software.

2 RELATED WORK

Before formulating the proposed process framework numerous research findings, existing solutions and scientific papers were verified and examined.

The QSOS method for qualification and selection of open source software (Atos Origin, 2006) is performed in four stages: definition of frames of reference, evaluation of software by identifying its main characteristics, defining its functional coverage and risks, qualification of the software using filters to translate needs and constraints to the selection of most suitable solution, and selection of software based on the outputs of the previous stages.

M. Cabano et al. (2007) based their context dependent evaluation methodology on common structural pattern shared by most widely known evaluation models like Open Source Maturity Model by B. Golden (2005), or Business Readiness Rating for Open Source by Carnegie Mellon West and Intel (OpenBRR, 2005). The assessment process of the context dependent methodology is composed of: context analysis, preliminary selection, and filtered selection.

R. Galoppini (2011) in his pragmatic methodology, indicates the best sources to find OSS and lists evaluation criteria to be applied.

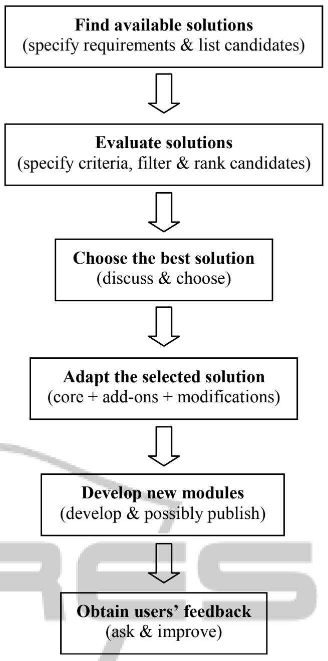
Open-source software evaluation process described by D.A. Wheeler (2011) consists of four steps: identify candidates, read existing reviews, compare the leading programs' attributes to the needs and analyze the top candidates in more depth.

A bit different approach, which concentrates on the software quality is described in a paper by G. Polancic et al. (2004). It is based on multiple criteria of quality, that can be checked using accessible quantitative data.

The main goal of the abovementioned OSS evaluation methods and frameworks is to identify, assess, sometimes also compare, and select open source products. The proposed FEChADO framework concentrates on software development process based on existing OSS components, it assumes that the OSS evaluation process must be followed by additional adaptation and development phases.

3 FEChADO FRAMEWORK

We call our process framework FEChADO which is an acronym of the six steps it consists of (see Fig. 1).



Find available solutions
(specify requirements & list candidates)
Evaluate solutions
(specify criteria, filter & rank candidates)
Choose the best solution
(discuss & choose)
Adapt the selected solution
(core + add-ons + modifications)
Develop new modules
(develop & possibly publish)
Obtain users' feedback
(ask & improve)

Figure 1: Overview of the FEChADO framework.

3.1 Stage 1: Find Available Solutions

Phase 1.1: Specifying Requirements. In the first step, the set of functionalities and non-functional requirements of the end-users has to be defined.

If the general description of the software fits into an established software category, a list of suggested requirements may be composed, based on characteristics of software belonging to that category. Otherwise, a brainstorming session involving both designers and users can be organized to obtain such list.

This list is then used as a base for a questionnaire to be filled in by the end users. They value each proposal and are also asked for their own proposals that were not mentioned in the questionnaire.

The results from the questionnaires are then processed and analyzed in order to obtain a list of requirements arranged in three groups: (1) core requirements (that must be met), (2) additional requirements (that should be met), (3) special requirements (that only specific types of end users believe should be met). The list is then presented to the end users for correction and re-evaluation.

Phase 1.2: Listing available Solutions. There are various sources that can be searched for solutions that possibly match the specified requirements (see, e.g., Golden, 2005). We recommend starting with searching open source project portals, such as sourceforge.net and freecode.com, as well as using web search engines.

Multiple search phrases should be tried, as different words may be used to describe equivalent functionalities. The search should be continued until no more matching solutions can be found.

3.2 Stage 2: Evaluate found Solutions

Phase 2.1: Specifying evaluation Criteria. The evaluation criteria are specified by assigning measurable fulfilment levels to the requirements. Every criterion should have at least one defined fulfilment level: *acceptable*; that is, solutions evaluated below this level must not be chosen. However, there could be more levels defined, so that solutions that pass the acceptable level could be compared between each other. A *satisfactory* level can be defined, such that no solution can be evaluated as better with regards to a given criterion if they both attain this level.

The evaluation criteria can be grouped, depending on their measurability, into: (1) objective and easily measurable, (2) objective and not easily measurable, (3) subjective.

The weights of criteria may be established using a simple ranking done by end users, or using more sophisticated approaches, like pair-wise comparisons (see, e.g., the AHP method – Saaty, 1980).

Phase 2.2: Preliminary evaluation. The goal of the preliminary evaluation is to shorten the candidate list by removing software that does not attain *acceptable* level for criteria based on the core requirements. In this phase only objective and easily measurable criteria should be considered.

Phase 2.3: Main evaluation. This phase starts with eliminating software that does not attain *acceptable* level for objective criteria (now also not easily measurable) based on the core requirements.

Both the criteria fulfilment levels and criteria weights are scaled and normalized, and then used to construct an aggregate measure for each candidate solution and group of criteria (core criteria, combined core and additional criteria, and all criteria). One or two highest-evaluated solutions are chosen for each group of criteria.

Phase 2.4: In-depth evaluation. The goal of this phase is to prepare full information that will be considered for making the choice, including subjective criteria. Evaluation is done by several people, working as a team or independently – the results are averaged in the second case. In contrast to the two previous phases, the end users should be involved in the evaluation.

3.3 Stage 3: Choose the Most Appropriate Solution

Phase 3.1: Discussion of evaluation Results. A meeting of the stakeholders should be organised that starts with presentation of the evaluation results after which every person should express their opinion and provide additional information that could impact the choice, e.g.:

- the end users should point to drawbacks or special advantages of respective solutions,
- the invited experts should clarify, if the mentioned drawbacks are specific to solutions, or they are merely results of improper usage,
- the members of the development team should declare if they are capable of fixing the drawbacks, and what the cost would be ,
- the sponsors of the project should declare if they will contribute necessary resources.

Phase 3.2: Making the Choice. The decision on choosing the solution for adoption is made by the sponsors of the project. It should be based on the results of the evaluation process, but if two or more solutions were evaluated closely, the decision maker(s) should pick one of them using their own opinion rather than the aggregated measure value.

3.4 Stage 4: Adapt the Solution

Adapting the solution consists of acquiring, installing, and configuring the core solution, then the required add-ons, and, finally, applying modifications required for the solution to meet the requirements.

What makes the modifications applied at this stage distinctive from those of the next stage is that they are aimed at the requirements of the specific end user group, and as such they will rarely be useful for other users, and that they are not usually defined as a separate entity (module or even function), being often a list of file/line updates.

Every modification made to the original solution must be explained in the technical documentation of the final product, in terms of its purpose, relation to other modifications, assumed conditions and possible risk factors. Regression testing should assure that the modification does not hurt stability, security or degrade performance of the system.

3.5 Stage 5: Develop New Modules

Sometimes, the modifications related to a certain group of requirements can be implemented as a new module for the chosen solution.

Developing a module should be done keeping straight with the guidelines defined in the solution's documentation. The new module should integrate seamlessly with the solution, and be configurable, preferably from the solution's administration panel, if such exists.

If the organization's internal regulations permit, and there are no redistribution constraints in the license of the original solution that could be violated, it should be considered to contribute the module to the open-source community.

Having decided to publish the new module, it should be pruned from elements that have any value only for the sponsor's organization. Depending on the number and character of such elements, it may be accomplished in three ways: (1) by turning the adaptation elements into a profile of module configuration settings, (2) by moving part of functions to an additional module, only for internal usage, (3) by forking the module into internal (full) and external (limited) versions.

If the module published as open source gains popularity, it may even attract external developers who may improve it. In such case, it should be checked if the improved version of the module could be used instead of the one developed internally.

3.6 Stage 6: Obtain Users' Feedback

It is important that the end users' opinions on the final product are gathered. The opinions that pertain to the implemented modifications can be used by the internal development team to improve them. Contrasting opinions should be resolved via discussion with the involved users.

The opinions that refer to the chosen solution should be passed to its original developers, especially bug reports and feature requests.

In order to facilitate the feedback process, a web form should be made available for the end users, so that their opinions could be reported easily.

4 PRACTICAL EXPERIENCES

The development process framework outlined in this paper has been applied in a complex project (Swacha et al., 2011), consisting of five components, four of which were adaptations of open source systems. The core of the framework was created after developing the first component of the mentioned project, and applied, in limited or full extent, to the remaining ones. The development of the framework was evolutionary: each application resulted in experiences that allowed for further improvement of the framework.

The framework has been positively evaluated by the respective decision makers and developers.

5 CONCLUSIONS

We have outlined a framework for software development process based on open-source components.

The framework described in this paper can be applied to any software development process that involves OSS adaptation. It helps achieve quality of the final products at the same time being simple and non-obtrusive.

REFERENCES

Atos Origin (2006). *Method for Qualification and Selection of Open Source software, version 1.6.* Retrieved Feb 1, 2012 from www.qsos.org/download/qsos-1.6-en.pdf.

Cabano, M., Monti, C., Piancastelli, G. (2007). Context-Dependent Evaluation Methodology for Open Source Software. In Feller, J. Fitzgerald, B., Scacchi, W. and Sillitti, A. (Eds.), *Open Source Development, Adoption and Innovation* (pp. 301–306). New York: Springer.

Galoppini, R. (2011). *How to find an Open-Source Alternative to a Commercial Software*. Retrieved Feb 1, 2012 from http://www.masternewmedia.org/open-source-software-tools-and-directories-where-to-find-them-how-to-evaluate-them.

Golden, B. (2005). *Succeeding with Open Source*. Boston: Addison-Wesley.

OpenBRR (2005). *Business Readiness Rating for Open Source*. Retrieved Feb 1, 2012 from http://docencia.etsit.urjc.es/moodle/file.php/125/OpenBRR_Whitepaper.pdf.

Polancic, G., Horvat, R. V., Rozman, T. (2004). Comparative assessment of open source software using easy accessible data. In Luzar-Stiffler, V. and Dobrić, V. H. (Eds.), *Proceedings of the 26th International Conference on Information Technology Interfaces* (pp. 673–678). Zagreb: IEEE.

Saaty, T. L. (1980). *The Analytic Hierarchy Process*. New York: McGraw-Hill.

Swacha, J., Muszyńska, K., Komorowski, T. and Drążek, Z. (2011). Development and maintenance of a multi-lingual e-Tourism website on the example of BalticMuseums 2.0 Online Information Platform. *Information Management*, 3, 237–246.

Wheeler, D. A. (2011). *How to Evaluate Open Source Software / Free Software (OSS/FS) Programs*. Retrieved Feb 1, 2012 from http://www.dwheeler.com/oss_fs_eval.html.