CONTEXT-DEPENDENT EVALUATION METHODOLOGY FOR OPEN SOURCE SOFTWARE

Michele Cabano, Cesare Monti, and Giulio Piancastelli
DEIS, Dipartimento di Elettronica, Informatica e Sistemistica
Alma Mater Studiorum — *Università di Bologna*
via Venezia 52, 47037 Cesena (FC), Italy
[michele.cabano, cesare.monti, giulio.piancastelli]@unibo.it

Abstract Many evaluation methodologies have been proposed to mitigate the risks of choosing Open Source Software as an effective solution to an enterprise's problem. This work extracts the shared traits from the most important and widely known evaluation models, and re-applies them to create a new methodology. This methodology has been designed both to be used for the creation of a common knowledge base, and to be specialized for application in the context of the particular breed of small- and medium-size enterprises found on the Italian ground.

1. Introduction

Selecting an Open Source Software (OSS) application as the appropriate solution to an enterprise's problem has always been an activity prone to many risks of different nature [4]. Proposed solutions to this choice problem have taken the form of evaluation methodologies [3, 1], aimed at assessing a software package's maturity for business IT adoption.

Open Source embracement is critical especially for small- and medium-size enterprises, where the consequences to a wrong decision in the picking of an Open Source alternative to a commercial product might not be as well-absorbed as in the context of bigger corporations. For this reason, the Italian region Emilia-Romagna funded the OITOS project,[1] started in 2005 with the goal of creating an Observatory for Innovation and Technological transfer on Open Source software. The project aims at strategic evaluations of Open Source solutions, in order to limit and control the risks for enterprise IT adoption. For that purpose, by re-applying a common structural pattern shared by the most important

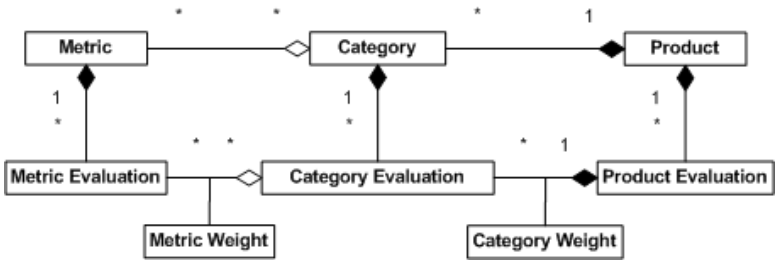

Figure 1. An UML diagram illustrating the evaluation meta-model, where a single metric can influence more than one category, and weights are used to take into account the final rating's dependency from the evaluation context.

and widely known methodologies, we designed a new OSS evaluation methodology, both to build a knowledge base classifying Open Source products, and to be applied in the specific business contexts of small- and medium-size enterprises participating in the OITOS project.

2. Modeling existing evaluation methodologies

We identified three most important works in the area of OSS evaluation methodologies, starting from the first Open Source Maturity Model [4], created by Bernard Golden at Navica; another Open Source Maturity Model has been later developed at the CapGemini consulting company [3]; finally, Intel Corporation, Carnagie Mellon University, and others are currently proposing a standard and open evaluation methodology for OSS, named Business Readiness Rating (BRR) [1].

Despite possible dissimilarities among the listed works, we believe each methodology always consists in the application of an *evaluation model* to one or more software packages during the execution of a well-defined *assessment process*. Both the process and the model can be reduced to a common pattern solving the Open Source evaluation problem.

2.1 The evaluation meta-model

As shown in Figure 1, all the examined evaluation models share a common meta-model subdivided along three layers. Those layers aim at refining a product's evaluation by aggregating raw data under ratings of similar assets, until an overall score is determined.

A metric is defined as a measurable property of an open source software project.[2] Metrics are organized into categories representing common areas of interest. The aggregation of categories represents the whole product under analysis. An evaluation is assigned to each metric, corresponding to the value reached by the software on that specific property.

Figure 2. The three phases in the skeleton of the assessment process.

The evaluations of metrics sharing their category are combined into category evaluations, which are then aggregated into the final product rating. Weights could be assigned to categories and metrics to account for the context-dependent importance that some aspects of a product may have.

The examined models feature common categories to evaluate quantitative data taken from objective sources, such as the project's infrastructure. Differences between models result in the remaining categories, which try to capture subjective data and normalize qualitative information taken from other sources, such as code analysis or product use.

2.2 The assessment process skeleton

As illustrated in Figure 2, the prototypical *assessment process* for OSS evaluation methodologies is composed by three phases: (1) Data Gathering; (2) Data Analysis through predefined metrics grouped by category; (3) Numerical Synthesis. The process can be refined, as it happens with the Quick Assessment filter of the BRR [1], or further detailed, as in the seven steps of CapGemini's Maturity Model [3]; but the outlined three phase pattern still maintain its validity.

The Data Gathering phase can be accomplished by collecting information indirectly through reuse of third party evaluations, or directly from primary data sources such as the project's support infrastructure and web sites promoting the software product. The Data Analysis and Numerical Synthesis phases represent the application of the evaluation model in two steps: first, following a numeric scale, a score is assigned to each metric, then weighted and aggregated on the basis of metric's areas of influence to calculate the rating of each category; second, product's evaluation is obtained by applying different weights to category ratings.

The assessment process must be frequently repeated over time for each software package, due to the responsive nature of Open Source projects.

2.3 Towards a pattern for software evaluation

We believe that the meta-model and process skeleton detailed in Section 2.1 and 2.2 form the structure of a *pattern*[3] for the creation of the two key elements in a software evaluation methodology. Patterns distill and provide a means to reuse the knowledge gained by experienced

practitioners [2]. In fact, the proposed software evaluation methodology pattern has been extracted from the solutions to the software evaluation problem proposed throughout the latest years. Besides, the creation of the BRR methodology shows that this solution pattern to the evaluation problem, on which the BRR is based, is considered mature enough to be openly discussed and worth of a standardization attempt.

3. The OITOS methodology

Started in 2005, the OITOS project is the first attempt made by the local government of an Italian region to spread Information Technology knowledge on its territory. The project aims at strategic evaluations of Open Source solutions under the perspective of enterprise IT adoption. The OITOS project involves a lot of companies which operate in several sectors. Different organizations have different technical requirements in the choice of software products: an assessment made for a company can rarely be totally reused for another. Thus, our primary task was finding a set of metrics which could always be used in any assessment, in order to create a knowledge base common to every organization.

3.1 Creating a knowledge base

Since existent evaluation models did not consider some important metrics the OITOS project's context demanded (*e.g.* migration costs or interoperability) we needed to create a new model to match our own requirements. The OITOS evaluation model was thus developed following the structure described in Section 2.1 without considering weight factors. The model was built to evaluate open source projects, rather than products, around three categories, aimed to measure a set of objective properties of general interest in a quantitative way: (1) Development; (2) Community; (3) Transition. The *Development* category measures the work made by project developers, and is composed by the Documentation, Developers, Tools, and Composability[4] metrics. The *Community* category measures all the resources offered by users, scientific communities and private companies, and is composed by the Visibility, Success Stories, Support, and Fundings metrics. The *Transition* category estimates how much the product is adaptable to existent organizational structures and configurable for its effective use, and is composed by the Configurability, Set-up, Use, Migration, and Interoperability metrics.

Needing general assessments in order to gain maximum reusability, we opted for the simplest possible evaluation process: it relied on a single phase called *Preliminary Assessment*, which included all the three steps highlighted in the prototypical process characterized in Section 2.2.

3.2 Context-dependent software evaluation

Companies participating in the OITOS project also needed evaluations customized to their particular technological context. Willing to reuse the objective and more general parts of each evaluation, the model and process described in Section 3.1 were extended piecemeal to produce a context-dependent methodology: metric weights from the meta-model in Section 2.1 were introduced, a new *Technology* category was added, and the process was extended to comprise three different phases.

Technology category groups all the metrics relative to features which an enterprise may consider important for its specific context. These metrics usually measure facets connected directly with the software product, never with the open source project's infrastructure. For this reason, rating those metrics is a subjective activity, but evaluations are more customized to their technological context than in other categories.

The assessment process is composed by three phases: (1) Context Analysis; (2) Preliminary Selection; (3) Filtered Selection. During the Context Analysis phase, information concerning a company are gathered in order to define its necessities. Each need is resolved by a software class which will be estimated by the model. After identification of software classes, a list containing possible products to introduce is edited, and a complete set of evaluation metrics is defined. In the Preliminary Selection phase, a set of most critical metrics is extracted. To each critical metric is associated a threshold, which must be surpassed to grant a software product access to the next phase. During the Filtered Selection step, software products passing Preliminary Selection are estimated once more, by the complete set of metrics defined in the Context Analysis phase. There are no thresholds assigned to the metrics: instead, each metric receives a 1 to 10 value which may be modified by a weight factor.

With reference to the process pattern outlined in Section 2.2, the Data Gathering phase has been split onto all the three phases in the OITOS methodology. During Context Analysis, data is collected about the company for which the evaluation is performed.[5] In the Preliminary Selection phase, we refined the process in the fashion of BRR's Quick Assessment filter [1]: data is gathered for critical metrics, and a first Data Analysis activity is performed to rule in or out software packages from the next step. During Filtered Selection, the complete evaluation of software is accomplished, featuring all the three phases in the pattern.

4. Open issues and future work

The Data Gathering and Data Analysis processes are driven by two sets of questions, which let evaluators collect the raw material to be

used for assessing a product's rating, and establish the influence of each metric on the higher category layer. These two sets are of fundamental importance, since they comprise the point of view that the methodology enforces on the software product being evaluated. Unfortunately, even if those questions are sometimes contained in accompanying documents such as user's guides or appendixes, more often they are nowhere to be publicly found, especially when consultancy agencies are involved.

Despite the endeavor to control subjective information and normalize qualitative data, a certain degree of discretion is still possible while performing a software evaluation. Subjectivity has not been eliminated: it has just been transferred to the normalization process, which at most can be driven by a set of best practices, not a well-defined specification. If the influence of the evaluator's point of view and the dependence from the context cannot be eliminated, they should at least be traceable during and after the evaluation process. For this reason, every software evaluation should be accompanied by a document explaining choices and relating facts to the actual ratings assigned to metrics and categories.

In the context of the OITOS project, not only the results of every evaluation are public, but the methodology itself is made open to feedback from OITOS participants. As a first result, steady interest has been shown for the introduction of a new category in the model, with the purpose of evaluating legal issues connected with software, such as private data management, copyright, and licence responsibilities.

Notes

1. Details and documents for the OITOS project are available at `http://www.oitos.it`, including a case study on Groupware applications where our software evaluation methodology has been tested.

2. We adopted the BRR lexicon to achieve a uniform vocabulary and avoid confusion.

3. A pattern is a general repeatable solution to a commonly occurring problem.

4. Composability aims to estimate how many projects use the product as a component.

5. We extended the meaning of Data Gathering to also include information not related to software, but still essential to an effective evaluation.

References

[1] BRR Project (2005) Business Readiness Rating for Open Source. BRR-2005-RFC1

[2] Buschmann F, Meunier R, Rohnert H, Sommerlad P, Stal M (1996) Pattern-Oriented Software Architecture – A System of Patterns. John Wiley & Sons

[3] Duijnhouwer FW, Widdows C (2003) Open Source Maturity Model. CapGemini Expert Letter

[4] Golden B (2004) Succeeding with Open Source. Addison-Wesley