SOFTWARE PROCESS IMPROVEMENT AND PRACTICE
Softw. Process Improve. Pract. 2006; **11**: 107–122
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spip.257

Evaluation Criteria for Free/Open Source Software Products Based on Project Analysis

Research Section

David Cruz[1,*,†], Thomas Wieland[2] and Alexander Ziegler[1]
[1] *Technische Universität München, Institut für Informatik, Boltzmannstr. 3, 85748 Garching, Germany*
[2] *University of Applied Sciences Coburg, Department of Electrical Engineering and Computer Science, 96450 Coburg, Germany*

Today many companies decide to select free/open source software (F/OSS) products for various reasons, for example, economical or quality reasons. For many areas of application, they can choose from a variety of packages provided by different communities. Introducing a software tool into a company – either for supporting a certain business process or for the development of its own products – may be more or less critical depending on the application, which may range from explorative to mission critical.

One of the most crucial problems in the usage of F/OSS is the actual decision to use a particular product. Although this is rather similar to the usual investment decisions, the consequences of an involvement in a software project are frequently underestimated. Whether a certain package is appropriate for the intended purposes in the given environment and is suitable to the overall business goals of the company is usually not determined systematically, but often arbitrarily. However, many important hints resulting from the development context of an F/OSS product as well as other technical and economical considerations could be helpful to make the product decision process more transparent and deterministic.

In this article, we give a systematic approach for supporting a decision to incorporate a particular F/OSS product into an enterprise. Systematic decision support is provided by a rational approach that facilitates evaluating and interpreting the relevant project circumstances of an F/OSS product to be chosen. By this means, F/OSS projects can be critically reviewed in the light of the particular product requirements of the choosing organisation. The article addresses software managers and engineers in industry and academia. Copyright © 2006 John Wiley & Sons, Ltd.

KEY WORDS: open source; OSS; F/OSS; product evaluation; project analysis; free software

* Correspondence to: David Cruz, Technische Universität München, Institut für Informatik, Boltzmannstr. 3, 85748 Garching, Germany
† E-mail: cruz@informatik.tu-muenchen.de

1. PROJECT CIRCUMSTANCES SUPPORTING A PRODUCT DECISION

When a company or any other organisation decides to select free/open source software (F/OSS) products, a selection process has to be undergone that is similar to that for commercial-off-the-shelf (COTS)-software, but differs in a number of aspects, Feller and Fitzgerald 2002. Not only the functionality, but also the various conditions of the project in which the F/OSS was or is created are relevant. In this article, we present a systematic approach for supporting a decision to utilise an F/OSS product by an enterprise. (Throughout this text, we understand 'F/OSS' as some piece of software that is released under a license approved by the Free Software Foundation or the Open Source Initiative.)

The project circumstances of the F/OSS project and its community have a clear influence on the product in its present and anticipated future form (Sharma *et al*. 2002, Seifert and Wieland 2003). From the analysis of the project, assumptions about its conditions, quality and reliability can be made (Crownston and Scozzi 2002, Asklund and Bendix 2002). For example, information from the project source code repository can be used and interpreted to make several hypotheses concerning the state of the project (Koch and Schneider 2002). Numerous essential characteristics of the software, like reliability, maintainability, or sustainability, cannot be identified by source code inspection alone, but have to include the environment in which it has been created – even if a fully detailed code review is possible within the given frame of resources. A thorough test of the software may reveal some other important facts, but is also just a technical snapshot. A company that is planning parts of its business to rely on an F/OSS package should thus conduct a systematic and detailed analysis before selecting a particular product. The central idea of our approach is to provide some support for determining the requirements on an F/OSS product and evaluating the extent to which these requirements are actually met.

The evaluation and selection of an F/OSS package has some similarities with the conventional process of selecting COTS-software. As this has been described repeatedly and thoroughly (see, e.g. (Kontio 1996, Morisio *et al*. 2002)), we can take most of these results as known and emphasise the differences in particular. For it is not only the availability of the source code that makes the F/OSS different from COTS-software, but it is also the entire creation environment that requires a special approach and leads to some completely new implications.

2. OUTLINE OF THE PRESENTED APPROACH

Starting from possible usage scenarios for F/OSS in a company environment, this contribution deduces that possible requirements can be posed to the F/OSS product and its project environment in Section 3. The requirements are grouped into several categories and are described in more detail in Section 4. Furthermore, Section 4 gives indications on when care should be taken that a specific requirement is met, if there are not enough usage scenarios to support such a decision. The article then presents several investigable criteria that may be determined by analysing the F/OSS product and its surrounding project in Section 5, also describing how the information for evaluating these criteria can be investigated. Finally, in Section 6 we show how to use the criteria to determine whether a specific requirement posed at the F/OSS can be met.

The coherence of the above topics is visualised in Figure 1.

A user of the presented approach can utilise the use cases to classify his situation and determine the requirements that should be posed at the F/OSS. The section about the criteria helps the user in finding information that supports the systematic analysis of the project circumstances. Now the user is ready to interpret the collected information,

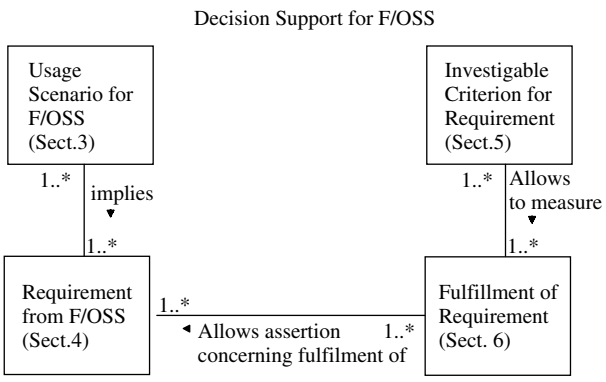

Figure 1. Overview of decision support approach

determine if the necessary requirements have been met and decide whether to incorporate the F/OSS into the company's usage.

3. USAGE SCENARIOS FOR F/OSS

In the following text plausible applications of F/OSS are presented. This is the defined entry point into the evaluation. It shows the possible usage scenarios for an F/OSS package. Various applications of F/OSS in different domains are conceivable. Classical application scenarios for F/OSS are web, mail or file server, firewalls, or embedded systems. Usual application models simply use an F/OSS product for daily business, as design choice for developing a new software product or as support providing business model, etc. (Grand *et al*. 2004, Hang *et al*. 2004). Furthermore, open source software products are already being used in several cases, for example, in bioinformatics and genetic research (Boyle *et al*. 2004), in drug discovery (DeLano 2005), or for medical image processing and visualisation (Yoo and Ackermann 2005). In addition, there are strong arguments for using F/OSS in security critical domains (transparency of code, amount of testers and speed of bug fixing) (Hissam *et al*. 2002, Payne 2002)

Several usage scenarios can motivate a manager to select an F/OSS product for a company. The usage scenarios listed below usually appear in various combinations. The relation of the usage scenarios with the requirements of Section 4 is presented in Table 1. Following the logical flow of the evaluation the requirements are detailed in Section 4.

All these usage scenarios for F/OSS will occur in various combinations and are multiply related to target-oriented requirements for an F/OSS product and like projects shown in Table 1. These requirements will be detailed in the following section.

4. REQUIREMENTS ON FREE/OPEN SOURCE SOFTWARE

This section discusses the question of requirements on F/OSS being reasonable. A close investigation of an F/OSS project can reveal many interesting details that clearly influence the decision in favour or against this software. But most of these must be viewed in the light of what the selecting company actually needs. The requirements give the chance to establish weights for the criteria laying particular emphasis on one aspect while more or less ignoring another one. In the same way the project-specific data is classified into several categories, that we will describe in complete detail in the next section, as the company-specific requirements range from technical and functional aspects to economical and even political characteristics. It is important that these requirements are elaborated in sufficient detail prior to the selection procedure. Only if the features of the entire system, in which the F/OSS is to be embedded are clear, an unbiased selection can be made. It can, however, be recommended to keep at least the functional requirements as simple and broad as possible. The required features should be listed by priorities. In this list, the number of unconditional features should be kept relatively short. Very strong functional requirements may limit the range unnecessarily.

In this section, we want to present a concise overview of various types of requirements that may affect the software. Some of them must be described item by item, like the functional requirements. Others are only qualitative aspects that should be arranged on a scale from 'negligible' to 'essential'. For each requirement category, we give a brief description and explain, whether a particular evaluation process should lay emphasis on it. So 'indication' means the circumstances under which this requirement should be added to the evaluator's catalogue and traced further on.

4.1. Functional Requirements

The most important requirement is certainly the functionality, the features the software has to cover. A schematic list for this aspect cannot be given, for it depends ultimately on the purpose for which the particular program or library is intended.

4.1.1. Required Functionality Covered

Description: The software comprises all the features it is supposed to. The coverage is considered sufficient if at least all the requirements indicated as 'essential' are fulfilled and the respective features are actually working.

Indication: This requirement is crucial for all usage scenarios, for it is the functionality why a particular piece of software is actually selected and used.

Table 1. Usage scenarios and related requirements on F/OSS

Usage scenario	Related requirements of Section 4
1. *Use F/OSS as a platform for a mission critical process.* (The F/OSS is existentially important for a company's core process; every problem causes considerable costs.)	*Functional*: Functionality covered, clear evolution direction *Technical*: Target platform supported, reliability, maintainability *Organisational*: Maintenance active, sufficient support, long life existence *Economical*: Sustainability, flexible maintainability according to individual needs *Political*: Possibility for influencing further development according to individual needs, decrease of proprietary dependencies, transparency over security
2. *Use F/OSS with a long-term consideration*. (The F/OSS is a part of the technological strategy of an enterprise.)	*Functional*: Clear direction of product evolution *Technical*: Target platforms supported, maintainability *Organisational*: Community exists, long life existence *Economical*: Sustainability, protection of investment *Political*: Possibility for influencing further development with respect to individual needs
3. *Use F/OSS as a cost reduction model*. (The F/OSS replaces a proprietary product and the license fee is saved.)	*Functional*: Required functionality covered, clear direction of product evolution *Economical*: Sustainability, protection of investment, cost reduction, division of development costs *Political*: Possibility for influencing further development with respect to individual needs
4. *Use F/OSS as exploration object, for example, for technology.* (The F/OSS is used as a study object to learn about new technologies or paradigms.)	*Organisational*: Community exists *Economical*: Increasing know-how
5. *Use F/OSS as an exhibition prototype*. (The F/OSS supports an exhibition of a product owned by the company or is exhibited as its own further development.)	*Functional*: Required functionality covered *Organisational*: Community exists *Economical*: Flexible maintainability according to individual needs, quick availability *Political*: Publicity, marketing effects
6. *Use F/OSS as a base line for further development and business model*. (The F/OSS is the base line for further proprietary development to be sold or to be subject of support contracts.)	*Technical*: Target platforms supported, maintainability *Legal*: No copyleft, liability, patent infringements, reselling obligations *Economical*: Flexible maintenance according to individual needs, increasing know-how *Political*: Transparency over security
7. *Use F/OSS to bridge a temporary bottleneck*. (The F/OSS fills a temporary gap in the migration from one software package to another.)	*Functional*: Required functionality covered *Technical*: Reliability *Organisational*: Sufficient support available *Economical*: Flexible maintainability according to individual needs, quick availability *Political*: Decrease of proprietary dependencies
8. *Use F/OSS for becoming independent of proprietary solutions and providers*. (The F/OSS helps to reduce dependence on software product companies with big market power and low customer orientation.)	*Functional*: Required functionality covered, clear direction of product evolution recognisable *Technical*: Reliability *Organisational*: Community exists, maintenance active, sufficient support available *Economical*: Sustainability, flexible maintenance according to individual needs *Political*: Possibility for influencing further development with respect to individual needs, decrease of proprietary dependencies

Table 1. (*Continued*)

Usage scenario	Related requirements of Section 4
9. Use F/OSS to gain transparency concerning safety and security. (The F/OSS is used to become independent from software companies not trustworthy in security concerns.)	*Functional*: Required functionality covered *Economical*: Sustainability *Political*: Decrease of proprietary dependencies, transparency over security
10. Use F/OSS for research purposes. (The F/OSS is used to support research activities.)	*Functional*: Required functionality covered, clear direction of product evolution recognisable *Technical*: Reliability *Organisational*: Sufficient support available *Economical*: Flexible maintenance according to individual needs *Political*: Possibility for influencing further development with respect to individual needs
11. Use F/OSS as a CASE tool. (The F/OSS is used for developing software.)	*Functional*: Required functionality covered, clear direction of product evolution recognisable *Technical*: Target platforms supported, reliability *Organisational*: Community exists, maintenance active, sufficient support available *Economical*: Sustainability, protection of investment, increase productivity, flexible maintainability according to individual needs, cost reduction, division of development costs *Political*: Possibility for influencing further development with respect to individual needs, decrease of proprietary dependencies

4.1.2. Clear Direction of Product Evolution Recognisable

Description: The community has developed clear thoughts and plans about which features will be changed or added in the future. Such a plan is sometimes also called a 'roadmap'. So it is obvious which additional functionality may be expected in which version. A time frame is desirable but not mandatory.

Indication: When deciding on a particular product, one has to keep in mind that this may not be a decision just for the moment. Generally the selected software is integrated with existing software components and systems by which a strong mutual dependence is generated. So the functional requirements should not only comprise the current ones, but also the ones of future versions that can be planned or anticipated today. This requirement makes sense only if there is already a clear view of the future usage of the software. In this case, it can increase the likelihood that the selection of today is still right tomorrow.

4.2. Technical Requirements

Besides the mere functional requirements, there are often numerous non-functional ones that are yet no less technical. When designing a piece of software, a particular technical environment in which it is supposed to run later is always anticipated. This environment (or maybe there are several of them) poses certain technical constraints on components, libraries, etc. that the software depends on.

4.2.1. Target Platforms Supported

Description: The software supports all the platforms in which it is intended to be used. 'Platform' in this context can mean the operating system and required libraries, and also virtual environments like the Java platform.

Indication: If a package does not run in the target platform, it is simply worthless for this purpose. The notion of the platform should, however, be defined carefully to cover all mutual dependencies. Platforms that are not in use today, but anticipated for the future should also be taken into account.

4.2.2. Reliability

Description: The software is mature enough so that all required features are working satisfactorily and are robust with respect to expected and unexpected failures.

Indication: Reliability is an important issue. The exact degree of reliability depends on the usage context. Before it can be decided whether the F/OSS in question is reliable enough, one has to determine how reliable the entire software system is supposed to be.

4.2.3. Maintainability

Description: This aspect is the ease with which a software system can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment (Institute of Electrical and Electronics Engineers 1990).

Indication: The importance of this requirement depends on the purpose for which the software is used. Experimental or short-term projects may accept that they only take one snapshot of the package and make no changes after the end of the project. Productive usage certainly requires a much larger degree of maintainability.

4.3. Organisational Requirements

Requirements that a company may expect from the community that develops or maintains the software of interest are grouped within this document as organisational requirements. The simplest example for such a requirement is that such a community exists at all, for there are many F/OSS systems in which the development has stopped and which are not maintained any more. This section identifies such requirements, describes their meaning within this document and points at simple indications that can show a company the relevance of such a requirement.

4.3.1. Community Exists

Description: As stated above, the existence of a community is the simplest organisational requirement. Within this document it means that there are persons who can be contacted to exchange information about the software. These persons must not necessarily be developers. For the community to exist, it is also sufficient if there are users with whom information about the software may be exchanged.

Indication: The only usage scenarios when this requirement may be unnecessary for the company are when the company developers have a full understanding of the software or when the software is only to be used for non-critical purposes. Another option is that the software product is so unique that the company intends to continue the project under its own auspices.

4.3.2. Product Evolution

Description: The software is being adapted to the changing environment in which it is used as well as to changing requirements.

Indication: If the company is not able to maintain the software completely by itself and is willing to use the software over a longer period of time, then this requirement should be met.

4.3.3. Sufficient Support Available

Description: When technical difficulties of any kind arise in working with (using or developing) the software there is a possibility of contacting someone who can help within an acceptable period of time.

Indication: Indications for this requirement could be, if the usage of the software is non-trivial or requires special knowledge. This requirement is also important, if the company wants to further develop the software and the software is complex either in size or in difficulty.

4.3.4. Long Life Existence

Description: The software and the developing/maintaining community is bound to still exist after a long period of time.

Indication: When the company does not have the means of maintaining and further developing the software itself and the software will be used in a company critical area or for a long period of time, then this requirement should be met.

4.3.5. Compatible Development Process

Description: The development process of the F/OSS and the development process of the company can be integrated to work as a whole. Sometimes it is not possible to combine the development processes, for example, when there are fixed deadlines for releases within the company process and the release decision for the F/OSS is made by a member of the community (Lussier 2004).

Indication: In some scenarios of usage, the company needs to take part in the development of the F/OSS. If the interaction with the development

process of the F/OSS community is high, i.e. the development contribution of the company is non-trivial then this requirement must be taken into consideration.

4.4. Legal Requirements

The possibility of having legal requirements may seem strange at first, but given the very different legal implications given by the different F/OSS licences it is essential that a company is certain about its own expectations (O'Mahoney 2003). This section will address some of the legal requirements to keep in mind when deciding on whether to use F/OSS.

4.4.1. No Copyleft Effect for Add-ons or Combinations
Description: The licence does not oblige the company to licence own add-ons under the same licence as the F/OSS. There are no obligations to licence combinations of the F/OSS and proprietary software under the same licence as the F/OSS.

Indication: If the company wants to develop add-ons for the F/OSS, which should not have to be free/open source then it is important for the F/OSS licence to allow such add-ons. The company may have an interest in keeping the licence of add-ons 'closed source' if it wants to sell the add-ons or if company confidential information is inside the source code. The same reasoning applies to combinations of proprietary software with the F/OSS.

4.4.2. No Liability for Third Party Code
Description: In many countries, the law will require that the company selling a product will be liable for its correct functioning. Given that the company selling the product did not write parts of the code, it may not want to be liable for the foreign code parts.

Indication: Depending on the country where the product is sold, this is a very strict requirement which may not be fulfilled. Therefore, this requirement should be investigated wisely. Indications are areas where the integrity of humans or animals is endangered (e.g. transportation) or areas where large amounts of money are involved in the software (e.g. financial software).

4.4.3. No Patent Infringements
Description: This requirement demands that the use of the F/OSS in question does not collide with any patent restrictions.

Indication: When the F/OSS is to be used for mission critical purposes, this requirement is absolutely necessary. For other scenarios, the need for this requirement has to be determined in each individual case.

4.5. Economical Requirements

Economical requirements are relevant, because the usage of an F/OSS is embedded in the context of an organisation under financial constraints that cannot be considered less important than technical issues because they are existential for the company using F/OSS. Therefore, criteria like 'Sustainability', 'Protection of investment', 'Increase of productivity', 'Flexible maintenance according to individual needs', 'Quick availability', 'Increasing know-how', 'Cost reduction' and 'Division of development costs' have to be discussed in this contribution. Different from the above stated requirements this is carried out from the particular point of view of the target environment.

4.5.1. Sustainability of the Usage of the F/OSS
Description: Sustainability means long-term availability of product, support and maintenance. For example, a firewall for an IT company should be sustainable.

Indication: It is important that the maintenance of the F/OSS product is granted for a long time if the purpose is to use it for several years. This can be achieved either by the using company itself, by a third party support company or by the open source community under consideration.

4.5.2. Protection of Investment for Migrating to the F/OSS Product
Description: The financial cost and time effort for migrating to the F/OSS product has to be prepaid. The purpose is to regain the invested money through the usage of the F/OSS product. For example, the effort for evaluating, testing, deploying and training an F/OSS should amortise through its usage.

Indication: The costs for the migration to the F/OSS product may be significant and must be amortised to get that business case profitable and acceptable.

4.5.3. Increase Productivity by Usage of the F/OSS
Description: Increasing productivity means improving the relation between business values and working time. For example, using an e-mail client should

support its users in their easy communication with each other and save time for further useful activities in the company.

Indication: Increasing the productivity of the processes supported by the F/OSS is necessary to ease given time or resource bottlenecks in the organisation.

4.5.4. Flexible Maintainability of the F/OSS According to Individual Needs
Description: The F/OSS is easily customisable to fulfil functional and non-functional requirements of the company.

Indication: Flexible maintainability of an F/OSS according to individual needs is often necessary when it is used in a very particular organisational context.

4.5.5. Quick Availability of the F/OSS
Description: The F/OSS is ready for use easily and in a short time. For example, an e-mail client should be easily downloadable for evaluation purposes.

Indication: If the possibility of use stands under time pressure, quick availability is most important.

4.5.6. Increasing Know-How for the Internal Staff by Studying the F/OSS
Description: The F/OSS provides inherent technological progress to be studied by the internal staff. For example, a file server with sophisticated technology or implementing a new paradigm can be useful for the internal staff to study and gain knowledge about these topics.

Indication: The provision of additional know-how may be important for a company exploring new possibilities to enrich products with state-of-the-art technology or to grow into a new technological segment. Increasing know-how of the staff means improving an important competition factor.

4.5.7. Cost Reduction Through Usage of the F/OSS Product
Description: The effort for running the F/OSS product in daily business is lower than the effort for the former solution. For example, running an F/OSS anti-virus software product saves the license fee with respect to a proprietary one.

Indication: Whenever the F/OSS is used to reduce costs, this requirement must be met.

4.5.8. Division of Development Costs Through Usage of the F/OSS Product
Description: Sometimes a company cannot afford the money and time for a needed software solution on its own and searches for others with similar interests and purposes. For example, smaller companies depend on F/OSS to share development costs with others when the market is too small for a proprietary product and individual development is too expensive.

Indication: A solution is needed, where a trade off between lower costs and suboptimal customising with respect to individual needs due to other project members are accepted.

4.6. Political Requirements

Like economic topics, political issues have to be considered very carefully because neither the F/OSS and its community nor the choosing company can be considered independently from the rest of the IT world, the rapid movements and changes of which always have side effects for related companies. For this reason factors like 'Possibility for influencing further development with respect to individual needs', 'Decrease of proprietary dependencies', 'Transparency over safety and security' and 'Publicity and marketing effects' have to be looked at.

4.6.1. Possibility for Influencing Further Development of the F/OSS with Respect to Individual Needs
Description: An F/OSS package often has to be customised according to company-specific requirements. Therefore, a sufficient willingness in the corresponding open source community is necessary. For example, if a company needs individual features for its F/OSS file server and has no own development resources, there should be a satisfactory possibility to place those interests in the community.

Indication: It must be possible to influence the further development of an F/OSS if the using company has no possibility for developing on its own.

4.6.2. Decrease of Proprietary Dependencies Through Usage of the F/OSS
Description: Often organisations using proprietary software products depend strongly on decisions made by the product provider and their needs are not respected sufficiently. For example, if a large

product vendor refuses to implement features of an e-mail client which are necessary for a smaller part of its customer community, choosing an adequate F/OSS could be a solution.

Indication: Independence from large software vendors and their market power is needed when their products, terms and conditions do not meet the wishes of a customer company any more.

4.6.3. Transparency Over Security Through Usage of F/OSS

Description: For information security reasons, transparency about the source code is needed for a software product that deals with a company's information assets. For example, if it is impossible to evaluate a proprietary operating system concerning security leaks, choosing an F/OSS platform could be an option because of source code availability and transparency.

Indication: In usage scenarios, in which an insight into the source code is needed to be sure about information safety and security, this requirement should be fulfilled.

4.6.4. Publicity, Marketing Effects Through Usage of and Participating in F/OSS

Description: It is publicly known that a company or organisation participates in an open source project. For example, if a large software vendor that does not provide an operating system participates publicly in a reputable F/OSS operating system there is a publicity effect.

Indication: Sometimes it may be important for the reputation of a company to be known as a participant in and user of a well-known open source project and product.

After the above stated possible requirements for F/OSS, it is necessary to present criteria that give hints about their fulfilment. These criteria follow in the section below. The connection between the criteria and the requirements is described in Section 6.

5. INVESTIGABLE DATA ABOUT A FREE/OPEN SOURCE SOFTWARE PRODUCT

There are a large number of aspects of a particular F/OSS project that can affect the decision in favour of or against using its software. Some of them are more abstract and hard to determine, while others are easy to be evaluated quantitatively. In this section, we discuss all the issues about a software project that are relevant to our approach and thus regarded as investigable. With the help of these data, the fulfilment of the respective requirements can be checked systematically. For each criterion, we not only give a brief description, but also point out how the fulfilment of this criterion could be determined or measured, respectively.

5.1. Functional Criteria

The functional criteria seem to be determined most simply. Almost every project has a description page that lists the features of the software. Sometimes, however, the terminology must first be learned to comprehend the feature and descriptions. At other times the feature list is far from complete that the actual features must be explored by running demos, test suites, or by analysing reference installations. The functionality is always a crucial aspect when selecting a certain software product. In case of F/OSS, however, there is no general way to determine it exhaustively. It always depends on the actual project and the type of software and how the feature list is set up.

5.2. Technical Criteria

There are numerous technical aspects that can serve as selection criteria. The most relevant ones are discussed in the following:

5.2.1. Actual Number of Bugs

Description: The number of bugs (failures, errors and inconsistencies) that are inherent in the F/OSS under consideration.

Measurement: This number can only be determined if the project maintains a public bug list. Common project portals (e.g. SourceForge, www.sf.net) offer and support this feature, but not all projects make use of it. Larger projects that rely on a bug tracking system (e.g. BugZilla, www.bugzilla.org) usually publish this data in full detail.

But even if there is a public bug list, it is not guaranteed that this really reflects the actual number of bugs. Often bugs that are noticed by a developer are not reported formally, but fixed immediately. Afterwards only a message in the

mailing list or a comment in the check-in protocol reminds one of this program error.

Related issues are the number of fixed bugs that can be determined similarly and the number of reported bugs. The latter means program errors that are noticed and reported by others than the developers themselves.

5.2.2. Number of Open Feature Requests
Description: The number of requests for new or modified features that have not been implemented.

Measurement: Like the number of bugs, the number of open feature requests can best be determined from public bug tracking systems, either via a portal site or a dedicated installation. Usually for each request its status is denoted here, e.g. open, assigned, realised, or discarded.

5.2.3. Code Metrics
Description: Statistical data about the source code, e.g. number of files, number of classes, average length of functions, fraction of comment lines *versus* code lines.

Measurement: This data can be evaluated automatically by shell scripts and other specialised tools.

5.2.4. Frequency of Changes
Description: Information about how often each project file has been changed and how frequently the code is checked in to the repository.

Measurement: Usually, the code repository is publicly accessible. So it can be evaluated by appropriate scripts from the repository log data (Chen *et al*. 2004).

5.2.5. Dependencies on Other Software
Description: The number of other software packages, commonly also F/OSS, on which the particular package depends.

Measurement: This criterion is normally well documented in the 'readme' files of the project. Moreover, it appears rather soon when compiling and running the software.

5.3. Organisational Criteria

The investigable information about the organisation, i.e. the organisation of the community developing the F/OSS in question is the following:

5.3.1. Number of Developers
Description: The number of developers that are involved in the development of the F/OSS in question.

Measurement: In most open source portals, the project page will show the number of developers involved in the project.

5.3.2. Number of Testers
Description: This document defines a tester as any person giving feedback, i.e. reporting bugs to the project developers.

Measurement: Usually, every F/OSS project has a means of letting users submit bug reports to the project developers. Bugs may be reported using a special bug reporting tool as BugZilla (www.bugzilla.org) or simply through a forum or mailing list. The number of testers is determined by exploring the means used to report bugs and counting the number of different identities reporting bugs, not counting the project developers.

5.3.3. Number of Users
Description: The number of persons or companies using the software.

Measurement: Unfortunately, this information can only be estimated. An estimate of this information may be obtained by observing the number of downloads of the software. Some Open Source Portals provide statistical information for the project where the number of downloads are included. If the Portal or Site does not provide this information, it may be possible to ask the project leader. This would also give an insight into the reaction to user questions. Another way of obtaining an estimate of the number of users is to take a look at the user mailing lists or user discussion forums, if available.

5.3.4. Development Process Patterns
Description: All forms of development obey a set of rules. The use of certain rules suggests that the development process follows a special development pattern.

Measurement: The rules of the development in an F/OSS project are usually described on the project site. If a developer wants to contribute to an F/OSS project, it is common to send a patch to a committee or the mailing list. The patch

will then be reviewed and only checked into the repository if its quality satisfies the reviewer(s). Further important rules may also be a part of the development process. If the rules are not published on the project site, the simplest solution is to ask the project manager.

5.3.5. Skills of the Developer Community
Description: The know-how and experience of the developers.

Measurement: By following the discussions of the developers in mailing lists, an expert in the area can determine, whether the developers have the expertise in the area of the use of the developed software (e.g. by determining, whether the correct terms are being used).

5.4. Legal Criteria

If the F/OSS product in question has a well-known licence, some of the legal properties of an F/OSS product can be determined rapidly by consulting websites and publications. For uncommon licences, it is usually wise to seek legal advice. Some legal criteria, which can eventually be determined rapidly are listed below:

5.4.1. Copyleft Effect of Licence
Description: Whether the licence of the F/OSS implies that combinations of the F/OSS and other software must be licensed under the licence of the F/OSS licence.

Measurement: The licence of the F/OSS is usually presented on the project site and it is usually packaged with the software when downloaded. As already mentioned, some properties (such as the copyleft effect of a licence) of well-known licences have been thoroughly discussed and are thus publicly available. As an example, the copyleft effect of the GNU GPL is publicly known.

5.4.2. Liability When Reselling
Description: Whether the company selling a product containing the F/OSS or parts of it is liable for errors in the F/OSS and to what degree.

Measurement: Determining the liability the company has to adopt for the F/OSS is done in three steps. Most F/OSS licences disclaim any liability for the software. The first step is to determine if this is the case for the F/OSS in question. Next it should be found if the country specific laws allow all liability to be disclaimed. As a third step it is useful to find out if the company itself is allowed to disclaim the liability for the third party software by the country specific laws.

5.5. Economical Criteria

5.5.1. Is There Sufficient Availability of Development Resources?
Description: What amount of infrastructure and resources through participating companies can be estimated? How much do large companies support the open source product with human resources, technical infrastructure and know-how? The better the support through resources, the better the vitality of the F/OSS.

Measurement: Often, but not mandatory, the project home page contains hints about which organisations or companies participate in the project.

5.5.2. What are the Estimated Migration Efforts?
Description: When migrating to an open source product sometimes a business should be calculated. The effort of time and money has to be estimated and compared with the expected benefit of the usage of the open source product. The fewer the efforts for the migration, the lower the financial risks.

Measurement: A cost benefit analysis considering and estimating the man power, know-how, time, money and effort needed to migrate to, train the staff for and roll out the F/OSS and the expected benefit in the form of, for example, cost reduction or increasing productivity should be carried out.

5.5.3. What are the Estimated Monthly Costs?
Description: How much are the running costs and what is the effort for using the open source product, for example, in the form of maintenance or in-house support? The less the running costs, the better the long-term economic efficiency.

Measurement: An analysis considering the running effort and the costs for in-house maintenance and support of the F/OSS has to be made.

5.6. Political Criteria

5.6.1. Which Well Reputed Companies are Involved?
Description: Which well-known companies support the open source project with man power, technical resources, know-how and political influence? The broader the political support for the F/OSS, the less risky it is to rely on it with a long-term consideration.

Measurement: On the project home page and in the relevant IT press one can learn about the overall support for and interests in the F/OSS by several well-known parties of the community.

5.6.2. Which Dependencies on Other F/OSS Exist?
Description: Are there any dependencies on other open source products for running the F/OSS under consideration and what effort can be estimated for respecting them? The more the dependencies, the less calculable is the risk to rely on it.

Measurement: On the project home page there should be links leading to existing dependent products.

5.6.3. In How Many Languages Does Documentation Exist?
Description: Is there any user documentation for the open source product and in what and how many languages is it available? The better and multi-lingual the documentation, the broader is the usage and the larger the community. This again is a liveliness criterion.

Measurement: On the project home page there should be links leading to existing documentation.

5.6.4. How is the Climate in Discussion Forums?
Description: Is there a constructive progress-oriented discussion climate in the forums? Does helpful advice dominate or 'rtfm'-like destructive answers? The more constructive the discussion climate, the more progressive the overall attitude, the more lively the F/OSS can be deduced to be.

Measurement: Look at the project discussion forums and consider the discussion threads.

5.6.5. Are There any Large Contributing Companies About to be Merged, Bought in or to be Insolvent?
Description: Are there any significant external changes in political, environmental or any other important circumstances to be expected for the open source project and product? The larger number of contributors who are about to retire from the project, the less is the liveliness to be estimated and the more risky it will be to rely on it for a long-term consideration.

Measurement: On the project home page and in the relevant IT press one can learn about the overall support for and interests in the F/OSS by several well-known parties of the community as well as movements and purposes at the finance market.

5.6.6. Are There Any Large Contributing Companies About to Have a Strategic or Executive Board Change?
Description: Are there any significant internal changes in political, environmental or any other important circumstances to be expected for the open source project and product? Analogous to the description for the criterion above, the larger contributors who are about to the number of retire from the project, the less is the liveliness to be estimated and the more risky it will be to rely on it for a long-term consideration.

Measurement: Again, the measurement of the criterion can be accomplished by browsing the web. On the project home page and in the relevant IT press one can learn about the overall support for and interests in the F/OSS by several well-known parties of the community and their enterprise strategic movements and purposes.

6. FULFILMENT OF REQUIREMENTS

After having discussed possible requirements for an F/OSS in Section 4, several investigable criteria for evaluating an F/OSS have been presented in Section 5. The main question is now how the fulfilment of the requirements can be estimated with the help of this information. Starting from the possible requirements it is shown how an interpretation of the corresponding F/OSS criteria can be carried out to support a product decision. The following creates the connection between the requirements described in Section 3 and the criteria discussed in Section 5. For each requirement, we check which criteria are suitable to confirm its fulfilment and give hints how these criteria could be applied or derived.

6.1. Fulfilment of Functional Requirements

The description page of a project usually contains a list of the features the software supports so they can be compared with the required functionality. But

this relationship between features and functionality is neither necessary nor is it sufficient. On the one hand, there may be a couple of features that are listed, but which do not work properly (yet). On the other hand, some required qualities may be missing on the list, although they are actually implemented and working. Both kinds of errors are hard to overcome. The first one is usually more serious for the selection of a program that does not work as it is supposed to cause severe problems.

The detection and clearance of these errors depends on the amount of time and resources the company is willing and able to invest in the selection procedure of each package. A step-wise approach can be recommended:

1. Compare the list of required features and list of reportedly available features. If essential features are missing and further test resources are not assigned, the product has to be discarded.
2. Check if the reportedly available features are really implemented and are working properly. This can be done by defining and realising appropriate test cases and by looking into the source code.
3. If features are missing, but test resources are on hand, check if these features are really absent. Again, this can be done by respective test cases or code inspection.
4. If required features are confirmed to be missing, estimate the effort and expenses for adding one's own implementation.

6.2. Fulfilment of Technical Requirements

6.2.1. Actual Number of Bugs

It is difficult to interpret the number of bugs objectively. A larger number can mean a more buggy or an unstable product. But it can also mean that there are a lot of users that find many bugs by running the software on various platforms. Similarly, a small number can mean that there are simply no users that could find any bugs. So before drawing any conclusions about a certain project with respect to the number of its bugs, the quality of these bugs should be analysed. If the project maintains no formal bug tracking, the mailing list commonly gives good hints on this issue.

6.2.2. Number of Open Feature Requests

In turn, the number of open feature requests gives a good indication how vivid the user community is and how satisfied the users are with the software. A larger number usually means a better user echo.

6.2.3. Code Metrics

Code metric data can give an impression about the complexity of a certain project. If the relationship of code lines *versus* comment lines is, e.g. rather bad, there may be a problem with maintainability or reliability. If the functions have more than 100 lines on an average, the code quality is normally not the best as stability and reliability may be affected.

6.2.4. Dependencies on Other Software

If the software product depends on several other projects, which in turn depend on further packages themselves, the entire configuration complexity is usually quite high. It is not recommended to use such a package for mission critical purposes.

6.3. Fulfilment of Organisational Requirements

Drawing conclusions about the organisation of the F/OSS project always has to be done by instinct. But the above criteria can help in estimating whether and to what degree some of the requirements stated above can be met.

6.3.1. Community Exists

To determine the existence of the community, one of the following criteria has to be fulfilled: The number of developers or the number of users or the number of testers has to be greater than zero.

6.3.2. Product Evolution

For the F/OSS product to evolve, its source code has to change. Therefore, the existence of a community or even discussions in developer mailing lists is not enough to show that the product can really evolve. To find out if the code really changes, the only possibility is to monitor the frequency of changes to the code repository. If a more detailed analysis is within budget and time constraints, the monitoring can be broadened to verify new development, reengineering, bug fixing, testing and other special cases.

6.3.3. Sufficient Support Available

For sufficient support to exist, as a first requirement a community has to exist. Other criteria aiding the fulfilment of this requirement are the existence of

documentation and the climate in discussion forums or mailing lists.

6.3.4. Long Life Existence
If an F/OSS has a large community, it implies that there is a great interest in using such software. Thus, the users of the software are bound to accept investments to be able to continue using it. Another criterion that signals a long life existence is the involvement of well reputed companies, because these companies are also bound to protect their investments made in the software.

6.3.5. Compatible Development Process
The development process pattern of the F/OSS indicates whether it can be combined with the company development process. It is important to also analyse the development processes of other software, that the F/OSS in question depends on. Usually this software is also open source.

6.4. Fulfilment of Legal Requirements

Although the legal requirements are usually dependent on the use case/business case followed by the company, once defined, the fulfilment of these requirements is mostly solely dependent on the licence and the applicable laws.

6.4.1. No Copyleft Effect for Add-ons or Combinations
Determining if the licence has a copyleft effect is fairly easy. Yet, it should also be analysed when the copyleft effect applies to add-ons or combinations. In many licences, there is a way of combining the F/OSS with other software and still avoiding a 'licence contamination'.

6.4.2. No Liability for Third Party Code
Examining whether the company is liable for third party code when selling it is determined solely by the criterion 'liability when reselling'.

6.4.3. No Patent Infringements
No criteria can measure whether the F/OSS in question violates some software patent, but if the company does not have the budget to research the possible patent infringements within the F/OSS, a good indication is to find other users. If large companies use the software it is likely that these companies researched the software for patent infringements before using it, unless, of course, the company using it holds the patent itself.

6.5. Fulfilment of Economical Requirements

The following section discusses how the fulfilment of economical requirements can be estimated by interpreting the corresponding criteria.

6.5.1. Sustainability
If there is a large, active and international community with reputable companies participating in the development and supporting the F/OSS it is most probable that sustainability is granted because this community does not seem to be about to die. Base factors are satisfactory feature coverage and an adequate direction of the product evolution.

6.5.2. Protection of Investments
Like the sustainability requirement and complemented by effort and cost-considering criteria it can be stated that if stable development circumstances are given and the migration and running of the F/OSS can be expected to remain economically justifiable, a sufficient protection of investments should be given. Because investment is not too high and the time span for amortising the investment by using the product can be expected to be long enough, protection of investments is given.

6.5.3. Increase Productivity
To increase productivity, it is important that a sufficient coverage of needed features is given and that the product can be recognised to evolve with further features useful for the company so the F/OSS remains productivity improving.

6.5.4. Flexible Maintainability According to Individual Needs
Sufficient flexible maintainability is given if starting from a satisfactory coverage of necessary features, the available skills and resources for further development of the F/OSS seem to be enough and the open feature requests tend to be closed with adequate pace.

6.5.5. Quick Availability
On the basis of an adequate feature coverage it is important to know the product dependencies and

their possible disturbing side effects to estimate if the product can be available fast enough.

6.5.6. Increasing Know-how
If the developer community seems to be skilled enough and the climate in the discussion forums is progressive and cooperative, it is highly possible that the in-house staff will be able to learn from the community and to increase its own know-how.

6.5.7. Cost Reduction
When sufficient feature coverage is given and the development tends to move in a helpful direction the basis for cost reduction is given. Further necessary are justifiable migration and running costs and a good documentation for the users to reduce support effort. Helpful for cost reduction are a minimum of product dependencies and a cooperative climate in the discussion forums to decrease maintenance effort.

6.5.8. Division of Development Costs
Division of development costs can be reached by feature coverage and the right evolution direction. Further a living and sufficient resourceful development community with stable participants is necessary. Existent documentation does not have to be written in-house and contributes to lower development effort.

6.6. Fulfilment of Political Requirements

In the following section, it is discussed how the fulfilment of political requirements can be estimated by interpreting criteria accordingly.

6.6.1. Possibility for Influencing Further Development with Respect to Individual Needs
The possibility for influencing further development of an F/OSS with respect to individual needs can be estimated according to several criteria. If the F/OSS evolves in a satisfactory direction and sufficient development resources are given, the F/OSS can be developed with respect to users' number needs. A minimum of product dependencies are helpful because this can be a crucial constraint for further development. The climate in the discussion forums shows how open the community is for change requests from product users. By the number of open feature requests in relation to time it can be seen how efficient the development progresses.

6.6.2. Decrease of Proprietary Dependencies
A decrease of proprietary dependencies can be achieved, if the feature coverage of the F/OSS is sufficient and it evolves in a useful direction. If the community has enough development resources and skills available then it is helpful to remain independent with the help of the F/OSS. It is useful if well reputed companies are involved and this relationship seems to be stable in the long run.

6.6.3. Transparency Over Security
In addition to the insight of the source code, it is helpful to gain transparency over security if the community is skilled enough to produce a readable design of the code and can be gained if the climate in the developer forums are cooperative. A minimum of dependencies from other products may increase the transparency further.

6.6.4. Increase Publicity, Marketing Effects
Publicity and marketing effects by participating in an F/OSS community can be increased if the whole project is well-known and reputable. This is often measured by well-known companies and by the number of developers, testers and users with sufficient development resources participating in an F/OSS project.

7. SUMMARY AND CONCLUSIONS

There are various business cases that can make the use of F/OSS within a company attractive. The problem is that often the companies have no experience in deciding when it is wise to use a specific F/OSS product or not.

This article provides an approach based on decision criteria to support decision-makers in the judgement of integrating a specific package of F/OSS into the company portfolio, by analysing the environmental circumstances of the F/OSS project.

A decision to integrate a piece of software into the company is highly dependent on the intended use of the software. Therefore, the article starts by listing possible scenarios for the usage of F/OSS. Different requirements towards the F/OSS and its project environment are derived from and put into relation with the scenarios of usage. For better lucidity, the requirements are structured into the aspects of various functional, technical, organisational, legal,

economical and political. This structure is pursued throughout the document.

The article then shows different criteria that can be extracted out of the project environment, which provide useful information about the F/OSS. This information can later be used to determine if the posed requirements are met.

Finally, the article illustrates which criteria to use and how to use them to decide whether the requirements posed at the F/OSS are actually met.

It is very important to notice that the presented approach is not and cannot be understood as an automated decision system for the usage of F/OSS products into a company. It is a guideline that eases the choice through a systematic analysis of relevant factors. Every decision maker has to adapt this approach according to the specific circumstances and individual preferences and needs.

REFERENCES

Asklund U, Bendix L. 2002. A Study of configuration management in open source software projects. *IEE Proceedings–Software* **149**(1): 40–46.

Boyle EI, Wenig S, Gollub J, Jin H, Botstein D, Cherry JM, Sherlock G. 2004. Go::TermFinder–open source software for accessing Gene Ontology information and finding significantly enriched Gene Ontology terms associated with a list of genes. *Bioinformatics* **20**(18): 3710–3715.

Chen K, Schach SR, Yu L, Offutt J, Heller GZ. 2004. Open-source change logs. *Empirical Software Engineering* **9**(3): 197–210.

Crownston K, Scozzi B. 2002. Open source software projects as virtual organisations: competency rallying for software development. *IEE Proceedings–Software* **149**(1): 3–17.

DeLano WL. 2005. The case for open-source software in drug discovery. *Drug Discovery Today* **10**(3): 213–217.

Feller J, Fitzgerald B. 2002. *Understanding Open Source Software Development*. Addison Wesley: London, UK.

Grand S, von Krogh G, Leonard D, Swap W. 2004. Resource allocation beyond firm boundaries: A multi-level model for open source innovation. *Long Range Planning* **37**: 591–610.

Hang J, Hohensohn H, Mayr K, Wieland T. Benefits and pitfalls of open source in commercial contexts. In *Free/Open Source Software Development*. Koch S (ed.). IDEA Group: Hershey, PA, 2004. 222–241.

Hissam SA, Plakosh D, Weinstock C. 2002. Trust and vulnerability in open source software. *IEE Proceedings–Software* **149**(1): 47–51.

Institute of Electrical and Electronics Engineers. 1990. *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*. The Institute of Electrical and Electronics Engineers: New York.

Koch S, Schneider G. 2002. Effort, co-operation and co-ordination in an open source software project: GNOME. *Information Systems Journal* **12**: 27–42.

Kontio J. 1996. A case study in applying a systematic method for COTS selection. In *Proc. 18th Int'l Conf. on Software Engineering (ICSE'96)*, Rombach HD (ed.). ACM/IEEE-CS Press: New York, Berlin, Germany, 201–209.

Lussier S. 2004. New tricks: How open source changed the way my team works. *IEEE Software* **21**(1): 68–72.

Morisio M, Seaman CB, Basili VR, *et al*. 2002. COTS-based software development: Processes and open issues. *Journal of Systems and Software* **61**(3): 189–199.

O'Mahony S. 2003. Guarding the commons: how community managed software projects protect their work. *Research Policy* **32**: 1179–1198.

Payne C. 2002. On the security of open source software. *Information Systems Journal* **12**: 61–78.

Seifert T, Wieland T. 2003. Prerequisites for enterprises to get involved in open source software development. In *1st Workshop of Open Source Software in Industrial Environments. Proceedings of Net.ObjectDAYS*, Erfurt, Germany.

Sharma S, Sugumaran V, Rajagopolan B. 2002. A framework for creating hybrid-open source software communities. *Information Systems Journal* **12**: 7–25.

Yoo TS, Ackermann MJ. 2005. Open source software for medical image processing and visualisation. *Communications of the ACM* **48**(2): 55–59.