Method for Qualification and Selection of Open Source software (QSOS) version 1.6

April 2006

Abstract

This document describes the QSOS method, conceived to qualify, select and compare free and open source software in an objective, traceable and argued way.

Contents

1 Licence notice

2 The QSOS Manifesto

2.1 Why a method?

For a company, the choice to opt for software as a component of its information system, whether this software is Open Source or commercial, rests on the analysis of needs and constraints (technical, functional and strategic) and on the adequacy of the software to these needs and constraints.

However, when one plans to study the adequacy of open source software, it is necessary to have a method of qualification and selection adapted to characteristics of this type of software. It is, for instance, particularly important to precisely examine the constraints and risks specific to open source software. Since the open source field is very rich and has a very broad scope, it is also necessary to use a qualification method allowing to differentiate the quite often numerous candidates to meet both technical, functionnal and strategic requirements.

Beside "natural" questions like:

- Which software meets best my actual/planned **technical** requirements?
- Which software meets best my actual/planned **functional** requirements?

Here follow some questions every company should answer before making any decision:

- What is the durability of the software? What are the risks of Forks? How to anticipate and manage them?
- What level of stability to expect? How to manage dysfunctions?
- What is the expected and available support level provided on the software?
- Is it possible to influence further development of the software (addition of new or specific functionalities)?

To be able to answer serenely these types of questions and thus set up an efficient risk management process, it is imperative to have a method allowing:

- software qualification by integrating the open source characteristics;
- software comparisons according to formalized needs requirements of weighted criteria, in order to make a final choice.

These are the various points which made Atos Origin conceive and formalize the method for Qualification and Selection of Open Source software (QSOS).

2.2 Why a free method?

For us, such a method as well as the results it generates, must be made available to all under the terms of a free licence. Only such a licence is capable of ensuring the promotion of the open source movement, via in particular:

- the ability for all to re-use already available work for qualification and evaluation;
- the quality and objectivity of documents generated, always perfected according to principles of transparency and peer reviews.

For these reasons Atos Origin decided to make available the QSOS method, and the documents generated during its application (functional grids, identity cards and evaluation sheets), under the terms of the GNU Free Documentation License.

3 Changelog

Version	Date	Authors	Comments
1.0		Raphaël SEMETEYS (Atos Origin)	Initial conception and authoring.
1.1		Olivier PILOT (Atos Origin)	Conception and proofreading.
1.2		Laurent BAUDRILLARD (Atos Origin)	Conception and proofreading.
1.3	17/11/2004	Raphaël SEMETEYS (Atos Origin)	First official release.
1.4	23/11/2005	Raphaël SEMETEYS (Atos Origin)	Corrections of misprints, addition of the "License notice" and "Changelog" sections. Replacement of the evaluation example by a link to `http://www.qsos.org`.
		Olivier PILOT (Atos Origin)	New logo.
1.5	19/01/2006	Gonéri LE BOUDER (Atos Origin)	Conversion to LaTeX . Re-licensing under the Gnu FDL.
		Raphaël SEMETEYS (Atos Origin)	Addition of the "QSOS Manifesto" section. First english version.
		Wolfgang PINKHARDT (Atos Origin)	English proofreading.
1.6	13/04/2006	Gonéri LE BOUDER (Atos Origin)	Criterion "Packaging" added, criterion "Integration" and subcriterion "Exploitability/Installation, deployment" removed

4 Introduction

4.1 Document objective

This document describes the QSOS method (Qualification and Selection of software Open Source), conceived by Atos Origin to qualify and select the Free and Open Source software as a basis of its support and technological survey services.

The method can be integrated within a more general process of technological watch which is not presented here. It describes a process to set up identity cards and evaluation sheets for free and open source software.

4.2 Targeted readers

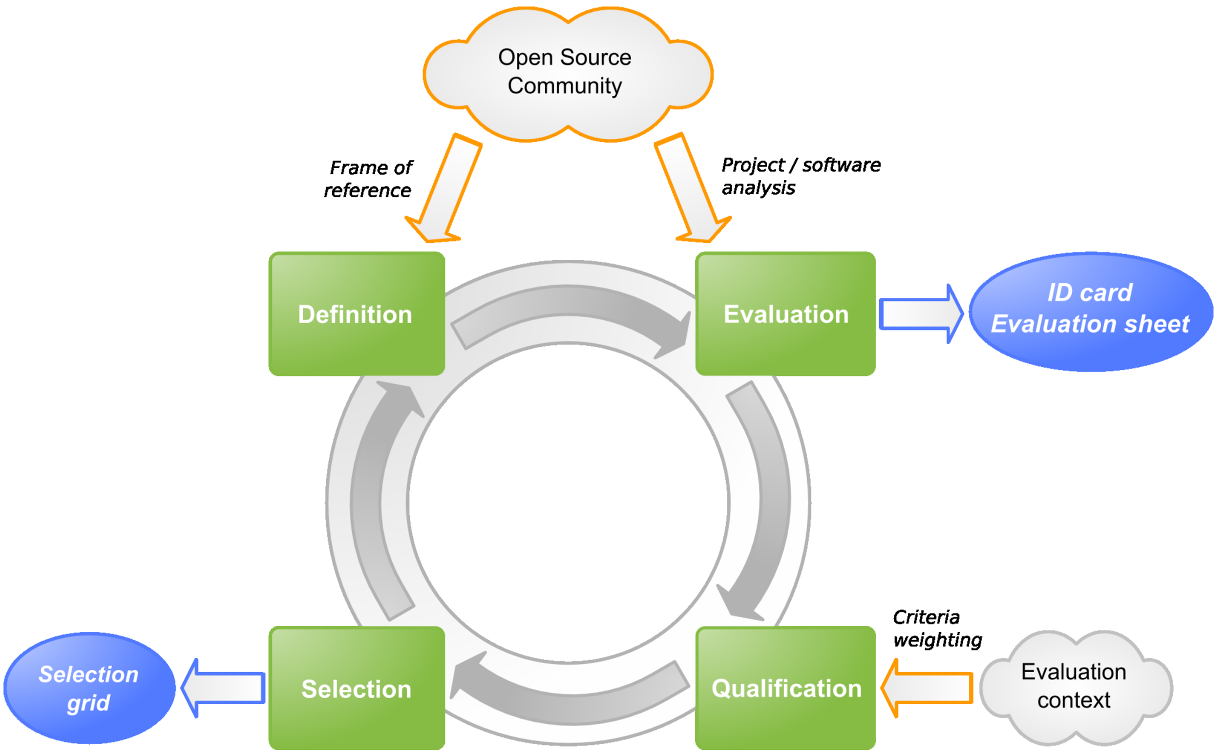
This documlent targets the following readers:

- people curious to know more about the method, on a professional or personal basis;
- communities of Free and Open Source projects;
- IT experts wishing to know and use the method in evaluating and selecting components to build systems either for their own needs or for their customers.

5 General process

5.1 Four steps

The general process of QSOS is made up of several interdependent steps (figure 1).



1: The 4 steps of QSOS

Each one of these steps is detailed further in this document.

5.2 Iterative process

The general process introduced here can be applied with different granularities. It enables the establishment of the desired level of detail for the process as well as advancement of the process by iterative loops to refine each of the four steps.

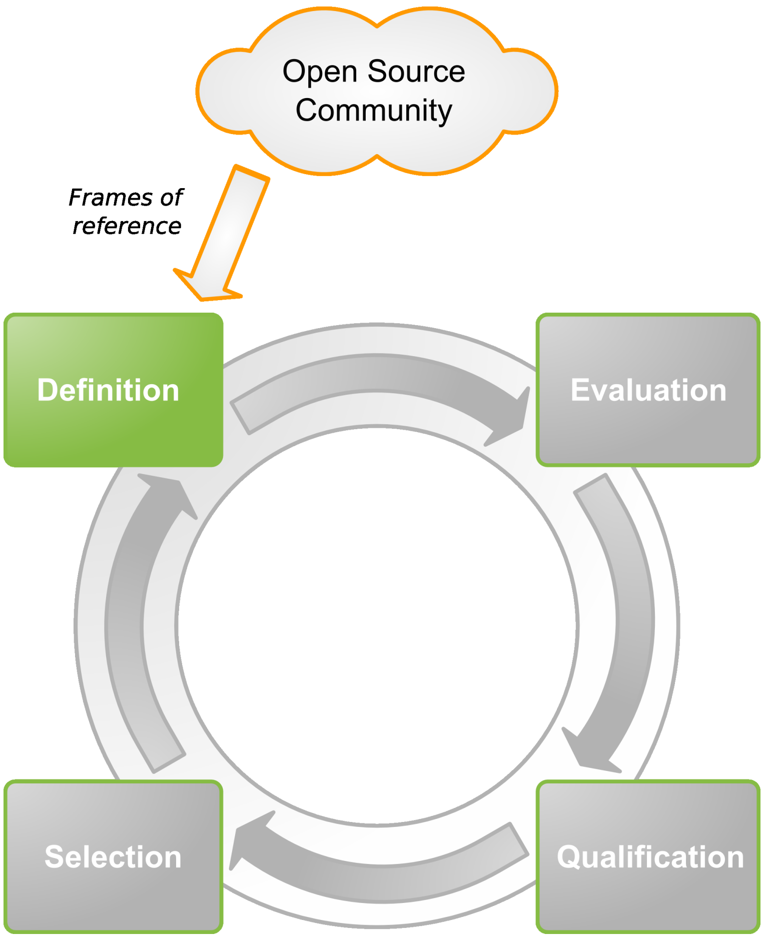
5.3 Tools

A single tool is being developped by Atos Origin to apply the QSOS method in a coherent way.

This tool, nammed O3S (Open Source Selection Software), will be made available to the community on the site `http://www.qsos.org` to coordinate creation, modification and use of QSOS evaluations.

Step	**Description**
1 - Definition	Constitution and enrichment of frames of reference used in the following steps.
2 - Evaluation	Evaluation of software made on three axis of criteria: functional coverage, risks for the user and risks for the service provider (independently of any particular user/customer context).
3 - Qualification	Weighting of the criteria split up on the three axes, modeling the context (user requirements and/or strategy set by the service provider).
4 - Selection	Application of the filter set up in Step 3 - "Qualification" of data provided by the first two steps, in order to proceed queries, comparisons and selections of products.

6 Definition



2: Step 1 - Definition

6.1 Objective

The objective of this step is to define various elements of the typology re-used by the three remaining steps of the general process.

The frames of reference are :

Software families: hierarchical classification of software domains and description of functional grids associated with each domain.

Types of licences: classification of free and open source licenses.

Types of communities: classification of community organizations existing around a free or open source software and in charge of its life-cycle.

6.2 Software families

This frame of reference evolves the most because as software evolves, it offers new functionalities that need to be added to the frame of reference.

6.3 Types of licences

This frame of reference lists and classifies the major licences used for free and open source software. The criteria chosen to describe such a license are:

Ownership: can the derived code become proprietary or must it remain free?

Virality: is another module linked to the source code inevitably affected by the same license?

Inheritance: does the derived code inherit inevitably from the license or is it possible to apply additional restrictions to it?

The table 3 lists a comparison of the most common licences showing the criteria formulated above.

License	**Ownership**	**Virality**	**Inheritance**
GPL	No	Yes	Yes
CeCILL	No	Yes	Yes
LGPL	No	Partial	Yes
BSD	Yes	No	No
Artistic	Yes	No	No
MIT	Yes	No	No
Apache v1.1	Yes	No	No
Apache v2.0	Yes	No	No
MPL v1.1	No	No	Yes
Common Public License v1.1	No	No	No
Academic Free License v2.1	Yes	No	No
PHP License v3.0	Yes	No	No
Open Software License v2.0	No	No	No
Zope Public License v2.0	Yes	No	No
Python SF License v2.0	Yes	No	No

3: Non comprehensive list of licenses

Note that a piece of software or code can be published under the terms of several licences (including closed source).

6.4 Types of communities

The types of communities identified to date are:

Insulated developer: the software is developed and managed by only one person.

Group of developers: several people collaborating in an informal or not industrialized way.

Organization of developers: a group of developers managing the software life-cycle in a formalized way, generally based on role assignment (developer, tester, delivery manager...) and meritocracy.

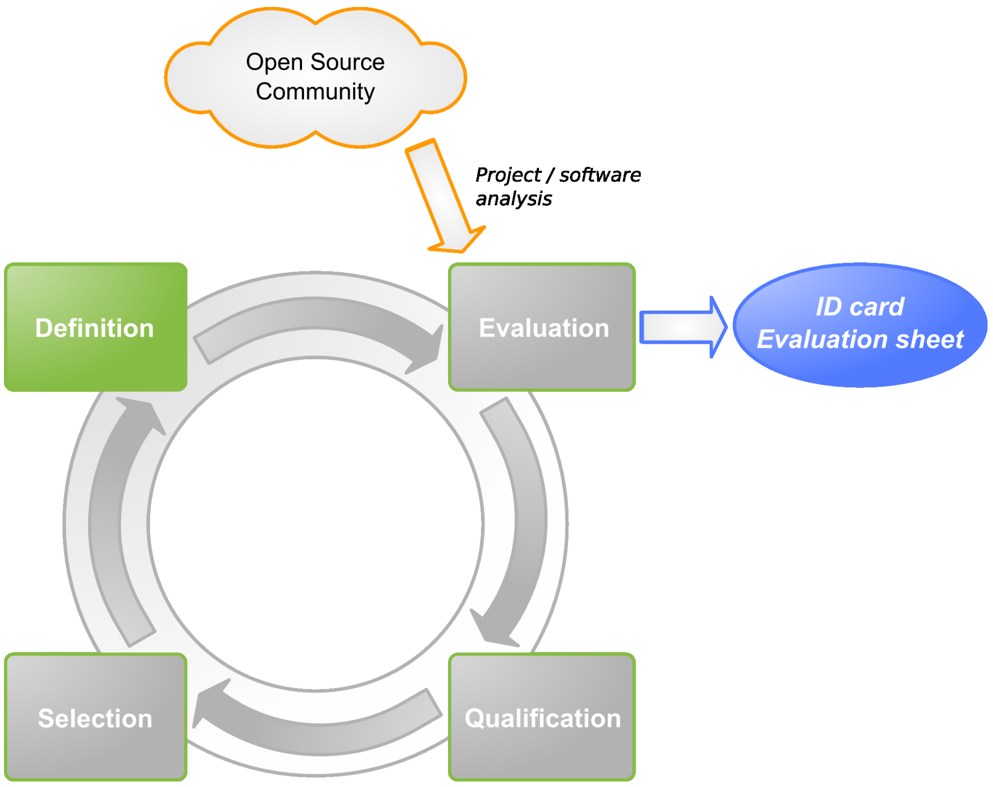
Legal Entity: a legal entity (in general non lucrative) that manages the community, generally possesses copyrights and also manages sponsorship and linked subsidies.

Commercial entity: the commercial organization employing the project's main developers who are remunerated by the sale of services or of commercial versions of the software.

6.5 O3S tool

The O3S tool is designed to be able to easily manage these frames of reference and to measure impacts generated by modifications on data already collected during others QSOS steps.

7 Evaluation



4: Step 2 - Evaluation

7.1 Objective

The objective of this step is to carry out the evaluation of the software. It consists of collecting information from the open source community, in order to:

- Build the identity card of the software
- Build the evaluation sheet of the software, by scoring criteria split on three major axis:
  - Functional coverage
  - Risks from the user's perspective
  - Risks from the service provider's perspective

This evaluation effort can be used in a more global approach of a technological watch which is not covered in this document.

7.2 Identity card

Data constituting the identity card is raw and factual and is not directly scored. However, it is used as a basis for the scoring process described below.

The main parts of an identity card are:

7.2.1 General information

- Name of the software
- Reference, date of creation, date of release of the ID card
- Author
- Type of software
- Brief description of the software
- Licenses to which the software is subjected
- Project's URI and demonstration site
- Compatible operating systems
- Fork's origin (if the software is a fork)

7.2.2 Existing services

- Documentation
- Number of contractual support offers
- Number of training offers
- Number of consultancy offers

7.2.3 Functional and technical aspects

- Technologies of implementation
- Technical prerequisites
- Detailed functionalities
- Roadmap

7.2.4 Synthesis

- General trend
- Comments

7.3 Evaluation sheet

Every software release is described in an evaluation sheet. This document includes more detailed information than the identity card as it focuses on identifying, describing and analyzing in detail each evolution brought by the new release.

7.3.1 Scoring

Criteria are scored from 0 to 2. These scores will be used in Step 4 - "Selection" to compare and select software according to the weightings, representing the user's requirements specified in Step 3 - "Qualifcation".

The following paragraphs describe the criteria used for each axis of evaluation. Note that the same criterion or similar criteria can appear on different axis.

7.3.2 Functional coverage

The functional grid is determined by the software's family and proceeds from the frame of reference of Step 1 - "Definition". Consult the site `http://www.qsos.org` for details of functional grids by software families.

For each element of the grid, the scoring rule is as follows:

Functionality	**Score**
Not covered	0
Partially covered	1
Completely covered	2

In certain cases it is necessary to use several functional grids for a same software, for instance when it belongs to more than one software family. In this case, the functional criteria are distributed on separated axis in order to be able to distinctly evaluate the functional coverage for each family.

7.3.3 Risks from the user's perspective

This axis of evaluation includes criteria to estimate risks incurred by the user when adopting free or open soure software. Scoring of criteria is done independently of any particular user's context (the context is considered later in Step 3 - "Qualification").

Criteria are split into five categories:

- Intrinsic durability
- Industrialized solution
- Integration
- Technical adaptability
- Strategy

The following tables detail each one of these categories, by specifying the rule of notation to be used for each criterion.

Intrinsic durability		**Score**		
		0	**1**	**2**
Maturity	**Age**	For instance less than 3 months	For instance between 3 months and 3 years	For instance more than 3 years
	Stability	Unstable software with numerous releases or patches generating side effects	Stabilized production release existing but old. Difficulties to stabilize development releases	Stabilized software. Releases provide bug fixes corrections but mainly new functionalities
	History, known problems	Software knows several problems which can be prohibitive	No known major problem or crisis	History of good management of crisis situations
	Fork probability, source of Forking	Software is very likely to be forked in the future	Software comes from a fork but has very few chances of being forked in the future	Software has very little chance of being forked. It does not come from a fork either

Intrinsic durability		**Score**		
		0	**1**	**2**
Adoption	**Popularity (related to: general public, niche, ...)**	Very few users identified	Detectable use on Internet (sourceforge, freshmeat, google, ...)	Numerous users, numerous references
	References	None	Few references, non critical usages	Often implemented for critical applications
	Contributing community	No community or without real activity (forum, mailing list,...)	Existing community with a notable activity	Strong community: big activity on forums, numerous contributors and advocates
	Books	No book about the software	Less than 5 books about the software are available	More than 5 books about software are available, in several languages

Intrinsic durability		**Score**		
		0	**1**	**2**
Development leadership	**Leading team**	1 to 2 individuals involved, not clearly identified	Between 2 and 5 independent people	More than 5 people
	Management style	Complete dictatorship	Enlightened despotism	Council of architects with identified leader (e.g: ASF, ...)

Intrinsic durability		**Score**		
		0	**1**	**2**
Activity	**Developers, identification, turnover**	Less than 3 developers, not clearly identified	Between 4 and 7 developers, or more unidentified developers with important turnover	More than 7 developers clearly identified, very stable team
	Activity on bugs	Slow reactivity in forum or on mailing list, or nothing regarding bug fixes in release notes	Detectable activity but without process clearly exposed, long reaction/resolution time	Strong reactivity based on roles and tasks assignment
	Activity on functionalities	No or few new functionalities	Evolution of the product driven by the core team or by user's request without any clearly explained process	Tool(s) to manage feature requests, strong interaction with roadmap
	Activity on releases	Very weak activity on both production and development releases	Activity on production and development releases. Frequent minor releases (bug fixes)	Important activity with frequent minor releases (bugs fixes) and planned major releases relating to the roadmap forcast

Intrinsic durability		**Score**		
		0	**1**	**2**
Independence of developments	**Independence of developments**	Developments realized at 100% by employees of a single company	60% maximum	20% maximum

Industrialised solution		**Score**		
		0	**1**	**2**
Services	**Training**	No offer of training identified	Offer exists but is restricted geographically and to one language or is provided by a single contractor	Rich offers provided by several contractors, in several languages and split into modules of gradual levels
	Support	No offer of support except via public forums and mailing lists	Offer exists but is provided by a single contractor without strong commitment quality of services	Multiple service providers with strong commitment (e.g: guaranteed resolution time)
	Consulting	No offer of consulting service	Offer exists but is restricted geographically and to one language or is provided by a single contractor	Consulting services provided by different contractors in several languages

Industrialised solution				Score
		0	**1**	**2**
Documentation	**Documentation**	No user documentation	Documentation exists but shifted in time, is restricted to one language or is poorly detailed	Documentation always up to date, translated and possibly adapted to different target readers (end user, sysadmin, manager, ?)

Industrialised solution		**Score**		
		0	**1**	**2**
Quality Assurance	**Quality Assurance**	No QA process	Identifies QA process but not much formalized and with no tool	Automatic testing process included in code's life-cycle with publication of results
	Tools	No bug or feature request management tool	Standard tools provided (for instance by a hosting forge) but poorly used	Very active use of tools for roles/tasks allocation and progess monitoring

Industrialised solution		**Score**		
		0	**1**	**2**
Packaging	**Source**	Software can't be installed from source without a lot of work	Installation from source is limited and depends on very strict conditions (OS, arch, lib, ...)	Installation from source is easy
	Debian	The software is not packaged for Debian	A Debian package exists but it has important issues or it doesn't have official support	The software is packaged in the distribution
	FreeBSD	The software is not packaged for FreeBSD	A port exists but it has important issues or it doesn't have official support	A official port exists in FreeBSD
	HP-UX	The software is not packaged for HP-UX	A package exists but it has important issues or it doesn't have official support	A tested package is provided for HP-UX
	MacOSX	The software is not packaged for MacOSX	A package exists but it has important issues or it doesn't have official support	The software is packaged in the distribution
	Mandriva	The software is not packaged for Mandriva	A package exists but it has important issues or it doesn't have official support	The software is packaged in the distribution

Industrialised solution		**Score**		
		0	**1**	**2**
Packaging	**NetBSD**	The software is not packaged for NetBSD	A port exists but it has important issues or it doesn't have official support	A official port exists in NetBSD
	OpenBSD	The software is not packaged for OpenBSD	A port exists but it has important issues or it doesn't have official support	A official port exists in OpenBSD
	RedHat/Fedora	The software is not packaged for RedHat/Fedora	A package exists but it has important issues or it doesn't have official support	The software is packaged in the distribution
	Solaris	The software is not packaged for Solaris	A package exists but it has important issues or it doesn't have official support (e.g: SunFreeware.com)	The software is supported by Sun for Solaris
	SuSE	The software is not packaged for SuSE	A package exists but it has important issues or it doesn't have official support	The software is packaged in the distribution
	Windows	The project can't be installed on Windows	A package exists but is limited or has important issues or covers only specific Windows releases (e.g: Windows2000 and WindowsXP)	Windows is fully supported and a package is provided

Industrialised solution		**Score**		
		0	**1**	**2**
Exploitability	**Ease of use, ergonomics**	Difficult to use, requires an in depth knowledge of the software functionality	Austere and very technical ergonomics	GUI including help functions and elaborated ergonomics (e.g: skins/themes management)
	Administration / Monitoring	No administrative or monitoring functionalities	Existing functionalities but incomplete and in need of improvement	Complete and easy-to-use administration and monitoring functionalities. Possible integration with external tools (e.g: via SNMP, ...)

Technical adaptability		**Score**		
		0	**1**	**2**
Modularity	**Modularity**	Monolithic software	Presence of high level modules allowing a first level of software adaptation	Modular conception, allowing easy adaptation of the software by selecting modules or even developing new ones

Technical adaptability		**Score**		
		0	**1**	**2**
By-products	**Code modification**	Everything by hand	Recompilation possible but complex without any tools or documentation	Recompilation with tools (e.g: make, ANT, ...) and documention provided
	Code extension	Any modification requires code recompilation	Architecture designed for static extension but requires recompilation	Principle of plug-in, architecture designed for dynamic extension without recompilation

Strategy		**Score**		
		0	**1**	**2**
License	**Permissiveness (to be weighted only if user wants to become owner of code)**	Very strict license, like GPL	Moderate permissive license located between both extremes (GPL and BSD), dual-licensing depending on the type of user (person, company, ...) or their activities	Very permissive like BSD or Apache licenses
	Protection against proprietary forks	Very permissive like BSD or Apache licenses	Moderate permissive license located between both extremes (GPL and BSD), dual-licensing depending on the type of user (person, company, ...) or their activities	Very strict license, like GPL

Strategy		**Score**		
		0	**1**	**2**
Copyright owners	**Copyright owners**	Rights held by a few individuals or entities, making it easier to change the license	Rights held by numerous individuals owning the code in a homogeneous way, making relicensing very difficult	Rights held by a legal entity in whom the community trusts (e.g. FSF or ASF)

Strategy		**Score**		
		0	**1**	**2**
Modification of source code	**Modification of source code**	No practical way to propose code modifications	Tools provided to access and modify code (like CVS or SVN) but not really used to develop the software	The code modification process is well defined, exposed and respected, based on roles assignment

Strategy		**Score**		
		0	**1**	**2**
Roadmap	**Roadmap**	No published roadmap	Existing roadmap without planning	Versionned roadmap, with planning and measure of delays

Strategy		**Score**		
		0	**1**	**2**
Sponsor	**Sponsor**	Software has no sponsor, the core team is not paid	Software has an unique sponsor who might determine its strategy	Software is sponsored by industry

Strategy		**Score**		
		0	**1**	**2**
Strategical independence	**Strategical independence**	No detectable strategy or strong dependency on one unique actor (person, company, sponsor, ...)	Strategical vision shared with several other free and open source projects but without strong commitment from copyrights owners	Strong independence of the core team, legal entity holding rights, strong involvement in the standardization process

7.3.4 Risks from the service provider's perspective

This axis of evaluation regroups criteria to estimate risks incurred by a contractor offering services around a free or open source software (expertise, intregration, development, support, ...). It is notably on this basis that its level of commitment can be determined.

Services provinding		**Score**		
		0	**1**	**2**
Maintenability	**Quality of source code**	Not very readable code or of poor quality, incoherence in coding styles	Readable code but not really commented in detail	Readable and commented code, implementing classic design patterns with a coherent and applied coding policy
	Technological dispersion	Use of numerous different languages	One main language with certain modules coded in other languages for specific and limited requirements	One unique language
	Intrinsic complexity	Very complex code requiring high level of expertise to perform modifications without generating side effects	Not very complex code but still requiring expertise in programming languages and software design	Simple code and design, easy to modify
	Technical documentation	No documentation (development guide or automatically generated doc like javadoc)	Incomplete or old documentation without conception and architectural considerations	Detailed and up to date documentation, including conception, architecture design and coding considerations

Services provinding		**Score**		
		0	**1**	**2**
Code mastery	**Direct**	No direct expertise of the source code	Mastery of code but limited to a single person or to only a part of the source	Several individuals mastering the code and covering together the totality of the source
	Indirect	No indirect expertise on the source code	Strong mastery via external expertise provided by partners	Partnership with the copyrights owner and/or the core team

7.3.5 Granularity of scores

As stated above, it is possible to iterate the QSOS process. At the evaluation step this brings the capacity to score criteria in three passes with different levels of granularity:

- First the five main categories
- Then the sub categories of each category
- Finally every remaining criterion

The general process is thus not hindered if not all of the scored criteria are available.

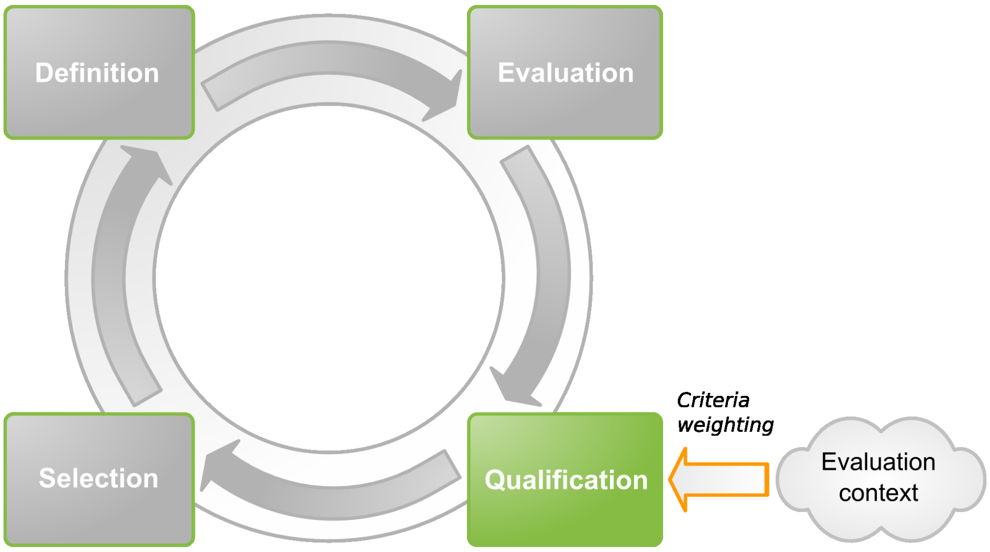
Once all criteria have been scored, the score of the firsts two levels is calculated by the weighed average of scores of the direcly inferior level.

7.4 O3S tool

The O3S tool allows the entry of raw data and the evaluation of software on the three majors axis, as well as generation of the identidy cards of evaluated software.

The granularity of evaluation is managed as follows: as long as all criteria composing a sub category are not scored, its score is not calculated but entered by the user. As soon as all criteria are scored, its score is then automatically calculated.

8 Qualification



5: Step 3 - Qualification

8.1 Objective

The objective of this step is to define filters translating the needs and constraints related to the selection of free or open source software in a specific context. This is achieved by qualifying the user's context which will be used later in Step - 4 "Selection".

8.2 Filter on ID card

A first level of filtering can be defined on data from the software's ID card.

For instance, it could be to consider software only from a given family or software that's compatible with a given operating system.

In general, although it is not mandatory, this filter does not include any weighting; it is mostly used to eliminate inadequate software in the specific context of the user.

8.3 Filter on Functional grid

Each functionality of the functional grid is attributed a requirement level selected among the following:

- required functionality
- optional functionality
- not required functionality

These requirement levels will be linked to weighting values at Step 4 - "Selection", according to the selected mode of selection.

8.4 Filter on User's risks

The relevance of each criterion of this axis is positioned according to user's context, as indicated in figure 8.4.

Relevance
Irrelevant criterion, excluded from filter
Relevant criterion
Critical criterion

This relevance will be converted into a numerical weighting value at the following step, according to the chosen mode of selection.

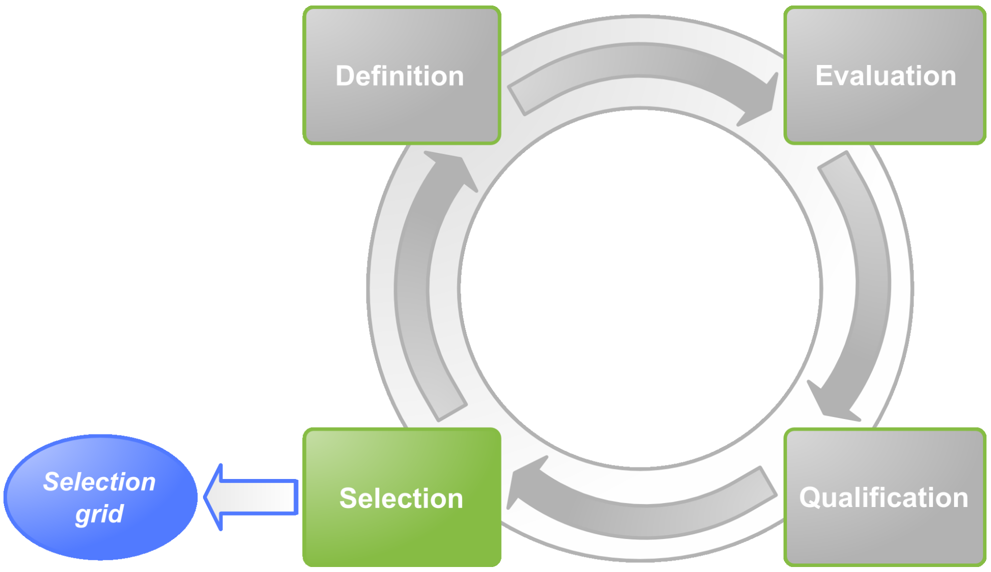
8.5 Filter on Service Provider's risks

This filter is used by a service provider to evaluate software and services to be integrated into its offering and to determine the associated levels of commitment.

8.6 O3S tool

The O3S tool allows the definition of these different filters, while being guided with data entry.

9 Selection



6: Step 4 - Selection

9.1 Objective

The objective of this step is to identify software fulfilling user's requirements, or more generally to compare software from the same family.

9.2 Selection

Two modes are possible:

- strict selection
- loose selection

9.2.1 Strict selection

Strict selection is based on direct elimination as soon as software does not fulfill the requirements formulated in Step 3 - "Qualification":

- Elimination of software incompatible with the filter on the ID card
- Elimination of software not providing functionality required by the filter on the functional grid
- Elimination of software whose scores on the user's risks axis do not meet the relevance defined by or with the user:
  - The score of a relevant criterion must be at least equal to 1
  - The score of a critical criterion must be at least equal to 2

This method is very selective and may, depending on user's requirement, return no eligible software.

Selected software is attributed a total score, calculated by weighting as for the loose selection.

9.2.2 Loose selection

This method is less strict than the previous because rather than to eliminate non-eligible software, it classifies them while measuring the gaps with applied filters.

The rules of weighting to use are detailed in the following paragraphs.

Weighting of functionalities The weighting value is based on the level of requirement defined on each functionality of the functional grid.

Level of requirement	**Weight**
Required functionality	+3
Optional functionality	+1
Not required functionality	0

Weighting on User's risk axis The weighting value is based on the relevance of each criterion on the user's risk axis.

Relevance	**Weight**
Irrelevant criterion	0
Relevant criterion	+1 or -1
Critical criterion	+3 or -3

The weight's value sign represents a positive or negative impact relating to the user's requirements.

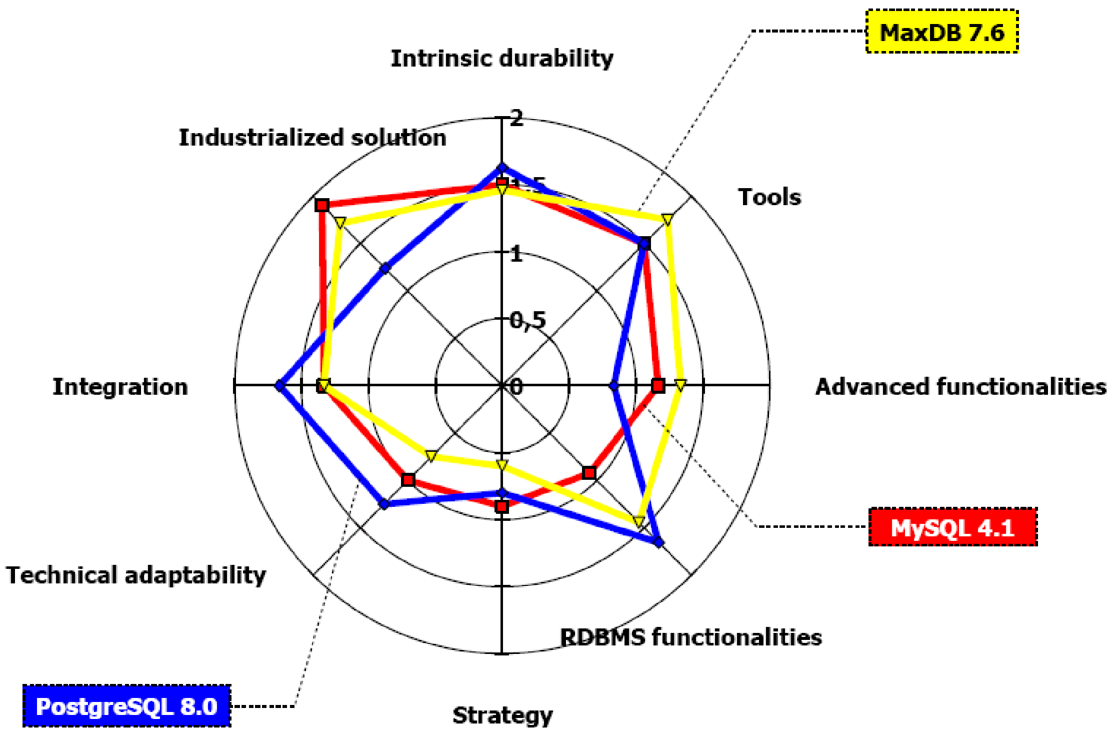
9.3 Comparison

The software of a same family (with a common functionnal grid) can also be compared by using weighted scores determined earlier.

Figure 7 illustrates the kind of synthesis available.

9.4 O3S tool

Besides implementing the strict and loose selection modes, the O3S tool also enables the consultation of data related to a specific software (ID card and evaluation criteria) and the comparison (integrally, by filtering or differentially) of software in the same family.



7: Comparison. This figure is provided as an example, therefore weightings on the various axis are not representative of all kind of RDMBS utilisations.

10 Internet website

The `http://www.qsos.org` website centralizes documents and information on the method itself and also creation, modification, certification of QSOS documents (functional grids, ID cards, evaluation sheets).

Glossary

Fork A fork is an event which sometimes occurs in a development project, typically in Community projects (as in many free and open source software projects), when opinions within the development team diverge on the direction to be taken and prove to be incompatible. The development of the software splits then in two different directions, at the instigation of both parties.

O3S Open Source Selection Software. Tool developed by Atos Origin, implementing the QSOS method, which will be used on the `http://www.qsos.org` website to create, modify and consult ID cards and evaluation sheets.

QSOS method Method for Qualification and Selection of Open Source software, designed and used by Atos Origin as a basis of its support and technological watch services. It is made available - under the terms of a free licence - on the `http://www.qsos.org` website.

Service provider Any company desiring to offer services around free and open source software (expertise, integration, development, support...).

User Any person, entity, company or administration using or planning to use free or open source software.