



Open Source Software is free, Why should we care?

IT Architect Master Program - Run 11

Thesis

Date: September 27th, 2009
Author: Robin Mulkers, Software IT Architect
robin.mulkers@be.ibm.com

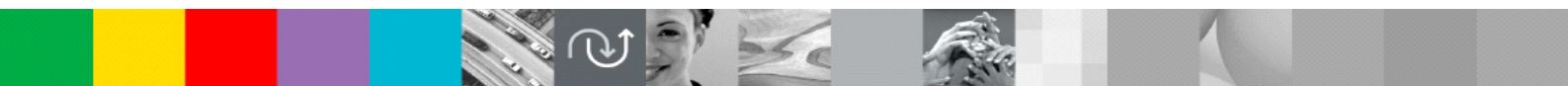


Table of Contents

| | | |
|-----------|--|-----------|
| 1. | SUMMARY..... | 2 |
| 2. | INTRODUCTION..... | 3 |
| 3. | OPEN SOURCE SOFTWARE REQUIRES A STRONG GOVERNANCE..... | 4 |
| 3.1. | ENSURING BUSINESS CONTINUITY..... | 4 |
| | SUPPORT MUST BE AVAILABLE..... | 4 |
| | MAINTENANCE MUST BE AVAILABLE..... | 5 |
| | THE OPEN-SOURCE PROJECT MUST BE ALIVE AND MATURE ENOUGH..... | 5 |
| | THE OPEN-SOURCE SOFTWARE MUST BE MATURE BUT NOT OBSOLETE..... | 6 |
| | THE OPEN-SOURCE SOFTWARE MUST BE TESTED AGAINST BACK-DOORS AND SECURITY FLAWS..... | 6 |
| 3.2. | PREVENTING LAWSUITS AND OTHER LEGAL ACTIONS..... | 6 |
| | CHECK THE LICENSE AND LOOK FOR INTELLECTUAL PROPERTY PROTECTIONS..... | 6 |
| | WATCH OUT FOR FREE SOFTWARE THAT IS NOT OPEN-SOURCE..... | 7 |
| 3.3. | GETTING TOTAL COST OF OWNERSHIP IN CONTROL..... | 7 |
| | ALWAYS CONSIDER MAINTENANCE AND SUPPORT..... | 8 |
| | AVOID REDUNDANCY..... | 8 |
| | MANAGE VERSIONS AND DEPENDENCIES..... | 8 |
| | LIMIT THE NUMBER OF OPEN-SOURCE COMPONENTS PER PROJECT..... | 9 |
| 4. | BLOCKING OPEN-SOURCE SOFTWARE IS NOT AN OPTION..... | 10 |
| 4.1. | OPEN-SOURCE SOFTWARE BRINGS A LOT OF ADVANTAGES..... | 10 |
| | NO LICENSE COST..... | 10 |
| | QUALITY COMPONENTS AND TOOLS ARE SOLVING CONCRETE PROBLEMS..... | 10 |
| 4.2. | COMMERCIAL SOFTWARE OFTEN INCLUDES OSS COMPONENTS..... | 11 |
| 4.3. | DEVELOPERS LIKE TO USE OPEN-SOURCE SOFTWARE..... | 11 |
| | DEVELOPERS IMPROVE THEIR SKILLS, LEARN FROM OSS..... | 11 |
| | DEVELOPERS GET A LOT OF SATISFACTION IN CONTRIBUTING TO OSS..... | 11 |
| | DEVELOPERS PLAY A MORE IMPORTANT ROLE IN THE SELECTION PROCESS FOR OSS..... | 11 |
| 5. | IT'S POSSIBLE TO MAKE AN EFFICIENT USE OF OSS..... | 12 |
| 5.1. | MAKE AN INVENTORY..... | 12 |
| | AN INVENTORY WILL MAKE PEOPLE REALIZE AND UNDERSTAND THE RISKS..... | 12 |
| | AN INVENTORY WILL IDENTIFY ANY ILLEGAL USE OF SOFTWARE..... | 12 |
| | AN INVENTORY IS A NECESSARY INPUT FOR A FUTURE OSS POLICY..... | 13 |
| 5.2. | INSTALL A GOVERNANCE PROCESS..... | 13 |
| | IT MUST BE A LIGHT PROCESS..... | 13 |
| | IT MUST LEVERAGE EXISTING GOVERNANCE PROCESSES AND BOARDS..... | 13 |
| | IT MUST INVOLVE ENTERPRISE ARCHITECTS AND SENIOR DEVELOPERS..... | 13 |
| | IT MUST MAINTAIN THE INVENTORY..... | 13 |
| 5.3. | CREATE AN OPEN-SOURCE SOFTWARE POLICY..... | 14 |
| | SUPPORT DIFFERENT RISK LEVELS LIKE FOR PRODUCTION SOFTWARE AND FOR TOOLS..... | 14 |
| | FORCE THE DEVELOPERS TO WONDER ABOUT LICENSING AND SUPPORT..... | 14 |
| | ALWAYS PROHIBIT COMMERCIAL OR SHAREWARE LICENSES WITHOUT A COMMERCIAL CONTRACT WITH THE VENDOR..... | 14 |
| | EXPLICITLY MENTION WHICH LICENSES ARE ALLOWED..... | 14 |
| 5.4. | INVEST IN TEST AUTOMATION..... | 15 |
| 6. | BIBLIOGRAPHY AND REFERENCES..... | 16 |
| 6.1. | ON-LINE REFERENCES..... | 16 |
| 6.2. | BIBLIOGRAPHY..... | 17 |
| 7. | APPENDICES..... | 18 |

1. Summary

Open-Source Software is the good and the evil of modern software.

Open-Source could mean innovative applications and tools maintained by very active communities for free or obsolete software donated by commercial vendors with the hope that their market share erosion will stop.

Open-Source Software is free and you should care.

Most companies do not know very much how far Open-Source Software is used to support their business. The risks associated to such a situation are numerous:

- business continuity might be at risk due to missing contractual support
- projects might be at risk due to increasing integration and maintenance costs
- business reputation might be at risk due to possible patent infringement

There are also lots of benefits in using Open-Source Software. There are thousands of innovative, leading-edge projects and tools which could help to bring a competitive advantage over the years.

The Open-Source movement is also a more democratic way to look at Software Development. Open-Source developers were early adopters of Web 2.0 social tools, they are proud to be geeks and will continue to influence and promote free software as a religion.

Open-Source Software is “cool”.

It is for sure possible to find the good balance between the risks and the advantages brought by Open-Source Software. For this, it is necessary to take concrete actions:

- make an inventory of all Freeware and Open-Source Software
- organize a smart governance process for Open-Source Software
- create a clear Open-Source Software policy

The policy must not be too restrictive and the governance process must not be too heavy. That governance must not be perceived as a pain. If not, the developers will continue to obfuscate Open-Source code in their projects without informing you as it happens today.

Open-Source Software is already in the house,
Ignoring it is not an option,
Governing it is a necessity.

2. Introduction

The nature of **Open-Source Software** (OSS) is variable and is constantly evolving. From Linux operating system code, higher-level Apache Java components, popular desktop applications like Mozilla Firefox to complete Enterprise Applications Suites like Open ERP, it's rather difficult not to be confused when talking about Open-Source Software, which one?

IBM is a major contributor to a number of Open-Source projects^{WEB-2} and is also integrating Open-Source software in many of its products. The work that is presented here is not derived from the IBM governance processes for Open-Source. It is the result of a project I did for a Belgian governmental agency, it shows the numerous impacts of an uncontrolled use of Open-Source software and provides some recommendations for governing and managing the use of Open Source software in any company.

This paper was originally focused on the use of Open-Source code and components in software development. However, I have discovered along the lines that some of the recommendations were applicable to a broader context than just software development. CIOs, IT strategy advisors or Enterprise Architects will find a lot of useful information to help them when considering Open-Source versus commercial software.

The true implications of using Open-Source software are often misunderstood. Opinions can range from "there won't be any impact", "it will cost us less" to "the company could be suited". OK, it's a little bit dramatized but in general, we ignore or under-estimate the implications of Open-Source Software.

It is true that some lawsuits were highly publicized, like the SCO claim against IBM about Copyright Infringement for the Linux^{WEB-15} operating system in 2003^{WEB-1}. For a large majority, the legal aspects related to the use of Open-Source software are of course important but are not the biggest source of concern.

Nowadays, the use of Open-Source Software is commonplace. Lots of people do have an internet connection at home, at work or on their mobile and use it to search for free software. Users get working Open-Source tools and applications, Developers get pieces of code and working components.

At first sight, this practice isn't really damageable. Why should we prevent it?

In Europe, the mood around Open-Source Software is even more positive. The European governments are promoting Open-Source as a new weapon to counter the US Software giants who are in dominant position on the software market.

In this paper, I'm going to explain that there is no free lunch. I'm not going to claim that Open-Source is a bad thing simply because I don't think so. I'm myself writing this paper using a Linux computer!

I'm just going to provide concrete recommendations and best practices to get Open-Source Software under control in your company, avoiding unnecessary risks and common pitfalls.

3. Open Source Software Requires a Strong Governance

... before bringing any benefit to a company.

The unrestricted, uncontrolled use of Open-Source Software always leads to a situation where it is not possible anymore to know where Open-Source Software is used and which Open-Source Software is used. The lack of inventory is typical with Open-Source Software and the risks associated to such a situation must be considered with attention.

3.1. Ensuring Business Continuity

Open-Source Software, like Commercial software, is supporting the business of a company. However, unlike Commercial software, Open-Source Software doesn't come out of the box with any support or warranty. If the software works initially, all kind of problems could surge months or years after the first installation.

The selection of an Open-Source Component is also essential because maintenance and support are highly dependent on the Open-Source project maturity and the size of its supporting community.

IT industry analyst firms like Gartner^{WEB-12} are unfortunately of little help here, they often cover only a fraction of the Open-Source projects, the biggest ones. Some smaller players like RedMonk^{WEB-13} are sometimes a better source for Open-Source software analysis.

Large System Integrators do also maintain an inventory or a ranking of Open-Source software but those inventories are not publicly available. There is one notable exception which is Atos-Origin who has made public its Open-Source assessment method, QSOS^{WEB-11}, and assessment results for a series of Open-Source software.

Depending on the business criticality of the software used, the need to have a reliable support might be important or not. In order to guarantee the business continuity, here are the aspects to consider in a first place:

Support must be available

This support could be professional:

- Contractual support from the company developing the Open-Source Software
Often as expensive as the support for a similar Commercial software
- Contractual support from a 3rd party
For example, from a hardware vendor providing support for a number of certified Open-Source Software on his platform. Or from a large software vendor for embedded Open-Source Components in his own products
- Internal skilled professionals
When there is no other professional support available, the only solution is to get in-house skills and competences. A very expensive solution that is affordable only for large companies

or the support could be less professional like:

- Community-based support
Typically, using an internet public forum or mailing-list to ask questions. This is the weakest form but the most commonly used form of support for Open-Source Software, there is no guarantee at all that the problem is going to be solved

Maintenance must be available

It must be possible to get fixes for defects in the Open-Source Software. The software should also follow the operating system or runtime platform releases (ex: Java runtime). The best way to assess this is to look after the release history of the Open-Source component and in particular, the time elapsed since the last release.

Some Open-Source projects accept donations and will prioritize the bug resolution or the implementation of a new feature based on the amount of the donation. Even in that case, there is no guarantee that the bug or feature is going to be finally implemented.

The Open-Source Project must be alive and mature enough

Marc Fleury, the former CEO of JBoss^{WEB-14} was classifying the Open-Source projects in 6 different business models^{WEB-9}:

Model 0. For the fun of it

The software is maintained and developed by volunteers

Model 1. Use it on your own job

The software is maintained and developed by people who're using it in their work but this is not their main activity (ISVs, Academic, Consultants)

Model 2. Training and Consulting

Open Source Software becomes a full-time job. It works only if the project is successful enough

Model 3. Dual Licensing

Software is available for free with an Open-Source license that does prevent the commercial redistribution (GPL) and is also available with a commercial license. In this model, the Open-Source version is used as a marketing vehicle. The authors hope that ISVs will OEM the commercial version

Model 4. On Ramp

The Open-Source project is supported by a commercial vendor. On top of this Open-Source platform, some commercial products are built (ex: Eclipse and IBM Rational). The Open-Source project is used as a marketing vehicle, often to get a large community of users that get used to the free open-Source version and are potential customers for higher-value software propositions on top of it

Model 5. Subscription and Support

Professionally organized support and maintenance services (ex: Jboss). It's a model very similar to the commercial vendors except that the revenue comes from subscription and support, not from licenses.

Open-Source Software companies are often playing on different levels of this model. As Open-Source software providers mature (and survive), they move up in the levels.

From a consumer perspective, the lower the level, the higher the risk that maintenance and support are not available at the time you're going to need them.

For the lowest levels, there is a high risk that the project never really takes off and gets abandoned by the main developer(s). This is called "Abandonware".

Abandonware is software that is not maintained anymore. It happens when the main developers of the Open-Source Software leave the project away.

This situation is not rare and happens all the time, it's a major risk for Open-Source consumers.

Main developers are getting married or have children, have moved to another project or have just stopped maintaining it because they were not successful in developing a profitable business around it.

It happened already in the past that a popular Open-Source project wasn't maintained anymore because the main developer was in jail...^{WEB-10}

The Open-Source Software must be mature but not obsolete

Some vendors are “donating” or “open-sourcing” some of their components or products. While this seems a very respectful practice, it often happens as soon as the product or component concerned is no more bringing any added value to the product line.

The rationale behind this donation is then not philanthropic but more an attempt to share the maintenance costs with the secret hope that a community takes care of the future maintenance.

While this Open-Source software might be of an excellent quality, the chances that it is going to be maintained in the future might be severely compromised from the beginning.

The Open-Source Software must be tested against back-doors and security flaws

Yes, Open-Source software is not perfect. Its defects are directly exposed to hackers simply because they have access to the source code. Is it more or less secure than commercial software? I don't know. Some people think that the Open-Source developers are smart guys and as soon as they discover a bug, they fix it. Some other people don't see the difference between an Open-Source developer and a hacker because they dress almost the same.

The reality is that it's important to care about the security of all your software, including the commercial software, the business applications and the Open-Source Software.

By chance, there are now tools that are able to find back-doors and exploitable security vulnerabilities in applications (ex: Rational Appscan^{WEB-20}). These tools are a must, use them.

3.2. Preventing lawsuits and other legal actions

Check the license and look for intellectual property protections

In August 2008, during a first of a kind court ruling^{WEB-3}, the US federal appeals court in Washington said that just because a software programmer gave his work away does not mean that it cannot be protected.

Open-Source Software is always free to download and to use but legal constraints described in the license are applicable when the Open-Source Software is redistributed.

The redistribution of Open-Source Software may have different flavors:

- Variation/Fork when the redistributed software is a modified version of the original Open-Source Software
- Derivative work when the redistributed software includes the original Open-Source Software unmodified

There is a myriad of Open-Source Licenses listed on the Open-Source Initiative website^{WEB-4}. These licenses describe different constraints for Variations or Derivative work. The typical restrictions are related to the license for the redistribution, copyright and other intellectual property protections. Some concepts are very important:

- **Viral licensing**

Is very restrictive, it imposes that the derivative work gets distributed under the terms of the included Open-Source component. If the Open-Source component is free to copy and redistribute, then the derivative work must be free to copy and redistribute as well.

The GNU General Public License^{WEB-16} (GPL) is a very popular example of such a

license because it is the license used for the GNU/Linux^{WEB-15} operating system.

- **Copyright, Patent and Intellectual Property Protections**

Can range from the obligation to explicitly mention that the redistributed version contains Copyrighted material or Intellectual Property from the Open-Source component to much more creative obligations. Some authors do show a real sense of humor when editing their license. I remember having seen an artistic license that was requiring the user of the Open-Source component to send a Postal Card to the original author ;)

These legal restrictions attached to Open-Source Software are almost only impacting some companies, like IBM, who are redistributing, selling commercial software products that include Open-Source code.

In the table here, we can see how the Open-Source original license affects the distribution of either Derivative work or a variation of it.

The GPL license is the most viral, other licenses impose to display some copyright or trademark information on the redistributed work^(*), some licenses don't impose any constraint.

| License Viral behavior: | Derivative Work "Work that uses" | Variation/ Fork |
|-------------------------|----------------------------------|-------------------------------|
| Commercial | OEM agreement | Not possible |
| GPL | GPL | GPL |
| LGPL | No restriction ^(*) | LGPL |
| Apache | No restriction ^(*) | No restriction ^(*) |
| BSD | No restriction | No restriction |

ISVs and Software Vendors should always involve their legal department to assess the constraints associated with an Open-Source Software before it becomes irremediably a part of a commercial product.

A majority of system integrators or companies are building applications for a particular customer or internal use and are in most of the cases not impacted by Open-Source Software license restrictions as long as there is no redistribution of the integrated software.

Watch out for Free software that is not Open-Source

There is a lot of confusion between Open-Source Software, Freeware, Shareware and Evaluation software.

Computer users and developers are not lawyers, they don't know the subtle differences between the licenses, they rarely read them. In addition, they almost never check for the availability of the source code, they just download the binary, ready to use, version of the software.

Commercial vendors as well are bringing confusion in this market, they assume that developers and computer users are influencing the software infrastructure decisions. Some vendors make everything possible to blur the lines between real Open-Source and commercial software.

For example:

- X Open licenses (which in fact are commercial ones)
- X Trial licenses (unrestricted use but for a limited duration)
- X Development licenses (Free but for development purposes only)
- X Community Editions of Commercial products (which are sometimes really Open-Source, sometimes not)

In these conditions, it is really possible to get and use a disguised commercial software without noticing.

3.3. Getting Total Cost of Ownership in control

Open-Source is free, so why am I talking about Total Cost of Ownership?

Is an Open-Source solution less expensive on the long term than a commercial product? This sole topic could deserve a complete article on its own.

With Open-Source, the license costs are non-existent. The TCO of Open-Source Software is mostly influenced by maintenance, support and added integration costs.

While Open-Source Software can be downloaded and used for free, the consequences of that freedom could be also an increase of the overall IT complexity.

The reason is that this unlimited freedom introduces more variation in the software components and result in less standardization of the software landscape.

Always Consider Maintenance and Support

As we've seen earlier, maintenance and support is key for business continuity. All Open-Source projects have a community-based support. Only a fraction of these projects are mature enough to propose a contractual support.

The cost of contractual support and maintenance can be quite high for Open-Source Software. In particular, some vendors have switched from commercial to an Open-Source sales model. The model is then not to get revenue from licenses but from support. In some cases, the access to detailed documentation is only available with a support contract.

When there is no contractual support, the support costs are often hidden in lost productivity. The IT staff people or application developers are much more heavily involved in incident and problem resolution.

Avoid Redundancy

Redundancy in Software is a major concern for Enterprise Architects^{WEB-17} because redundancy increases costs in application development and maintenance.

What should be specifically done for managing the redundancy of Open-Source software? In theory: nothing, Open-Source should be managed using processes and organization as those in place to care about the current and future use of commercial software and business applications.

The problem here is that it is much harder to control the use of Open-Source Software through the Enterprise Architecture or IT management processes in place. Unlike commercial software which requires a contract before being used, Open-Source Software doesn't require anything like that because it can be freely downloaded from the internet and used directly. As its purchase cost is null, the vendor management office is never involved.

The result is that, in many companies, the application developers use their favorite Open-Source components in their own projects, leading to duplicates. The overall number of components to maintain is then bigger than if there was no Open-Source possible.

For example, the inventory I'm referring to in the next chapter, showed that there were in total 3 different tools used to generate PDFs from the business applications, 1 was commercial and 2 Open-Source. As a result it was necessary to keep the knowledge and care for the maintenance of these 3 components while only one or 2 could have been enough.

Manage versions and dependencies

Open-Source components are often strongly dependant on other Open-Source components. It is not rare to find more than 10 dependencies for one Open-Source

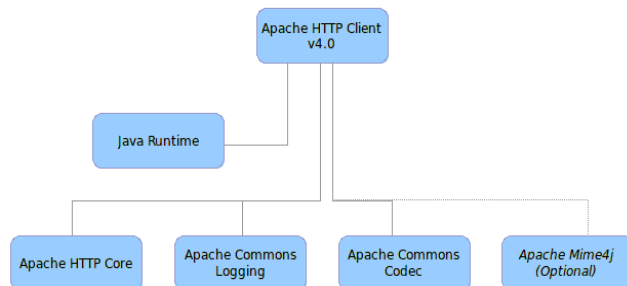
Software component.

Application developers have to manage the situation where for example 2 different Open-Source Components A and B are dependant on a 3rd component C but with 2 different versions. A requires Cv1 and B requires Cv2. In the lucky cases, both versions of the 3rd component Cv1 and Cv2 are compatible and the 2 Open-Source Components A and B may co-exist in their project with the more recent version of C: Cv2 in our case.

In worst cases, they are not and both components A and B are then not compatible. It is quite difficult to imagine how this could seriously be a problem with 3 components but this is definitely a preoccupation in real-world applications^{BOOK-1}.

A simple popular component like for example the Apache HTTP Client^{WEB-7} has 3 required and 1 optional runtime dependencies.

Another server side component, Apache myFaces^{WEB-8}, has 8 dependencies while a more complex one like Apache JMeter^{WEB-5} has 29 dependencies!



It is then very important to properly manage the versions and dependencies between Open-Source components in order to make their proper integration possible.

If not, it will rapidly lead to higher integration costs (integration hell) due to incompatible dependencies in the largest applications.

Limit the number of Open-Source components per project

The granularity of Open-Source components is in general smaller than Commercial products. In the previous paragraph, we have seen that a fully functioning Apache Jmeter test tool is in fact composed of 29 different Open-Source components. A corresponding Rational software for example, would provide that same functionality in a single product.

While some commercial products are often already integrated, the selection and integration of Open-Source components for a particular project is time-consuming and is not a negligible cost in any large project.

Limiting the number of Open-Source components will prevent the project integration costs to grow exponentially.

4. Blocking Open-Source Software is not an option

After reading the previous chapter and getting aware of all the risks related to Open-Source, a radical reaction could be to prohibit all Open-Source Software. Well, I'm not convinced that it's the right thing to do. Let me explain why.

4.1. Open-Source Software brings a lot of advantages

No License cost

This is probably the #1 argument for adopting Open-Source Software. As I've already explained, there is no free lunch. It's not because the initial cost is zero that the Total Cost of Ownership of a solution based on Open-Source software is always lower than the same solution based on a commercial product.

At current, the atmosphere around Open-Source is perhaps a little bit euphoric. But it's time to remember that some very popular Open-Source Software like Open-Office, MySQL or Eclipse are in fact supported by large commercial vendors. What's going to happen to these free Open-Source software if the vendor supporting them is in financial trouble? For some other vendors, Open-Source is only a marketing strategy, a simple way to get a foot in the door and get revenues from maintenance and support contracts instead of licenses.

It's a fact that Open-Source Software is accelerating the "commoditization" of the IT software industry. Commercial software vendors are getting under pressure by a growing number of Open-Source Software providers.

Quality components and tools are solving concrete problems

The community based nature of the real Open-Source Software development is fundamentally different from the traditional commercial software development. The development process for commercial software development is Top-Down. Product managers manage the specifications, taking into consideration both the requirements of their customers and their own product strategy. Those specifications are then used as input for designing and developing the final software product.

Open-Source software development, on the contrary, uses a Bottom-Up development process. There is no product manager. Committers accept or decline changes proposed by the many contributors.

Here, the users are also the main contributors of the technology.

The Open-Source portfolio is composed of thousands of small projects which eventually become popular and are adopted by a large number of users. Often, these few popular projects are inspired and much brilliant than their commercial equivalents. They solve concrete problems in a much elegant way by focusing on what is really necessary rather than getting flooded with useless capabilities.

The granularity of Open-Source Software is also different. There are lots of small Open-Source components and tools and few integrated suites as we can find in Commercial Software.

4.2. Commercial software often includes OSS components

It's a fact that commercial software products include Open-Source components. There are several reasons why Open-Source components are bundled in a commercial product:

- To ease the migration of applications from Open-Source infrastructures/runtimes to their commercial equivalents. In this case, only the interface or API of the Open-Source component is kept but the implementation is highly modified and integrated with the commercial product.
- To support de-facto standards (for example, the popular Apache web server^{WEB-18})
- Because, software vendors, like application development teams don't want to reinvent the wheel and reuse Open-Source components when applicable if the license is compatible with a commercial re-distribution.

In these conditions:

- it's rather difficult to persuade developers to avoid Open-Source Software
- developers are going to use Open-Source frameworks and APIs anyway, often modified versions of these frameworks, when creating applications with commercial tools

4.3. Developers like to use Open-Source Software

Almost every Open-Source project has its own merchandising. Those goodies and wearables are really popular in the geek communities. My guess is that when using Open-Source software like Firefox^{WEB-19} for example, people are able to differentiate themselves and increase their geek level :-).

Seriously speaking, developers do have stronger motivations:

Developers improve their skills, learn from OSS

Popular Open-Source projects are written by very knowledgeable developers, the way the software is constructed, the patterns, the tools and the process used to build and document the software are a great source of inspiration for developers who are able to download it, have a look at the code they are using, get inspired and improve their development skills.

Developers get a lot of satisfaction in contributing to OSS

Only few developers contribute to Open-Source projects. Often the most respected developers in your organization. Contributors to Open-Source projects do develop their network and knowledge much faster and eventually, become respected and popular. Some of them are invited to conferences and maintain successful blogs, some others are hired by famous software companies like Google. In any case, contributing to an Open-Source project is a good career accelerator.

Developers play a more important role in the selection process for OSS

As there is no good coverage of IT analyst firms for Open-Source, the only way to properly select this software is to involve the people who're using it on a daily basis. Developers now play a role in the selection process for components and tools, it's an important social recognition for them.

5. It's possible to make an efficient use of OSS

In the previous chapters, I've identified and commented the benefits and the potential pitfalls when using Open-Source software instead of commercial software. I hope that it was interesting but the question is now: What can we do for it?

I'm going now to give away 4 concrete recommendations to make possible a controlled use of Open-Source Software, avoiding the risks while still getting the benefits.

5.1. Make an inventory

The inventory is the first step in getting back control of the Open-Source Software assets in your company. Realizing it through the mean of questionnaires or by writing down the list of Open-Source Software during a workshop is maybe not enough. This inventory should be executed like an audit. It must include a scanning of deployed production applications.

Open-Source components will be found in the library folders of the deployed applications or in the software repositories while Open-Source tools will be found on the desktops.

An inventory will make people realize and understand the risks

The typical questions for each Open-Source Software found are:

- What is the license? Is there any copyrighted material, what are the constraints?
- Where is the source code? Are you sure the source code you have corresponds to the binary that is used?
- How do you get support? Is there a forum? How long does it take to get an answer on this forum?
- Is the OSS not obsolete? Are there regular releases?

My experience is that the simple fact of asking these questions makes sure that people understand what you are talking about and that they realize the risks.

It happens that the software downloaded some time ago doesn't exist anymore, making impossible to answer those questions. I have experienced some difficulties in getting the license for one component. It appeared after an intense research that this software used in production was proprietary and for evaluation purposes only...

An inventory will identify any illegal use of software

This inventory results in a better understanding of the different Open-Source software in use.

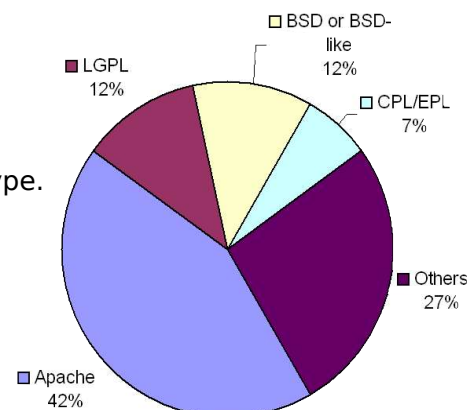
In the inventory I did, there were 63 different Open-Source software and 17 freeware.

In the diagram here, we can see the split per license type.

There are lots of "Other" licenses which were in fact variations of BSD or LGPL licenses with additional trademarks or restrictions.

There was no trace of GPL licenses.

GPL is very much common for GNU/Linux and also PHP while for Java-based Open-Source, Apache license is dominant.



The inventory will be used to detect non-compliant OSS and disguised commercial software and plan their replacement or deprecation.

An inventory is a necessary input for a future OSS Policy

The inventory will be really helpful to know the weaknesses and set-up useful policies and governance. It is key for establishing a realistic Open-Source Software Policy later on. It will help in understanding where are the major problems and defining the right priorities in addressing them. The inventory is also going to reveal classes of usage like for example:

- Productivity OSS, which contains the development and test tools as long as replacing them is transparent for the business
- Business Critical OSS, which contains the Production Software and all the software directly used by the business users

In some companies, there might be additional service levels which could be worth to deserve with a specific Open-Source Software policy.

5.2. Install a governance process

The governance process will care about the control of Open-Source Software after the first inventory. The governance process must describe how new requests for Open-Source Software are submitted, who takes care of handling them and which person or board approves or rejects them.

It must be a light process

The process should be really simple and open. A simple template document, an e-mail and here we go. A too heavy process with approval from many managers might prevent people from using it. The decision process must be reactive enough. It should not be perceived as a pain at all.

I've included in the Appendix an example of an application form^{Appendices}.

It must leverage existing governance processes and boards

The Open-Source governance must not co-exist apart of the overall IT governance. At a certain point, it is useful to know about the IT standards and also be able to detect if the functionality brought by the new Open-Source software is not overlapping with an existing software, Open-Source or commercial.

The Open-Source governance doesn't absolutely require the creation of a new board. An existing Architecture board will qualify and will consider the handling of new Open-Source Software requests as an additional activity.

It must involve Enterprise Architects and Senior Developers

As developers as the main contributors for getting Open-Source Software in-house, it's logical that a Senior developer participates to the selection process. Some components are very technical or specific and the Enterprise Architects might not get what advantage the new Open-Source Software brings to the company.

It must maintain the inventory

The Open-Source governance must maintain an updated inventory of Open-Source Software used in the company. A special attention should also be taken to deprecate or mark obsolete software for removal.

The list of standard Open-Source software must be publicly available (for example on the intranet) so people already know which Open-Source Software they can use or not.

5.3. Create an Open-Source Software policy

The Open-Source Software policy is the reference document that will describe the process and rules applicable to govern the use of Open-Source Software.

The tone of this Open-Source Policy is very important. If it is perceived as too restrictive, developers will continue to use Open-Source software, even copy/pasting Open-Source code in their own code to obfuscate it a little bit more.

The important elements to consider when creating such an Open-Source Policy are:

Support different risk levels like for production software and for tools

In my first experience, while I had identified several classes of usage, I thought that one policy could rule them all. I was plain wrong, it didn't work. It's a necessity to have a relaxed policy for the software and tools that are not really business critical.

If such a tool is not working anymore, replacing it must be an acceptable solution. If the impact is much more important, then maybe this software has to be considered with a higher service level and a much more restrictive policy.

On the other extreme, business critical software used in Production should not pass through without the guarantee that a serious support is available.

Force the developers to wonder about licensing and support

The application form used to submit a new Open-Source Software request must include questions over the license type and support. The person who's going to submit the request is then forced to gather this kind of information. He might as well discover that the software is in fact not updated since a long time or that its forum or mailing list is not active anymore.

The governance process and the policy force people to think twice before introducing a new Open-Source Software.

Always prohibit commercial or shareware licenses without a commercial contract with the vendor

That's a direct corollary of the Open-Source governance process. As we're looking at licenses and source code, disguised commercial software is also detected as well. It's then important to involve the project manager or the vendor management office as this will represent an additional license cost for the project.

Explicitly mention which licenses are allowed

In my case, some licenses were explicitly allowed, representing +/- 80% of the Open-Source software listed in the inventory. For example, the acceptable licenses for production software were:

- ✓ Apache License
- ✓ LGPL
- ✓ BSD
- ✓ Mozilla Public License
- ✓ Common Public License

This list depends on the future use of the software and in particular if there is a plan to redistribute it or not.

The target is to let know that Open-Source Software, distributed under the terms of these licenses, is much more likely to be accepted than another.

It will reduce the involvement of the legal department to a small fraction of the requests.

5.4. Invest In test automation

Testing is very important for embedded Open-Source Software. While a majority of Open-Source Software is released with unit tests, it's a necessity to get a clear idea on how the use of these embedded Open-Source Components affects the business application developed on top of them.

Ideally, these tests should be automated but that's a broader best practice which is not specific to Open-Source.

The goal when automating the tests is to identify much more rapidly if the new version of an Open-Source component breaks the system or not.

The tests to consider are:

- system integration (testing the application including the Open-Source parts)
- performance under load
- security scan (searching for vulnerabilities and back-doors)

The security tests are also very important for Business Continuity but as it is often unknown to what degree the new version of an Open-Source component is less secure, it is important to run security scans more frequently.

6. Bibliography and References

6.1. On-line References

- WEB-1 SCO Files Lawsuit Against IBM
SCO, March 7th 2003
<http://ir.sco.com/ReleaseDetail.cfm?ReleaseID=103273>
- WEB-2 Open Source at IBM
<http://www-03.ibm.com/linux/ossstds/oss/ossindex.html>
- WEB-3 United States Court of Appeals for the Federal Circuit
United States District Court for the Northern District of California, August 13th 2008
<http://www.ca9.uscourts.gov/opinions/08-1001.pdf>
- WEB-4 Open-Source Initiative
<http://www.opensource.org/licenses>
- WEB-5 Apache JMeter project
<http://jakarta.apache.org/jmeter/index.html>
- WEB-6 Open ERP
<http://openerp.com/>
- WEB-7 Apache HTTP Client
<http://hc.apache.org/httpcomponents-client/index.html>
- WEB-8 Apache myFaces
<http://myfaces.apache.org/>
- WEB-9 Professional Open Source
Presentation done at Javapolis 2006 by Marc Fleury
<http://www.parleys.com/display/PARLEYS/Home#slide=1;talk=6972;title=Professional%20Open%20Source>
- WEB-10 File system killer Reiser rejected 3-year sentence
The Register news
http://www.theregister.co.uk/2008/07/10/reiser_rejected_voluntary_manslaughter_plea/
- WEB-11 QSOS
Atos Origin
<http://www.qsos.org>
- WEB-12 Gartner
<http://www.gartner.com>
- WEB-13 RedMonk
<http://redmonk.com>
- WEB-14 JBoss
<http://www.jboss.com/>
- WEB-15 GNU Linux
<http://www.gnu.org/gnu/linux-and-gnu.html>



- WEB-16 GNU General Public License
<http://www.gnu.org/copyleft/gpl.html>
- WEB-17 Enterprise Architect
http://en.wikipedia.org/wiki/Enterprise_architect
- WEB-18 Apache Web Server
<http://httpd.apache.org/>
- WEB-19 Mozilla Firefox
<http://www.mozilla.com/firefox/>
- WEB-20 Rational Appscan
<http://www-01.ibm.com/software/awdtools/appscan/>

6.2. Bibliography

- BOOK-1 Mining Open Source Component Behavior for Reuse Evaluation
Ji Wu, Chun Wang, Xiao-xia Jia, and Chao Liu
School of Computer Science & Engineering, BeiHang University, Beijing
ISBN: 978-0-7695-3398-8

7. Appendices

Appendix A: Open-Source Software Application Form

Open-Source Software Application Form

Name of the software:

Date submission:

Submitted by:

Description: <Max 5 lines, what does this software exactly?>

Motivation: <Max 10 lines, why should we use it?>

List of the environments this software will be deployed on <Prod, Pre-Prod, Development, Developers PC, System administrators PC, ...>

Projects where the software will be used: ...

Pre-requisites: <list of other underlying software required>

Is the software directly used by business users Yes/No

Impact if the software is mal-functioning: ...

Supplier:

Version:

Project Website: <url of the open-source project website>

Licence: <Licence type>

Licence Page: <url of the licence page for this open-source software project>

Download Website: <url link to the latest version download page>

Community Website: <url of the forum / wiki / Jira / mailing list support for this software>

This document does not constitute any form of offer by IBM. Information contained in this document are of indicative nature only. Although our companies may exchange proposals or other materials, neither company will have any obligations or liability to the other unless and until our companies' authorized representatives sign definitive written agreements. Exchanged terms are non-binding to the extent they are not included in definitive agreements. This document is confidential and cannot be reproduced without IBM prior written consent. This message is intended for use of the individual or entity to which it is addressed and may contain personal data that is privileged, confidential and exempt from disclosure under applicable data privacy law. Any review, retransmission, dissemination or other use of, or taking of any action in reliance upon, this information by persons or entities other than the intended recipient is prohibited. If you received this in error, please contact the sender and delete the material from any computer.

© Copyright International Business Machines Corporation 2009. All rights reserved.