JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2007**

Vol. 6, No. 6, July-August 2007

Openness

John D. McGregor, Clemson University and Luminary Software LLC, U.S.A.

Abstract

"Open" seems to be the adjective of choice lately in software engineering circles. Open source, open interface, and open standard are all terms in popular use, but "open" doesn't mean the same thing in every use. What they have in common is that the decision to make something open has strategic implications for the organization. It changes fundamental aspects about how products are produced. In this issue of Strategic Software Engineering I will explore some aspects of openness.

1 INTRODUCTION

I use Eclipse and Eclipse-hosted tools in a number of my classes, research projects, and consulting engagements [Eclipse 07]. There are several reasons for this. The tools are free, they are good quality, and the tools are developed using an open source development strategy. Why do I care that the tools were created as open source? Because open source gives me additional options about how I create my products.

This strategy, and the options that accompany the strategy, have become attractive to a wide range of individuals and corporations. Individuals take advantage of the strategy to facilitate their own user-innovations. Corporations can use the strategy to form communities that ultimately commoditize those products that are complementary to the corporation's target market. More on these options later.

There are several similar uses of the term "open" in the development community. Open standards refer to standards that are developed via a process in which anyone can participate. The process is open with regard to structuring the initial definition and proposing changes to the maturing standard. These standards are usually developed under the auspices of a not-for-profit organization independent of any one commercial company. For example, the Internet Engineering Task Force (IETF) has produced a number of standards for protocols and other tools. [IETF 07]

Open interface refers to an interface whose description is publicly available to anyone who wants to supply an implementation of one side of the interface. These interfaces often are based on open standards. Openness does not preclude a proprietary implementation of an open definition. Encouraging as many customers as possible to provide their own solutions on the other side of an interface from your own solution

Cite this column as follows: John McGregor "Openness", in Journal of Object Technology, vol. 6, no. 6, July - August 2007, pp. 7-14 http://www.jot.fm/issues/issue_2007_05/column1

provides the widest possible market for your product. The ideas of open interfaces and open standards are sufficiently clear that I will leave them here and focus on open source software in the remainder of this paper.

The idea of being "open" appeals to the altruistic nature of some and to the business sense of others. Open source software (OSS) has become a buzzword that is applied to a wide range of approaches, but I will follow the Open Group's definition which comprises three aspects:

- **"Open Source Methods:** A set of tools and processes for managing an extremely loosely-coupled development team.
- **Open Source Licensing:** The particular form of covenant between collaborators embodied in Open Source licensing seems to have resulted in more successful collaboration.
- **The Open Source Community:** The Open Source community is a very-loosely-coupled group of individuals, companies, and organizations that are united only in the use of a particular paradigm of software development and licensing." [Open 07]

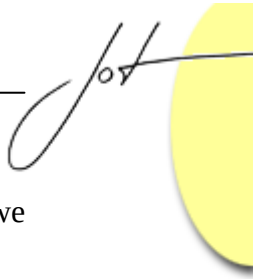
Some have taken to using FOSS or OSS/Free to indicate an apparent close relationship between open source and free software. However, many types of OSS allow uses that violate the principles of the FSF and that are strategically significant. The Free Software Foundation (FSF) defines free software in terms of four freedoms:

- "The freedom to run the program, for any purpose (freedom 0).
- The freedom to study how the program works, and adapt it to your needs (freedom 1). Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help your neighbor (freedom 2).
- The freedom to improve the program, and release your improvements to the public, so that the whole community benefits (freedom 3). Access to the source code is a precondition for this." [FSF 07]

The Open Group's definition seems to be more flexible and the basis for more collaborations so it will be the basis for the rest of this column.

Gianugo Rabellino, of the Apache Software Foundation, advocates what he terms "Open Development" [Rabellino 07]. Open development includes a number of characteristics of the development process that don't always appear in discussions of open source software. In particular it includes an evolutionary approach to developing a product, publicly available means of building the product so that anyone can easily modify the source and produce the product, and a truly non-discriminatory means of achieving membership in the project.

I am interested in open communities from several perspectives. As a researcher I am interested in what makes open source work when it violates some of the basics of economics. As a software development consultant I am interested in what we can learn from open source projects that can be applied to commercial development. As always, I am interested from the perspective of producing a product line of products.



In the rest of this article I will explore some specific facets of openness and how we can use those facets to strategic advantage.

2 PULLING IT APART

One distinguishing characteristic of various open source strategies is the license under which the source code is released. In fact, it is the license that makes open source open. Not being a lawyer I have no intention of explaining the nuiances of these documents, which on the surface seem simple but in some cases have significant implications. The one item I will mention is the "viral" issue. Some licenses try to spread an agenda to any software that uses their software in its implementation. Like most viruses, coming into contact with a viral license can have long term negative effects.

A successful open source project is a complex ensemble of techniques and strategies. Like any ensemble, the individual components may not have the desired effects if used separately. I am going to pull out and list individually several of these components only for the purpose of understanding the whole.

General issues

Trust – Organizations, particularly ones that rely on non-co-located, volunteer labor, are dependent on individuals trusting each other. Open source developers tend to trust those who deliver quality work on time and they quickly quit trusting a developer who does not. Commercial organizations tend to operate more on hierarchical controls than trust but trust is certainly evident in successful commercial teams that have a stable history of working together.

Transparency – Transparency supports trust building across a distributed organization. Openness implies that the activities surrounding the artifacts are available for scrutiny by anyone. The OSS community considers that the many eyes on OSS code tend to more quickly identify bugs. This transparency includes management decisions, such as who has write access to the source code repository, and development discussions about subdividing modules. Consensus building is necessary in a committee driven management process such as that used by most open source projects, but it can be time consuming. A commercial project would be hard pressed to make progress and maintain this level of debate and interaction. Open source projects make participation in operating the organization part of the reward structure for their committers and contributors.

Software process issues

Evolutionary product development process – This is one aspect of Rabellino's Open Development. It is a highly iterative process with feedback from users of initial versions. Most open projects provide access 24/7 to the source code. Users can download the very latest code and begin using it immediately. Users contribute bug reports which drive much of the effort of developers. This level of user involvement can happen in

commercial projects and does happen in some cases such as in-house development efforts. It can also happen between the core asset developers and product builders in a software product line organization. The product builders can provide valuable feedback quickly allowing the core asset developers to react and change their future releases.

As the open source movement has matured, the processes have matured. The OpenUP, an implementation of the Unified Process for open source projects, is one example of explicit processes developed with the unique characteristics of OSS in mind [OpenUP 07]. The process defines 20 artifacts that a project following OpenUP should produce to some degree of formality. The process is intended for the small teams that work on specific tasks in an open source project. It may also turn out to be a reasonable process for the product teams in a product line where all they do is use core assets to assemble products.

Source code available for modification – One of the benefits touted for open source software is the ability to modify the software for the user's purpose. The question is whether that code is structured in such a way that it is feasible to modify the code. In reality, most reuse involves black box reuse behind an interface or via other pre-determined variation mechanisms. So, in evaluating open source software it is more important to examine the variability mechanisms provided than to rely on the openness of the source code.

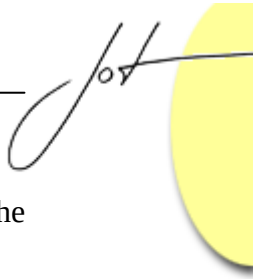
Software architecture - Being architecture-centric is a characteristic of both commercial and open source projects. The architecture plays a critical role in the success of both types of projects. In particular, the success of an open source project is related to having an architecture that is sufficiently modular to support massively parallel development work [Baldwin 06]. The more modular the architecture the easier it is for project leaders to provide interesting work for willing contributors.

Organizational issues

Volunteerism - Economists have studied open source organizations and often focus on the altruism involved. However, one study showed that 38% of Linux contributors did their work on company time. In the "patron" model of OSS project, a large corporation supports the work with contributions of previously developed software, dues for its member contributors, and perhaps cash contributions.

Virtual organization – Like many proprietary software development companies, an open source project is staffed by a number of non-co-located workers. In projects that have supporters that are large corporations, there may be clusters of workers at company locations, but largely the developers are widely distributed. Tools, such as bug trackers, wikis, and configuration management tools are used to facilitate the interaction among these workers.

Control – One of the concerns that some have about open source projects is how the direction and rhythm of a project can be controlled with a large number of volunteers. One control model is meritocracy, used by projects such as Eclipse. Project contributors are rewarded for the size, frequency, and quality of their contributions with additional



privileges such as participation in high-level design decisions or write access to the source code base.

3 EVOLUTIONARY DEVELOPMENT

Evolutionary development is not just iterative, incremental development nor is it limited to OSS. Evolutionary development includes early release of the product to selected users and a feedback mechanism to capture user innovation for future releases. This adds an extra dynamic to the ensemble of techniques and strategies.

Evolutionary development shares some characteristics with agile development. There are three items of particular interest.

Architecture

The architecture of the software product being evolved must evolve as well. One way to accommodate this evolution is to begin with a very high-level architecture. Yes, that means vague and means it is difficult to know the characteristics of products produced from the architecture. Another technique, and the preferable one from my perspective, is to provide variation points in the original definition of the architecture. The latter approach is obviously more desirable but it does require some feel for the directions in which the product will evolve. The product line practices of understanding relevant domains, scoping, and technology forecasting provide information to help guide the placement of those variation points. This is the topic of the OSSPL family of research workshops [OOSPL 07]. In OSS projects there is often a high level architecture linked directly to projects. The more detailed architecture of each subsystem is in the hands of the project team who shepard the assignment of modules and/or the creation of subprojects.

Model-driven

For evolutionary development to succeed there must be tools supporting the development activities. These tools must automate as many activities as possible. For evolutionary development to be optimal, it should be model-driven. This allows the project to more rapidly and accurately incorporate changes.

In some sense every project is model driven because even requirements written in English are a model. But I mean model driven in the sense that there are machine readable representations of each set of artifacts: requirements, architecture, detailed design, and implementation. This is my single biggest problem with open source software. Seldom are any artifacts other than code, and the occasional high level architecture, available. If my sole intention was to reuse the code as is and there were sufficient interfaces to do that I would not care about other artifacts. Its when the evolution I want requires extending or modifying the OSS that these other artifacts become useful.

Traceability

Given the number of changes in an evolutionary development project, it is essential that the dependencies among artifacts be explicitly represented. Then the automation must propagate changes from one artifact to all the others that need to accommodate the change. In OSS projects this traceability is often in the heads of the contributors and committers. In other cases, for example many dependencies in Eclipse, they are explicitly captured in a configuration file or header.

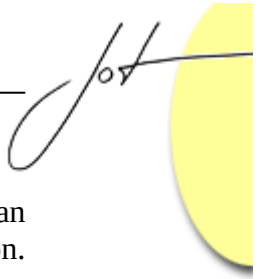
4 STRATEGIC IMPLICATIONS

The OSS approach to development is one of the more completely thought out strategies in the software world. IBM has strengthened their position in several arenas by strategic contributions of assets to open source projects such as Eclipse. An extensive community has grown up around Eclipse and this has commoditized certain segments of the development tools industry. Eclipse has well over 50% of the Java Development Tools market depriving other vendors of significant amounts of revenue. This has increased the demand for consulting businesses while reducing the license revenue of business rivals who follow a more traditional business model. In this case development environments have been driven to commodity pricing levels in order to increase demand for other products and services.

On the other side of the coin, consumers of software, particularly solution providers, also gain from open source software. They gain by being able to offer their products at a lower cost thereby making them affordable to more people. Many companies have found opportunities within the ecosystem surrounding a successful OSS project. For example, Instantiations offers a specialized application builder that extends the Eclipse platform. Their value added is described in the excerpt from their web page shown in the box below.

> By purchasing the professional version you will acquire such benefits as
>
> - Includes SWT Designer, Swing Designer and GWT Designer
> - All features available in professional version. Check our roadmap for details.
> - Free upgrades to new versions for 90 days (or one year*).
> - Free e-mail and newsgroup support for 90 days (or one year*).

Corporations are not the only beneficiaries of open source contributions. Individual developers who ascend to committer status in successful projects gain reputation and increase their value in the market place. One study also showed that the salary of developers who participated in open source projects correlated with their status in their



open source project. Individuals can pursue their own strategy by rising to status in an existing project where they have more influence and maybe even a leadership position. The ultimate way to puruse an individual strategy is to start your own OSS project. Prime the pump with some amount of software you have written and attempt to attract others to your idea. Many successful projects start this way, Linux being the canonical example. Incubators such as SourceForge provide you with the necessary platform from which to launch your effort.

Individual users have the opportunity to realize their own innovations thereby achieving their own goals. This can happen through user-written extensions to the OSS or through user-submitted bug reports. OSS projects use the bug tracking system to manage this type of information. For example, I have built a number of development tools that achieved a higher level of maturity than would have been possible if they were not built as extensions to the Eclipse toolset.

5 SUMMARY

Open source software development is a family of attractive development strategies that fundamentally affect the way an organization develops and delivers software products. You probably will not make any money by just producing open source software. You must have a strategy in which the OSS enables other opportunities that do provide value:

commerical software through a dual licensing arrangement, or
complementary services for those using the OSS, or
the less tangible gains in reputation leading to enhanced employment.

OSS projects form communities. In the January-February 2005 issue of this column1 discussed technical and professional communities. The OSS strategy relies on the formation of a community as a source of labor, ideas, and revenue. Set your sights. Choose a community in which you are interested and participate.

OSS strategies are bringing changes in software development practices such as architecture definition, customer interface management, and structuring the organization. The results of this movement are providing strategic advantage to those who can align their business goals with successful open source approaches. I have tried to delineate some of the variations among open source approaches in hopes that you will be better equipped to select the approach that will provide you or your organization with the greatest competitive advantage.

ACKNOWLEDGEMENTS

Thanks to John Hunt of Covenant College for insightful comments that greatly improved the paper.

REFERENCES

[Baldwin 06] Carliss Y. Baldwin and Kim Clark. „The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model?", **Management Science** 52(7)1116-1127, July 2006.

[Eclipse 07] Eclipse, http://www.eclipse.org, 2007.

[FSF 07] Free Software Foundation, www.fsf.org, 2007.

[IETF 07] Internet Engineering Task Force, www.ietf.org, 2007.

[Open 07] Open Group, www.opengroup.org, 2007.

[OpenUP 07] OpenUP, http://www.eclipse.org/epf/general/OpenUP_Basic.pdf, 2007.

[OSSPL 07] Open Source Software Product Line Workshop, http://www.itea-cosi.org/modules/wikimod/index.php?page=OssPl07, 2007.

[Rabellino 07] Gianugo Rabellino. http://feather.planetapache.org/?p=46, 2007.

About the author

Dr. John D. McGregor is an associate professor of computer science at Clemson University and a partner in Luminary Software, a software engineering consulting firm. His research interests include software product lines and component-base software engineering. His latest book is *A Practical Guide to Testing Object-Oriented Software* (Addison-Wesley 2001). Contact him at johnmc@lumsoft.com.