Open Source: A Boundaryless Information Architecture
Draft 0.1 2003-10-17

Author
Michael Tiemann, CTO Red Hat

Abstract
The goal of a Boundaryless Organization is to improve corporate performance by reducing hierarchy, changing organizational structure, altering boundaries, and overcoming barriers within the organization. Open Source software development model not only mirrors the Boundaryless organization model, but it can actually act as the catalyst for a company's Boundaryless transformation.

This paper will describe the parallels between the Boundaryless model derived by GE and the consultants they hired (Ron Ashkenas et al) and and the Boundaryless models observed within the Linux development community (and specifically Red Hat). It will then describe a seventh business model to add to The Open Group's current six reference business models. Finally, it will discuss some specific examples of how open source software may facilitate or function as components of the Reference Architecture that The Open Group has proposed as part of its vision to support Boundaryless Information Flow.

Table of Contents

Software's Enemy: The External Boundary

The concept of a "Boundaryless Organization" turns on its head many long- and dearly-held notions of the traditional organization. Whereas the traditional organization used boundaries to separate people, tasks, processes, and places, the Boundaryless Organization recognizes that such boundaries do not have intrinsic business value, and may in fact be detrimental. And like any new paradigm, it takes more than a little discipline and commitment to keep from backsliding to the historic patterns of the past.

The traditional software industry is predicated on the notion of developing proprietary technologies and then licensing those technologies under very restrictive terms--a strong and intrinsic external boundary. But this boundary brought with it certain problems never examined from the traditional paradigm:

- investment-driven innovation was often frustrated by lack of (possible) standards and strategic behavior of other companies;
- it was maddeningly difficult to identify, hire, train, and retain qualified programmers;
- with rapidly changing technologies and long development and QA cycles it was virtually impossible to meet customer requirements (especially quality and deployability requirements) in any consistent fashion.

While traditional software companies may suffer the ill effects of other boundary conditions (vertical, horizontal, global), the industry's inability to consider, let alone solve, the external boundary problem has become increasingly apparent. Indeed, it may well be the case that when a company cannot bring itself to solve a fundamental external boundary problem, it will be unable to solve many other boundary problems as well.

The open source software paradigm, typified by the GNU General Public License (GPL), is a broadside attack on the external boundary problem intrinsic to the traditional software model. Whereas proprietary software licenses provide some limited permissions while claiming "All Rights Reserved", the GPL provides quite liberal rights and only one restriction. The rights--to read, modify, and share source code--make it possible for innovation anywhere, anytime, by anyone. The one restriction the GPL puts in place is that no additional restrictions (such as the restriction against commercial use) can be applied when selling, or sharing, otherwise distributing such software. No more radical than the notion of a Boundaryless Organization, the GPL is no

less radical, either.

Users as Innovators: Removing External Boundaries

Open Source software is not the only movement making short work of external boundaries. In the scholarly work "Customers and Innovators: A New Way to Create Value" by Stefan Thomke and Eric von Hippel (published in the Harvard Business Review, April 2002, see http://userinnovation.mit.edu/papers/2.pdf for a downloadable copy), Thomke and von Hippel identify a number of companies in industries as diverse as food products, plastics, silicon technologies, and computer manufacturing who achieved remarkable results by redefining their external boundaries and rearchitecting their product development process to leverage that change.

They observed that the traditional model of product innovation made product R&D a costly and inexact science. In their words:

> In a nutshell, product development is often difficult because the "need" information (what the customer wants) resides with the customer, and the "solution" information (how to satisfy those needs) lies with the manufacturer. Traditionally, the onus has been on manufacturers to collect the need information through various means, including market research and information gathered from the field. The process can be costly and time-consuming because customer needs are often complex, subtle, and fast changing. Frequently, customers don't fully understand their needs until they try out prototypes to explore exactly what does, and doesn't, work (referred to as "learning by doing").
>
> [...]
>
> With the customers-as-innovators approach, a supplier provides customers with tools so that they can design and develop the application-specific part of a product on their own. This shifts the location of the supplier-customer interface, and the trial-and-error iterations necessary for product development are now carried out by the customer only. The result is greatly increased speed and effectiveness.

Though "Customers as Innovators" never mentions the term Boundaryless Organization, they spend considerable time analyzing a case study from GE plastics which clearly exhibits Boundaryless behavior.

> GE Plastics does not design or manufacture plastic products but sells

> resins to those that do, and the properties of those resins must precisely match that of both the end product (a cell phone, for instance) as well as the process used to manufacture that product. With the formation of the Polymerland division in 1998, GE Plastics allowed customers to order plastics on-line and later took the step of making 30 years of its in-house knowledge available on a Web site. Registered users were given access to company data sheets, engineering expertise, and simulation software. Customers could use that knowledge and technology to conduct their own trial-and-error experiments to investigate, for example, how a certain grade of plastic with a speci c amount of a particular type of reinforcement would ow into and ll a mold. The approximate cost of bringing such sophisticated tools on-line: $5 million.
>
> [...]
>
> Today, the Web site attracts about a million visitors per year who are automatically screened for potential sales; that information accounts for nearly one-third of all new customer leads, thus fueling much of GE Plastic's growth. And because the cost of acquiring new business has decreased, GE Plastics can now go after smaller customers it might have ignored in the past. Specifically, the sales threshold at which a potential customer becomes attractive to GE's field marketing has dropped by more than 60%. The on-line tools have also enabled GE Plastics to improve customer satisfaction at a lower cost. Before the Web site, GE Plastics received about 500,000 customer calls every year. Today, the availability of on-line tools has slashed that number in half.

GE's leap of insight was the realization that the proprietary knowledge they'd kept from their customers was keeping their business from performing to its potential. And GE's act of courage, which was to invest significant money to transfer their proprietary knowledge to their customers, thereby cutting through the external boundary they so successfully maintained, turned a decent business into a truly valuable one.

To this point we have focused on just one boundary: the external boundary. We have referenced examples showing that proprietary knowledge (including proprietary software) is not intrinsically valuable. Rather, value is a function of what the customer actually gets. When external boundaries (which are necessarily created by proprietary information) prevent the customer from getting what they need, value is created by eliminating those boundaries. We shall now examine the other three boundaries: vertical, horizontal, and global, in the context of the open source and free software movements.

Boundaryless Bazaars

For the sake of this paper, we shall imagine that open source projects are organizations. After all, they do have leaders, developers, evangelists (marketing), customers (users), and they experience many of the same challenges and pitfalls of traditional organizations--competition, communication issues, resource constraints, etc. common to "real" organizations. Indeed, one of the seminal books on the nature of open source projects, The Cathedral and the Bazaar, by Eric Raymond, observed most early open source projects were just as rigidly maintained as any traditional software development project! These were the Cathedral-style projects, many of which were very successful, such as the GNU tools and utilities and sendmail.

Linux came out of nowhere, and turned the conventional (!?) open source wisdom on its ear. Quoting Raymond:

> Linus Torvalds's style of development - release early and often, delegate everything you can, be open to the point of promiscuity - came as a surprise. No quiet, reverent cathedral-building here - rather, the Linux community seemed to resemble a great babbling bazaar of differing agendas and approaches (aptly symbolized by the Linux archive sites, who'd take submissions from anyone) out of which a coherent and stable system could seemingly emerge only by a succession of miracles.
>
> The fact that this bazaar style seemed to work, and work well, came as a distinct shock. As I learned my way around, I worked hard not just at individual projects, but also at trying to understand why the Linux world not only didn't fly apart in confusion but seemed to go from strength to strength at a speed barely imaginable to cathedral-builders.

The technologies that enable open source are fairly basic compared with more complex workflow models found in traditional software development organizations. For example, sophisticated groupware applications are replaced with email and the broadcast medium of developer mailing lists. Sophisticated email management databases are obviated by a technique invented by Linus Torvalds: when his mail file gets to large, he just deletes it and starts from scratch. After all, if something is so important that somebody can be bothered to send it again, they will. (This certainly gives new meaning to the notion of Boundaryless Information Flow!)

But, as Eric Raymond observed, beneath such apparent madness lay a deep

and remarkably effective method: a forest of source code repositories, labeled, coordinated, and managed by maintainers, accessible to any and all with the ability to access and test the code under a such a variety of conditions as to actually exceed imagination. The technologies that make this possible are worth examination.

Technologies Enabling Open Source Information Flow

The most basic technology required is that of a source code repository. Though this has long been CVS (for which Brian Berliner recently received a STUG award, see http://www.usenix.org/directory/stug.html), a new technology called BitKeeper, which interoperates with CVS, has become the new favorite of some developers. Regardless of one's choice of tool, the most important feature is the ability to communicate, at the source code level, set of changes that implement a feature, fix a bug, or otherwise represent a unit of change against some reproducible reference point. This is what makes it possible to actually participate in the development of Linux (and other bazaar-style) code development.

Bug (and feature) tracking software serves two roles for open source developers. The first (and most obvious) is that it provides a framework for linear progress. The second is that it provides points of entry for would-be developers, testers, and users. Many of the most prolific open source contributors began by reading bug reports and deciding one day "hey, I bet I can fix that!". Bugzilla, developed by Netscape and now maintained by Red Hat, has become a standard (see https://bugzilla.redhat.com/bugzilla/index.cgi).

Of course any given source code requires a suite of tools for developing, building, and testing the code. Because open source permits the direct observation of specific bugs, it is not mandatory to use the same build tools that others do: properly diagnosed, a bug is a bug, no matter how it came to be observed. While most developers tend to use the same tools for efficiency of development, few developers are willing to exclude alternative tools for the sake of conformity. In particular, everybody has a different favorite code editor, and most would be loath to accept an authoring environment that forced conformity to one editor or another.

Because there is little consistency in build environments from one developer to the next (and because it is very difficult to encourage, let alone enforce such consistency), packaging becomes an all-important technology. Early on, Red Hat popularized RPM, the RPM Package Manager.

Whatever the diversity in build environments, RPM (and subsequent technologies like Kickstart, PXE-boot, and Red Hat Enterprise Network) makes it possible to achieve consistency of installation, which is essential to effect any sort of operational standard. RPM goes well beyond mere installation: it supports digital signatures, package building/re-building, package testing, dependency analysis, remote operations, and more. See http://rpm.redhat.com/ for more information.

Above this level of basic technology is the process of open source development. While different projects differ in their details, all project processes are ultimately defined by their "maintainers". There are over 200 people listed in the MAINTAINERS file of the Linux kernel. A Linux kernel maintainer (see http://lwn.net/Articles/3521/) is the final arbiter of what code goes into the main source tree and what gets tossed out. These maintainers have an architectural role, an aesthetic role, a feature and functionality role, and serve to mediate discussions that inevitably cross technical boundaries. The maintainers are loosely organized as a hierarchy, with Linus Torvalds being the BP (Big Penguin). Wonderful features have been vetoed by the BP, and sometimes the BP reverses himself when he concedes to the logic of a superior argument and/or implementation. For a great example, see "New Scalable Scheduler: http://kt.zork.net/kernel-traffic/kt20020114_150.html#4".

To my knowledge, no single company employs more than 10% of these maintainers, and most of the companies that employ the largest number of maintainers (Red Hat and SuSE, for example) are competitors in the market. However, when a maintainer is doing their job, the question of market competition is secondary to the question of technical excellence. This discipline has resulted in such product integrity that despite billions invested in Linux by the very actors who fragmented Unix, Linux has not fragmented. The technical results--performance, scalability, cost benefits, etc., speak for themselves. Linux has not fragmented, despite the pundits' dire warnings, precisely because it was legally impossible to create the kind of barriers that were so intractable in the world of proprietary Unix.

When innovation occurred in Unix, competitors were forced either to play catch up or declare to the market that the approach in question was somehow deeply flawed and that an alternative must be developed. This naturally, rapidly, and repeatedly led to fragmentation that all but destroyed the economic value of Unix.

When innovation occurred in Linux, consensus or criticism occurred first in the arena of the vendor-neutral forum of kernel.org. This is not to say that vendor opinions weren't considered: Red Hat (and others) would frequently say “If you do this, it will make it difficult for us to do XYZ.” As long as XYZ was seen as

valuable (as opposed to, for example, killing competition), debate would continue until everybody was as happy as they could be.

Technically speaking, Linux hasn't fragmented (the technical term is *forked*), although there are many *branches*. A *branch* is a version of the code that is isolated from the mainstream so that developers can focus on a specific set of problems without being distracted by too many external variables. When a problem is sufficiently well solved on a branch, the branch's maintainer can then submit the solution to the mainstream kernel, where it can be integrated and made available broadly.

Whereas the open source paradigm takes natural aim at a single boundary--the external boundary--the Bazaar-like nature of Linux development shatters all barriers--vertical, horizontal, external, and global. What remains is unfettered innovators working at the speed of thought.

Red Hat's Boundaryless Experiences

Breaking down barriers and creating great technology is all well and good, but those alone are not enough to create or sustain a profit-making organization. Red Hat has created a profitable business based on open source software that exhibits a number of boundaryless features.

Red Hat was fortunate to be a small company when it decided to orient its core processes to serving customers and then orient its work around these core processes. While Red Hat does have a hierarchy and does have several independent groups all working in their respective areas, the fact that customers get source code (and the freedom to modify that source as they choose) forces Red Hat to keep vertical and horizontal barriers low. Red Hat also has offices in 20+ countries, and again, the open source model makes it impossible for Red Hat to limit how customers use or deploy our software (regardless of geography). This forces Red Hat to focus on value, not boundaries, as the defining feature of the business.

Can other companies profit from the value shift that Red Hat has leveraged in its open source business? Back to Thomke and von Hippel:

> Perhaps the most important lesson to be learned from GE Plastics is that a company that adopts the customers-as innovators approach must adapt its business accordingly. Furthermore, we've found that because the value that tool kits generate tends to migrate, a company must

> continually reposition itself to capture that value.
>
> [...]
>
> In other words, one long-term result of customer tool kits is that manufacturers lose a portion of the value they have traditionally delivered. But if the conditions are ripe for the technology to emerge in a given industry and if customers will benefit from it and our research shows that they will then suppliers really don't have a choice.

Red Hat's recent financial reports (see "Red Hat Delivers Strong Operating Performance in Second Quarter", http://www.corporate-ir.net/ireye/ir_site.zhtml?ticker=RHAT&script=410&layout=-6&item_id=450143)
prove that open source is a strong foundation for building a profitable business that can grow profitably. But managing that business is not easy, and boundaryless information flow is crucial to our success.
Herein is a strawman of a seventh business model for The Open Group's Boundaryless Information Flow Reference Architecture.

Open Source: A Seventh Boundaryless Information Flow Architecture

This section follows the format of the six models presently described in the Open Group's working document “Boundaryless Information Flow Reference Architecture” (see http://www.opengroup.org/cio/ReferenceArc-Final1.pdf).

Scope

The scope of boundarylessness required for the Open Source business model exists across the participating organization and the community and includes:

- Project maintainers and developers
- Third parties (ISVs and possibly hardare OEMs) using or packaging project components
- User-Innovators

Objectives

The objectives of this form of boundarylessness are:

- Continued support of rapid and robust innovation

- Maintenance of standards (APIs, ABIs, packaging, monitoring, security, etc)
- Growth of the community
- Explicit facilitation of Open Source Architecture (see below)

Constraints

The entire implementation must be open source, preferably GPL. This does not mean there cannot be competing non-OSS implementations, but no part of the implementation can require technology that is not open source. Most open source developers refuse to use proprietary software, especially as a long-term solution. Therefore, to achieve community adoption, the Reference Architecture must itself be open source.

Additional Considerations (The Open Source Architecture)

This model is primarily motivated to give additional process support to a community of people already working together very successfully. The goal of the model, therefore, is to enhance and standardize what already exists, and to make it easier to work more consistently when possible. In particular, to support system-level functionality like integrated security, system virtualization, and to enable enterprise system management, new architectural standards must be defined and implemented broadly across thousands of systems, at little to zero incremental cost to individual projects.

An historical success of this kind has been the internationalization and addition of accessibility (Section 508) to many open source applications. The internationalization and accessibility infrastructure became an architectural feature that was, once defined, reasonable to integrate with existing applications and became "free" for new applications.

The Boundaryless model for open source businesses must address the issue of facilitating architectural implementations and enhancements to enable other broad-based features as well.

A second consideration is that many open source projects are already quite boundaryless: developers have no traditional business relationships (such as employee-manager, contractor-company, etc.) and are often so geographically remote that they often do not meet in real life, nor even communicate except via the asynchronous processes of email and/or code releases. Reputations, all-important among open source developers, are rarely portable to more formal business contexts,

leading to an unintended boundary between those who produce and those who could market and/or better use what is produced. Reputation systems that can provide a meaningful currency across widely varying contexts may lead to greater efficiency by lowering secondary boundary effects induced by the utter boundarylessness of open source development.

Key Common System Architectures

- Workflow management
- Messaging
- Information
- User Interface and Ontology
- Collaborative Work
- Repuatation Systems