Chapter XIV
A Framework for Understanding the Open Source Revolution[1]

Jeff Elpern
Software Quality Institute, Inc., USA

Sergiu Dascalu
University of Nevada–Reno, USA

ABSTRACT

Traditional software engineering methodologies have mostly evolved from the environment of proprietary, large-scale software systems. Here, software design principles operate within a hierarchical decision-making context. Development of banking, enterprise resource and complex weapons systems all fit this paradigm. However, another paradigm for developing software-intensive systems has emerged, the paradigm of open source software. Although from a traditional perspective open source projects might look like chaos, their real-world results have been spectacular. This chapter presents open source software development as a fundamentally new paradigm driven by economics and facilitated by new processes. The new paradigm's revolutionary aspects are explored, a framework for describing the massive impact brought about by the new paradigm is proposed, and directions of future research are outlined. The proposed framework's goals are to help the understanding of the open source paradigm as a new economic revolution and stimulate research in designing open source software.

INTRODUCTION

The *open source revolution* is having a dramatic impact on the computer industry. Web services based on open source technologies play a major role in the Internet. The Linux® operating system has achieved the dominant position within the embedded controller segment of the telecommunication industry. Recently, open source applications have passed Mac applications in penetration into the PC market. Why is this happening? Should we be surprised? Is this a major, self-sustaining phenomenon? This chapter proposes a framework for understanding the open source revolution by identifying a number of market forces driving the revolution and placing these forces within historical perspective. From the birth of open source–the socialism of the GNU Manifesto–to the dominance of current Web services, we show that the open source revolution is a natural response, and part of a continuing effort by users to increase their returns from technology by controlling the market power of commercial software developers. The core argument is based on economics. As users pursue optimal economic returns of their software portfolios, they gravitate toward software solutions that limit the market power of commercial developers. An example of this is the movement toward more and more standards. The adoption of open source is a natural next step for users in the battle for the control of market power. Thus, the open source revolution is the current "front line" in the battle between software developers and users on how economic returns from technologies are allocated between the two. In addition, open source is shown to be a "disruptive technology" in the sense defined by Clayton Christensen in his "The Innovator's Dilemma" (Christensen, 2003). This market force explains the "why now?" issue. As the current commercial software leaders' effort for "sustaining technology innovations" exceeds the users' ability to absorb new features and power, the seeds for the entry of a disruptive technology are sowed. Open source software fits all three criteria for a disruptive technology, which are discussed in this chapter. The disruptive technology framework is also used to provide behavioral and economic models of personal and organizational participation in open source development and delivery. It is important to note that a sweeping paradigm shift like this–the shift from proprietary code to open source–always changes the faces of winners and losers and the processes and the economic models, and, thus, will affect everyone in the industry.

Many discussions on open source software center on questions about what this "new" and "strange" idea is about. How could free source code ever work, isn't its quality poor, and who would ever work on such a project? Yet the reality is that most computer users interact with open source technologies every day as Google, Yahoo and eBay all utilize open source software. Open source is a key component of the most dynamic segment of computing, the Internet. Almost 70% of Web page accesses are provided via the over 40 million Apache™ servers (Netcraft, 2005). This is a market share two and half times greater than the nearest commercial technology.

The dominance of open source Web servers is one example of a stealth *paradigm shift* (Kuhn, 1996) taking place and discussed in this chapter. And, let us make it clear that when we say *paradigm shift*, we are referring to the full scientific revolution concept[2] defined by Kuhn.

Kuhn said that all the powerful stakeholders of the old paradigm will resist and belittle the new paradigm. It is with this insight that we start the discussion of a new, powerful force present today within the technology market. From this view we will be able to understand how the excitement of ever-growing success within the open source community co-exists with the skepticism and hostility of the established software community. This is the classic, painful process of an old paradigm being replaced by a new paradigm.

WHAT IS THE OPEN SOURCE REVOLUTION?

Any discussion of the *open source revolution* needs to begin with the observation that open collaboration, open publishing of findings, and building on the breakthrough efforts of others are at the heart of the scientific process. Society emerged from the Dark Ages and has experienced a 400-year period of accelerating technological innovation using this process.

However, in the current climate of hypercommercialization, ever-increasing amounts of research and innovation are closed-off in proprietary technology. The *open source* movement was born as a reaction to this trend and at its core is a return to the scientific method.

There is no question that the open source paradigm shift started as a social movement. However, it is our position that the open source paradigm has evolved and now economics is the driving force. Let us quickly look at this evolution.

Social Movement

We believe that the 1984 *"GNU Manifesto"* (Stallman, 1984) is the first articulation of an *Open Source* creed. Richard Stallman's almost religious reference to the *golden rule*[3], and how it dictates his developer behavior, launched the *Free Software* movement as a social issue.

Today the GNU Project (GNU, 2008) provides the majority of components of the wildly successful GNU/Linux distributions (usually just referred to as Linux).

The GNU initial position seemed to include both the ideas of free as in *freedom of use*–ability to acquire the source code, modify and redistribute–and free as in *no economic charge*. Over time the emphasis has moved to mainly focus on *freedom of use*. The legal license that enforces Stallman's vision–the GNU Public License (GPL)–defines freedom of use terms (more details are provided in the "Open Source Characteristics" section of this chapter). The GPL license is compatible with for-profit business models.

The Free Software movement is a precursor to the Open Source movement. It is hard to distinguish one from the other except at the philosophical level. Free Software is a social movement about equality, equity, and the return to the scientific method. Open Source focuses on the power of access (no vendor lock-in), limits on market power, and accommodations for prior source code intellectual property being packaged with open source.

Thus, our starting point for the *open source revolution* is more than 30 years ago. Open source, just like a Broadway star, has worked hard to become an instant success.

Collaborative Development

Although Stallman is usually associated with free software and the GPL license, his most stunning contribution may be the *Collaborative Development* process (Williams, 2002). The concept that loosely associated, widely dispersed developers could produce complex, high-quality software was first proved in the GNU project. This was a radical departure from the structured environment, central planning, and hierarchical managed software development processes of proprietary software projects.

Many see involvement with open source projects as a developer's hobby. However, hobbies build model airplanes not jet fighters. Something more that casual "hacking" is taking place. Collaborative development is a fundamental new process.

Today the Linux environment–a huge collection of kernel modules, operating system components and application packages–is developed and tested by thousands across the world without the rigid management hierarchy of corporate software development. Clearly something interesting is happening.

The open processes–including access to the source code–consistently extend the number and the diversity of direct contributors when compared with proprietary closed processes. The new paradigm includes "users as developers." Thus, the people most knowledgeable about requirements and usage are directly impacting the code. Research is starting to quantify the advantages of diversity in the open source software development and testing process (Bosio, Little, Strigini, & Newby, 2002).

Economics Driven

From the moment the GNU Project was started economics played a certain part. However, the *tipping point* (Gladwell, 2000) where massive adoption of Open Source is driven by economics seems to have happened with a series of publications by Raymond, including the "Cathedral and the Bazaar" (Raymond, 1999), providing insight into this new paradigm. Raymond's work provided the first framework for outsiders to understand how this new process could produce world-class technology.

We believe that the open source revolution has now moved from a social statement of individuals to a mainstream economic strategy of corporate technology users. Technology users view open source as a means of controlling the *market power* of software suppliers.

Corporate users purchase technology to generate economic value. The return to the user is the generated economic value less the cost of the technology. Thus, the greater is the market power (pricing power) of the software supplier, the less is the return to the user.

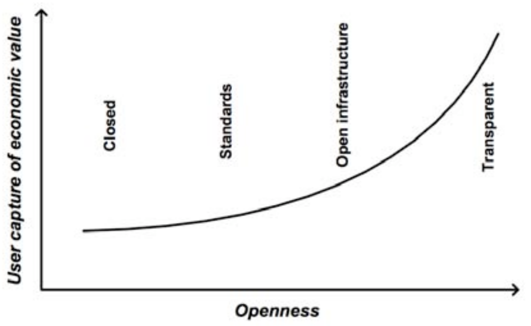
The graphic in Figure 1 shows the user's return increasing at various stages of "openness." The lowest return experienced by the user happens when a software developer has a completely closed product that does not adhere to any standards.

The user increases returns by imposing standards such as SQL for data base technology, Internet RFCs set by the IETF, or XML domain schemas. Each of these provides alternatives and thus reduces the market power of the software vendor.

Open source is the next step in *openness* after standards. Access to the source reduces dependency on a single vendor. This reduces the software vendor's market power, and the return to the user increases.

We believe there is a state beyond just access to source. In the *transparent state* all the information a user needs to make an informed decision is available from the software developer. For example, all outstanding bugs are known, the prior rate of bug fixing is provided, all benchmarks are available, performance to development schedules is provided, and so forth.

Figure 1. User's capture of economic value



Thus, the open source approach is a key strategy available to technology users desiring to increase returns by making software more of a commodity, thus reducing the market power of technology suppliers.

OPEN SOURCE CHARACTERISTICS

The open source revolution is a concept powered by a movement and protected by a legal structure. This section outlines the legal structure.

It is imperative to understand that open source code is different from public domain source code. Public domain software can be used in any way one wishes. Open source software always has a license that grants certain rights and imposes certain restrictions, plus a copyright holder that has legal standing to enforce the license.

The initial free/open software license was the GNU Public License (GPL). The Free Software Foundation–the maintainer of the GPL license-defines free software as having four kinds of freedom as follows:

- *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); and*
- *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.*

(The Free Software Foundation, 2008)

These are the core concepts that are implemented in legal language within the GPL.

Over time, many alternative open source licenses have been developed. Some deal with specific legal details and some offer very substantive differences such as the required "openness" of derivative works.

The Open Source Initiative (OSI)™ is a nonprofit corporation dedicated to managing the Open Source Definition (Open Source Initiative, 2008), a broad statement of objectives, and approving licenses as meeting the objectives. For example, Apache Software License, New BSD license, GPL, IBM Public License, MIT License, Mozilla Public License, and PHP License are all approved as OSI Certified Open Source Software.

In summary, many instances of open source licenses exist that grant the user broad powers and prevent "vendor lock-in."

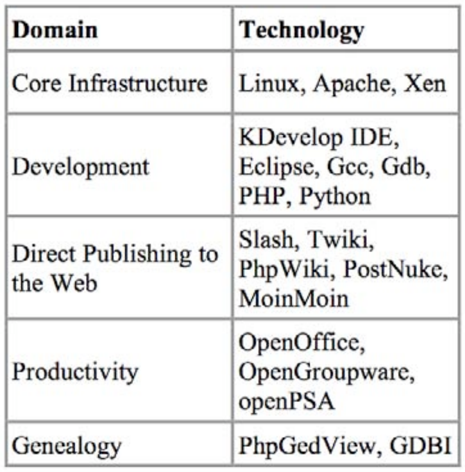
BREADTH OF OPEN SOURCE

Open source is not just Linux. It is a new software paradigm that touches every segment of software development. Initial efforts focused on building the base of an open source operating system and development tools. This work was *by* computer science engineers *for* computer science engineers. This is why open source has a reputation for being very "techy" oriented.

This initial goal was achieved in the late 1990s as Linux matured into a stable and powerful operating system. Work continues on the base but the majority of open source development has shifted to the application areas. The leading collaborative development Web site alone hosts over 100,000 projects, predominately applications, and 1 million registered users (SourceForge, 2005).

One measure of the scope of open source development is to compare the above 100,000 active projects to the approximately 1,400 commercial software packages available on CompUSA's Web site (CompUSA, 2008). Table 1 presents

Table 1. Examples of breadth of open source



Domain	Technology
Core Infrastructure	Linux, Apache, Xen
Development	KDevelop IDE, Eclipse, Gcc, Gdb, PHP, Python
Direct Publishing to the Web	Slash, Twiki, PhpWiki, PostNuke, MoinMoin
Productivity	OpenOffice, OpenGroupware, openPSA
Genealogy	PhpGedView, GDBI

some examples of the breadth of open source technology.

Even very narrow market segments such as genealogy now have active open source development efforts. And, mainstream segments have so many active projects it is difficult to just stay informed.

FRAMEWORK FOR A REVOLUTION

The previous sections outlined the events and current state of open source. This section provides the fundamental framework driving the revolution. First, a business cycle theory of why technological paradigm shifts are always driven by new players is presented. The specific software forces that fit within the theory are covered.

Theory of Disruptive Innovations

The Theory of Disruptive Innovations (Christensen, 2003; Christensen & Raynor, 2003) presents the dynamics of why established firms always create an opportunity for firms with new technologies to win. In short, successful firms establish a market position around a set of technologies servicing a set of clients. Initially, the clients have needs (requirements) that are not met; the "under served" market state. The firm invests in *sustaining innovation* capabilities to close the gap between the product's capabilities and the client's requirements. Inevitably, the *sustaining innovation* closes the gap and then starts producing new features and performance faster then the clients can adopt the new product capabilities; the "over served" market state. This dynamic is presented in the chart shown in Figure 2.

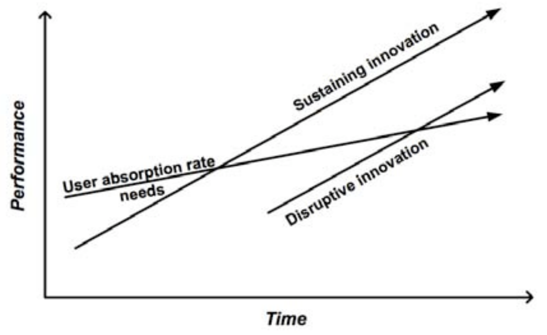
The graphic presents the view of the average client. In reality clients are distributed from light to heavy users of the product's capabilities. Thus, light users would reach the "over served" state well before the average users. And, a point could exist where the majority of the clients have all their needs fulfilled, but the heavy users would still be pressuring the firm to enhance the product.

This is the point of opportunity for a disruptive innovation, usually substantially cheaper, to enter the market even though it may initially be a less capable product. The "over served" users are more than willing to trade features they do not use for cost reductions.

Of course, once established in the market the disruptive innovation establishes its own sustaining innovation rate. At every increment in capability the new, cheaper disruptive innovation eats market share.

This is one of the reasons why a framework such as the one we propose is needed to understand open source vs. proprietary source.

Figure 2. A disruptive innovation emerges



Cutting Edge to Commodity

After the quarter of a century of microcomputer technological innovation and half a century of server technology innovation, computer science algorithms that were cutting-edge at one point have become basic technology. For example, the Xerox Star pioneered graphic interfaces followed by the Apple Lisa, the Apple Mac, and finally Microsoft Windows. Each development team struggled with performance issues as the algorithms for clipping regions and stacking order were developed and matured.

In the early 1980s this knowledge was a valuable trade secret. Now, computer science graduate students around the world routinely develop graphics engines and GUI interfaces as exercises. The exotic has become the routine.

The same is true for relational databases. Oracle and IBM were cutting edge-pioneers in the 1980s. Now relational database algorithms are core computer science technology. Further, this move from exotic to routine is true across the application space as well.

In short, the value and access to technology has changed, and now the industry must follow.

Limit on Software as Intellectual Property

Proprietary software companies are structured around the concept that the source code is the company's main asset, that it has substantial intellectual property (IP) value–mostly know-how but sometimes also patents—and the user will pay premiums for access to this IP. All true in the early formation of each industry segment.

Over time each market segment experiences the following: (i) first, sustaining innovation rapidly addresses all of the needs of the less demanding users, next it addresses the average users, and soon it addresses only the most demanding segment of the market; (ii) early innovations become computer science basic knowledge, and

(iii) all users continue to pay a premium for IP and the sustaining innovation.

Thus, the least demanding users are paying for IP that is now a commodity and for sustaining innovation that they will never use. The seeds for change are thus planted.

Software as a Service

The Open Source software model moves the value from the code base to the people. This is a service model.

In the early evolution of the open source paradigm these services were *traded*. Communities formed around projects. Participants contributed services in return for access to the collective work. The larger projects, in many cases, formed non-profit organizations to co-ordinate and facilitate this *"swap of services."*

Although the collective swap has been a very successful business model for open source it is only one of many possible configurations of the business model (more details are provided in section "A New Business Model" of this chapter).

Darwin at Work

The Open Source Revolution is driven not just by the processes with each project but also by the overall force of *natural selection*. Barriers to entry are low. Barriers to trial are low. And, the programming talent can easily move. In the open source world good projects flourish and bad projects die quickly for lack of support. In economic terms, "the market is efficient."

Superior Process of Open Source

Open source is a true paradigm shift. It is not just a cheap hack of the proprietary code process. The open source model provides a diversity of coding talent, code review, testing and bug fixing that cannot be duplicated in the proprietary environment. For many software segments this may be a superior process (Bosio et al., 2002).

A NEW BUSINESS MODEL

Individuals and businesses new to open source always have the same question: "How can anyone make money when the software is free?" We see this as a larger question: "What are the open source business models?"

First, a historical viewpoint, a similar question from the 1960s: "How can anyone make money on operating system software when it comes free with the mainframe?" This is just a reminder that technology markets change rapidly and that the future product bundles will not be exactly the same as the present product bundles.

Now, about the future business models. Let us start with understanding what "buy software" really means.

Software "Derivatives"

The Black and Scholes (1973) model for pricing financial derivatives won the Noble prize in 1997 (Nobel Prize, 1997). With the model, a financial product such as a stock could be separated into "derivatives," stock without its dividends, dividends only, a call option on the stock, and so forth. These derivatives could then be sold as new financial products, either individually or in new bundles.

We can use this insight about derivatives to think about business models in the open source world. When people say they "bought some software" what do they really mean? A software package is really a bundle of derivative products:

- A set of bits on a CD (the software);
- A right-to-use license;
- An option for installation support (usually at no additional charge);
- An option to purchase software maintenance;
- An option to purchase support;
- An option to buy a future version at a reduced price (upgrade price); and
- An option to purchase training.

Each of these derivatives has a "stand alone" value. One analysis (Lefkowitz, 2005) places the value of the upgrade plus maintenance derivative at more than 80% of the pack cost and the software RTU license at less than 20%. In general, we make claim that the service derivatives–support, maintenance, training–exceed the value of the license derivative.

So the point is that even if the value of the license derivative was driven to zero the above package would retain more than half its value.

One of the issues with closed or proprietary source products is that of "captive" derivatives. When the source is closed, only the original developer can offer some of the derivatives: RTU license, maintenance, future versions. Third parties can offer installation, training and support derivative.

In the open source world all the derivatives are available to every firm. And, an additional derivative, customization, is available because of the access to the source.

So the process of creating an open source business model is one of deciding which of the software derivatives are bundled into the product and how to price that package.

Software as a Service Industry

The above analysis indicates that even today most of the value in a proprietary software product is the services. Thus, a service model is the natural place to start constructing open source business strategies.

A *Close to the Client Application(s) Provider* business model would include: (1) selecting an application, or portfolio of applications, as a specialty; (2) building configuration and source understanding in the application(s); and (3) packaging a yearly professional service product that includes the following derivatives: installation, training, bug fixing, maintenance and upgrades.

This business model captures the majority of the derivative value even though the firm is not the primary developer. It should be noted this model will not return the monopoly margins of a "vendor lock-in" for these derivatives but, if executed well, will return the high margins achieved by good consulting firms.

Another model for well-established and very stable technologies, for example, a Linux/Apache Web server, could be that of a *Close to the Client IT Provider.* Here the derivate set would include only installation and maintenance updates. All application expertise and bug fixing resides with the original developer organization.

Additional business models offering different sets of the software derivatives are also available.

Software as a Community Core Expertise

The open source paradigm also enables organizations banding together to produce software supporting core processes. Outstanding examples are the University portal project (JASIG, 2008) and the Collaboration and Learning Environment for higher education (Sakai Project, 2008). In both cases a number of universities have pledged resources over a number of years to the projects.

Economics and "close to the requirements" are the driving forces of the new software development paradigm. Community software has a huge potential cost advantage over proprietary software. For community software the user group is known and committed so all the sales and marketing costs associated with proprietary software are avoided, as is the profit margin. Together these average about 80% of the revenues of a proprietary software company. Thus, the members of the community are funding only the 20% of normal proprietary software costs that go to development.

OPEN SOURCE FRAMEWORK: BENEFITS PROVIDED AND SIGNIFICANCE FOR DESIGNING SOFTWARE INTENSIVE SYSTEMS

The proposed framework describes an open source revolution that is generating high quality components at every level of the software stack. From operating systems to Web services to relational databases to SOA middleware to applications, viable technologies are available. This impacts software development at both the cost and control levels.

Open Source to Reduce Costs

A design team can insert some open source components into the software stack to reduce the per user license fees. Examples of this strategy are:

- Most wireless handset manufacturers–Motorola, Nokia, Samsung, and so forth–have selected Linux for the operating system component of the software stack. The handset control application is proprietary software for each manufacturer. This "mixed stack" provides a royalty-free solution for the manufacturer.
- Most of the Web pages serviced on the Internet are from a software stack that has at least some open source components. Some large search engine providers use Linux and Apache Web servers as the operating and Web services part of the software stack. There, IP is at the application level where proprietary software is used to index the Web and generate search results.

In both examples, open source has driven the per-user or per-system royalties for third party software to zero. The use of open source is a huge economic win when manufacturing hundreds of thousands of cell phones or deploying tens of thousands of servers to crawl the Web.

Open Source to Increase Control

A design team can insert open source components into the software stack to increase their control. Examples are:

- Using an open source operating system when the application will need modifications to the kernel to perform at required levels.
- Using an open source Web server in an embedded environment of a small device such as a set top cable box. This allows the development team to fit the technology into the hardware by stripping out any functionality not used.

Both of these examples demonstrate a general principle: the higher the open source content is within the software stack, the greater end-to-end development control the team has because of the transparency of the code and the ability to modify it.

LOOKING AT ALTERNATIVES

In this chapter we present a strong case for open source as a viable development strategy. Nevertheless, in the real world, we expect to see users and corporate IT departments pursue a mixed strategy with software portfolios consisting of both proprietary and open source applications. Also, we expect new software development teams to pick the proprietary code strategy in some cases and open source in others.

Alternatives for the Users

The question of when an open source application becomes a viable alternative is driven by the maturity of the open source technology and by the skill set of the user.

Users should select proprietary software applications when at least one of the following applies:

- The software developer has a patent or other intellectual property that makes the application uniquely capable of solving specific requirements.
- The support infrastructure required by the user's technical skills–training, installation consulting, direct support, reference books and internal knowledge–is only available via proprietary software products.
- Standards such as the Structured Query Language (SQL) or the Internet IETF RFCs have imposed a level playing field for software developers and thus have generated a competitive market.

Users should select open source projects when:

- The core technology of the application category is widely understood, that is, a potential resource pool of engineering talent exists for open source projects.
- A well-established open source project meets the users' requirements.
- Users have the expertise to take advantage of the open source technology. For example, users may modify and integrate source code into a larger solution.

Stated at a higher level, users should look to proprietary software for state-of-the-art solutions. On the other hand, as any technology becomes more of a commodity, users should look toward open source applications to reduce costs.

Alternatives for the Software Developers

For software development teams the following considerations drive the proprietary vs. open source decision. Developers should select a proprietary source strategy when:

- They hold a patent of unique IP that makes it almost impossible for another development firm to directly compete.
- They have investors that require a proprietary strategy.
- They are a "first mover" and believe they will be able to defend an established market position.

Developers should select an open source strategy when:

- They plan to enter a market where the existing products are "over serving" the low-end users with a low-end disruptive technology.
- They need to build a community of outside resources to have any possibility of completing the full user requirements.
- They want to band together with similar organizations to reduce costs of "core competency" systems.

Stated at a higher level, software developers should use a proprietary source strategy to maximize returns when there is a user "lock-in." On the other hand, they should use an open source strategy to enter a market when the core technology has become a commodity and leading applications are still proprietary software.

FUTURE RESEARCH DIRECTIONS

The open source revolution has proven that large scale and complex systems such as the Linux operating system and the Apache Web server can be developed using collaborative development methodologies. At the same time many open source projects have failed to produce a lasting technology. In the section "Darwin at Work" of this chapter we presented this as an efficient market where resources quickly flowed to the best projects. However, research opportunities exist that may contribute to a systematic improvement in both the percentage of successful projects and the efficiency of the end user in selecting open source technologies.

The first research directions outlined below will provide fundamental understanding for development teams. We propose studying the impact of systems architecture and project organization on the success of open source projects. Another research direction proposed is focused on providing enhanced analysis tools for the user as he or she selects an open source project. All the research suggested is directed at improving open source effectiveness by providing insight at the early stages of development strategies and technology selection.

These research directions are supported by the availability of a resource unique to open source; access to a vast number of source code repositories. For example, SourceForge.net (www.sourceforge.net) has over 150,000 registered projects. Within the source code revision control of each open source project is information about the number of contributors, the turnover of contributors, the programming language, the growth and change rate of the code base, and interfaces with other technologies. This is a vast pool of information just waiting to be analyzed. In addition, many of the code repositories are accompanied by source and binary download services. The statistics on downloads are a good indicator of user interest and acceptance.

Impact of System Architecture on the Success of Open Source Projects

The first line of research proposed is into the effects of the core system architecture on a project's success. This is a straightforward computer science question: "Do the open source projects that utilize 'best practices,' as currently understood by the computer science community, show a higher rate of success than projects that use other practices?"

Algorithms to extract methodology metrics from a project's source code repository can be developed. Algorithms to extract measures of project success from download statistics can also be designed. The data set could then be analyzed to identify relationships between system architecture practices and the ultimate success.

An associated research issue is how the system architecture affects the collaborative development process. For example, do object-oriented architectures and program languages—which isolate the scope of the code—make the collaborative process more effective? Or, is the open source methodology equally effective across programming languages and architectures?

Impact of Organizational Model on the Success of Open Source Projects

Another line of research could focus on the impact of the organizational model on a project's chance of success. The research could develop organizational categories (decentralized contributors, strong-core contributors, legal foundation of non-profit organization, corporate sponsor, consortium sponsor, etc.), map open source projects into these categories, and develop metrics of success. The research can then look for relationships between organizational models and the project's chances of success.

Contributor Participation as Leading Indicator of Success

The end-user or corporate information technology (IT) organization selecting an open source technology is making the decision without any of the business metrics that normally are used to assess stability because open source projects do not have income statements and balance sheets. Thus, alternative measures of viability are needed.

This line of research could focus on using the size and frequency of activity of the contributing developer community as a measure of project viability. Algorithms to extract participation measures–number of contributing developers, number of major developers, turnover of developers, coverage of multiple developers, growth of source base, and so forth–could be developed. Algorithms to extract measures of project success from download statistics could also be created. Further, the data set could be analyzed to identify relationships between developer analytics and the project's success.

Visualization could play a major role in this research direction. The focus here would be on presenting developer metrics in a form that mainstream users can understand. The goal should be to create visualizations that enable users to easily estimate the success of open source projects. Also, it would be desirable to have these projections rank-ordered close to the quantitative prediction, that is, a user projection of "very likely to succeed" will correlate with a high probability of success metric.

Along these lines, a very interesting site moving in the direction of advanced code analysis and visualization is ohloh (www.ohloh.net). This site crawls source code repositories to produce a graph of code growth, a simple rating of development activity and a list of top developers each with a contribution graphed by month. Ohloh shows Mozilla Firefox having a code history of 1 million lines in January 2000 growing to almost 2 million lines by January 2007. The contributor statistics show over 400 contributing developers with the top 10 having over 1,000 code commits each. From the above, one can see that this is clearly a large, stable project. IBM has also done very exciting and useful research in visualizing the contribution of authors to Wikipedia pages (IBM, 2008). This approach could be used to visualize source code development over time.

Thus, the market is responding to the user needs for assessing a project's stability. The majority of users would most likely assess a project of hundreds of contributing developers as stable and a project with one developer as risky. But what about the risk level of a project with 5 or 10 developers? Research in this direction could provide a deeper understanding of "risk assessment" across all sizes of developer communities. Thus, the users would be better equipped to draw the line between projects that are really "lab research" and those that are stable, commercially ready.

CONCLUSION

Open source software represents a fundamental paradigm shift in developing software. New development processes are emerging. New business models are being tested. Across technology suppliers, the open source revolution is creating new winners and losers. However, one thing is clear even in the midst of the paradigm shift chaos: technology users are, and will continue to be, the big winners.

However, while we can observe the evolution of software development driven by the market forces and we can see winning products emerge, the theory of why this new paradigm works is lacking. While many open source projects are winners, many fail. While high quality products like Linux and the Apache Web server exist and thrive, other open source projects are poorly designed and bug-ridden. Just making a project open source is not the recipe for success.

The framework proposed in this chapter describes a software development paradigm that is economically driven, long-term sustainable, and unique in terms of the processes involved. It is our hope that the framework will stimulate research into the design and development of the open source software. The open source paradigm offers a new alternative for software-intensive system development. Some of the main issues now are to build theoretical models for successful open source development and identify more precisely the application domains that open source projects can address most effectively.

REFERENCES

Black, F., & Scholes, M.S. (1973). The pricing of options and corporate liabilities. *Journal of Political Economy, 81*(3), 637-654. University of Chicago Press.

Bosio, D., Little, B., Strigini, L., & Newby, M. J. (2002, February). Advantages of open source processes for reliability: Clarifying the issues. In *Paper presented at the Workshop on Open Source Software Development*, Newcastle upon Tyne, UK.

Christensen, C. M. (2003). *The innovator's dilemma*. HarperBusiness Essentials.

Christensen, C.M., & Raynor, M. E. (2003). *The innovator's solution: Creating and sustaining successful growth*. Harvard Business School Press.

CompUSA. (2008). *Software*. Retrieved March 23, 2008, from http://www.compusa.com/products/products.asp?cmid=topnav&N=200163

Gladwell, M. (2000). *The tipping point: How little things can make a big difference*. Back Bay Books.

GNU. (2008). *The GNU operating system*. Retrieved March 23, 2008, from http://www.gnu.org

IBM. (2008). *Visualizing the editing history of wikipedia pages*. Retrieved March 23, 2008, from http://www.research.ibm.com/visual/projects/history_flow/index.htm

JASIG. (2008). *JASIG's U-Portal: Evolving portal implementations from participating universities and partners*. Retrieved March 23, 2008, from http://www.uportal.org/

Kuhn, T. S. (1996). *The structure of scientific revolutions* (3rd ed.). IL: University of Chicago Press.

Lefkowitz, R. (2005, July 21). *Calculating the true price of software*. Retrieved March 23, 2008, from http://www.onlamp.com/pub/a/onlamp/2005/07/21/software_pricing.html

Netcraft. (2005, August-October). *Market share for top servers across all domains*. Retrieved March 23, 2008, from http://news.netcraft.com/archives/2005/10/index.html

Nobel Prize. (1997). *The Nobel Prize in Economics 1997 – press release*. Retrieved March 23, 2008, from http://nobelprize.org/nobel_prizes/economics/laureates/1997/press.html

Open Source Initiative. (2008). *The open source definition*. Retrieved March 23, 2008, from www.opensource.org/docs/definition.php

Raymond, E. S. (1999). *The cathedral and the bazaar: Musings on Linux and open source by an accidental revolutionary*. O'Reilly & Associates.

Sakai Project. (2008). *About Sakai*. Retrieved March 23, 2008, from http://www.sakaiproject.org/index.php?option=com_content&task=view&id=103&Itemid=208

SourceForge. (2005). *SourceForge.net®, world's largest open source development site surpasses 100,000 projects hosted on the site*. Retrieved March 23, 2008, from http://www.ostg.com/pdfs/SourceForgeProjects_PR_Final1.pdf

Stallman, R. (1984). *GNU manifesto*. Retrieved March 23, 2008, from http://www.gnu.org/gnu/manifesto.html

The Free Software Foundation. (2008). *The free software definition*. Retrieved March 23, 2008, from http://www.fsf.org/licensing/essays/free-sw.html

Williams, S. (2002). *Free as in freedom: Richard Stallman's crusade for free software*. O'Reilly & Associates.

ADDITIONAL READING

Aberdour, M. (2007). Achieving quality in open-source software. *IEEE Software, 24*(1), 58-64.

Carrington, D., & Kim, S.-K. (2003). Teaching software design with open source software. In *Proceedings of the 33rd Annual Frontiers in Education Conference* (Vol. 3, pp. S1C/9-14). IEEE Computer Society Press.

Christley, S., & Madey, G. (2007). Analysis of activity in the open source software development community. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, (pp. 166b/1-10). IEEE Computer Society Press.

Crowston, K., Annabi, H., Howison, J., & Masango, C. (2004). Effective work practices for software engineering: Free/libre open source software development. In *Proceedings of the 2004 ACM Workshop on Interdisciplinary Software Engineering Research*, (pp. 18-26). ACM Press.

Dinkelacker, J., Garg, P.K., Miller, R., & Nelson, D. (2002). Progressive open source. In *Proceedings of the 24th International Conference on Software Engineering*, (pp. 177-184). IEEE Computer Society.

Ebert, C. (2007). Open source drives innovation. *IEEE Software, 24*(3), 105-109.

Edwards, J. (1998). The changing face of freeware. *Computer, 31*(10), 11-13.

Feller, J., & Fitzgerald, B. (2000). A framework analysis of the open source software development paradigm. In *Proceedings of the 21st International Conference on Information Systems*, (pp. 58-69). ACM Press.

Fink, M. (2002). *The business and economics of Linux and open source*. Prentice Hall.

Fitzgerald, B. (2004). A critical look at open source. *Computer, 37*(7), 92-94.

Fogel, K. (2005*). Producing open source software: how to run a successful free software project*. O'Reilly Media.

Fowler, D. (2000). Open season. *netWorker, 4*(2), 18-25.

Ghosh, R.A. (Ed.). (2006). *Study on the economic impact of open source software on innovation and the competitiveness of the Information and Communication Technologies (ICT) sector in the EU –final report* (Tech. Rep. contract ENTR/04/1112). Lead contractor UNU-MERIT, The Netherlands. Retrieved March 23, 2008, from http://ec.europa.eu/enterprise/ict/policy/doc/2006-11-20-flossimpact.pdf

Ghosh, R.A., Glott, R., Wichmann, T., Spiller, D., Krieger, B., & Robles, G. (2002). *Free/libre and open source software (FLOSS): survey and study–final report* (Tech. Rep., several parts). International Institute of Infonomics, University of Maastricht, The Netherlands and Berlecon Research GmbH, Berlin, Germany. Retrieved March 23, 2008, from http://www.infonomics.nl/FLOSS/report/index.htm

Glass, R.L. (1999). The loyal opposition of open source, Linux … and hype. *IEEE Software, 16*(1), 126-127.

Golden, B. (2004*). Succeeding with open source*. Addison-Wesley.

Goth, G. (2005). Open source meets venture capital. *IEEE Distributed Systems Online*, *6*(6), 1-7.

Hecker, F. (2006). Setting up shop: The business of open-source software. *IEEE Software, 16*(1), 45-51.

Hoepman, J.-H., & Jacobs, B. (2007). Increased security through open source. *Communications of the ACM, 50*(1), 89-83.

Koponen, T. (2006). Evaluation framework for open source software maintenance. In *Proceedings of the International Conference on Software Engineering Advances,* (pp. 52/1-5). IEEE Computer Society Press.

Laplante, P., Gold, A., & Costello, T. (2007). Open source software: Is it worth converting? *IT Professional, 9*(4), 28-33.

Lawton, G. (2002). Open source security: Opportunity or oxymoron? *Computer, 35*(3), 18-21.

Madanmohan, T.R., & Dé, R. (2004). Open source reuse in commercial firms. *IEEE Software, 21*(6), 62-69.

Mindel, J. L., Mui, L., & Sameer, V. (2007). Open source software adoption in ASEAN member countries. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences,* (pp. 226b/1-10). IEEE Computer Society Press.

Mockus, A., Fielding, R.T., & Herbsleb, J.D. (2002). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology, 11*(3), 309-346.

Norris, J.S. (2004). Mission-critical development with open source software: Lessons learned. *IEEE Software, 21*(1), 42-49.

Neumann, P.G. (1999). Inside risks: Robust open-source software. *Communications of the ACM, 42*(2), 128.

O'Reilly, T. (1999). Lessons learned from open-source software development. *Communications of the ACM, 42*(4), 32-37.

Ousterhout, J. (1999). Free software needs profit. *Communications of the ACM, 42*(4), 44-45.

Paulson, J.W., Succi, G., & Eberlein, A. (2004). An empirical study of open-source and closed-source software products. *IEEE Transactions on Software Engineering, 30*(4), 246-256.

Rosen, L. (2004*). Open source licensing: Software freedom and intellectual property law.* Prentice Hall.

Ruffin, C., & Ebert, C. (2004). Using open source software in product development: A primer. *IEEE Software, 21*(1), 82-86.

St. Laurent, A.M. (2004*). Understanding open source and free software licensing.* O'Reilly Media.

Samoladas, I., Stamelos, I., Lefteris, A., & Apostolos, O. (2004). Open source software development should strive for even greater code maintainability. *Communications of the ACM, 47*(10), 83-87.

Scacchi, W. (2004). Free and open source development practices in the game community. *IEEE Software, 21*(1), 59-66.

Spinellis, D. (2006). Open source and professional advancement. *IEEE Software, 23*(5), 70-71.

Toth, K. (2006). Experiences with open source software engineering tools. *IEEE Software, 23*(6), 44-52.

Weber, S. (2005*). The success of open source.* Harvard University Press.

Weiss, A. (2001). The politics of free (software). *netWorker, 5*(3), 26-31.

Wikipedia® – OSS (2008). *Open-source software. Wikimedia Foundation, Inc.* Retrieved March 23, 2008, from http://en.wikipedia.org/wiki/Open_source_software

Wikipedia® – FSF (2008). *Free software foundation. Wikimedia Foundation, Inc.* Retrieved March 23, 2008, from http://en.wikipedia.org/wiki/Free_Software_Foundation

Woods, D., & Guliani, G. (2005*). Open source for the enterprise: managing risks, reaping rewards.* O'Reilly Media.

Wu, M.-W., & Lin, Y.-D. (2001). Open source software development: An overview. *Computer, 31*(10), 11-13.

ENDNOTES

1 All marks are the properties of their respective owners.

2 Kuhn (1996) defined science, in this case computer science, as taking place within a belief set, with changes accepted by a small set of authorities and with the next set of scientists selected and trained by these authorities. He termed this Web of belief, authority and acceptance a *paradigm*. From time-to-time a new scientific belief set is established–Copernicus, Newton, Einstein–that is not just a linear extension of an existing paradigm but is radically different. He calls this a *paradigm shift*. One could postulate that the paradigm of "big software projects" is well established in computer science with the many attendant issues of scaling, managing large teams, long design cycles, and so forth. One could also postulate that open source concepts and methodologies are a paradigm shift, that is, the way Apache or Linux or KDE environments are developed are not just sloppy, or underfunded, or lucky exercise of the old paradigm but the living proof of a new paradigm. From this framework it is easy to see why computer scientists using the beliefs and tools of the big-project paradigm are baffled by how the "chaotic" processes of open source can deliver short cycle, feature rich, high quality products. This is similar to the scientist using the Newtonian belief set struggling to understand Einstein's theory that distance could be relative.

3 From the "*Why I Must Write GNU* "(Stallman, 1984) section of the Manifesto: "I consider that the golden rule requires that if I like a program I must share it with other people who like it. Software sellers want to divide the users and conquer them, making each user agree not to share with others. I refuse to break solidarity with other users in this way. I cannot in good conscience sign a nondisclosure agreement or a software license agreement. For years I worked within the Artificial Intelligence Lab to resist such tendencies and other inhospitalities, but eventually they have gone too far: I could not remain in an institution where such things are done for me against my will. So that I can continue to use computers without dishonor, I have decided to put together a sufficient body of free software so that I will be able to get along without any software that is not free. I have resigned from the AI lab to deny MIT any legal excuse to prevent me from giving GNU away."