The Commons as New Economy

& What This Means for Research

Richard P. Gabriel
IBM Research
rpg@{us.ibm.com
dreamsongs.com

Abstract

Suppose the entire social and commercial fabric supporting the creation of software is changing—changing by becoming completely a commons and thereby dropping dramatically in cost. How would the world change and how would we recognize the changes? Software would not be continually recreated by different organizations, so the global "efficiency" of software production would increase dramatically; therefore it would be possible to create value without waste, experimentation and risk-taking would become affordable—and probably necessary because firms could not charge for their duplication of infrastructure—, and the size and complexity of built systems would increase dramatically, perhaps beyond human comprehension. As important or more so, the activities of creating software would become the provenance of people, organizations, and disciplines who today are mostly considered consumers of software—there would, in a very real sense, be only a single software system in existence, continually growing; it would be an ecology husbanded along by economists, sociologists, governments, clubs, communities, and herds of disciplines. New business models would be developed, perhaps at an alarming rate. How should we design our research to observe and understand this change? There is some evidence the change is underway, as the result of the adoption of open source by companies who are not merely appreciative receivers of gifts from the evangelizers of open source, but who are clever thieves re-purposing the ideas and making up new ones of their own.

Introduction

Sometimes something new happens at a scale that both researchers and practitioners are either unable or unwilling to observe. An example of this in recent memory has been the emergence of emergence as a field of study, in the form of complexity science. For centuries a sort of phenomenon that is now regard as possibly central to many scientific disciplines was simply not observed or was considered not worthy of serious thought.

Researchers in and practitioners of open source* are enamored of licensing, tools and their usage, community building, and how effective and efficient the open-source methodology is at producing software. However, something much larger is going on that could be changing the landscape of computing and not just adding some knowledge to the discipline of software engineering.

Over the last 10 years, companies have been contributing a stupendous amount of software to (let's call it) the open-source world. For example, Sun Microsystems recently computed that, using conventional means for assigning a monetary value to source code, it has contributed over $1 billion in code. IBM and possibly other large corporations are not far behind. Of particular interest is that Sun has made a decision to open-source all of its software, and it appears they are well on their way to doing that. At the same time, Sun is not placing all of its revenue expectations on their hardware: they expect to make money with their software.

Sun: A Case Study (Brief Overview)

Sun started in 1982 as a company based on open standards and commodities: BSD Unix, Motorola 68000 processors, and TCP/IP. In the late 1990s it began to experiment with open-source ideas and true open source: Jini (not true open source, but an interesting experiment in open-source concepts and practices combined with strategies for creating markets), Netbeans, Juxta, and OpenOffice were early experiments, followed by Glassfish, Grid Engine, OpenSparc, OpenSolaris, Open Media Commons, and most recently Java. Throw in Java.net and an interesting landscape emerges. Sun is clearly experimenting with the whole concept of the commons. OpenSparc is a hardware design that was licensed under an open-source license for the purpose of creating markets; Open Media Commons is primarily a DRM open-source project, but it is also looking at the question of what intellectual property rights means in the 21st century. Java.net is a sort of meta community aimed at creating markets

* I use this term for simplicity and to avoid politics.

around Java. Solaris and Java are considered Sun's software crown jewels.

Throughout this experimental era at Sun—which is still going on—there were emphases on governance and business models.

Sun is pushing four open-source-related business strategies:

- to increase volume by engaging software developers and lowering the barriers to adoption
- to share development with outside developers and established open-source projects for software required by Sun's software stacks
- to address growing markets whose governments or proclivities demand open source, such as Brazil, parts of the European Union, Russia, India, and China
- to disrupt locked-in markets by providing open-source alternatives

Sun makes an interesting set of observations about how the point has changed over time where monetization of software happens. In the 1970s, software was primarily part of a complete hardware package. People would buy a complete system—hardware and software. In many cases, hardware companies would provide the source code for their customers to customize—and nothing was considered unusual about this.

During the two decades from 1980 to 2000, hardware companies started to unbundle their software, and software companies sprang up to sell software to do all sorts of things, including operating systems. What these two periods had in common was that software was monetized at the point of acquisition. And it seemed at the time there was no choice: you wanted to use something, so you needed to buy it first.

With open source and the right business models, this can change, and that change started in the early 2000s. Open source is typically free to use—that is, no cost. However, there are auxiliary things companies and in some cases individuals are willing or eager to pay for: support and maintenance, subscription for timely updates and bug fixes, indemnification from liability, and patent protection. In these cases, monetization can occur when the final product is deployed. That is, in such cases it costs nothing to explore an idea for a product to the point of putting it completely together for sale or distribution. Then, if the producer wishes, one or several of these services can be purchased.

By delaying some of the costs of coming up with new products and possibly new companies, likely many more new ideas can be explored considered over the entire market. The barriers for experimentation are very low.

The full repertoire of business models Sun has identified are as follows:

- subscription (as described above) including indemnification and patent protection by extending a company's umbrella of intellectual property over parties who subscribe
- dual license, in which newer versions of the code are sold and older ones are open source
- stewardship, in which a standard is used to attract developers using the standard and to whom other products are services are sold
- embedded, in which the code is part of something else—usually hardware—that is sold
- consulting, in which a person's or company's expertise in particular source code base is sold as, typically, heads-down programming services
- hosting, in which services provided by open-source software is running on servers and access to the running services are sold or other revenue streams are attached to the running code (like advertisements)
- training and education—of the source base and also of open-source methodologies

Sun open-source theoreticians view these observations as implying a *virtuous cycle* in which by finding a place for added value in code in the commons, a company (or person) can create a monetization point without having to invest alone in a large code base, and thereby produce a product or service at lower overall cost.

What This Means for Software

Suppose that Sun is not an isolated situation and that companies and other organizations (including individuals) are preparing to alter their business and software development models to be based on the Sun-described virtuous cycle. How would the entire enterprise of producing software change and what would this mean for software engineering?

Let's paint the picture. The vast majority of software would be in the commons and available for use. Nothing much would be proprietary. There would be pressure from the customer base for there to be some unifications or simplifications. For example, why would there need to be multiple operating systems aside from the needs of different scales, real-time, and distributed systems (for example)? On the other side, finding new value might cause pressure on firms to fork source bases to create platforms or jumping off points for entire categories of new sources of value. How would this balance play out?

Because the barriers to entry to almost any endeavor would be so low, there will be many more players—including small firms, individuals—able to be factors in any business area. With more players there would be more opportunities for new ideas and innovations. How will these play out in the market? Will, perhaps, firms try to become repositories of intellectual property in order to offer the best indemnification? Will other entities like private universities or pure research labs become significant players because they can offer a potent portfolio of patents to use to protect their cli-

ents? Looking at large portfolios such as owned by IBM or Microsoft, it would seem that they would continue to dominate; however, in new or niche areas, small organizations or even individuals could hold the key patents.

Some obvious considerations immediately come up. What about licensing? At present large systems are put together from subsystems (to pick a term) licensed under different licenses. What is not permitted is to be able to mix pieces from differently licensed source bases. Will there be pressure to put all code under the same license or will the pressure be the other way—to create new licenses for specialized purposes?

What This Means for Software Engineering

Because few companies would "own" an entire system or application area, there could be some pressure on code bases to drift regarding APIs, protocols, data formats, etc. And if so, where would the countermanding pressure come from? Would standards bodies handle it, would governance structures like the Apache Foundation or the IETF be created? Or would firms spring up to define application or system structure as was done with the personal computer in the early 1980s. In that case, a set of *design rules* were set up by IBM stating what the components of a PC were and how they interacted. [1] This enabled markets to form around the different components and the nature of design in computer systems changed. Today this way of looking at design has spawned a new approach to software engineering problems: *economics-driven software engineering*.

Software and computing education would change because all the source code would be available for study (and even improvement as part of the teaching/learning process). In this way, developers would be better educated than they have ever been before.

Programming would become less a matter of cleverness and invention, and more a process of finding existing source code that's close and either adapting or adapting to it. Licensing would either help or hinder this.

With pressure lessened to build everything from scratch, it would be possible to construct larger and larger systems with achievable team sizes. This would bring out the issues and challenges associated with ultra-large-scale systems.† To quote from the call for position papers for a workshop on this topic [2, 3]:

> *In a nutshell, radical increases in scale and complexity will demand new technologies for and approaches to all aspects of system conception, definition, development, deployment, use, maintenance, evolution, and regulation. If the software systems that we focus on today are likened to buildings or individual infrastructure systems, then ULS systems are more akin to cities or networks of cities. Like cities, they will have complex individual nodes (akin to buildings and infrastructure systems), so we must continue to improve traditional technologies and methods; but they will also exhibit organization and require technology and approaches fundamentally different than those that are appropriate at the node level. The software elements of ULS systems present especially daunting challenges. Developing the required technologies and approaches in turn will require basic and applied research significantly different that that which we have pursued in the past. Enabling the development of ULS systems—and their software elements, in particular—will require new ideas drawing on many disciplines, including computer science and software engineering but also such disciplines as economics, city planning, and anthropology.*

The switch from proprietary to commons-based software would hasten the age of ultra-large-scale systems which will differ qualitatively because of their massive scale. If that happens, the inadequacies of our tools including programming methodologies and languages would be placed in high relief.

What This Means for Research

The habit of research in computing is to look deeply and narrowly at questions. In a sense, researchers loves puzzles. Gregory Treverton wrote this about puzzles versus mysteries in a paper on/for the intelligence community [4]:

> *Now, intelligence is in the information business, not just the secrets business, a sea-change for the profession. In the circumstances of the information age, it is time for the intelligence community to "split the franchise" between puzzles and mysteries. Puzzles have particular solutions, if only we had access to the necessary (secret) information. Puzzles were the intelligence community's stock-in-trade during the Cold War: how many missiles does the Soviet Union have? How accurate are they? What is Iraq's order of battle? The opposites of puzzles are "mysteries," questions that have no definitive answer even in principle. Will North Korea strike a new nuclear bargain? Will China's Communist Party cede domestic primacy? When and where will Al Qaida next attack? No one knows the answers to these*

† This is the topic of a workshop I'm leading on Tuesday at ICSE.

questions. The mystery can only be illuminated; it cannot be "solved."

Finding evidence of the sea-change from proprietary software to commons-based software in the commercial world is part of a mystery, not a puzzle, and so our traditional methods might not hold up well. But certainly studying the engineering methods open-source projects use will not illuminate the larger context—that context being how the entire enterprise of creating software changes when corporations change their business models to embrace the commons. The concerns of firms are not the same as the concerns of someone using a bug-tracking tool, editing code with Emacs, and automating a tricky part of the testing process. Moreover, because bottom-line concerns dominate sticking to certain ideals of engineering, for example, we are likely to see ideas we in the software engineering community have not thought of.

Here is a small example, again from the Sun case study. A Japanese automobile manufacturer contacted Sun's Open Source Group to learn about open-source. The group was responsible for the creation of the bulk of the company's applications. They claimed to not have a single coder in their direct employ, but outsourced—primarily to India. They were concerned that the Indian companies they were using were not as adept with interpreting the specs they were given as made financial sense for the car company. So the VP of the group was interested whether the Sun Open Source Group could help them figure out how to impose an open-source methodology (but not reality) on the Indian outsourcing companies so that the applications group could monitor progress, run the nightly builds, observe email and wiki-based communications, and etc, to both judge how the project was going and to correct it on the fly, perhaps by using open-source techniques.

Not a line of code would be released to the outside world; there would be no license. It would be simply a management tool. Researchers who would notice and report on such innovations and activities would come from a business school, or would be economists or perhaps anthropologists. Therefore what I see required is a broader view, a more interdisciplinary view—this is in concert with the conclusions reached by the authors of the ultra-large-scale systems report.

Another part of the sea change is that software researchers would be able to do real science on naturally occurring software, systems, frameworks, etc. For example, it would start to make sense to get a handle on how many times a piece of data is transcoded on its way from a database to a client screen somewhere, a number that could be very high particularly if the system doing the overall transmission were made of a number of separately developed frameworks. Today, gathering such information requires a special relationship with a corporation—a relationship that I suspect is quite rare.

Conclusions

One can wonder whether Sun's directions are predictive or iconoclastic. If the latter, then Sun is merely a curiosity; but if the former, it behooves those of us who straddle the research / practitioner boundary to figure out a sort of research program that will help us notice the changes in order to record and study them.

Author Bio

Richard P. Gabriel was until recently one of Sun's open-source experts and a member of its Open Source Group. In 1998 he accepted the assignment from Bill Joy, one of Sun's founders, to make Sun an open-source company. From 1998 until 2004 he did just that, culminating in the book *Innovation Happens Elsewhere* with his colleague Ron Goldman, the formation of the Open Source Group, and the announcement by Sun's CEO that Sun would open-source all of its software. After spending 2 years in Sun Labs doing early studies of ultra-large-scale software systems, he returned briefly to the Open Source Group before moving on.

References

[1] Baldwin, C. & Clark, K. *Design Rules: The Power of Modularity*. MIT Press, 1999.

[2] http://www.cs.virginia.edu/~sullivan/ULS1/

[3] The Software Engineering Institute (SEI), "The Software Challenge of the Future: Ultra-Large-Scale Systems," June 2006, http://www.sei.cmu.edu/uls/.

[4] Treverton, Gregory F. Reshaping Intelligence to Share with "Ourselves" Commentary No. 82, Canadian Security Intelligence Service, July 2003.