Epilogue: Open Source outside the Domain of Software

Clay Shirky

The unenviable burden of providing an epilogue to *Perspectives on Free and Open Source Software* is made a bit lighter by the obvious impossibility of easy summation. The breadth and excellence of the work contained here makes the most important point—the patterns implicit in the production of Open Source software are more broadly applicable than many of us believed even five years ago. Even Robert Glass, the most determined Open Source naysayer represented here, reluctantly concludes that "[T]here is no sign of the movement's collapse because it is impractical."

So the publication of this book is a marker—we have gotten to a point where we can now take at least the basic success of the Open Source method for granted. This is in itself a big step, since much of the early literature concerned whether it could work at all. Since even many of its critics now admit its practicality, one obvious set of questions is how to make it work better, so that code produced in this way is more useful, more easily integrated into existing systems, more user-friendly, more secure.

These are all critical questions, of course. There are many people working on them, and many thousands of programmers and millions of users whose lives will be affected for the better whenever there is improvement in those methods.

There is however a second and more abstract set of questions implicit in the themes of this book that may be of equal importance in the long term. Human intelligence relies on analogy (indeed, Douglas Hofstadter, a researcher into human cognition and the author of *Gödel, Escher, Bach: An Eternal Golden Braid* suggests that intelligence is the ability to analogize). Now that we have identified Open Source as a pattern, and armed with the analytical work appearing here and elsewhere, we can start asking ourselves where that pattern might be applied outside its original domain.

I first came to this question in a roundabout way, while I was researching a seemingly unrelated issue: why it is so hard for online groups to make decisions? The answer turns out to be multivariate, including, among other things, a lack of perceived time pressure for groups in asynchronous communication, a preference in online groups for conversation over action; a lack of constitutional structures that make users feel bound by their decisions, and a lack of the urgency and communal sensibility derived from face-to-face contact. There is much more work to be done on understanding both these issues and their resolution.

I noticed, though, in pursuing this question, that Open Source projects seemed to violate the thesis. Open Source projects often have far-flung members who are still able, despite the divisions of space and time, to make quite effective decisions that have real-world effects.

I assumed that it would be possible to simply document and emulate these patterns. After all, I thought, it can't be that Open Source projects are so different from other kinds of collaborative efforts, so I began looking at other efforts that styled themselves on Open Source, but weren't about creating code.

One of the key observations in Eric Raymond's seminal *The Cathedral and the Bazaar* (2001) was that the Internet changed the way software was written because it enabled many users to collaborate asynchronously and over great distance. Soon after that essay moved awareness of the Open Source pattern into the mainstream, we started to see experiments in applying that pattern to other endeavors where a distributed set of users was invited to contribute.

Outside software production, the discipline that has probably seen the largest number of these experiments is collaborative writing. The incredible cultural coalescence stimulated by *The Cathedral and the Bazaar* led to many announcements of Open Source textbooks, Open Source fiction, and other attempts to apply the pattern to any sort of writing, on the theory that writing code is a subset of writing, and of creative production generally.

Sadly, my initial optimism about simple application of Open Source methods to other endeavors turned out to be wildly overoptimistic. Efforts to create "Open Source" writing have been characterized mainly by failure. Many of the best-known experiments have gotten attention at launch, when the Open Source aspect served as a novelty, rather than at completion, where the test is whether readers enjoy the resulting work. (Compare the development of Apache or Linux, whose fame comes not from the method of their construction but from their resulting value.)

The first lesson from these experiments is that writing code is different in important ways from writing generally, and more broadly, that tools that support one kind of creativity do not necessarily translate directly to others. Merely announcing that a piece of writing is Open Source does little, because the incentives of having a piece of writing available for manipulation are different from the incentives of having a piece of useful code available.

A good piece of writing will typically be read only once, while good code will be reused endlessly. Good writing, at least of fiction, includes many surprises for the reader, while good code produces few surprises for the user. The ability to read code is much closer, as a skill, to the ability to write code than the ability to read well is to the ability to write well.

While every writer will tell you they write for themselves, this is more a statement of principle than an actual description of process—a piece of writing, whether a textbook or a novel, needs an audience to succeed. A programmer who claims to writes code for him or herself, on the other hand, is often telling the literal truth: "This tool is for me to use. Additional users are nice, but not necessary."

The list of differences goes on, and has turned out to be enough to upend most attempts at Open Source production of written material. Writing code is both a creative enterprise and a form of intellectual manufacturing. That second characteristic alone is enough to make writing code different from writing textbooks.

This is the flipside of Open Source software being written to scratch a developer's particular itch; Open Source methods work less well for the kinds of things that people wouldn't make for themselves. Things like GUIs, documentation, and usability testing are historical weaknesses in Open Source projects, and these weaknesses help explain why Open Source methods aren't applicable to creative works considered as a general problem. Even when these weaknesses are overcome, the solutions typically involve a level of organization, and sometimes of funding, that takes them out of the realm of casual production.

Open Source projects are special for several reasons. Members of the community can communicate their intentions in the relatively unambiguous language of code. The group as a whole can see the results of a proposed change in short cycles. Version control allows the group to reverse decisions, and to test both forks of a branching decision. And, perhaps most importantly, such groups have a nonhuman member of their community, the compiler, who has to be consulted but who can't be reasoned with—proposed changes to the code either compile or don't compile, and when compiled can be tested. This requirement provides a degree of visible arbitration absent from the problem of writing.

These advantages allow software developers to experience the future, or at least the short-term future, rather than merely trying to predict it. This ability in turn allows them to build a culture made on modeling multiple futures and selecting among them, rather than arguing over some theoretical "best" version.

Furthermore, the overall value built up in having a collection of files that can be compiled together into a single program creates significant value, value that is hard to preserve outside the social context of a group of programmers. Thus the code base itself creates value in compromise.

Where the general case of applying Open Source methods to other forms of writing has failed, though, there have been some key successes, and there is much to learn from the why and how of such projects. Particularly instructive in this regard is the Wikipedia project (http://wikipedia.org), which brings many of the advantages of modeling culture into a creative enterprise that does not rely on code.

The Wikipedia is an open encyclopedia hosted on a *wiki*, a collaborative Web site that allows anyone to create and link to new pages, and to edit existing pages. The site now hosts over 200,000 articles in various states of completion, and many of them are good enough as reference materials to be on the first page of a Google search for a particular topic.

There are a number of interesting particularities about the Wikipedia project. First, any given piece of writing is part of a larger whole—the cross-linked encyclopedia itself. Next, the wiki format provides a history of all previous edited versions. Every entry also provides a single spot of contention—there can't be two wikipedia entries for Islam or Microsoft, so alternate points of view have to be reflected without forking into multiple entries. Finally, both the individual entries and the project as a whole is tipped toward utility rather than literary value—since opposing sides of any ideological divide will delete or alter one another's work, only material that both sides can agree on survives.

As a reference work, the Wikipedia creates many of the same values of compromise created by a large code base, and the history mechanism works as a version control system for software does, as well as forming a defense against trivial vandalism (anyone whom comes in and deletes or defaces a Wikipedia entry will find their vandalism undone and the previous page restored within minutes).

Open Source methods can't be trivially applied to all areas of creative production, but as the Wikipedia shows, when a creative endeavor takes on some of the structural elements of software production, Open Source methods can create tremendous value.

This example suggests a possible reversal of the initial question. Instead of asking "How can we apply Open Source methods to the rest of the world?" we can ask "How much of the rest of the world be made to work like a software project?" This is, to me, the most interesting question, in part because it is the most open-ended. Open Source is not pixie dust, to be sprinkled at random, but if we concentrate on giving other sorts of work the characteristics of software production, Open Source methods are apt to be a much better fit.

A key element here is the introduction of a recipe, broadly conceived; which is to say a separation between the informational and actual aspects of production, exactly the separation that the split between source code and compilers or interpreters achieves. For example, there are two ways to get Anthony Bourdain's steak au poivre—go to Bourdain's restaurant, or get his recipe and make it yourself. The recipe is a way of decoupling Bourdain's expertise from Bourdain himself. Linus Torvalds's operating system works on the same principle—you don't need to know Torvalds to get Linux. So close is the analogy between software and recipes, in fact, that many introductory software texts use the recipe analogy to introduce the very idea of a program.

One surprise in the modern world is the degree to which production of all sorts is being recipe-ized. Musicians can now trade patches and plug-ins without sharing instruments or rehearsing together, and music lovers can trade playlists without trading songs. CAD/CAM programs and 3D printers allow users to alter and share models of objects without having to share the objects themselves. Eric von Hippel, who wrote the chapter in this book on user innovation networks, is elsewhere documenting the way these networks work outside the domain of software. He has found a number of places where the emergence of the recipe pattern is affecting everything from modeling kite sails in virtual wind tunnels to specifying fragrance design by formula.

Every time some pursuit or profession gets computerized, data begins to build up in digital form, and every time the computers holding that data are networked, that data can be traded, rated, and collated. The Open Source pattern, part collaborative creativity, part organizational style, and part manufacturing process, can take hold in these environments whenever users can read and contribute to the recipes on their own.

This way of working—making shared production for projects ranging from encyclopedia contributions to kite wing design take on the characteristics of software production—is one way to extend the benefits of Open Source to other endeavors. The work Creative Commons is doing is another. A Creative Commons license is a way of creating a legal framework around a document that increases communal rights, rather than decreasing them, as typical copyrights do.

This is an almost exact analogy to the use of the GPL and other Open Source licensing schemes, but with terms form-fit to writing text, rather than to code. The most commonly used Creative Commons license, for instance, allows licensed work to be excerpted but not altered, and requires attribution for its creator. These terms would be disastrous for software, but work well for many forms of writing, from articles and essays to stories and poems. As with the recipe-ization of production, the Creative Commons work has found a way to alter existing practices of creation to take advantage of the work of the Open Source movement.

Of all the themes and areas of inquiry represented in *Perspectives on Free and Open Source Software*, this is the one that I believe will have the greatest effect outside the domain of software production itself. Open Source methods can create tremendous value, but those methods are not pixie dust to be sprinkled on random processes. Instead of assuming that Open Source methods are broadly applicable to the rest of the world, we can instead assume that that they are narrowly applicable, but so valuable that it is worth transforming other kinds of work, in order to take advantage of the tools and techniques pioneered here. The nature and breadth of those transformations are going to be a big part of the next five years.