point

Up from Alchemy

Eric S. Raymond, *Open Source Initiative*

Open source is a vital step in moving software engineering from arcane art to science.

Three hundred years ago, alchemy became chemistry. Arcane art became science when its practitioners abandoned secrecy to embrace process transparency and peer review. Today, software engineering is undergoing a similar transition, moving from closed to open source development. Only when we complete this transition, adopting open source development as a normal practice, can software development assume its place as a mature engineering discipline.

Open source: A path to quality

Both evidence and theory confirm that open source delivers better reliability, lower costs, shorter development times, and a higher quality of code (including better security). The Fuzz papers' empirical results, showing a substantially lower error rate in open source Unix utilities relative to closed-source equivalents, are well known (www.cs.wisc.edu/~bart/fuzz/fuzz.html). Recently, Damien Challet and Yann Le Du developed a mathematical model of software development that provides a formal, generative theory to explain those results (http://arxiv.org/pdf/cond-mat/0306511).

But the truly powerful evidence for the superiority of open source development lives not in academia but the real world. The Linux operating system, the Apache Web server, and the rest of the Internet's open source infrastructure have become so pervasive in recent years that it's sometimes difficult to remember how preposterous their story would have seemed before 1995–96. Amateurs successfully challenging Microsoft's operating-system monopoly and actually wresting leading-edge markets away from them? A volunteer-run effort maintaining the server that runs 60 percent of the world's Web sites? Absurd, but true.

Increasingly, too, customers are demanding the assurance open source provides that software they rely on is controlled by them and can't be arbitrarily modified or withdrawn by a vendor decision that's not in the customers' interest. Open source restores balance to relationships in which vendors have had the whip hand. It prevents rent-seeking activities (meant to avoid competitive or market pressure) and encourages the growth of healthy and sustainable service-for-revenue exchanges.

This isn't to claim that open source development is a final solution to the software quality problem. As Fred Brooks famously observed, there are no silver bullets. When we learn better techniques for managing a given level of software complexity, we tend to pocket those gains by escalating complexity to the point where we are again just

Continued on page 90

counterpoint

Back to the User

David G. Messerschmitt, *University of California, Berkeley*

Open source fails organizational and nonprogrammer users.

Software offers greater value when it's a single unfragmented solution, it's widely used and technically sophisticated, and a great many programmers depend on it. For software meeting these criteria (such as operating systems or programming tools), open source software has had notable successes. But most software doesn't fit this mold. Consider software embedded in an airplane or telephone switch or nuclear plant, where programmers have no access to the facilities needed for testing and refinement; indeed, in situ debugging can be life-threatening. Open source doesn't apply here.

More generally, many applications have few programmers among their users (supply chain or customer relation management are examples). Programmers have little direct motivation to solve somebody else's problem, but financial incentives are a good substitute. Nonprogrammer users shouldn't be testers and debuggers; they need stable working solutions, not technical acumen.

Quality doesn't mean bug-free

Open source software takes us back to an ancient prebartering economic system where everything is placed in and extracted from a commons. This works in closed and trusting communities, but nonprogrammer users are the beggars roaming the commons, helplessly dependent on it without input or influence. In commercial software, users are appropriately put squarely in the driver's seat through competitive choice and as the origin of all wealth.

In applications, the emphasis is often on user satisfaction and coherence to processes and organizations (rather than specifications or technical sophistication), areas less appreciated by programmers. "Defects" are typically misunderstandings of user needs; identifying such defects and their fixes benefits little from access to source. Programmers often lack domain knowledge and sufficient empathy for naïve users, so testers, product managers, marketers, salespeople, and customer support play a crucial role in deeply understanding user needs and communicating them to programmers. Most open source software lacks usability from the perspective of naïve users.

Applications often have a diversity of uses and legitimate variations in needs that evolve over time. Efficiently capturing commonalities while accommodating variation and evolution is crucial to expanding the application base. Open source can't accommodate this gracefully: the argument

Continued on page 90

point

continued from page 88

at the edge of being able to cope. And there are a few unusual edge cases in which open source doesn't make economic sense.

But the overall trend is clear. Technically, closed source is sinking under the weight of escalating error rates. Economically, even large firms are finding they can't afford to hire enough in-house developers to meet the demands of today's huge project scales—such loads must be spread across a wider base. This is increasingly being done through both formal consortia such as the Apache Software Foundation and informal cooperative efforts regulated by open source licensing. Resource pooling across corporate boundaries is moving from an option to an essential cost-control tool.

Who wins? Who loses?

The last five years have dispelled a lot of the negative mythology that used to be associated with open source development. It used to be widely believed, for example, that the process was unsustainable at scales above garage-project level because open source software couldn't capture sufficient market returns to fund development. Extreme forms of this argument claimed open source spelled doomsday for the livelihood of programmers.

Indeed, some people will lose out in the change—mainly, software monopolists and speculative investors. Open source software businesses must be service-centered rather than product-centered; thus, as with other service businesses, scaling up returns just by pumping in capital isn't possible. There's no fast, lucrative exit for venture capitalists. Tomorrow's pure-play software house will probably look more like a legal or medical practice than like Microsoft or Oracle. Proportionally more salaries will be funded by companies for whom open source software is part of a bundled good (which, for example, is why companies like IBM and Hewlett-Packard are embracing the method).

But programmers' fears were born of economic ignorance. In fact, open source has probably raised average programming salaries for exactly the same reasons that the wages of automobile designers and mechanics go up when car prices drop. When a bundled good becomes less expensive and more rewarding, customers put more money, rather than less, into the most expensive bottleneck component.

Professional software engineers therefore have no reason to resist the trend toward open source. Indeed, they have every reason to embrace it, because one of its effects is that programmers—the people who do the work—get to take back control of their art from people who merely manage and market things. Perhaps the most subversive thing about open source development in the long term is the revelation that throwing the suits overboard actually *works*.

Eric Raymond is cofounder and president of the Open Source Initiative, an educational organization that builds bridges between the hacker community and business with the aim of spreading the open source development method. Contact him at esr@thyrsus.com or www.catb.org/~esr.

counterpoint

continued from page 89

that users can modify the code to their specific need places unreasonable demands on technical proficiency and results in forking, fragmentation, and a self-defeating, increasingly proprietary code base. The motivations attributed to individual open source programmers don't apply to user organizations, which won't and shouldn't invest in development without being rewarded with differentiation from their competitors; sharing source code fails this litmus test.

We can capture commonality while accommodating variation and evolution. Acquired components support variation through selection, configuration, and system evolution by component upgrade. Open interfaces allow piecemeal upgrades, and open application programming interfaces allow sanctioned context-specific extensions by users. User communities are establishing evolving open standards, a process emphasizing user community consensus rather than fun and challenge for programmers, and stability and interoperability rather than rapid technical innovation and change.

These techniques emphasize issues near and dear to users rather than to programmers; unlike open source, they are persistent in the face of funding vagaries and technology fads and are consistent with competing solutions. Their emphasis on overall planning and user community drivers are anathema to open source proponents, although they can be combined with open source (and commercial) software in various ways.

The cost of quality

Open source can yield high-quality code, but at what cost? Having hundreds or thousands of programmers pouring over the totality of source is hardly productive, although this isn't evident with volunteer labor. This is justifiable for some widely used infrastructure, but the programming workforce doesn't have the spare capacity to globalize this inefficiency.

Open source is incomplete as a software creation process. Somebody has to understand needs; design an architecture accommodating long-term evolution; create a working prototype as a starting point; provide a talented, centralized person or group to coordinate and govern; and provide support, training, and certification. These critical tasks hardly differ between open source and traditional processes. If programmers do all the coding and debugging in their spare time, what will they do on their paying jobs? Probably these leftover tasks, which many will find less interesting and challenging than what they do now.

Commercial vendors make extra margins on their successful software products (given practically zero manufacturing and distribution costs) and use them to finance losers in a product portfolio and to finance research and risky sunk investments in new products. In an all open source industry, products would be replaced by marginally profitable services, and research and risk-taking would be gravely damaged.

David G. Messerschmitt is the Roger A. Strauch Professor of electrical engineering and computer sciences at the University of California, Berkeley, and coauthor of *Software Ecosystem: Understanding Technology and Industry* (MIT Press, 2003). Contact him at messer@eecs.berkeley.edu.

Eric Responds

Most of the objections in David's essay can be met with the simple observation that nothing in open source development stops users from paying programmers to get slick user interfaces or anything else they want. His image of beggars roaming the commons is touching but mistaken; these "beggars" have fistfuls of florins to wave around, and programmers will be no slower than other kinds of skilled artisan to notice this.

When users need to be in the driver's seat, they can buy that privilege quite directly by employing programmers to improve the open source base on their behalf. The only option open source forecloses is that of cornering the results for exclusive resale. Most users, most of the time, don't care—they need software to solve business problems, not to be in the software business themselves.

So, if you want "coherence to process and organization," pay for it. Open source doesn't change this, it just peels away overhead—all your money buys programmer time rather than corner offices for some other firm's executives. Because the programmers will spend less of their time implementing checklist features for someone else's requirements, they can spend more of it improving and innovating with respect to yours.

3M discovered years ago that the central problem of innovation is not eliciting it but how not to smother it under bureaucracy, turf wars, and process overhead. The fluidity, low cost of experiments, and peer review process in open source give it a huge and sustainable edge in fostering innovation.

David Responds

Alchemy? If selling proprietary products in a competitive market is the criterion, then all mature industries practice alchemy. Eric confuses commerce with scholarship.

Eric's examples, which fall within the scope of my first sentence, do suffer winner-take-all effects, and thus open source is an antidote to monopolistic behavior. It has major strengths and weaknesses, but the market (not the profession) will be the ultimate determinant, and success won't be uniform across all types of software. Open source (and its cousins such as open-access publication) aren't subversive; they represent healthy competition among methodologies. But why such an invective against venture capitalists (a.k.a. speculators)? VCs are a needed antidote to stodginess and manipulation and encourage unfettered innovation while actually paying programmers well.

Eric says that given unlimited resources and absent pesky scheduling or stakeholder constraints, we can create defect-free software. Without annoying users or managers injecting themselves, programmers can have satisfying careers. This is credible, but managers are necessary to allocate limited resources, arbitrate among stakeholders, and meet aggressive schedules. Actively incorporating users (and their representatives) dramatically increases impact.

Other approaches have many of the same advantages touted for open source. Brad Cox mirrors economists in arguing eloquently that a market is the ultimate complexity tamer. Buying components from suppliers achieves resource sharing across companies without requiring technical acumen and without encouraging free-riders.

New methodologies such as open source, component software, superdistribution, and Extreme Programming are welcome. Open source is surprising and successful and works well for some purposes. It will play a significant role, but one more limited than Eric asserts.