

Sharing Source Code with Clients: A Hybrid Business and Development Model

Mikko Rieppula, Aalto University

// A practical hybrid licensing model helps both business-to-business software vendors in vertical domains and consultancies that maintain separate quasi-products to benefit from interaction with select clients. //



USER INNOVATION AND customer-centric development are hot topics in software and almost all service sectors, manifested by growing interest in crowdsourcing and user-contribution systems.¹ However, success stories of companies that have switched from a traditional closed-source to an open source software (OSS) business remain scarce.

Access to source code will likely increase—especially in commoditized software markets—but without the software being released as OSS. Companies are becoming more comfortable working with inbound OSS and adopting OSS-like processes. For certain

types of products and vendors, *client-shared source*—a hybrid between the traditional and OSS business logic—can present a new opportunity.

Consider a business-to-business (B2B) software vendor whose clients are willing and able to modify source code. The client-shared-source model is particularly applicable when the vendor must customize or extend the product for many clients—for example, in different regions or industry sectors. In this situation, the customizations or extensions are many and varied, yet potentially reusable by other clients not in direct competition.

The client-shared-source model is a potential development and business model with practical implications for software architects, developers, and especially development managers. It's not yet common, but I predict its emergence on the basis of recent OSS and innovation research as well as my own research and experience in telecommunications software. This model can be implemented by vendors of less differentiated products and by consultancies and integrators doing similar projects for different clients, the results of which have been traditionally maintained as separate quasi-products.

In the client-shared-source model, the vendor lets clients access the source code—and more—through a common platform. Scott Cook uses the term *user contribution system* (UCS).¹ Satish Nambisan, Priya Nambisan, and Robert Baron urge us to profit from *virtual customer environments* (VCEs) in product idea generation, development, testing, and support in almost any business domain, not just software.^{2,3} Both approaches aim to engage clients in product or service development and delivery. We can regard the client-shared-source model, or its supporting infrastructure, as a case of implementing a software-specific UCS or VCE. However, giving clients access to source code has fundamental implications for a firm's business logic and how it should design, develop, and manage its software.

The Changing Landscape

In the early stages of any high-tech industry, market success depends on product innovation based on proprietary technology. Most software vendors share this background. However, once a dominant design emerges, commoditization begins. To remain competitive, companies must innovate

with marketing, efficient operations, after-sales support, and so on. The focus moves from product innovation to business innovation. This is where many software vendors stand today. For them, this has typically meant services around the core product.⁴ However, it increasingly means new types of licensing and delivery methods—for example, software as a service (SaaS), cloud computing, or even OSS—that is, new business models. Client-shared source can be such a differentiating aspect for some vendors.

Open Source Is Here to Stay, but Not Everywhere

Today, the question is no longer whether OSS can survive but mainly what kind of software might become available as OSS and under which kind of license. We haven't seen OSS much in industry verticals—that is, in low-volume, complex, domain-specific applications.^{5,6}

Various OSS business models have emerged. The usual model is based on services—for example, support and maintenance, or customizations. Another is *dual licensing*, in which the same software is simultaneously available both for free under a viral license such as the General Public License and for a fee under a nonviral license that probably requires assigning rights to any incremental developments back to the vendor.

User innovation is a distinct benefit of OSS to both the original contributor (vendor) and the user. Knowledgeable, motivated “lead users” can often best determine how to improve a product and, sometimes, even how to modify the source code.⁷ Because they aren't in a position to market their own small incremental developments, they're happy to contribute them back. Also, companies participating in OSS projects often want the community to maintain their incremental developments so that future versions remain compatible with

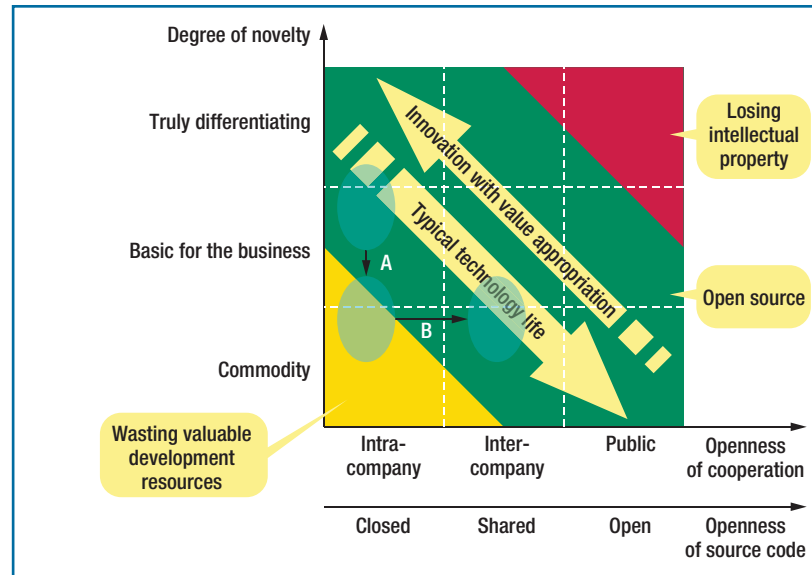


FIGURE 1. The shift from differentiating software to commodity over time, with the two corners to avoid. The x-axis represents the level of openness in cooperation; the y-axis indicates the software's degree of novelty (adapted from the work of Frank van der Linden and his colleagues⁹).

their needs. Such companies are driven by commercial self-interest as opposed to purely intrinsic motivation.⁸

Most Software Is Fast Becoming a Commodity

A good recipe for business failure is to market a product while remaining unaware of how other solutions address the need. Development managers should realize that the most cost-effective way to address their clientele's needs is often with someone else's software.⁹

Figure 1 demonstrates how the typical technology life cycle results in commoditized software. The development of already commoditized features is a waste of resources (yellow). By definition, such functionality is cheaper to obtain through licensing or might even be available as OSS. Releasing differentiating software too openly is asset leakage (red). The x-axis can be regarded as a question of code ownership and source code availability, not just as a question of technology sourcing.

Traditionally, software development has fallen in the left column. Too many software businesses have unwillingly descended into the yellow zone

(transition *a*) and entered fierce price competition.

When lacking product innovation capability, a software vendor should examine how the other two columns relate to its business. Instead of discussing pure OSS (the right column), I focus on involving your clients (the center column, transition *b*) as a way to stay in the green zone in the present era, in which many companies are comfortable adopting OSS-like practices and collaboration tools.

Client-Shared Source

The term *client-shared source* refers to this arrangement:¹⁰

- Access to source code becomes part of the business arrangement at a price that's typically higher than that of a runtime license. Thus, the client pays for the right to have the source code and participate in a restricted client/developer community. The vendor continues to sell regular runtime licenses.
- Suppliers (employees and outsourcing partners) provide their work as nonclient participants in the

community, using OSS-like processes and tools.

- The software vendor requires, or at least expects, participants to assign copyright to all incremental developments back to the vendor, as in OSS dual licensing.
- The participants can't further license the source code to third parties.

The license agreement might be limited to the client's internal use, or it can even allow granting runtime licenses to third parties, whether under the original vendor's or client's (reseller's) brand. So, resellers and integrators can similarly act as participants. The vendor should also use the same technical platform for sharing much of the documentation from requirements analysis all the way to testing.

The vendor might even pay a motivated client who substantially contributes to product development. However, for access to the platform, vendors

to contribute, lower maintenance and support costs, and so on. Such licensing can instill the client's trust in the vendor¹¹ and naturally eliminates the need to negotiate separate, potentially costly, source code escrow arrangements. However, don't expect clients to pay high premiums until you have a working community!

You can extend the basic model to cover, for example, joint requirements analysis and testing, which would be good examples of customer-supplier relational processes¹² or a case of "an overarching VCE."³ Empirical research has also found that interactions by and between clients in value co-creation can be an important source of value and shape their future participation in such co-creation.³

The closest equivalents to client-shared source in current industry practice are the development licenses software vendors grant to their close partners—for example, integrators. Unfortunately, these usually don't capi-

By opening up your bug- and issue-tracking tools to clients, you further instill trust in the client relationship.

should charge clients a positive fee, whether fixed, recurrent, or proportional to the client's use of the product. The price can no longer be tied to a specific software version. The vendor should then award any particular client's contributions on a per-task or results basis, just as it would reward subcontractors, if at all.

So, clients will support the software vendor, which can still extract more license revenue owing to the higher value associated with a source code offering. Too good to be true? The higher value to the client is explained by user innovation and the participants' self-interest

talize on concurrent distributed development or user innovation and involvement the same way client-shared source does.

Finally, I must mention Microsoft's "shared source" initiative. The reasons for Microsoft's offering their "shared source" differ considerably from the above logic as Jason Matu-sow outlines.¹¹ We shouldn't generalize on the basis of any example set by such a dominant vendor; few Microsoft clients could ever expect to get their own incremental developments incorporated into subsequent product versions.

Challenges and Limitations

Client-shared source can only be envisaged in a B2B setting in which the clients are willing to work with the source code, either with in-house or outsourced development resources. Clients might have vested interests in, for example, critical operational support systems over which they need great control or resale opportunities. In both cases, the clients benefit from direct access to the source code and the closed development community.

Not all the motivational factors that have led to successful OSS projects are automatically present in the client-shared-source setting. Clients might well expect the original vendor to provide the mundane but necessary support tasks. Research in a similar context revealed that the primary motivations for contributing code were reciprocity (the obligation or desire to conform to the community's norms) and future improvements (to retain compatibility and benefit from expert discussions).¹³ Less surprisingly, participants were clearly driven by need as opposed to fun and enjoyment, as is often the case in OSS projects.

In the beginning, clients that aren't used to working with OSS projects can be expected to jealously hide their new developments against their own best interests. Dialogue and facilitation are recommended.

The software vendor must take an active role in quality assurance and resolving conflicts among participants' incompatible developments. There's a definite need to find the right balance in how the vendor handles the community because the participants' sense of fairness can supersede their economic self-interest.^{3,14}

Implications for Developers and Managers

The client-shared-source model poses additional requirements for internal developers and development managers.

Managers should be happy to see these being taken into account, considering the promise of benefiting from external resources in exchange.

Software developed in a shared-source community must exhibit a fairly clean, modular structure if paying clients are to further develop it—for example, by writing add-on modules or localized implementations. Clear APIs are nice, but when sharing source code, you don't necessarily need to implement them as runtime or linked libraries. The most important thing is to document the interface design, which, for performance and convenience, can still be implemented in compile-time code.

Designs based on widely adopted programming paradigms stand the best chance of success. If your company has developed an elaborate programming framework with a steep learning curve, consider whether your clients will make the effort to learn it. The more clients you expect to attract, the simpler it must be and the more universal the required tools should be.

Revision control and configuration management are cornerstones of any well-working collaborative environment; by now, the industry has learned how to master them even in distributed environments. Tool support exists, whether based on de facto tools such as svn or others, and many tool vendors offer complete platforms for collaborative development—for example, CollabNet.

However, some subtle points can be of major importance. The style of the documentation, including code comments, might need cleansing to avoid revealing business secrets, offending particular clients, containing strong language or any “self-incriminating” text, or inadvertently infringing on any third parties’ (secrecy) rights or interests. Managers should make a high-level code review to ensure that what’s being sold as source code is actually the

vendor’s property. If parts aren’t, the company must procure or negotiate a license that lets them be shared in the source code, if necessary.

Finally, by coordinating testing among clients, you can better benefit from their help in all areas of testing. Develop testing plans that clients can execute on those pieces of software that

zaar. Here are three options.

The first is to incorporate the changes into your main branch of development; your clients have a clear interest in giving you the source code to maintain so that they don’t lose the respective functionality in subsequent upgrades. Because you gave them access and incentives, they can even do this

After all, what your company is
(or should be) selling
is a transparent relationship.

they’re most interested in using. Establish policies as to what level of testing you expect them to perform before they make their incremental developments visible to the rest of the community. If you can centrally manage the information on your clients’ testing, you can direct your own testing into areas they haven’t covered, while winning time with your limited testing resources. By opening up your bug- and issue-tracking tools to clients (while managing their rights at the appropriate level), you help them fix issues critical to them (even if those issues were less critical to you and others) and further instill trust in the client relationship. After all, what your company is (or should be¹²) selling isn’t just the product or the source code but a transparent relationship in which clients have a concrete say in your product roadmap.

By now, you’re probably worried about your ability to manage the software as it evolves with different clients. In business terms, if your clients have the need and ability to customize the software, then having multiple variants out there satisfying different needs is fundamentally a good thing! As software engineers, we obviously prefer a common codebase, whether it’s a vendor-controlled one or an OSS-like ba-

mostly by themselves. You just need to decide how much control you want to retain over such code commits. At minimum, you should oversee who’s allowed to do what in the meritocracy of your own developer community.

The second option is to conditionally include client-specific features. If you suspect not everyone would be thrilled about the changed functionality, let your clients choose their preferred way, be it on a separate user-settings page, in a configuration file, or even by compile-time directives. Even mutually exclusive features can be reconciled into a common codebase—the question is just at what level.

The last option is to isolate client-specific changes from main development. Occasionally, you might come across a quick-and-dirty workaround you’d rather not see in your code. First, think twice—has the product found an unanticipated application area? If not, isolate it, but help your client reapply their local patches systematically whenever they upgrade the software.

The point in setting up a shared development environment is, of course, to promote the first two options over the third. When you decide to market a certain baseline as a new version, don’t

forget your existing shared-source clients who already have access to the latest code but might have deferred deployment of newer versions precisely because they have custom developments they want to protect. By convincing them to upgrade, you might not get much immediate cash from them. How-

vendors and clients.¹² They challenge the predominant view of a solution as a customized, integrated combination of goods and services for meeting a customer's business needs, and they demonstrate how suppliers' and customers' views of solutions typically differ. According to their empirical study,

In business terms, having multiple variants satisfying different needs is fundamentally a good thing!

ever, you have the ideal opportunity to exercise the first two options for both your and their benefit: your product gets the new functionality, and they get compatible upgrades.

Getting Started

In the beginning, the vendor must alleviate its new shared-source clients' concerns and provide them with a gradual transition until the offering is solid and self-evident. Some clients might regard the license terms as fair up front. Others might see it as an interesting partnership opportunity but need to be offered small perks as they cling to the old mindset: "If I give you something, you should give me something in exchange." Carefully plan how the upfront premium for getting the source code, the associated limitations, the yearly membership fee, and so on will evolve over time and number of participants. To ease bootstrapping, you might want to entice the first customers by not charging extra license fees, just to show you have a community. After all, your offering's value grows with the number of active participants in your community.

Discussion

Kapil Tuli and his colleagues have examined what "a solution" means to

customers tend to view a solution more broadly as a set of customer-supplier relational processes.

In Tuli's and his colleagues' research, "a supplier had noted the importance of flexible source-code software as an enabler of effective solutions."¹² Client-shared source shouldn't be viewed as an end solution to the problem of declining profits or resource-consuming maintenance but as an enabler of and means for truly client-serving solutions and lasting client relationships.

Client-shared source could also be a way for IT consultancies to capitalize on their client-specific projects and quasi-products. The same market change factors are at play for service houses as for software product vendors. Although IT systems are critical, they're seldom differentiating in the banking or telecommunications sectors, for example. Service house clients no longer have a strong interest in requiring strict confidentiality and sole ownership of their custom information systems when the systems in question aren't strategically differentiated anymore. The clients' investments are lower and better secured if the systems aren't custom ones but are maintained (even if in low volumes) as products whose development and maintenance costs are shared across multiple par-

ties. This also allows for more freedom in choosing support and maintenance partners. The service house might lose in short-term turnover but win in development costs and long-term competitiveness.

In vertical domains, business requirements are more complex and demand more specialized knowledge. Compared to general-purpose, horizontal OSS, much less OSS has emerged in and for vertical domains.^{5,6} This might imply that we're going to see more client-shared source and other hybrid models still leaning toward the traditional, closed value appropriation models than OSS, particularly in vertical domains. Certain types of businesses also exhibit what researchers call *network externalities*: the more users, the more value to each user. Technically compatible companies can benefit from such network effects; an alliance formed around the same software is the most compatible of all.

Despite the recent interest in purely free open models, user innovation and participation are not a priori incompatible with premium licensing schemes. Market pressures might well make the client-shared-source model more popular among vendors whose commodity products are suitable for such collaborative development and even among consultancies, who could thus better leverage their quasi-products. Programming is no longer such a rare or expensive skill that someone with a specification and the willingness couldn't devise a software product similar to (or better than) yours. So, don't be so jealous about your source code! There's probably much more value in your client relationships (just as there is in knowing your clients' specific requirements), and client-shared source is one way to nurture them.

I've tried to assess the client-shared-

source model's applicability in terms of product type and company and market attributes. The challenges aren't insurmountable to many innovative companies. Most (successful) companies shouldn't open up their source code even as widely as suggested by client-shared source. Client-shared source is a mid-tier solution for the "me too" companies, particularly in vertical domains, who don't have or foresee significant differentiators in their core product. It will enable them to survive and develop new profitable business by incremental improvements. Client-shared source certainly doesn't represent such a radical leap of faith as a transition to a pure OSS model would require. At the other extreme, given enough commoditization, one of the pure OSS business models might be a better bet even if the chance of success is slim—but that's what commoditization in the extreme implies. Besides, nothing prevents doing that later, so if you've been contemplating releasing something as OSS, you should probably first try client-shared source. 

Acknowledgments

I thank all my colleagues and interviewees for sharing their experiences and opinions along the way. Thanks also to Matti Rossi and Juho Lindman for the encouragement that led to this article.

References

1. S. Cook, "The Contribution Revolution: Letting Volunteers Build Your Business," *Harvard Business Rev.*, Oct 2008, pp. 60–69.
2. S. Nambisan and P. Nambisan, "How to Profit from a Better 'Virtual Customer Environment,'" *MIT Sloan Management Rev.*, vol. 49, no. 3, 2008, pp. 53–61.
3. S. Nambisan and R.A. Baron, "Interactions in Virtual Customer Environments: Implications for Product Support and Customer Relationship Management," *J. Interactive Marketing*, vol. 21, no. 2, 2007, pp. 42–62.
4. M. Cusumano, *The Business of Software: How to Survive and Thrive in Good Times and Bad*, Free Press, 2004.
5. B. Fitzgerald, "The Transformation of Open Source Software," *MIS Q.*, vol. 30, no. 3, 2006, pp. 587–598.
6. L. Frank, P. Tyrvaäinen, and E. Luoma, "Future Issues in a Software Market: A Delphi Study

- on the Telecommunications Industry," *Proc. 8th Conf. Telecom, Internet & Media Technologies*, IEEE CS Press, 2009, pp. 1–7.
7. E. von Hippel, "Democratizing Innovation: The Evolving Phenomenon of User Innovation," *J. für Betriebswirtschaft*, vol. 55, no. 1, 2005, pp. 63–78.
8. K. Ven and H. Mannaert, "Challenges and Strategies in the Use of Open Source Software by Independent Software Vendors," *Information and Software Tech.*, vol. 50, nos. 9–10, 2008, pp. 991–1002.
9. F. van der Linden, B. Lundell, and P. Marttiin, "Commodification of Industrial Software: A Case for Open Source," *IEEE Software*, vol. 26, no. 4, 2009, pp. 77–83; doi:10.1109/MS.2009.88.
10. M. Riepula, "A Licensing and Business Model for Sharing Source Code with Clients—Leveraging Open Client Innovation in the Proprietary World," *Software Business*, Lecture Notes in Business Information Processing 51, Springer, 2010, pp. 13–25.
11. J. Matusow, "The Shared Source Initiative:

ABOUT THE AUTHOR



MIKKO RIEPULA is a researcher and PhD student at the Aalto University School of Economics. His interests include software business and licensing in general, open source software, open innovation, and client-driven development. Riepula has an MSc in information technology and computer science from Helsinki University of Technology. Contact him at mikko.riepula@aalto.fi.

The Microsoft Perspective," *Perspectives on Free and Open Source Software*, J. Feller et al., eds., MIT Press, 2005, pp. 329–346.

12. K.R. Tuli, A.K. Kohli, and S.G. Bharadwaj, "Rethinking Customer Solutions: From Product Bundles to Relational Processes," *J. Marketing*, vol. 71, no. 3, 2007, pp. 1–17.
13. S.K. Shah, "Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development," *Management Science*, vol. 52, no. 7, 2006, pp. 1000–1014.
14. W. Güth, R. Schmittberger, and B. Schwartz, "An Experimental Analysis of Ultimatum Bargaining," *J. Economic Behavior & Organization*, vol. 3, no. 4, 1982, pp. 367–388.



Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.

ADVERTISER INFORMATION • JULY/AUGUST 2011

Advertisers

ICSE 2012

Seapine Software

Cover 2

Cover 4

Advertising Personnel

Marian Anderson: Sr. Advertising Coordinator
Email: manderson@computer.org
Phone: +1 714 821 8380 | Fax: +1 714 821 4010

Sandy Brown: Sr. Business Development Mgr.
Email: sbrown@computer.org
Phone: +1 714 821 8380 | Fax: +1 714 821 4010

IEEE Computer Society
10662 Los Vaqueros Circle
Los Alamitos, CA 90720 USA
www.computer.org

Advertising Sales Representatives

(display)

Western US/Pacific/Far East: Eric Kincaid
Email: e.kincaid@computer.org
Phone: +1 214 673 3742
Fax: +1 888 886 8599

Eastern US/Europe/Middle East: Ann & David Schissler
Email: a.schissler@computer.org, d.schissler@computer.org
Phone: +1 508 394 4026
Fax: +1 508 394 4926

Advertising Sales Representatives

(Classified Line/Jobs Board)

Greg Barbash
Email: g.barbash@computer.org
Phone: +1 914 944 0940

This article was featured in

computing|now

ACCESS | DISCOVER | ENGAGE

For access to more content from the IEEE Computer Society,
see computingnow.computer.org.



IEEE  computer society

Top articles, podcasts, and more.



computingnow.computer.org