Susan Elliott Sim
ses@intdexwis.com

Research Statement

The software that we use on our computers, mobile phones, and other devices is created programmers writing source code in a logical programming language. A modern luxury car has approximately 80 million lines of code—far too much for any individual to read in a lifetime, but nevertheless needs to be evolved by adding new features and taking out bugs. Any software system is a complex collection of features that has been negotiated by engineers, designers, and managers. The software that we use is created through a complex dance between what is technically feasible, job responsibilities, political agendas, and individual personalities. Probing the relationships between people, roles, and technology helps to understand both the limits of software and how they are mutually constituted [11].

Problem

Historically, for-profit companies have developed software using a "closed source" model. In other words, the source code for the software is not distributed along with the compiled executable program. It was believed that this approach would protect intellectual property. Within the last fifteen years, "open source" software has emerged as an important phenomenon for both culture and innovation. In the simplest terms, open source distributes the source code along with the compiled executable, so that users could modify the program to suit their own purposes. This distribution and licensing model has challenged notions of ownership, control, and software process. Open source is here and changing how software is developed, but not all organizations are able make use of it due to intellectual property issues. There is a spectrum of corporate responses ranging from enthusiastic embrace of open source to strictures against the adoption. Nevertheless, firms need to understand how to make best use of it, so they can leverage a global supply chain for software components and the know-how necessary to implement new product and service ideas to keep up with societal, economic, and corporate demands.

My current work looks at the middle ground between the extremes between "free and open" and "proprietary and closed" models of software development. Most individuals and organizations don't have the ideological luxury of choosing one of these poles, and instead must work with the constraints and resources available. This middle ground, which I call "hybrid source," is where the bulk of professional software developers work. They often use source code from the web, but they don't usually share their software assemblages with the world. I am studying two aspects of hybrid source: a) the source code that is being reused; and b) the remix culture in open and hybrid source.

Methods

Before delving into the research results, I need to describe my research methods. I use an interdisciplinary approach in my work that is driven by empirical data, motivated by theory informed by an understanding of the technology. Let me unpack each of these terms.

Driven by Empirical Data. When studying socio-technical systems, it is important to look closely at what is actually there, rather than making assumptions about what should be there. This means collecting data about the participants, artifacts, and interactions [6, 7]. I draw on qualitative and quantitative data collection and analysis techniques primarily from the social sciences, such as sociology, anthropology, and psychology.

Motivated by Theory. The data is collected not for its own sake, but to help us understand, that is, to build explanatory theories about a phenomenon or relations. I draw on the humanities for both theories to direct research and critiques that can be used in the construction of new analyses. STS has special insight into technology within a social context over the broad sweep of history. Although, computer technology

is on the leading edge of innovation now, we as humans have been here before; technologies such as books, libraries, and movable type have led to advances in knowledge and new kinds of sharing.

Informed by Technology. Many studies of technology tend to treat the object of study as a black box; they don't necessarily understand the internals and are satisfied with studying the production and impact of the technology. In other words, one widget, whether it is a can opener or a computer program, is pretty much the same as another. I draw on my background in computer science to provide a deep understanding of the artifacts and processes involved in the production of software.

These principles provide distinct perspectives on hybrid source. This mutual interrogation of technology and social systems provides an integrated (and sometimes conflicting) view of a phenomenon. However, this intellectual breadth leads to a depth of understanding that cannot be obtained any other way, which leads to insights for future action.

Source Code

In my study of hybrid source software development, we investigated precisely what source code was being reused. By some estimates, there are billions of lines of source code on the web in open source projects, example repositories, tutorial sites, and blogs. When a programmer takes source code from the web, they could be taking entire components or just a few lines (a "snippet"). My pioneering work in this area [15, 16] has led to an NSF CAREER grant. We have found that open source projects tend to be used like building supplies, whereas snippets tend to be used as know-how [13, 14]. Although, software has not been built from scratch for decades, this opportunistic reuse tends to violate traditional component boundaries and ignores conventional wisdom that advises against modifying the source code for a component.

Snippets are examples that encapsulate a small nugget of knowledge about how to complete a small programming task. An important characteristic of snippets is they take abstract information and make it concrete for a particular use context. Consequently, snippets are a form of executable know-how that travels well. This stands in stark contrast to textbooks and manuals that take specific knowledge and generalize it in the form of abstract principles. While this abstraction is admired in computer science, the information is harder to use because it needs to be first re-contextualized by the reader, which leads to greater cognitive load.

Remix Culture

Key to the success of open source is its "remix culture," the set of ethics and norms that places the collective rather than the individual as central to the production of creative works. In open source, there is no single owner or designer who can be given economic and legal credit for the intellectual property of a project. Central practices of remix culture are appropriation of available resources, recombination of resources into a new creative work, attribution of the appropriated resources, and circulation of the new creative work.

Although open source software is being taken up by professionals and students alike, we have found that remix culture has not been adopted to the same degree. While they regularly appropriate and recombine source code during software development, they less frequently circulate their own software assemblages for criticism and reuse by the community. There is an awareness of issues around attribution, but a lack of mechanisms for footnoting or citing sources. Even in settings where there is a culture that promotes knowledge sharing, collaboration, non-market motives for creative work, such as BarCamps and universities, source code assemblages are rarely shared. In a review of 40 software tools to support

opportunistic programming, most (34) support appropriation, 9 help with recombination, 25 have some features for attribution, and only 8 support circulation. None provide support for all four practices.

In future work, we will take this research in two directions. One, what are the reasons underlying the mismatch between the adoption of open source and the adoption of remix culture? Two, what new models of authorship and software tools are needed, so that software assemblages can be attributed properly [4]? This work will have implications for law, economics, and software ethics.

Other Research

There are other projects where I used a similar approach, i.e. combining computer science subject matter expertise with an STS sensibility and empirical data to arrive at an integrated, holistic understanding of the phenomenon. These projects include studies of benchmarking for software tools [12], agile software development [2, 3, 5, 9], and expertise in software development [8, 10].

In my dissertation research, I looked at the role of benchmarking in advancing scientific research. This work involved a deep understanding of both STS and computer technology. I found that there was a tight relationship between a benchmark and the scientific paradigm of a discipline that is responsible for the leap forward [12]. A Kuhnian scientific paradigm is the dominant view of a science consisting of explicit technical facts and implicit rules of conduct. A benchmark operationalizes a scientific paradigm; it takes an abstract concept and turns it into a guide for action. In effect, a benchmark is a statement of the discipline's research goals and it emerges through a synergistic process of technical knowledge and social consensus proceeding in tandem.

References

[1] Carey, K., "Decoding the Value of Computer Science," in *The Chronicle of Higher Education*, 2010.

[2] Cohn, M.L., Sim, S.E., and Dourish, P., "Design Methods as Discourse on Method," in *Proceedings of the 2010 International Conference on Supporting Group Work (GROUP10)*, Sanibel Island, FL, 2010.

[3] Cohn, M.L., Sim, S.E., and Lee, C.P., "What Counts as Software Process? Negotiating the Boundary of Software Work through Artifacts and Conversation," *Computer Supported Cooperative Work: The Journal of Collaborative Computing,* 2009.

[4] Cohn, M.L., Sim, S.E., and Philip, K., "Technical Authorship: Refiguring the Designer-User Conflict and the Visioning of Collective Technical Futures," in *Third Annual iConference* UCLA, Los Angeles, CA, 2008.

[5] Gallardo-Valencia, R.E. and Sim, S.E., "Agile Validation Is Continuous and Collaborative: A Field Study of Agile Requirements Knowledge," in *Second International Workshop on Managing Requirements Knowledge (MaRK'09)*, Atlanta, GA, 2009.

[6] Lethbridge, T.C., Sim, S.E., and Singer, J., "Studying Software Engineers: Data Collection Techniques for Software Field Studies," *Empirical Software Engineering: An International Journal,* vol. 10, pp. 311 - 341, July 2005.

[7] Patterson, D.J., Sim, S.E., and Aiyelokun, O., "Overcoming Blind Spots in Interaction Design: Lessons from Designing for African Aids Orphan Care Communities," *Information Technologies and International Development,* 2009.

[8] Ratanotayanon, S. and Sim, S.E., "When Programmers Don't Ask," in *Second International Workshop on Supporting Knowledge Collaboration in Software Development*, Tokyo, Japan, 2006.

[9] Ratanotayanon, S. and Sim, S.E., "Supporting Program Comprehension in Agile with Links to User Stories," in *Agile 2009*, Chicago, IL, pp. 26-32, 2009.

[10] Sim, S.E., Alspaugh, T.A., and Al-Ani, B., "Marginal Notes on Amethodical Requirements Engineering: What Experts Learned from Experience," in *16th International Requirements Engineering Conference*, Barcelona, Catalunya, Spain, pp. 105-114, 2008.

[11] Sim, S.E., Cohn, M.L., and Philip, K., "The Work of Software Development as an Assemblage of Computational Practice," in *Cooperative and Human Aspects of Software Engineering (CHASE09)*, Vancouver, Canada, pp. 92-95, 2009.

[12] Sim, S.E., Easterbrook, S., and Holt, R.C., "Using Benchmarking to Advance Research: A Challenge to Software Engineering," in *Twenty-fifth International Conference on Software Engineering*, Portland, OR, pp. 74-83, 2003.

[13] Sim, S.E. and Philip, K., "Software Source Code Mashups as Transnational Circuits of Know-How," in *CSCW 2008 Workshop: Tinkering, Tailoring, and Mashing: The Social and Collaborative Practices of the Read-Write Web*, San Diego, CA, 2008.

[14] Sim, S.E. and Philip, K., "Tracing Transnational Flows of IT Knowledge through Open Exchange of Software Development Know-How," in *Third Annual iConference* UCLA, Los Angeles, CA, 2008.

[15] Sim, S.E., Umarji, M., Ratanotayanon, S., and Lopes, C.V., "How Well Do Search Engines Support Code Retrieval on the Web?," *ACM Transactions on Software Engineering and Methodology,* 2012.

[16] Umarji, M., Sim, S.E., and Lopes, C.V., "Archetypal Internet-Scale Source Code Searching," in *4th International Conference on Open Source Systems*, Milan, Italy, p. 7, 2008.

[17] Wing, J.M., "Computational Thinking," in *Communications of the ACM*. vol. 49, pp. 33-36, 2006.