Susan Elliott Sim
ses@intdexwis.com

Research Statement

The software that we use on our computers, mobile phones, and other devices is created by developers (or 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 management. 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].

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 what it means to participate in the process of creating software.

A central theme of my research over the last two decades has been exploring boundaries and exclusions in software development. Using an approach that combines computer science and STS (science and technology studies), I have addressed questions such as: What counts as software process? Who decides what features to include? What constitutes a feature? My current research looks at computer software, specifically source code, as a form of digital media that is circulated and remixed. I am interested in the tools, techniques, and ethics that are in play when programmers take source code from the web and recombine it in a novel software assemblage. In future, I will look more closely at issues of authorship in software development. I will now discuss each of these projects further.

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 [4, 5]. 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. In addition, I construct software tools to provoke novel behaviors and provide new contexts for observing established practices.

Prior Research

What counts as software process? Many modern software products, such as web applications, are developed by small teams working on short business cycles with little formal, written documentation. In these settings, software developers and managers often feel that they were not using a "real" software process because they were not following a written prescriptive model [3]. Despite delivering successful software, they were defensive about their deviations from prescribed procedures. At the same time, they felt that their adaptations and improvisations were the only thing that could work in their context. We concluded that software process models and enactments are a generative system, where the prescribed procedures and local improvisations are resources [1]. In particular, specific actions are chosen by participants to ensure that their contributions are counted as inside or outside the software process.

Who decides what features to include? Programmers are often stereotyped as unkempt "geeks" who possess a special ability to talk to the computer, but not necessarily to users. At the same time, users are stereotyped as having unreasonable priorities, such as fixating on the color of software early in the process. While stereotypes have their limits, they also contain a grain of truth: programmers and users aren't very good at communicating with each other [5]. Yet they must, because users have a business or organizational need that must be filled using software and the programmers are tasked with providing the software. Usually, a "business analyst" or "product manager" is tasked with serving as a bridge between the two worlds. These individuals serve as obligatory points of passage, by taking the incomplete information provided by users and translating it into incomplete information for programmers [10].

What constitutes a feature? The concept of "feature" is a boundary object, plastic enough to be used in multiple contexts by diverse stakeholders yet robust enough to retain identity across groups. Consequently, it doesn't mean exactly the same thing to different people at different phases of software development. To probe this question, my students and I have implemented a series of software tools that allow programmers to create maps of features [6-9]. These tools allow developers to create documentation that is simultaneously situated in the code and locally contingent. Examining these maps allows us to investigate the lifecycle and epistemology of features.

Current Research: What are the limits of hybrid source?

My current work looks at the contested space between the extremes of "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 positions, 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 the web and sometimes use open source components, but they usually don't share their own software assemblages. Hybrid source offers a site of cultural production to test claims about sharing of knowledge and transnational circuits, to investigate the effects of novel computational tools on how software is constructed, and to study the entanglement of social and computational systems. My pioneering work in this area [14, 15] has led to an NSF CAREER grant.

When source code is downloaded or copied, it is used both as a material to be used in programming and as knowledge [12, 13]. We have found that open source projects tend to be used as construction supplies, whereas examples that consist of a few lines ("snippets") tend to be used as know-how. 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. We have been building tools to help programmers share and use snippets, and we are looking how they phrase questions and answers that use snippets.

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. 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 [2]? This work will have implications for law, economics, and software ethics.

References

[1] 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.

[2] 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.

[3] Gallardo-Valencia, R.E. and Sim, S.E., "Enacting Software Processes through Prescription and Improvisation," in *ACM Crossroads*. vol. 14, p. 3 (8 pages), 2007.

[4] 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.

[5] 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.

[6] Ratanotayanon, S., Choi, H.J., and Sim, S.E., "Feature Location Using Induced Changesets to Join Semantic and Syntactic Knowledge," in *25th IEEE/ACM International Conference on Automated Software Engineering*, Antwerp, Belgium, 2010.

[7] Ratanotayanon, S., Choi, H.J., and Sim, S.E., "My Repository Runneth Over: An Empirical Study on Diversifying Data Sources to Improve Feature Search," in *Eighteenth IEEE International Conference on Program Comprehension*, Braga, Portugal, 2010.

[8] Ratanotayanon, S. and Sim, S.E., "Discovering Task-Based Concern Maps by Merging Concern Fragments," in *Seventeenth IEEE International Conference on Program Comprehension (ICPC)*, Vancouver, Canada, pp. 299-300, 2009.

[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. 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.

[13] 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.

[14] 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.

[15] 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.