Three Positions on the Future of Open Source Research

Dirk Riehle
Friedrich-Alexander-University of Erlangen-Nürnberg
Martensstr. 3, 91058 Erlangen, Germany

dirk@riehle.org, http://dirkriehle.com, @dirkriehle

For the Computing Community Consortium's 2010 Workshop on the Future of Open Source Research, held in February 2010 at and near UC Irvine, I provide the following opinions on three major open source research areas.

1. Quantitative Analyses of Actual Programmer Behavior

We design software development processes and tools based on what we think helps the process and programmers best. The design of a process and of supporting tools then reflects the beliefs of the designers of what holds true in software development. Until the advent of empirical software engineering, it used to be a lot of guesswork of what makes a good process and a good tool. But even early empirical software engineering research found it hard to abstract from one project to another, not to mention to get sufficient data in the first place.

Open source has changed that. We now have extensive data at hand for the analysis of actual software development processes and actual programmer behavior. We should analyze this data on all dimensions and we should compare it with what people previously believed was true about software development (as applied to open source). First, this will give us a better understanding of how things actually work, and from this we can generate new insights and new hypotheses. Second, by comparing actual data with previously held beliefs that influenced process and tool design, we can uncover problems with those designs and improve them.

Various consequences follow: (a) The research community needs a broad and commonly shared database of project data; efforts like FLOSSmole are only the beginning. (b) The software engineering research community needs to brush up its empirical research skills because unearthing beliefs requires new methods previously unknown or unused in computer science. (c) Publication organs need to be more accommodating of research that takes a natural science or social science perspective rather than an engineering perspective. (d) The community needs to find ways to apply the same research efforts to closed source software.

2. Improved Open Source Process and Tooling

Quantitative analyses of open source projects can only show how things are done today. But what about tomorrow? Where do groundbreaking new practices and tools come from? Open source software development, when compared with other approaches like plan-driven and agile methods, has many advantages but also is clearly lacking in many dimensions. Many of these problems surface, when developers are no longer "scratching their own itch". In such situations, more traditional methods of product and project management can and should be applied.

Thus, we need to see which practices of plan-driven and agile methods can be applied and carried over to open source software development. Some agile methods practices, for example, seem easy to apply: Refactoring, test-driven development, and continuous integration, just to name a few, seem compatible with an open source value system. Nevertheless, they pose research questions, for example, why open source projects tend to prefer code review or user testing over test-driven development. Other practices like consistent product management are virtually unheard of in open source software development. Here open source can learn and benefit a lot.

Thus, the research question is: Which existing traditional plan-driven and agile methods practices are compatible with open source software development, would be beneficial to be integrated, and how should they be integrated? Such research work is highly empirical qualitative hypothesis driven and can build on the prior mentioned quantitative analyses. The end result could be a new process framework combining the best of plan-driven, agile methods, and open source software development.

3. Decision Models for Industry Participation in Community Open Source

Community open source is open source software that is owned by a community of stakeholders, sometimes through a proxy like a non-profit foundation. Today, neither software development nor software user firms have decision models that help them decide whether and to what extent to get engaged with such projects. What is the return on investment for SAP to participate in the Apache Software Foundation? For Bank of America to support the Eclipse Foundation? Without proper decision models and economic reasoning, software user and developer firms are likely to remain more cautious than necessary and end up underinvesting.

Thus, the proposed research is to develop economic models that aid industry decision makers when making investment decisions for their software infrastructure and projects. A recent paper outlines some of the economic benefits that software developer and user firms can expect when investing in community open source like reduced development costs, increased sales of complementary products, and a larger addressable market [DR1]. Some of these benefits can be addressed in an analytically closed form and, from an economists perspective, may be "mere applications" of established theory. Other benefits, like how positional advantages gained in a foundation can be translated into increased revenues, may only open to simulation and related approaches.

The main challenge in this research is the complexity of modeling the real world and the need to combine deep understanding of technology and software with economics expertise, commonly not found in one person. Thus, it can only addressed by teamwork between computer scientists and economists. A second challenge is to make it usable for practitioners, which requires both relevance of the research to the real world as well as ease of use of the models.

[DR1] Dirk Riehle. "The Economic Case for Open Source Foundations." IEEE Computer, January 2010. Page 86-90.