

Thoughts on Continuous Design and Open Source

Richard P. Gabriel

Design is the thinking done before building something.

—Richard P. Gabriel PhD MFA

Open source is primarily a social activity, and therefore its most important aspect is harnessing the community to accomplish such things as the following:

- determine the right design, especially for end-users, more quickly than traditional development processes can
- continue to evolve the design as requirements are discovered and change
- get to a high quality implementation more quickly than traditional development processes can
- create exemplary software artifacts to form the basis for a literature of software (source)



Currently open-source projects are weak on engaging end-users and encouraging diversity. Diversity is important to finding design principles. Think of how hard it would be for people to come to understand how to build bridges if only the builders of bridges were allowed to observe them.

Engaging end-users is a serious issue. Consider the agile community which is renowned for trying to do iterative design and development. One of their principles is the following:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software....

<http://agilemanifesto.org/principles.html>

Reading their principles carefully, one comes to see that the “customer” is the organization paying for the work, not the end-user. Recently there has come to light disturbing facts about how developers and customers can conspire (perhaps innocently) to thwart the good of end users. Voting machine companies serve election officials in local governments in a developer/customer relationship. During the ongoing IEEE Voting Equipment Standards work (IEEE Project 1583) it has become clear that the customer (local voting officials) wanted control over the software and audit trails, and the developers wanted to comply, saying that it was important to trust the voting officials. Missing in the equation are the end-users, who in this case are the voters. With less than adequate security, it’s possible for local voting officials and others to tamper with an election, which is not what the end-users want, and there is evidence that such voting machine tampering has taken place.



Educating software designers is not done the same way other sorts of designers are educated. Design is typically taught by practice while reflecting, under the tutelage of a master designer, along with a heavy dose of critical reflection on works in the design area.

Currently there is no primary literature of software source, neither at the code, design, nor architectural levels, there is no critical literature of software-related design, there are no schools nor curricula aimed at teaching or training in software design, there are few or no masterpieces, few or no masters, nor even a way to talk about design qua design in software. This primary literature initially could come from open-source projects.

The only systematic attempts I know of to capture techniques and practices of software design have been made by the software patterns community, whose members have proceeded by attempting to create a second-order literature by mining common patterns of practice and expressing them in natural language and simple diagrams. The software patterns community has been working at this for about 10 years now, largely outside academic institutions. The software patterns community has its own conferences, workshops, publications, editors, reviewers, practices, and literature. This community has not yet attempted a systematic mining of open-source software. Perhaps there are ways to encourage this.



We need to understand how open-source communities work. We need to know how roles and attitudes change over time, especially as participants move from the periphery to the center of a project and how this affects design. We need to understand the role of written communication in the open-source design process—on the surface this would seem to reinforce Knuth’s literate programming ideas and also form an audit trail of design and other decisions.



We also need to understand what constitutes a good design and then determine whether open-source processes improve the design over time.