Designing for Participation in Socio-technical Software Systems

Yunwen Ye[1,2] and Gerhard Fischer[1]

[1] Center for LifeLong Learning and Design, Univ. of Colorado, Boulder, CO80503, USA
[2] SRA Key Technology Laboratory, 3-12 Yotsuya, Shinjuku, Tokyo 160-0004, Japan
{yunwen,gerhard}@colorado.edu

Abstract. Participative software systems are a new class of software systems whose development does not end at the deployment but requires continued user participation and contribution. They need to provide both solutions to users and a participation framework that entails technical and social challenges. Meta-design is a promising approach to guide the development of participative software systems. Drawing on lessons learned from a systematic analysis of Open Source Software projects, this paper described general issues that need to be addressed to enable and encourage continued user participation during the meta-design process.

Keywords: meta-design, participative software system, socio-technical environment, system evolution, community of practice, Open Source Software.

1 Introduction

We have been observing the rapid emergence of a new type of software systems that are based on the contributions by a community of users [18]. Systems, such as Wikipedia, Flickr, and Open Source Software (OSS) projects, that are created through the collaboration of many contributors who are regarded as equal partners by bringing their unique set of skills and expertise to shape the functionality and utility of the software systems. We call such software systems as *participative software systems (PSS)* whose design does not end at the time of deployment and whose success hinges on continued participations and contributions of users at use time. Participative software systems need to be evolved continuously at the hand of users to achieve the best fit between the system and its ever-changing context of use, problems, domains, users, and communities of users.

In such systems, the roles of users and developers are blurred and design extends into use time. The design of participative software systems, therefore, presents a challenge of creating new methodological frameworks that re-delineate the roles of developers and users, re-distribute the design activities over the life cycle of the software systems, and give equal importance to the design of technical functionality and the design of social conditions for wide and sustained participation of users.

Meta-design [3] is a new design methodology that we have proposed to address the above challenge. Meta-design characterizes objectives, techniques, and processes for

C. Stephanidis (Ed.): Universal Access in HCI, Part I, HCII 2007, LNCS 4554, pp. 312–321, 2007.

creating new media and environments that allow "owners of problems" (or users) to act as *designers*. A fundamental objective of meta-design is to create socio-technical environments [7] that empower users to engage actively in the continuous development of systems rather than being restricted to the use of existing systems. Meta-design aims at defining and creating not only *technical infrastructures* for the software system but also *social infrastructure* in which users can participate actively as co-designers to shape and reshape the socio-technical systems through collaboration.

User participation, however, does not come automatically. Specific design decisions have to be made conscientiously to enable and encourage user participation and collaboration. This paper discusses the issues that need to be addressed during the meta-design process to achieve sustainable user participation. After the articulation of the concept and defining features of participative software systems in Section 2, we describe the lessons that we have learned from a systematic analysis of OSS systems. Drawing from the lessons, we present a general framework of designing for participation in Section 4, followed by a summary in Section 5.

2 Participative Software Systems

Software systems are knowledge artifacts whose creation requires a wide range of knowledge from computation domains and problem domains. Systems that require relatively little domain knowledge or in domains where requirements can be clearly articulated up front can be delegated to professional developers after the users have clearly identified the requirements. When the requirements can be only partially understood or defined previous to the construction of the system, professional software developers need to work in close collaboration with domain experts (a system design methodology pursued in *participatory design approaches* [15]).

Most complex problems are ill-defined problems that *cannot be delegated* because they require the *integration of problem framing and problem solving* [13], making it impossible to define requirements in advance. Ill-defined problems require that "back-talk" of a problem goes to the owners of the problem helping them iteratively to gain a deeper understanding of the problem [14] during the process of constructing the solution. Continued user participation and involvement in the design and development of software system is needed. We use the term *participative software system (PSS)* [11] to refer to this kind of software systems.

The development of PSS does not end at the time of deployment but extends into use. PSS is a living entity and a socio-technical system [7] capable of integrating computing infrastructure and participation process in one single platform and supporting collaboration not only about design artifacts but also about the goals of the design activity. In a PSS:

1. users can participate in the evolution of the system according to their capabilities and on the basis on their own interest or needs;
2. user participation (at various levels) not only benefits the user, but it also shapes the platform for other participants to collaborate; and
3. as a result of participation, users and the software system co-evolve to adapt the whole PSS to new social and technical demands.

2.1 Re-defining the Roles of Users and Developers

In the world of software, users and developers are conventionally regarded as two mutually exclusive groups of people. Users are those people who own a problem, and developers are those who construct software systems for the users. However, with the widespread use of, and the society's increasing reliance on, software, the distinction between users and developers is quickly disappearing. More and more people are not only using software but also getting involved in developing software to widely varying degrees (Fig. 1) to solve problems.

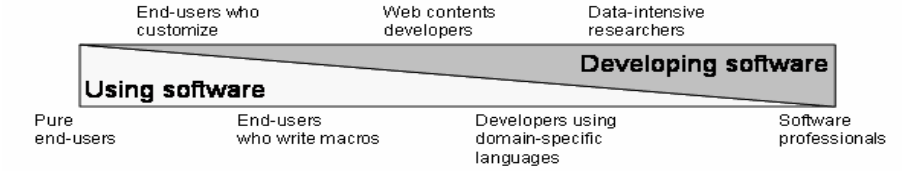

Fig. 1. The spectrum of software-related activities

To make software development easier, two major research fields have been established to attack both ends of the above spectrum (Fig. 1). *Software engineering* focuses on the group of people on the right, who call software development their profession. They develop software systems that are used by users other than themselves. *End-user development* [6, 8] aims to find ways of creating software systems that can be adapted by end-users to their own unique needs. It focuses on the group of people on the left of the above spectrum.

In the middle are people who have certain software development skills but are not interested in software per se. They do not develop software for other people; rather they are developing software to solve specific problems that they own. This group of people can be called *domain expert software developers* (or *domain experts*) [1].

2.2 Redistributing the Design Activity

In all design processes, two basic stages can be differentiated: design time and use time [4]. At *design time*, system developers (with or without user participation) create environments and tools for the *world as imagined* by them to anticipate users' needs and objectives. At *use time*, users use the system in the *world as experienced*. The bridging of these two stages into a unique *"design-in-use" continuum* encompassing an ongoing conversation both with the design material and among participants differentiates meta-design from other (more established) design frameworks.

Existing design frameworks are based on the assumption that major design activities end at a certain point after which the system enters use time. Meta-design complements and transcends these design methodologies by creating *open and continuously evolvable systems* that can be collaboratively extended and redesigned at use time *by users and user communities*. However, meta-design is not merely end-user modification and programming. Meta-designed software systems not only provide the technical means for users to customize and extend the systems but also

provide social and technical mechanisms to facilitate user participation and collaboration during the design activities.

3 Designing for Participation: Lessons from Open Source Software Development

To understand how user participation can be sustained in PSS, we studied successful examples of a typical class of PSS: *Open Source Software (OSS)* systems.

OSS development is an activity in which a community of software developers collaboratively constructs systems to help solve problems of shared interest and for mutual benefit. The original designers of an OSS system do not provide a complete solution that addresses all problems of potential users, rather he or she provides an "*under-designed seed*" as a solution space that can be evolved by its users at use time via making the source code available [2, 12]. The ability to change source code, the technological means of sharing changes over the Internet, and the spontaneous social support among community members are the enabling conditions for collaborative construction of software by changing software from a fixed entity that is produced and controlled by a closed group of designers to an open effort that allows a community to design collaboratively.

However, not all OSS systems are successful in terms of active user participation. A study [10] of 90,902 Open Source Software projects hosted in the sourceforge.net has found that 66.7% of the projects have only one developer.

To understand the socio-technical factors that make some OSS development successful PSS, we have conducted studies of five OSS projects: GNU, Linux, PostgreSQL, Jun and GIMP [21, 22]. One critical factor that enables the continual evolution of an OSS project is the forming of a vibrant and sustained *community of practice* [20] of developers, users, and user-turned-developers. The right to access and modify source code itself does not make OSS projects different from most "Closed Source Software" ones. All developers in a project in any software company would have the same access privilege. The fundamental difference is the *role transformation* of the people involved in a project. In Closed Source Software projects, developers and users are clearly defined and strictly separated. In OSS projects, there is no clear distinction between developers and users: all users are potential developers.

3.1 Roles and Community Structure in OSS Communities

People involved in a particular OSS project create a community around the project. Members of an OSS community assume roles according to their personal interest in the project, rather than being assigned by someone else. A member may have one of the following eight roles [9]:

Project Leader. Project Leaders are often the person who has initiated the project. They are responsible for the vision and overall direction of the project.

Core Member. Core Members are responsible for guiding and coordinating the development of an OSS project. Core Members are those people who have been involved with the project for a relative long time and have made significant contributions to the development and evolution of the system.

Active Developer. Active Developers regularly contribute new features and fix bugs; they are one of the major development forces of OSS systems.

Peripheral Developer. Peripheral Developers occasionally contribute new functionality or features to the existing system. Their contribution is irregular, and the period of involvement is short and sporadic.

Bug Fixer. Bug Fixers fix bugs that either they discover by themselves or are reported by other members. Bug Fixers have to read and understand a small portion of the source code of the system where the bug occurs.

Bug Reporter. Bug Reporters discover and report bugs; they do not fix the bugs themselves, and they may not read source code either. They assume the same role as testers in the traditional software development model.

Reader. Readers are active users of the system; they not only use the system, but also try to understand how the system works by reading the source code.

Passive User. Passive Users just use the system in the same way as most of us use commercially available Closed Source Software. They are attracted to OSS mainly due to its high quality and the potential to be changed when needed.

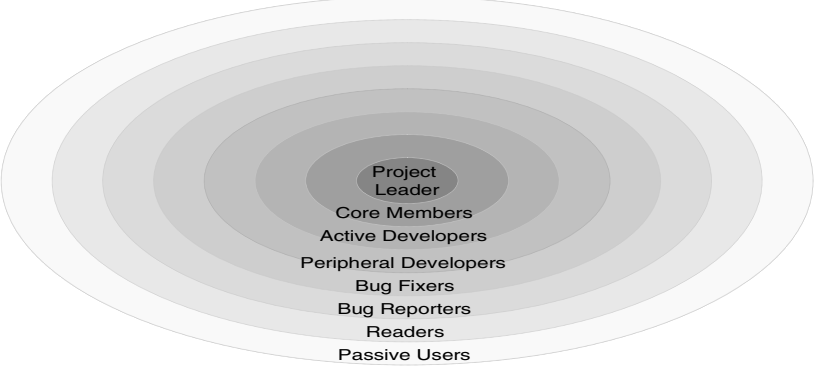

Fig. 2. General structure of an OSS community

Although a strict hierarchical structure does not exist in OSS communities, the structure of OSS communities is not completely flat. The influences that members have on the system and the community are different, depending on the roles they play. Fig. 2 depicts the general layered structure of OSS communities, in which roles closer to the center have a larger radius of influence. Passive Users have the least influence, but they still play important roles in the whole community. Although they do not directly contribute to the development of the system technically, their existence contributes socially and psychologically by attracting and motivating other, more active, members, to whom a large population of users is the utmost reward and flattery of their hard work [12].

3.2 Co-evolution of OSS Systems and OSS Communities

The roles and their associated influences in OSS communities can be realized only through contributions to the community. Roles are not fixed: members can play larger roles if they aspire and make appropriate contributions. As members change the roles they play in an OSS community, they also change the social dynamics, and thus reshape the structure of the community, resulting in the evolution of the community.

For an OSS project to have a sustainable development, the system and the community must co-evolve. A large base of voluntarily contributing members is one of the most important success factors of OSS. The evolution of an OSS community is effected by the contributions made by its aspiring and motivated members. Such contributions not only transform the role and influence of their contributors in the community and thus evolve the whole community, but they are the sources of the evolution of the system. The opposite is also true; any modification, improvement, and extension made to an OSS system not only evolves the system but redefines the role of the contributing members and thus changes the social dynamics of the OSS community (Fig. 3).

The role that an OSS member plays in the community is not pre-assigned, and is assumed by the member as he or she interacts with other members. An aspiring member can become a Core Member through the following migration path:

- New members are attracted to an OSS community because the system can solve one of their own problems.
- The depth and richness of good OSS systems often drives motivated members to want to learn more, to read the system [16]. The new members now migrate from Passive Users to Readers. As they gain more understanding of the system, they are able to fix the bugs that are either encountered by themselves or reported by others. They may also want to add a new twist to the system to make the system more powerful and more suitable for their own tasks.
- As their developed programs are made publicly available to other community members, their roles as Bug Fixers and Peripheral Developers are recognized and established in the whole community. The more contributions they make, the higher recognition they earn, and finally, they will enter the highly selected "inner circle" of Core Members.

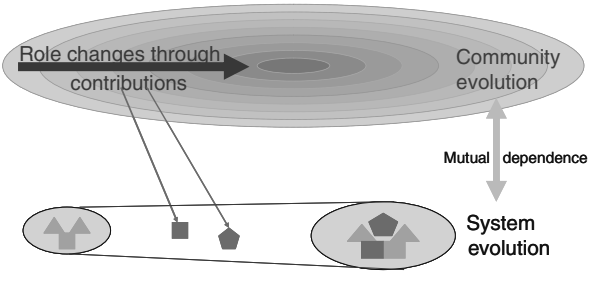

Fig. 3. The co-evolution of OSS systems and OSS communities

The above path describes an abstract model of role changes of aspiring members. Not all members want to and will become Core Members. Some will remain Passive Users, and some stop somewhere in the middle. The important point is that Open Source Software makes it possible for an aspiring and technically capable software developer to play a larger role through continual contributions and engagement

4 Designing for Participation: A General Framework

Drawing from the lessons learned by systematic analysis of OSS projects from the meta-design perspective, this section describes challenging issues that need to be considered during the meta-design process of PSS to enable and encourage continued user participation.

4.1 Embracing Users as Co-designers

To embrace users as co-designers, designers of PSS need to bear in mind that they are not only providing a solution to users, but also a solution space [18] within which users can develop new solutions to their specific needs. The solution space contains technological instruments that users can use for their design activities, and determines the degree that users can evolve the original design. Currently available technology in software systems provides a variety of choices, ranging from the modification of options, the customization of menus and functions, the plug-in structure for extension, the published services for being mashed up with other services, the publication of system API for integration with other systems, and the source code that offers the highest freedom for user development. Meta-designers of PSS have to make a conscientious decision according to how much they want to get user involved.

4.2 Providing a Common Platform

Design contributions made by one individual user are limited because one particular user is only interested in creating solutions for his or her own needs. The power of distributed user design comes from the fact that the evolution of systems is pushed by a large number of users with diversified needs and skills who each makes small contributions. For this to happen, users need to have a common platform so that they can share with each other and integrate design solutions of others. Meta-designers need to either create an associated common toolkit or utilize a set of common tools widely available to all users to facilitate easy sharing and integration. The concept of OSS has been pioneered by Richard Stallman (with the term Free Software) in the 80s but the huge success of OSS systems becomes possible only when software development tools—such as Emacs, Eclipse, and CVS—becomes widely available and the de facto standard tools for most software developers.

4.3 Enabling Legitimate Peripheral Participation

A transparent policy and procedure is needed to incorporate some of user contributions into the participative software systems. Users who made contributions need to see that their contributions make a recognizable influence on the system. In

other words, user participation has to be legitimate [20] and their design activities are regarded as an integral part of shaping the direction and functionality of the system.

The possibility for newcomers to participate peripherally is another key aspect [19]. To attract more users to become developers, the system architecture must be designed in a modularized way to create many relatively independent tasks with progressive difficulty so that newcomers can start to participate peripherally and move on gradually to take charge of more difficult tasks. The way a system is partitioned has consequences for both the efficiency of parallel development—a prerequisite for OSS—and the possibility of peripheral participation. The success of Linux is due in large part to its well-designed modularity [17].

Another approach to afford peripheral participation is perhaps to *intentionally release under-designed system* to users by leaving some non-critical parts unimplemented to facilitate easy participation. The TODO list of most OSS systems creates guidance for participation.

4.4 Sharing Control

While the original meta-designers of the PSS may retain the major control of the direction of the system, active participating users need to be granted certain controls commensurate with their interest, technical skill, and contributions. The roles that a domain expert user can play in the system are different depending on their levels of involvement. Each level has its own responsibility and authority. Responsibility without authority cannot sustain users' interest in further involvement. When users change their roles in the PSS by making constant contributions, they should be granted the matching authority in the decision-making process that shapes the system. The meta-designer needs to find a strategic way to transfer some of the control to aspiring and contributing users. Granting those users controlling authority has two positive impacts on sustaining user participation: (1) users who gain controlling authority become stakeholders and require ownership in the system and are likely to make further contributions; and (2) it can attract and encourage new users who want to influence the system development to make contributions. Successful OSS projects invariably select skilful user-turned-developers and grant them access privilege to contributing directly to the source base.

4.5 Promoting Mutual Learning and Support

Users have different levels of skill and knowledge about the system. To get involved in contributing to the system or using the system, they need to learn many things. Peer users are important learning resources. A PSS should be accompanied with knowledge sharing mechanisms that encourage users to learn from each other. In OSS projects, mailing lists, discussion forums, and chat rooms provide an important platform for knowledge transfer and exchange among peer users [5].

4.6 Fostering a Social Rewarding and Recognition Structure

Motivation to participation is essential for the success of PSSs. Factors that affect motivation are both intrinsic and extrinsic. The precondition for motivating users to get involved in contribution is that they must derive an intrinsic satisfaction in their

involvement by shaping the software system to solve their problems. Intrinsic motivation is positively reinforced and amplified when social structure and conventions of the community recognize and reward the contributions of its members.

The social fabric inherent in OSS communities reinforces the intrinsic motivation for participating in OSS projects. Members close to the center of the community enjoy better visibility and reputations than do peripheral members. As new members contribute to the system and the community, they are rewarded with higher recognition, trust, and influence in the community. Rewarding contributing members with higher recognition and more important roles is also important for the sustainability of the community and the system development, because it is the way that the community reproduces itself.

Developers of PSSs therefore need to establish a social norm in the user communities by recognizing publicly contributing users and promoting the social status in the community by granting matching authority.

5 Summary

PSSs represent the rapidly emerging class of software systems whose development does not end at the point of deployment and continues to evolve at the hand of participating users. The success of many such systems is mostly accidental resulting from the insights of their original designers. Existing software design methodologies that have mainly focused on engineering software systems to the needs of users at design time are not well suited for PSSs. For the past several years, we have developed the *meta-design framework* to address this challenge. In this paper, we described general issues that need to be considered to design socio-technical environments that enable and encourage user participation, drawing on a systematic study of existing OSS projects.

Acknowledgements. The authors would like to thank Kumiyo Nakakoji, Yasuhiro Yamamoto, and the members of the Center for LifeLong Learning & Design at the University of Colorado, who have made major contributions to the research described in this paper. The research was supported (1) by the National Science Foundation, Grants (a) IIS-0613638 “SoD-Team: A Meta-Design Framework for Participative Software Systems”, and (2) by SRA Key Technology Laboratory, Inc., Tokyo, Japan.

References

1. Costabile, M.F., Fogli, D., Fresta, G., Mussio, P., Piccinno, A.: Building Environments for End-User Development and Tailoring. In: Proc. of 2003 IEEE HCC’03, Auckland, New Zealand pp. 31–38 (2003)
2. DiBona, C., Ockman, S., Stone, M. (eds.): Open Sources: Voices from the Open Source Revolution. O’Reilly and Associates, Sebastopol, CA (1999)
3. Fischer, G., Giaccardi, E.: Meta-Design: A Framework for the Future of End User Development. In: Lieberman, H., Paternò, F., Wulf, V. (eds.) End User Development: Empowering People to Flexibly Employ Advanced Information and Communication Technology, The Netherlands, pp. 427–457. Kluwer Academic Publishers, Dordrecht (2006)

4. Henderson, A., Kyng, M.: There's No Place Like Home: Continuing Design in Use. In: Greenbaum, J., Kyng, M. (eds.) Design at Work: Cooperative Design of Computer Systems, pp. 219–240. Lawrence Erlbaum, Mahwah (1991)
5. Lakhani, K.R., von Hippel, E.: How Open Source Software Works: Free User to User Assistance. Research Policy 32, 923–943 (2003)
6. Lieberman, H., Paternò, F., Wulf, V.: End User Development - Empowering People to Felxibly Employ Advanced Information and Communication Technology, The Netherlands. Kluwer Publishers, Dordrecht (2006)
7. Mumford, E.: Socio-Technical System Design: Evolving Theory and Practice. In: Bjerknes, P.G., Ehn, P., Kyng, M. (eds.): Computers and Democracy. Averbury, Aldershot, UK, pp. 59–76 (1987)
8. Myers, B.A., Ko, A.J., Burnett, M.M.: Invited Research Overview: End-User Programming. Proceedings of Human Factors in Computing Systems (CHI2006), Montreal, pp. 75–80 (2006)
9. Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., Ye, Y.: Evolution Patterns of Open-Source Software Systems and Communities. In: Proceedings of International Workshop on Principles of Software Evolution (IWPSE 2002) Orlando, FL, pp. 76–85 (2002)
10. Ohira, M., Ohsugi, N., Ohoka, T., Matsumoto, K.-i.: Accelerating Cross-Project Knowledge Collaboration Using Collaborative Filtering and Social Networks. In: Proceedings of International Workshop on Mining Software Repositories, St. Louis, MO, pp. 111–115 (2005)
11. Pangaro, P.: Participative Systems (2000)
12. Raymond, E.S.: The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly, Sebastopol, CA (2001)
13. Rittel, H.: Second-Generation Design Methods. In: Cross, N. (ed.) Developments in Design Methodology, pp. 317–327. John Wiley & Sons, New York (1984)
14. Schön, D.A.: The Reflective Practitioner: How Professionals Think in Action. Basic Books, New York (1983)
15. Schuler, D., Namioka, A. (eds.): Participatory Design: Principles and Practices. Lawrence Erlbaum Associates, Mahwah (1993)
16. Tiemann, M.: Future of Cygnus Solutions. In: DiBona, C., Ockman, S., Stone, M. (eds.): Open Sources: Voices from the Open Source Revolution. O'Reilly, Sebastopol, pp. 71–89 (1999)
17. Torvalds, L.: The Linux Edge. Communications of ACM 42, 38–39 (1999)
18. von Hippel, E.: Democratizing Innovation. MIT Press, Cambridge (2005)
19. von Krogh, G., Spaeth, S., Lakhani, K.R.: Community, Joining, and Specialization in Open Source Software Innovation: A Case Study. Research Policy 32, 1217–1241 (2003)
20. Wenger, E.: Communities of Practice — Learning, Meaning, and Identity, England. Cambridge University Press, Cambridge (1998)
21. Ye, Y., Kishida, K.: Toward an Understanding of the Motivation of Open Source Software Developers. In: Proceedings of, International Conference on Software Engineering (ICSE'03), Portland, OR (2003), pp. 419–429 (2003)
22. Ye, Y., Nakakoji, K., Yamamoto, Y., Kishida, K.: The Co-Evolution of System and Community in Open Source Software Development. In: Koch, S. (ed.) Free/Open Source Software Development. Idea Group Publishing, pp. 59–82 (2004)