A Distributed Software Development Team Meets a Distributed Community of Practice: Participatory Meetings

Abstract
Computer systems can be used to propagate expertise within a professional community of practice, e.g. decision support systems, to drive standardization and systematization of processes, e.g. supply chain management systems. In this paper, we describe a case study that spans fifteen years and documents the co-evolution of a community of software users, which is distributed across continents, and a distributed software development team, which is distributed across different development sites. We describe how a combination of annual community meetings and computer collaboration tools supported a participatory evolution of a major software system.

Keywords
Participatory Design, Condition-based Maintenance, distributed software development

Introduction
Our research is based on an ongoing development activity, started over a decade ago, which targeted a range of systems for maintenance management of civil infrastructure (roadways, piers and wharves, sewers, etc.). Over the years, the technological and
organizational contexts of these systems have undergone a number of changes: stand-alone applications became environments for multi-user collaboration, single user systems were replaced by network-centric, loosely connected collaborative toolset [1]. Both the developers and especially the users of these systems were and still are distributed across several organizations, spanning a number of national and international locations. In this paper we will describe what was done in order to enable and promote user participation in the design and redesign process.

The Case Study
Our case is situated in the context of maintenance of civil infrastructures, such as roadways, airfields, bridges, ports, water utilities, etc [4]. Within this domain, software systems for engineering maintenance management have the goal of helping civil engineers in making good maintenance management decisions, achieving the best result with limited founds. The key idea of these systems is to focus on proactive maintenance of civil infrastructures, in other words the idea is that it is less expensive to maintain an infrastructure in good condition than it is to let it degrade and then replace it when it fails [5].

The main tasks of software systems supporting civil engineering maintenance are: managing the inventory (a description of infrastructures being managed); collecting and storing data measuring the infrastructure conditions; predicting the future conditions based on appropriate models; planning and budgeting maintenance activities. We will focus on a specific system, which has been developed and maintained since the early '90s (PAVER). The software has undergone disruptive evolution several times because of both technology-drivers and domain-based changes.

In order to manage the participation of users in the redesign process, there have been annual workshops (for more than a decade). These workshops bring together sponsors for new development work, civil engineers, power users, and normal users along with software architects and developers to review the previous work, share ideas about evolving civil engineering best practices, in light of the emerging tools, and plan a course for future development.

Sponsors normally represent governmental founding agencies, while engineers are mainly working for organizations in charge of infrastructure maintenance, such as governmental bodies, or for consulting companies. This international user community evolves over time, both in terms of organizations involved as well as people; some individuals have been involved for more than ten years and others for a few years. It is a distributed community that comes together annually but has looser contact over the rest of the year.

The Development Team has been distributed geographically from the start and has used tools like change management software, defect tracking, and collective repositories to share the status of the design and implementation.

This collaborative process has been followed through the evolution of PAVER across major disruptive software technology changes: (1) from DOS to Windows; (2) from 16 bit Windows to 32 bit Windows; (3) from partial object-orientation to a fully object-oriented .NET version; (4) from Windows to the
During that same time, there has also been evolution of the domain algorithms, particularly in the area of work planning and project formulation.

The basic approach has been as follows: the developers, under the supervision of senior pavement engineers, originally brought together mockups of the new system and put these more interactive sessions before the users for feedback. In later meetings, users were given a chance to react to prototype and released software. From that point onward, the meetings have included both the presentation of what was ready for user distribution and what was on the way. In the workshops there were often two projectors: one for slides and one for demos.

Users were presented both emerging prototypes to try and suggest changes, while they are still cheap, and "blue sky" ideas that only exist on paper. During early stages of development, printouts of screen shots and/or mockups were distributed to the users who marked them up with notes. In addition, a set of computers was provided to allow the users to try new software features. Users were able to bring their own data in order to test new versions in their usual context.

During the sessions with released, emerging, and imaginary software, the group compiled a list of proposed revisions. These can include minor tweaks, which can be agreed by the group without further discussion, modifications to systems that go beyond minor tweaks and need to be proposed, funded, and separately tracked. These can also include major new features or paradigms for old features.

For all except the minor tweaks, the group then votes on the relative priorities of the proposed development activities. In all the meetings, a librarian had the task of collecting proposed items on the list of new features. That list was printed and distributed in the last session, when it was used for voting. Priorities are then presented to the sponsors, many of whom heard the discussion among the users about priorities, for possible funding. The development agenda for the next year flows out of the interaction of the priorities of the group and the priorities of the funding agencies.

**Lessons Learned**

From the analysis of several years of user participation for system evolution, we propose several important lessons about how to build collaborative systems in complex application domains:

- Change Management means something different for the community of practice and the development community but these two context are interlocking and represent co-evolution;
- The community of users must keep custody of the domain model and have real ownership;
- Software must be designed and implemented with extension points and "standardized non-standardization" (i.e. protocols for extension);
- Forcing the two, normally separate, communities to spend an extended period of time together breaks down barriers and enriches the perspectives of both groups;
Limiting the time span of this period allows participation of decision makers as well as users of the system;

Formal procedures like, use case models, screen mockups and voting on new features empower the user community and provide a structure to feedback that helps developers;

Successful systems like the one described require human and technological support for “emergent” patterns of professional practice and software tools (pre-defining all of the features does not work).

Conclusions
The lessons listed in the previous section are grounded on practice, allowing us to contextualize the lesson learned [2]. In summarizing these lessons, our first conclusion is the importance of designing for extensibility [3]. An extensible system will allow developer to provide immediate positive feedback to users, rather than putting forward technological reason for not accepting users’ input.

Extensibility is not only a matter of technological design. In order to support long term evolution, systems must be emergent in nature, embodying the domain deeply in the system architecture. Using domain modeling for the overall system allows the user to provide input that directly impacts the system itself, possibly with the mediation of the software architects and developers.

Acknowledgements
We wish to thank all of the sponsors for this work, especially Dr. Mo Shahin of the Corps of Engineers’ Construction Engineering Research Laboratory-ERDC, who oversees this entire program, the Federal Highway Administration, which sponsored the first few workshops and fostered the development of the community, and the U.S. Army, Navy, Air Force, and FAA, among others, who have provided funding for the pavement engineering research and software development. Finally, we wish to acknowledge the participation of users from around the world, who have given of their time and experience to make the community and the software better.

References and citations