ABSTRACT
We aim to improve the usability of software systems by re-examining the software development process with a focus on evaluation and participation of end users and collaboration among members of multi-disciplinary design teams. We propose an evolutionary model of systems development in which the expected usability of a system is improved by active, computer-based support. In our approach, active support is provided by human-centered, intelligent agents. We discuss two types of agents. The first monitors end users working with prototype systems and provides feedback to members of a development team, enabling participatory design. The second monitors design decisions by different system developers and relays related (sometimes conflicting) design information based on that developer’s current task.

INTRODUCTION
Design methodologists [1,2] emphasize the need for integrating problem framing and problem solving. They argue that one cannot gather information meaningfully unless one has understood a problem, but one cannot understand a problem without information about it. For instance, Curtis and colleagues observed the predominant activity in designing complex systems is the participants’ teaching and instructing each other, that understanding the problem is the problem [3]. Because complex problems require more knowledge than any one person possesses, communication and collaboration among all the “stakeholders” involved is necessary [4,5]. Stakeholders include members of a software development team, with potentially diverse backgrounds, and representatives of the intended end users.

We propose an evolutionary model of software development in which the expected usability of a system is improved by active, computer-based support for 1) feedback from users of prototype systems to developers and 2) communication and collaboration among developers. In our approach, active support is provided by human-centered, intelligent agents. Agents are proactive, automated mechanisms that operate in otherwise passive information repositories to find and deliver information to those who need it [6]. More specifically, we discuss two types of agents. The first monitors end users working with prototype systems and provides feedback to members of a development team, enabling participatory design. The second monitors design decisions by different system developers and relays related (sometimes conflicting) design information based on that developer’s current task.

The model and approach are based on a real development experience and current research into agent-based software environments. A system called Bridget was developed by the intelligent interfaces group at NYNEX with the explicit goal of producing a more usable system for customer sales representatives who create new accounts for small businesses. The development of Bridget illustrated a process in which participatory design and communication and collaborative design were carried out by the participants manually. The goal our work is to add computer support through intelligent agents to facilitate the process. In Bridget, we already used agents to support end users in performing their tasks, agents that detect conflicts and automatically initiate database accesses when appropriate. Such agents would now be extended to support system developers.

AGENTS SUPPORTING PARTICIPATORY SOFTWARE DESIGN
To increase the usability of a system, constant participation of end users in the development process is necessary. Such participation increases the system developers’ understanding of the end users’ tasks and shows how end users would interact with the proposed system. Developers have certain expectations of how users should perform tasks with a system. If these expectations do not match actual use, the users might be performing tasks suboptimally or the developers’ expectations might be based on false assumptions.

Agents can help in observing “actual use” in contrast to “expected use.” Such agents would know about a repertoire of tasks the system is intended to support and the sequence of user interaction the developers envisioned for performing these tasks. In the case of a mismatch, the following should happen: 1) developers should be notified of the discrepancy, 2) users should receive an explanation message suggesting a “better” way of performing the task and they
should be able to accept or reject this suggestion, optionally attaching their rationale, 3) and the developers should see all user feedback. In sum, this type of agent initiates a dialog between users and developers in which the developers communicate their intent to the users and the users have the opportunity to respond. Additional, follow-up discussion could take place outside of the agent-based software development environment. The advantage of the agent-based approach over simply keeping a log of the users’ interactions and breakdowns is that the users have the opportunity to reply to the developers’ expectations while still being in the context of the task and breakdown situation.

The potential benefit of such agents was demonstrated in the development of Bridget. Users participated in the development process by redoing tasks they had to do with their existing system. The developers observed the users and asked for explanations of unexpected behavior after the task was completed. The users also used Bridget unsupervised and compiled a list of suggestions for changes. Using the proposed agents would have made it possible to involve more users in parallel and information could have been exchanged more efficiently.

AGENTS SUPPORTING COMMUNICATION AND COLLABORATION AMONG SOFTWARE DEVELOPERS

The development of software is increasing a collaborative effort among experts with diverse backgrounds. Systems programmers, domain analysts, and graphic artists all make decisions which influence the usability of resulting systems. The varying expertise each developer brings to their part of the system development provides a valuable information resource. But, it also poses a challenge, namely, how to communicate and coordinate this information to others.

Our work has focused on allowing members of a development team to work separately but to be alerted to the existence of potential overlap or conflict between their work and the work of others [7]. A virtual cooperation is created that spans both space and time. It spans space in the sense that developers may be located at different sites which is a conventional view of cooperative work. However, the cooperation may also span time in the sense that previous system designs may be relevant to the work of a current development team. Previous developers and users become stakeholders in the present effort. This model of cooperation intensifies the need for active support since developers are unlikely to be aware of past efforts.

In a growing number of software development teams, design discussions, elaboration, and annotation take place on line. By channeling communication through an agent-based software environment, valuable information can be captured by the system with no additional effort on the part of developers. For instance, an interaction history consisting of chronologically ordered e-mail messages, electronic post-it notes, and artifact changes are an important source of domain knowledge that can help developers to understand what work has been done, by whom, and why the work was done. As passive information spaces, interaction histories are not of much use to designers; the effort required to find specific information is prohibitive. Agents having knowledge about interactions and strategies for parsing interaction histories should accept user requests on how and when to make this knowledge available to designers. In this way, information that is lost with paper technology, and is too unwieldy to use with passive database technology, is turned into a design knowledge resource that uses computational agents.

CONCLUSIONS

Complex relationships exist among people, tools, and tasks. One of the crucial aspects of these relationships is that they all change over time in a co-adaptive manner: people adapt their practices and problem-solving approach to the affordances of a tool, and people adapt their tools to better support their practices and problems [4,8]. Our approach supports co-adaptation by supporting communication of design intent among software developers, from software developers to end users, and from end users back to developers. Alan Kay notes that “many are just discovering that user interface design is not a sandwich spread [9]”; our approach seeks to improve the usefulness and usability of a software system by focusing on the process of systems development and not just on the end result.

REFERENCES
