CONTINUOUS COORDINATION WITHIN THE CONTEXT OF COOPERATIVE AND HUMAN ASPECTS OF SOFTWARE ENGINEERING

Ban Al-Ani, Erik Trainer, Roger Ripley, Anita Sarma, André van der Hoek, David Redmiles
University of California, Irvine
Department of Informatics
444 Computer Science Building
Irvine, CA 92697-3440 USA
Phone: +1(949) 824-2776

{ balani, etrainer, rripley, asarma, andre, redmiles}@ics.uci.edu

ABSTRACT
We have developed software tools that aim to support the cooperative software engineering tasks and promote an awareness of social dependencies that is essential to successful coordination. While the tools share common characteristics that can be traced back to the principles of the *Continuous Coordination* (CC) paradigm, the development of each sprung from carrying out a different set of activities during its development process. In this paper, we outline the principles of the CC paradigm, the tools that implement these principles and focus on the social aspects of software engineering. Finally, we discuss the socio-technical and human-centered processes we adopted to develop these tools. Our conclusion is that the *cooperative* dimension of our tools represents the cooperation between researchers, subjects, and field sites. Our conclusion suggests that the development processes adopted to develop like-tools need to reflect this cooperative dimension.

Categories and Subject Descriptors
H.4.3 [**Information Systems Applications**]: Office Automation-Groupware; H5.3 [**Information Interfaces and Presentation**] Group and Organization Interfaces – Computer Supported Cooperative Work.

General Terms
Management, Documentation, Design, Human Factors.

Keywords
Continuous coordination, awareness, social dependency, collaboration, cooperation, software engineering.

1. INTRODUCTION
Software engineering (SE) by its very nature is a complex cooperative process, which requires the collaboration of stakeholder teams (e.g. managers, end-user, designers…etc). This cooperation is made even more complex with the increase in the number of stakeholders involved and can lead to a failure in coordination and a loss of information pertaining to social dependencies amongst software engineers [2][7].

An increase in the size of the software engineering team can lead to developers lacking an awareness of social dependencies within the team. Awareness of the social dependencies is typically difficult to recognize or to identify dynamically in software teams, yet they are essential to the coordination process. Some studies have concluded that the interactions that occur during various SE activities create social dependencies that may continue throughout the development lifecycle. Recognizing, defining and documenting these dependencies can lead to heightened awareness and can mean a more successful SE experience [5].

Our work is concerned with overcoming some of the challenges of cooperative software engineering by implementing the principles of the *Continuous Coordination* (*CC*) paradigm. Our implementations provide alternative means of communication and identify social dependencies amongst stakeholders visually. In previous work, we presented a report of the tools that implement this paradigm within varying contexts and to support diverse software engineering activities (e.g. [14] [1]). In this paper, we report the development processes adopted to develop tools that visualize the interdependencies that typically evolve during a cooperative software engineering endeavor. Consequently, we present an outline of the CC tools that visualize the social interdependencies in the following section (Workspace Activity Viewer, Ariadne and World View) and discuss the development process activities in the final section of this paper

2. CONTINUOUS COORDINATION (CC): VISUALIZING SOCIAL ASPECTS OF SE
The CC paradigm is based on four main principles, namely: identifying *how* much of *what* information needs to be shared with *whom* and *when*. We concluded these principles from the observation we derived from both field studies and from existing literature, namely: that software engineers typically need differing

levels of information abstraction at different stages of development while carrying out different developmental tasks. When implementing these principles we sought to provide this information at a time suitable to developers such that it becomes part of their *peripheral awareness* in an unobtrusive way.

We present an outline of the tools that implement these principles and visualize the social aspects of cooperative software engineering in the following sections, namely Workspace Activity Viewer (WAV), Adriane and World View (WV). We present screen shots of CC tools in figure 1.

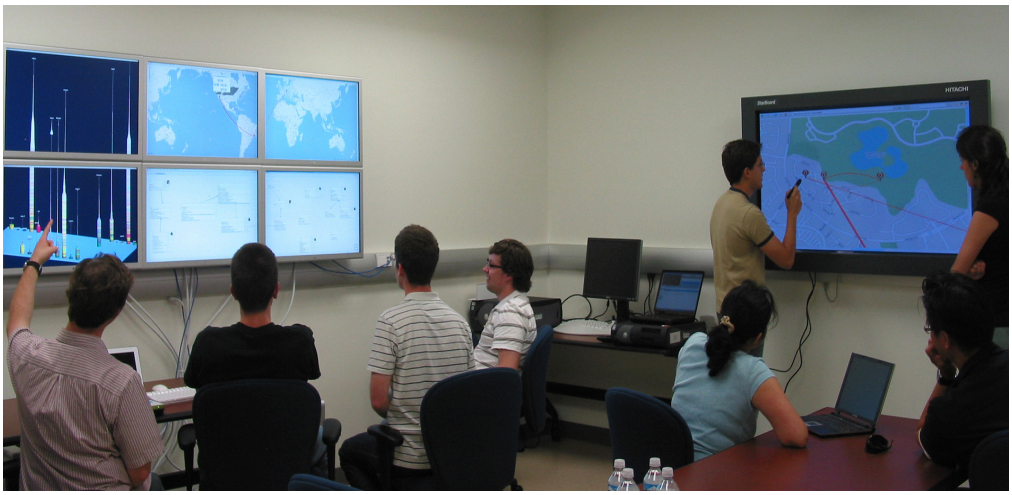

Figure 1. An open discussion of the CC tools.

2.1 Workspace Activity Viewer (WAV)

The Workspace Activity Viewer project is based on our observations of problems in the real world. It visualizes the developers and artifacts in a project using a 3D metaphor and gives managers an overview of ongoing activities in a project, using information extracted directly from developers' workspaces. The tool can visually represent either artifacts or developers as stacks of cylinders. When artifacts are represented each segment of the cylinder denotes a developer that has modified that artifact, whereas when the tool represents developers each segment of the cylinder denotes an artifact that developer has touched. Thus, the height of the stacks of cylinders represents the amount of changes to the artifact or the reach of a developer's activity. As stacks (artifacts or developers) become dormant, the associated stack of cylinders slowly moves to the back of the display. WAV conveys the magnitude of a change (currently calculated as the relative number of lines of code that have changed) by the width of each cylinder. Thus, the tool can visualize both the social evolution and the social dependencies the developers are seeking

The WAV reveals social evolution by allowing its user to play back the state of the project, showing what developers are active when, and to which types of artifacts they contribute. The workspace activity viewer also shows social dependencies by revealing when developers are simultaneously working on the same artifact. Since the tool shows the state of a developer's local workspace, before the changes are committed to the repository, it can show when developers are working on the same artifacts at the same time.

We have utilized WAV to visualize ongoing activities in an open-source project and observed the social aspects of development evolving. For example, we observed that a particular developer started the project, but then become inactive. Later, new developers joined the project and one of them becomes the new lead on the project and attracts additional developers. Some developers primarily contributed graphics for the user interface, while others focused on actual code.

2.2 Ariadne

Ariadne is a visual collaborative software development tool that highlights the socio-technical relationships between source-code artifacts and the developers implementing those artifacts. Based on theoretical predictions and empirical studies (including our own), we have observed that dependencies in source-code artifacts create dependencies between the developers implementing those artifacts. In response, we created Ariadne, a tool that exploits the automated identification of dependencies in the source-code and the automated mining of development teams' CM repositories for authorship information. The tool utilizes these exploitations to infer social dependencies between developers.

A social dependency can be described as a dependency between developers as a result of the calls to each other's code. Awareness and understanding of these social dependencies is a critical first step in coordinating work efforts from both a managerial and development perspective.

Our previous field studies of collaborative software development [6] highlight the fact that software development demands awareness of the activities being conducted as well as knowledge about who to talk to regarding these activities (even when the right people are not available). Similar studies have confirmed these results [8][4]. One class of findings in our studies indicates a need for managers and developers to know which team members have started to integrate their code and when they have done so. Similarly, we found that developers needed to know when other developers began to make calls to their code so that they would have time to make final changes before a release. Secondly, our fieldwork shows that developers often spend a good deal of time trying to determine who knows what information.

Ariadne's visualization is intended to aid users in finding answers to these very questions. The tool's interface displays a project's socio-technical dependencies, relating people to the artifacts they use, arranged vertically along the longest screen dimension. Developers and managers can improve their coordination strategies and execute development activities with greater awareness by making socio-technical relationships explicit in the visualization rather than leaving them implicit as they exist in a repository. Alternatively, an absence of a critical dependency, such as an integration of two large subsystems, would serve as a red flag to developers and managers, indicating that immediate action should be taken.

We have performed a preliminary evaluation of Ariadne's visualization using multiple inspection methods appropriate for visual interfaces. The initial results suggest a need for clearer status and feedback mechanisms, such as cues indicating that the user can hover over certain elements to see more detailed information, as well as features that allow the end user to configure elements of the visualization in meaningful ways. We hypothesize that by allowing the individual to arrange the visualization in a way that closely matches a given project's configuration of social and technical resources, the better it will inform coordination decisions. In our future work, we plan to test this hypothesis by visualizing real-world software development projects and showing the results to software development practitioners.

2.3 World View (WV)

World View is a software development tool designed to support distributed development. The tool provides managers, team leaders and developers in general with a central repository that can derive, retain and visualize the structure of distributed teams, the availability of its members and their locality. Existing coordination tools fueled its development process in addition to advent of new technologies that allowed us to use the world map metaphor to present team dynamics and social interdependencies in globally distributed projects at varying levels of abstraction.

The tool derives the social interdependencies among teams from a number of sources, such as IDEs, CM systems. It visually represents each derived dependency as a color-coded line between two dependent teams at the highest level of abstraction or between two team members at a lower level of abstraction. A developer can utilize the tool's "zoom" feature to move from one level of abstraction to another. The thickness of the lines represents the extent of sharing and consequent social-interdependence: the thicker the lines, the larger the number of shared artifacts.

The tool also visualizes the structure of development teams to promote awareness of developers and their organizational roles (displayed by "mousing over" a site). Developers can determine the availability of others on two levels of abstractions, namely: regional and local. WV visualizes the availability of teams within a specific region by shading a region in the world map, thus showing time zones and national holidays. The availability of specific team members can also be determined by "zooming" in to a particular team member, as each team member is represented visually by a single node. Each node is labeled with the team member's name and includes their contact information, role within the team in addition to their availability (e.g. online, busy or offline).

WV demonstrates the feasibility of providing a visual contextualized representation of inter-dependencies, roles, locality and availability in an integrated environment by implementing CC principles of non-obtrusive integration, socio-technical issues, and multiple views. The tool's design and implementation contributes to developers' awareness about co-workers. The need for such awareness has been identified as one of the most frequently sought information [8].

We are planning on working with industry partners to evaluate the tool in real world setting. We also plan to conduct controlled experiments in order to assess its usefulness and gather information on how to improve it. Future work includes integrating World View with different tools that provide additional project-related information, such as issue trackers and task management systems.

3. A HUMAN AND COOPERATIVE RESEARCH PROCESS

The CC paradigm and each of the tools discussed above all address different aspects of the human and collaborative nature of software development. However, in reflecting on the above concepts and systems, we realized that our research process also represented, and perhaps necessarily so, a human and even cooperative approach. Although human-centered design [12] and socio-technical design [16] are not novel concepts, they represent a meta-issue not to be overlooked in this workshop. These approaches vary greatly nowadays. Our systems and concepts provide a valuable data point and example for other researchers in the area to consider.

A key component to our research approach is observation in the field. During the development of Ariadne, two key field studies were performed over consecutive years, each 2-3 months in duration. The first year's study provided insight into our defining *continuous coordination*, revealing the critical nature of awareness and informal communication in collaborative software engineering. The second reified the ideas from the first, but brought out in both sets of data the role of dependencies in coordinating work. These summaries are glossing over many details, but the important point is that no system was proposed until after the second field study. The system arose out of the field observations similar to a *grounded theory* approach [17].

Like many systems, Ariadne went through iterative development. As the interface stabilized, it could be demonstrated to colleagues to make general suggestions. However, to keep the evaluation and further refinement linked to human needs, we adapted usability inspection methods [10] and cognitive theories [13] to evaluate the interface against typical usage tasks observed in the original field studies. We were able to make use of cognitive walkthrough [19], cognitive dimensions [13], heuristic evaluation [11], and general visualization principles [18]. Inspection methods are not a substitute for eventual trials with real users, but they play a critical role that subject evaluations can not always support. Namely, they provide a rationale or even a "mapping" between system features / design choices and human users' needs. Observations with human subjects provide kinds of performance data, but in complex tasks such as software engineering, often leave the rationale for the performance slightly speculative.

In many cases, an idea for a system comes from a general problem known to the community. Such was the case for the WAV system. It is commonly known that different developers contribute different amounts of code, at different times, over the lifetime of a project during software development (and particularly open source projects). WAV visualizes the history of development. WAV went through iterative development however. The iterations involved a kind of evaluation. Parts of WAV were compared to related projects in visualization research. A rough characterization of this refinement process would be to competitive usability analysis [9]. As the purpose of the interface emerged, related systems become a comparison point.

WV development was initiated as a result of observing general problems in the real world (typically associated with distributed development) and motivated by advances in technology. The iterative, and ongoing, development process is driven by the need to overcome the challenges reported in related literature and the integration of new tools as they emerge. We have made use of general visualization principles and plan further evaluations through cognitive walkthroughs. Consequently, it is possible to conclude that WV evolved as a result of a hybrid of the approaches adopted to develop both WAV and Ariadne.

We have also performed and intend to perform more trials with real world data and in real world settings to complete a human-centered approach. We applied both Ariadne and WAV against open source projects and gathered feedback from industry partners to assess the usefulness of WV. In some cases, the results of the application were provided to participants in those projects for comment. We consider these steps towards a complete human-centered approach. Influencing real world developers to adopt research tools is an ongoing challenge in the research community. However, future work we are undertaking includes integrating our tools with Jazz-based applications. Conducting this up-stream integration can increase our chances for real world adoption.

In the beginning of this section, we mentioned that our research process was both human-centered and *cooperative.* The human-centered aspect was emphasized in the intervening paragraphs. The *cooperative* dimension in some senses really is the cooperation between researchers, subjects, and field sites. We would like to suggest, in the sprit of socio-technical design, that our systems are ultimately for improving the quality of experience for real end users, as well as the quality of the process and product.

4. ACKNOWLEDGMENTS

This research was supported by the U.S. NSF under grants 0534775, 0326105, 0093489, and 0205724.

5. REFERENCES

[1] Almeida da Silva, I., M. Alvim, R. Ripley, A. Sarma, C.M.L. Werner, and A. van der Hoek, *Designing Software Cockpits for Coordinating Distributed Software Development*, First Workshop on Measurement-based Cockpits for Distributed Software and Systems Engineering Projects, August 2007, 14–19.

[2] Brooks, F., No silver bullet: essence and accidents of software engineering, *Computer 20:4*, Apr. 1987, 10-1.

[3] Carmel, E. and Agarwal, R. 2001. Tactical approaches for alleviating distance in global software development. *IEEE Software. 18, 2* (Mar. 2001), 22-2.

[4] Cataldo, M., Wagstrom, P.A., Herbsleb, J.D. and Carley, K.M. Identification of Coordination Requirements: implications for the Design of Collaboration and Awareness Tools, *20th Conference on Computer Supported Cooperative Work*, ACM Press, Banff, Alberta, Canada, 2006.

[5] de Souza, C. R., Redmiles, D., Cheng, L., Millen, D., and Patterson, J. 2004. Sometimes you need to see through walls: a field study of application programming interfaces. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work* (Chicago, Illinois, USA, November 06 - 10, 2004). CSCW '04. ACM, New York, NY, 63-71.

[6] de Souza, C.R.B., Quirk, S., Trainer, E. and Redmiles, D. Supporting Collaborative Software Development through the Visualization of Socio-Technical Dependencies , *ACM Conference on Supporting Group Work,* ACM Press, Sanibel Island, FL, 2007.

[7] Dunbar, R.I.M The social brain hypothesis. *Evolution. Anthropology.* 19986: 178-190.

[8] Ko, A. J.; DeLine, R.; Venolia, G., Information Needs in Collocated Software Development Teams," *Software Engineering, 2007. ICSE 2007. 29th International Conference on* , pp.344-353, 20-26 May 2007

[9] Nielsen, J., *A home-page overhaul using other Web sites2,* IEEE Software, V. 12, N. 3, May 2995, pp. 75-78.

[10] Nielsen, J., Mack, R., *Usability Inspection Methods*, John Wiley & Sons, Inc., New York, 1994.

[11] Nielsen, J.K. Heuristic Evaluation., in J. Nielsen, R. Mack (eds.), *Usability Inspection Methods*, John Wiley & Sons, Inc., New York, 1994.

[12] Norman, D.A., Draper, S.W. (eds.), User Centered System Design, New Perspectives on Human-Computer Interaction, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[13] Petre, M., A. Blackwell, T. Green, *Cognitive questions in software visualization, in Software Visualization: Programming as a Multi-Media Experience*, MIT Press, Cambridge, MA, 1997, pp. 453–480.

[14] Redmiles, D.; van der Hoek, A.; Al-Ani, B.; Hildenbrand, T.; Quirk, S.; Sarma, A.; Filho, R.S.S.; de Souza, C. & Trainer, E., *Continuous Coordination: A New Paradigm to Support Globally Distributed Software Development Projects*. In: Wirtschaftsinformatik Journal, Vol. 49 (2007), S28–S38.

[15] Scacchi, W., Socio-Technical Design, in W. S. Bainbridge (ed.), *The Encyclopedia of Human-Computer Interaction*, 656-659, Berkshire Publishing Group, 2004.

[16] Scacchi, W., Socio-Technical Design, in W. S. Bainbridge (ed.), *The Encyclopedia of Human-Computer Interaction*, 656-659, Berkshire Publishing Group, 2004.

[17] Strauss, A. and J. Corbin (1998). *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. SAGE Publications, Thousand Oaks, CA.

[18] Tufte, E., *Beautiful Evidence*. Graphics Press, Cheshire, CT.. 2006.

[19] Wharton, C., Rieman, J., Lewis, C., Polson, P., *The Cognitive Walkthrough Method: A Practitioner's Guide*, in J. Nielsen, R. Mack (eds.), Usability Inspection Methods, John Wiley & Sons, Inc., New York, 1994.