This is a post-peer-review, pre-copyedit version of this article. The final authenticated version is available online at: https://doi.org/10.1109/CHASE.2012.6223024

ProxiScientia: Toward Real-Time Visualization of Task and Developer Dependencies in Collaborating Software Development Teams

Arber Borici*, Kelly Blincoe†, Adrian Schröter*, Giuseppe Valetto† and Daniela Damian*
*Software Engineering Global Interaction Lab
University of Victoria, Victoria, BC, Canada
borici@uvic.ca, schadr@acm.org, danielad@cs.uvic.ca
†Department of Computer Science
Drexel University, Philadelphia, PA, USA
kelly.blincoe@drexel.edu, valetto@cs.drexel.edu

***Abstract*—This paper introduces ProxiScientia, a visualization tool that provides awareness support to developers, as they engage in collaborative software development activities. ProxiScientia leverages streams of fine-grained events that are generated by team members as they interact with software artifacts in their development environments. The main goal of the tool is to make each developer aware of coordination needs and opportunities as they arise, by depicting ego-centered views of the developers and tasks that most closely impact their work, and showing how they change in real time. In this paper, we illustrate the conceptualization of ProxiScientia and discuss its initial evaluation.**

***Keywords*-Socio-technical; task context; visualization; tools; proximity; awareness; collaboration**

I. Introduction

In contemporary software development projects, the efficient coordination of concurrent activities among team members remains a challenge. The detection of existing and emerging coordination needs (also called Coordination Requirements, or CRs) among team members is difficult, especially in large-scale projects with many work dependencies that evolve dynamically. Furthermore, it is critical that detection of Coordination Requirements be timely, so that developers can become aware of, and possibly act upon, them as early as possible. We call this form of awareness "coordination awareness." Achieving and maintaining coordination awareness is already complex in medium-scale software projects. It becomes daunting in large-scale projects with hundreds of teammates, thousands of tasks and artifacts, and countless acts of work occurring concurrently, often dispersed across multiple geographical workplaces [1].

Research on the socio-technical aspects of Software Engineering has produced methods to detect Coordination Requirements and proposed a metric, called Socio-Technical Congruence (STC), which indicates how many of those CRs get fulfilled by a software organization [2]. Empirical evidence suggests that high levels of STC are beneficial to developer and project productivity [3], and therefore important for the efficient governance of collaborative software development [4]. However, the timely and actionable detection of CRs in a way that enables developers to recognize and resolve them early and quickly is still problematic, since most of the proposed CR detection methods have been reliant on mining software development traces in project repositories, such as source control and bug tracking systems, and are therefore retrospective. Such historical data cannot provide for actionable coordination awareness. To that end, Blincoe et al. [5] recently proposed a technique, which allows detection of CRs as they emerge, and provides a quantitative relationship, called *proximity*, indicating the strength of the need to coordinate between pairs of developers or pairs of development tasks. Proximity can be thus used to elaborate ranks and distances between developers or tasks.

Building on the proximity concept, we have begun to develop *ProxiScientia*, a tool that gathers data from the work environment of developers, continuously computes proximity, and presents to each developer appropriate visualizations of her coordination needs. ProxiScientia displays an ego-centered view of all other tasks that may impact a given developer's current work, or all developers with whom it may be important to collaborate.

In the remainder of this paper, we first provide background on Coordination Requirements and proximity. Then we discuss other visualization tools that aim at supporting coordination awareness in software engineering. Next, we describe the conceptualization of ProxiScientia and illustrate the design decisions for its current prototype. We then discuss the strategy for its evaluation, together with some early results. Finally, we offer our conclusions and future work.

II. Background and Related Work

Software development environments are moving towards more team-based environments and away from the traditional individual development environments. Perhaps the most prominent example of a team-based environment is IBM Jazz® [6]. Jazz facilitates collaboration among developers of the same project. For instance, developers can view

© 2012 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

the feed of recent activity or start a chat with teammates.

However, Jazz and other team-oriented environments on the market do not explicitly make developers aware of emerging work dependencies and coordination needs. Recent case and field studies underline the necessity of understanding social and technical relationships in complex collaborative software projects [5], [7]. A visualization tool providing this awareness has two main objectives. First, the tool should accurately detect work dependencies between developers and tasks that originate coordination needs. Second, the tool should enable developers to address those needs efficiently. In what follows, we review the relevant literature in light of these two objectives.

A. Detecting Coordination Needs in Software Development

The analysis and support of work dependencies and coordination in software development teams originates with Conway [8] and Parnas [9], who recognized that it is possible to reduce coordination needs by minimizing technical dependencies between software components under the responsibility of different developers or teams. More recently, Baldwin et al. [10] showed how modular software design enables more parallel development, paving the way for larger development teams. With scale comes an increased need to coordinate developers through organizational structures, processes, and communication and coordination mechanisms [7]. A method to detect Coordination Requirements (CR) [2] led to the development of the Socio-Technical Congruence (STC) index, which compares acts of coordination actually carried out vs. perceived coordination requirements [3]. The authors mine archival data to compute CRs and STC, which is more useful for retrospective analysis rather than for providing coordination awareness.

Recently, Blincoe et al. have developed *proximity*, a method that reveals how close the work contexts of developers or tasks are to one another by considering an intersection of the corresponding artifact working sets and the actions developers have with those artifacts, weighted based on the type of interaction [5]. Our tool, which can visualize CRs in real time as they occur during a software project, is based on the proximity concept and metric.

B. Visualization Tools that Support Coordination Awareness

Other works have addressed the visualization of socio-technical relationships in a software project. Tukan [11] is an online Smalltalk plug-in that conducts semantic analysis on programs to determine which artifacts are related. It provides awareness for developer activities as well as potential conflicts. The visual presentation involves a graph, wherein nodes may represent artifacts (such as classes or methods) or relationships (such as inheritance or composition). Augur [12] employs program analysis to visualize developer activity in temporal and contextual details in distributed development teams. It presents a listing of program lines in code artifacts with a zoom option. Ariadne [13] visualizes social and code dependencies, whereas Tesseract [14] builds upon Ariadne by integrating information from multiple communication repositories. FastDASH [15], which targets mainly small teams, notifies developers of artifact changes and provides a dashboard showing all concurrent activities happening within the team. It does not, however, analyze those activities to infer and abstract coordination requirements. All of these tools obtain information about artifact changes from the configuration management system of the project. The drawback of such tools is that fresh data is not available until code has been committed. For that reason, the awareness they provide to developers may not be actionable, since when code is committed much of the development work is often already completed.

Palantir [16] has a different approach, since it monitors both the centralized code repository and the workspace of individual developers. It visualizes in real-time differences between code files in a developer's workspace and those same files in other workspaces. While Palantir can provide timely notifications as soon as these conflicts appear, they represent only a small subset of coordination requirements that can emerge between a pair of developers.

Like Palantir, RaisAware [17] monitors edit events in the developer's workspace. It expands the reach of Palantir by detecting and showing visualizations of direct same-file conflicts as well as indirect work dependencies that may originate from both syntactic dependencies between software artifacts and logical dependencies based on the "files changed together" heuristics, derived from past history of commits. RaisAware visualizes complex networks and artifacts linked together, which are valuable for exploration of the common work environment, but does not try to synthesize or quantify CRs. It also looks only at edited artifacts, and not at the whole working set, which also includes consulted artifacts.

C. The Proximity Algorithm

Proximity is a measure of similarity between developers' activities expressed in terms of overlaps between their *task contexts*. A task context aggregates the working set of artifacts for a task, together with the metadata about the developer's interactions with those artifacts. These interactions can be captured by tools such as Mylyn [18], which records interactions like artifact modification and consultation as they happen. To compute proximity between a pair of developers A and B, the algorithm collects all events produced by A and B into two working sets, WA and WB. The proximity between WA and WB is then computed by applying weights to each artifact that overlap in the two sets. The weights differ based on the type of interactions. A higher weight is assigned to modification, and alower, but still significant, weight to consultation. Complete details on proximity can be found in [5].

III. Conceptualization of a proximity visualization tool

We have leveraged a framework proposed in [1] to establish the goals of ProxiScientia, define its stakeholders and decide how to best translate the coordination insight provided by the proximity into effective visual awareness support. The framework has guided the elicitation of the tool requirements and the implementation of the current prototype.

The framework comprises five dimensions: intent, information, presentation, interaction, and effectiveness. Intent describes the motivation behind the proposed tool. The information dimension captures the data sources which will be used by the tool. Presentation entails the way the information will be visualized for viewing by the stakeholders. It also captures the various visualization variables and techniques that will be employed. The interaction dimension captures the temporal operability of the tool (online or offline; real-time or not) along with the different user interaction modes. Finally, effectiveness refers to the usability of the tool and the proposed evaluation.

A. Intent

ProxiScientia is aimed at visualizing CRs in real time to maintain and increase awareness of coordination needs and opportunities in distributed software development projects. Developers are the main users of the real-time visualizations. For any complex collaborative software project, our tool must help the individual developer become aware of coordination needs induced by dependencies of her current task with other ongoing tasks. Becoming aware of these CRs in a timely fashion can result in more streamlined and efficient coordination, and can save developer's time. For example, by eschewing communication breakdowns, awareness can result in more productive and organized software development and improved quality [19]. Also project managers are ProxiScientia stakeholders. The tool can help managers to alert developers of the need to coordinate their activities before complications arise. The tool can also be integrated with existing decision support systems to streamline the development process and its governance.

In terms of cognitive support, ProxiScientia reveals *who* is working on *what* in the context of the current project, and aims at minimal disruption of the developer's mental context. It can also be extended, for example to provide insight on the rationale and chronology of developer activities that are carried out on a working set of artifacts.

B. Information

ProxiScientia is based on the proximity algorithm [5]. Proximity extracts context data as each developer interacts with her development environment at different levels of artifact granularity—files, classes, and class elements. The amount of data generated this way can become quite significant in a large-scale software project, as many developers work concurrently on a number of tasks. ProxiScientia deals with such a large amount of information by filtering the vast majority of it behind the scenes, and synthesizing a simple numeric measure. That measure must offer each developer adequate cognitive support for her coordination awareness needs and must be useful for the presentation of results pertinent to that developer.

C. Presentation

The tool uses awareness information in two ways: to build developer-centric and task-centric proximity clouds, i.e. graphical representations of the proximity semantic relationships. In particular, we consider ego-centered views through radar graphs (already implemented) and heat maps (conceptualized, but not yet implemented).

For the developer-centric view, the tool mines data relevant to the given developer for the task in focus to automatically and periodically compute proximity with all other developers in real time. The view is a radar graph with the current developer positioned in the center. The edges denote the proximity relationships of the current developer to all other developers. This view may also be construed as a user-centric social network. Figure 1 illustrates a developer-centric view for developer `Sue`. It shows, for instance, that `Kate` and `Jack` are the two developers closest to Sue.

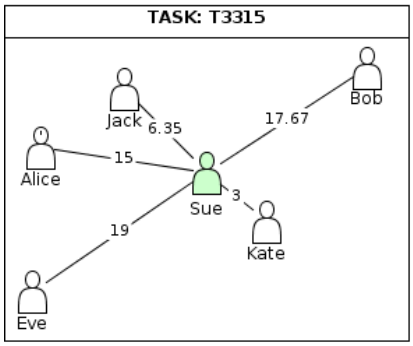

Figure 1. Developer-centric radar view of proximity relationships for task `T3315`. The view is generated for `Sue`, highlighted in green.

The task-centric view is also visualized as a radar graph but with the current task positioned in the center and all related tasks revolving around the central task. The edges, again, represent the proximities between the current task and all relevant tasks computed in real time. A sample task-centric view is shown in Figure 2. Higher proximity values indicate greater need to coordinate. The visualizations display the reciprocal values of the scaled proximity values on the edges, for a more intuitive representation, based on the distance between developers' or task icons. The tool visualizes only the closely related developers/tasks that may need coordination. Unrelated developers/tasks are left out

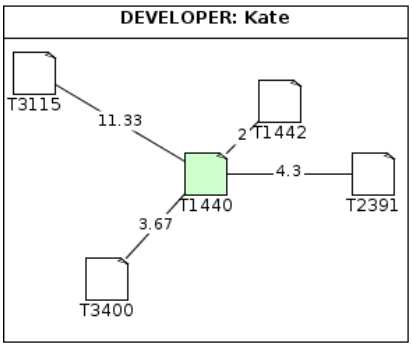

Figure 2. Task-centric radar view of proximity relationships for developer `Kate`. The current task in focus, `T1440`, is highlighted in green.

of the visualization to minimize the amount of information presented. Also, in its present implementation, ProxiScientia filters out developers or tasks whose proximity value is larger than two standard deviations from the average proximity; this filtering is a default that must be validated and could be changed as an individual preference setting.

Besides radar graphs, we envision that future versions of ProxiScientia must include additional views to satisfy the needs of all of its stakeholders, from developers to managers to testers. We are considering a heat map view on top of a spatial representation of the working set of a user or a task. Each concurrent activity that intersects with that working set must yield heat, proportional to the proximity score calculated between that activity and the current user or task. The various regions on the heat map shall thus vary in luminosity over time. In addition to this visual feature, the heat map shall also provide contextual information upon request by being interactive. The user can click on the heated regions for additional information, or even to observe the stream of events other developers have been generating relative to those artifacts.

D. Interaction

As entailed in the intent dimension, ProxiScientia aims at providing cognitive support in real time. The tool periodically updates and stores context information automatically and recomputes proximity for the current developer and the task in focus based on the context information. In a future release, it will also allow the user to explicitly request additional information from the tool, for example, by hovering over a particular area in the proximity cloud to view detailed annotations or locally generated reports.

As with any awareness tool, there is a tradeoff between providing enough information to make the user aware and minimizing disruptions to the user's work. Interaction with the tool could pose a contingent disruptive interruption threat to the user's prospective memory, which entails remembering a task that needs to be done in the future [20] (cited in [21]). In particular, the disruption has the largest effect on workers whose information workload is especially large and critical. Other variables such as task complexity and duration, as well as the number of interruptions, must be taken into consideration when evaluating the effectiveness of the tool. We present below a relevant cognitive support strategy that attempts to address the awareness vs. disruption tradeoff and to minimize the interruption threat.

1) The "Push and Pull" Cognitive Support Strategy: In ProxiScientia we want to establish a dialectic relationship between the developer and her work environment as the main form of cognitive support towards coordination awareness. We envision to that end a "push–pull" strategy, according to which ProxiScientia users can be presented with timely and relevant awareness information quickly and with minimal disruption, but are also enabled to explore, seek more detail, and hence gain a deeper understanding about coordination from the awareness information that has been pushed to them. In the push mode, ProxiScientia enriches the developer's experience and environment by proactively presenting cues that the tool deems relevant for the developer's coordination needs: the radar views described in the Presentation dimension above represent instances of the push mode. The complementary pull model, which is not yet fully implemented, enables the developer to follow the cues presented to her, drill deeper, and request additional information. For example, the ability to interact and click on the heat map visualization is an instance of the pull mode.

Developers will also be given the option to customize presentation settings, such as the refresh rate at which the tool conveys visual information about proximity, and the filter threshold for proximity scores. ProxiScientia can be integrated into the developer's work environment as a plugin. Currently, the ProxiScientia prototype is implemented as a plugin for the Jazz platform (also known as Rational Team Concert), whose development tool suite heavily borrows from the Eclipse IDE. When the developer logs onto Jazz, ProxiScientia loads the developer-centric and task-centric views with the most recent context data and then refreshes the view in real-time. Context data is obtained through the Mylyn plugin [18] for the Eclipse IDE and is sent to a centralized database on the server. The server also computes the proximity scores among all users, by default every 30 seconds, and sends them back to the visualization plugin which acts as the client. This way, the ProxiScientia visualization can also be deployed as a stand-alone application for stakeholders who do not employ an IDE such as a project manager.

E. Effectiveness

Two key aspects of effectiveness are the scalability and the evaluation of the visualization tool. A detailed discussion on the evaluation of the tool is found in the next section. In regard to scalability, the awareness provided by ProxiScientia must be beneficial to small, medium, or large software development teams alike, in the face of their diverse

coordination requirements [22]. The key threat to scalability is the vast amount of context data generated throughout the development cycle of a given project that is going to be fed to the server for proximity calculation. The handling of that data and the computation of pairwise proximity scores and the corresponding coordination relationships must remain efficient. There may be a trade-off between the frequency with which that computation occurs and the number of pairs in the system to which proximity must be applied. A related aspect is determining a minimum value of the proximity metric that can be used as a threshold of significance for the coordination needs of pairs. That threshold can be seen as a tuning parameter for ProxiScientia, which needs to be determined via empirical validation.

F. Comparison to Other Tools

We have used the same framework that has guided us in conceptualizing the tool to carry out a side-by-side comparison of ProxiScientia with three closely related visualization tools: Tukan, Palintir, and Tesseract (see Table I). The salient findings are reported below.

In terms of intent, ProxiScientia provides timely cognitive support to a wider audience of stakeholders than current existing tools. It provides authorship information for changes to artifacts, but does not provide a rationale for those changes, such as does Tesseract, for example. In terms of information, ProxiScientia does not conduct semantic analysis of program code or make use of syntactic units like the other three tools. Instead, its CR conceptualization descends from contextual data extracted from the working sets of developers and their artifact interactions, which allow it to visualize developer or task dependencies in real time. All of the tools visualize awareness information graphically. ProxiScientia uses ego-centered networks (radar graphs) and heat maps. The evaluation of the current prototype of ProxiScientia, which has been developed in accordance with the conceptualization we have just described, is under way as we write this paper. We report some early results and our overall evaluation strategy in the subsequent section.

Table I
SUMMARY OF PROXISCIENTIA AND OTHER AWARENESS VISUALIZATION TOOLS. THE TABLE AND THE DATA ABOUT TUKAN AND PALANTIR ARE ADAPTED FROM [1].

	ProxiScientia	Tukan	Palantir	Tesseract
Team size	Any	Any	Any	Any
Developer	Yes	Yes	Yes	Yes
Manager	Yes			
Tester/Documenter	Yes			
Researcher	Yes			Yes
Present	Yes	Yes	Yes	
Recent past	Yes	Yes	Yes	Yes
Historical				Yes
Authorship	Yes	Yes	Yes	Yes
Rationale				Yes
Syntactic units		Yes	Yes	Yes
Semantic Analysis		Yes		Yes
Context Data	Yes			Yes
Online	Yes	Yes	Yes	
Customizable	Low		Low	Low
Zoomable	Yes			Yes
Hypertext				
Graphical	Yes	Yes	Yes	Yes
Graph view	Yes	Yes	Yes	Yes
Other views	Yes			Yes
Visual variables	Yes	Yes	Yes	Yes
Animation	Partial			
Availability			Yes	
Adopted				
Case Study		Yes	Yes	Informal
User Study	In progress			

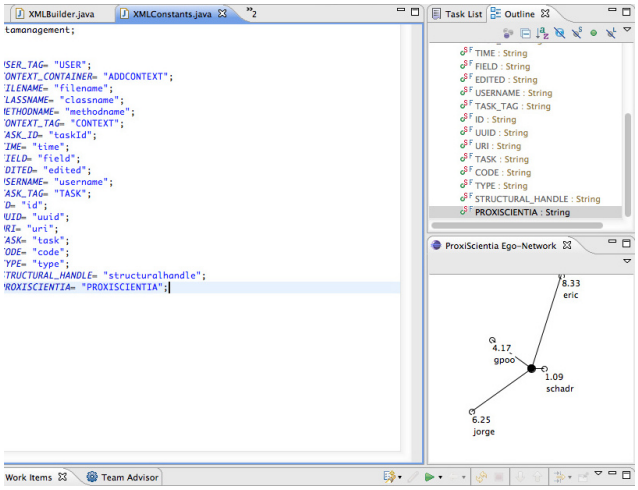

Figure 3. A cropped screenshot of ProxiScientia integrated in the Eclipse IDE: User-to-user proximities displayed in the lower-right view.

IV. EVALUATION

As a first step in evaluating ProxiScientia, we carried out an expert judgment study involving six participants having strong background knowledge of the Eclipse IDE and experience in distributed software development. A sample screenshot of the simulated proximity visualizations that we employed for that study is provided in Figure 3. According to the short survey we conducted, ProxiScientia was considered useful to maintain awareness in terms of both tasks and developers by finding related entities. The responses also showed that most users do not believe ProxiScientia poses a large disruptive interruption threat. To further evaluate ProxiScientia we have designed a controlled field study ending April 2012. The study focuses on an agile development project that will be completed during a course on global software development held the University of Victoria, Canada, in conjunction with a similar course held at Aalto University, Finland. Students of both institutions are placed into three geographically distributed teams, which need to collaborate to complete one unified project. Each team is assigned user stories of similar complexity and type. The user stories introduce both intra-team and inter-team work

dependencies. We hypothesize that the increased awareness of coordination needs provided by our tool will increase productivity and lower coordination issues. To validate that principal hypothesis, and control for any differences and idiosyncratic properties of the three teams, we intend to vary the project iterations (agile "sprints") during which each team is allowed access to the tool visualizations. The course project will include five two-week sprints.

All teams will be be introduced to the tool, its purpose and its semantics during the first sprint, which will serve as a "practice" sprint to allow the teams to get familiarized with the whole set of processes, tools and best practice they need to adopt in the context of a globally distributed software development project. Following that first sprint, one team will be given access to ProxiScientia during sprints two and three and another team will be given access to the tool during the last two sprints. The remaining team will not be given access to the tool and will serve as a control group throughout the course of the project. We will collect context events, proximity scores and communication traces during the sprints. We will also collect user surveys after each sprint to elicit user feedback on their perceived coordination needs, as well as the use of the tool. Our goal is to answer the following research questions:

RQ1: Does the support by ProxiScientia make CR an actionable concept?

This research question aims at determining if ProxiScientia provides timely coordination awareness to team members. We will be collecting context data and computing proximity for all three teams regardless of their use of the tool. To answer this research question, we will use the earliest time at which proximity indicates a coordination requirement for each pair of developers. We will look to see if the time between the emergence of a CR and when a developer acts upon it is less during those sprints in which members of a team are equipped with ProxiScientia, and hence provided with a visualization of their CRs.

RQ2: Does the coordination awareness offered by ProxiScientia make a difference in the SW project?

For this question we will look comparatively at various productivity measures, such as the time to complete an issue, for the three teams and check whether productivity is increased by the use of ProxiScientia.

V. Discussion

The design of ProxiScientia aims at fostering coordination awareness of software developers with real-time visual cues in order to make the concepts of CRs and STC actionable.

We are in the process of carrying out a field study to corroborate and validate ProxiScientia against that goal, and to evaluate the benefits it could bring to individual developers and the development team as a whole. In the meanwhile we are considering to extend the features and support capabilities of ProxiScientia in the following directions:

A. Customizability

At its present state, ProxiScientia is displayed as a bi-color graphical Eclipse plugin with two available views: developer- and task-centric radar graphs (Figure 3). The nodes of the graph are denoted with simple circles annotated with a unique entity label (such as the developer's name or task ID) and a proximity value. The tool should allow the user to customize the appearance of the graph, as well as the way labeling is displayed or how much information the labels reveal. For instance, a developer may choose to hide proximity values. Avatars (or small photos) could also be an available choice of customization. For example, developers working with the IBM Jazz® environment can choose to use the Jazz photos of participating developers. That could induce more awareness, given the personal contacts and acquaintances of team members and their social implications.

B. Faster Coordination

ProxiScientia should enable an option for quickly initiating explicit coordination actions between developers whenever coordination needs arise. This could be achieved by integrating synchronous communication mechanisms (e.g. instant messaging, audio and video chats, and screen sharing) or asynchronous ones (e.g. emails or discussion boards). To enable these forms of communication for coordination, a simple drop-down menu could be generated as soon as the user clicks on a particular node on the ego-centric graph. These features have already been experimented with in the Jazz platform, although they were not associated to any cognitive support dealing with the concept of CRs. Therefore it should not difficult to integrate them with the ProxiScientia prototype, which is also a Jazz plugin.

C. More Contextual Information

Currently, ProxiScientia provides information about the nodes of the developers or task view in a static way. Only the value of the proximity relationship is dynamic, as it is frequently refreshed to remain up to date. Therefore, ProxiScientia can be enriched with the ability to allow the user to hover over a node on the ego-centric network to retrieve more contextual information. This is in accord to the push–pull cognitive support strategy discussed in the tool conceptualization section above, and will enables the user to "drill down" on demand and pull additional data about a CR, once she believes ProxiScientia has brought an important piece of awareness information to the forefront. This feature can be integrated with the explicit coordination support described above, to speed up the resolution of coordination requirements. Finally, we are considering adding a menu of additional tasks, whereby an ego-centric graph can be

generated for gathering more contextual information of other ongoing activities besides the task in the current work focus of the user. This could support project managers in enacting efficient coordination of developers. Further effectiveness studies need to be conducted in order to test the viability of the features and properties proposed here.

VI. Conclusions

ProxiScientia is a visualization tool that provides cognitive support to coordination awareness in software projects, by providing developers with information about the presence and strength of work dependencies they have with co-workers. This tool has the potential to improve the management of coordination needs that emerge—but often remain undetected until they have percolating damaging effects—in large-scale or distributed software development projects. ProxiScientia is based upon the proximity metric, which is a new, timely way to conceptualize coordination requirements and measure their intensity.

We have presented ProxiScientia in terms of a formative assessment framework for visualization tools in software engineering, which allowed us to motivate and describe the design choices we have made with respect to five dimensions: its intent, the information it consumes and presents, its presentation strategy, its interaction with the intended users, and how its effectiveness can be explicated and evaluated.

A prototype of ProxiScientia has been developed as a visual plugin to IBM's Rational Team Concert collaborative development environment, also known as Jazz. A preliminary evaluation has been conducted, and a controlled field study in the context of a distributed agile development project is under way.

References

[1] M.-A. D. Storey, D. Čubranić, and D. M. German, "On the use of visualization to support awareness of human activities in software development: A survey and a framework," in *Proc. ACM Symposium on Software Visualization*, 2005, pp. 193–202.

[2] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley, "Identification of coordination requirements: implications for the design of collaboration and awareness tools," in *Proc. 20th Anniversary CSCW*, Banff, Canada, 2006, pp. 353–362.

[3] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-technical congruence: A framework for assessing the impact of technical and work dependencies on software development productivity," in *Proc. 2nd ACM-IEEE Int. Symposium on Empirical Software Engineering and Measurement*, Kaiserslautern, Germany, 2008, pp. 2–11.

[4] M.-A. D. Storey, L.-T. Cheng, I. Bull, and P. Rigby, "Shared waypoints and social tagging to support collaboration in software development," in *Proc. 20th Anniversary CSCW*, Banff, Canada, 2006, pp. 195–198.

[5] K. Blincoe, G. Valetto, and S. Goggins, "Proximity: a measure to quantify the need for developers' coordination," in *Proc. CSCW*, Seattle, WA, 2012, pp. 1351–1360.

[6] L.-T. Cheng, S. Hupfer, S. Ross, and J. Patterson, "Jazzing up eclipse with collaborative tools," in *Proc. OOPSLA Workshop on Eclipse Technology eXchange*, Anaheim, CA, 2003, pp. 45–49.

[7] M. Cataldo, M. Bass, J. D. Herbsleb, and L. Bass, "On coordination mechanisms in global software development," in *2nd IEEE Int. Conf. Global Software Engineering.*, 2007, pp. 71–80, iD: 1.

[8] M. Conway, "How do committees invent," *Datamation*, vol. 14, no. 4, pp. 28–31, 1968.

[9] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Commun. ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[10] C. Y. Baldwin, *Design Rules: The Power of Modularity*. MIT Press, 2000.

[11] T. Schümmer and J. M. Haake, "Supporting distributed software development by modes of collaboration," in *Proc. 7th European Conf. on CSCW*, Bonn, Germany, 2001, pp. 79–98.

[12] J. Froehlich and P. Dourish, "Unifying artifacts and activities in a visual tool for distributed software development teams," in *Proc. 26th Int. Conf. Software Engineering*, Washington, DC, 2004, pp. 387–396.

[13] C. R. de Souza, S. Quirk, E. Trainer, and D. F. Redmiles, "Supporting collaborative software development through the visualization of socio-technical dependencies," in *Proc. Int. ACM Conf. on Supporting Group Work*, Sanibel Island, FL, 2007, pp. 147–156.

[14] A. Sarma, L. Maccherone, P. Wagstrom, and J. Herbsleb, "Tesseract: Interactive visual exploration of socio-technical relationships in software development," in *Proc. 31st Int. Conf. Software Engineering*, Washington, DC, USA, 2009, pp. 23–33.

[15] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson, "Fastdash: a visual dashboard for fostering awareness in software teams," in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, San Jose, California, USA, 2007, pp. 1313–1322.

[16] A. Sarma, Z. Noroozi, and A. van der Hoek, "Palantir: Raising awareness among configuration management workspaces," *Int. Conf. Software Engineering*, p. 444, 2003.

[17] J. Costa, R. Feitosa, and C. de Souza, "Tool support for collaborative software development based on dependency analysis," in *6th Int. Conf. Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom),*, 2010, pp. 1–10.

[18] M. Kersten and G. C. Murphy, "Using task context to improve programmer productivity," *Proc. 14th ACM SIGSOFT*, p. 1, 2006.

[19] D. Damian, L. Izquierdo, J. Singer, and I. Kwan, "Awareness in the wild: Why communication breakdowns occur," in *2nd IEEE International Conference on Global Software Engineering*, 2007, pp. 81–90.

[20] J. Ellis and L. Kvavilashvili, "Prospective memory in 2000: Past, present, and future directions," *Applied Cognitive Psychology*, vol. 14, no. 7, pp. S1–S9, 2000.

[21] M. Czerwinski, E. Horvitz, and S. Wilhite, "A diary study of task switching and interruptions," in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, Vienna, Austria, 2004, pp. 175–182.

[22] J. M. Costa, M. Cataldo, and C. R. de Souza, "The scale and evolution of coordination needs in large-scale distributed projects: Implications for the future generation of collaborative tools," in *Proc. Annual Conf. on Human Factors in Computing Systems*, Vancouver, Canada, 2011, pp. 3151–3160.