

# Visualizing a Requirements-centred Social Network to Maintain Awareness Within Development Teams

Irwin Kwan, Daniela Damian and Margaret-Anne Storey  
University of Victoria  
Department of Computer Science  
3800 Finnerty Road, Victoria, British Columbia  
{irwink,danielad,mstorey}@cs.uvic.ca

## Abstract

*When the requirements in a software system change, we should notify every contributor who participates in the analysis, design, implementation, and testing of the requirement to reduce rework. However, the network of contributors working on a requirement is constantly changing, making it not only difficult to seek expertise from other team members, but also difficult to send requirements-change information to team members.*

*To promote communication and improve awareness among contributors working on the same requirement, in this position paper we suggest using a visual representation called a requirements-centred-social-network diagram. Using the social-network diagram, a contributor can learn about another contributor's communication patterns around the development of a requirement, or send requirements-change-awareness notifications to every member of a team working on the same requirement. This social network can automatically expand to include contributors who work on a requirement but may not have been included in a project plan. The requirements-centred social network therefore captures not only the relationships among an initial team, but also emergent relationships among peripheral contributors.*

*We believe that, by providing visual feedback of communication patterns within a contributor's expanding social network and promoting communication among team members, we can improve awareness of the work done by other contributors and maintain awareness of requirements change.*

## 1. Introduction

When building software, the software requirements specification forms the basis for the design and the code

of the system, and the final product, the software, is verified against the specification. This process of *developing a requirement* may involve the specification and analysis of the requirement, the design of a software architecture, programming, and testing. No matter what process a software project follows, a large amount of collaboration must occur among the project members, or *contributors*, to ensure that the artifacts are properly built. A significant barrier to effective collaboration is the fact that software continually evolves, and over time, more contributors are involved in the development of a requirement than initially planned [1, 5]. Communication in this situation becomes problematic due to the dynamic nature of the team. Maintaining awareness among those working on a requirement, especially requirements-change awareness, becomes difficult because notifications may not reach every contributor working on the requirement.

If changes are not promptly communicated to contributors, the costs of rework can be high [7]. To reduce the costs of rework, a contributor must be notified of changes in requirements in a timely fashion. We believe that, by facilitating collaboration among contributors, we can achieve heightened *requirements-change awareness*.

Although multiple contributors may be assigned to develop the same requirement in a project plan, they might not belong to the same team. This creates a situation where a contributor must seek information from outside of his team to develop the requirement. For example, a programmer working on a requirement may be on a programming team, whereas a tester who must write test cases for this requirement may be on a quality assurance team. The programmer may need to consult with the tester during the development of a requirement, but if a programmer does not have good *peripheral awareness*, which is awareness of what others are doing within his team, he may consult with the wrong people. Alternatively, in a project where a team is assigned per component, each team must coordinate with each other

team during integration. Ehrlich, et al, has shown that a contributor often seeks information from outside of their software team [5]; this finding supports the need for a system that provides communication links to other contributors working on the same requirement.

The outline of this paper is as follows. We introduce the requirements-centred social network and discuss the visualization method in Section 2. We discuss the challenges associated with this work in Section 3. We review the related work in visualization tools for collaborative software engineering in Section 4, and present our research agenda in Section 5.

## 2. The Requirements-centred Social Network

### 2.1. The Dynamic Nature of a Requirements-centred Team

A *requirements-centred team* is a team whose members are each working on the same requirement, be they a requirements analyst, designer, coder or tester. This network of contributors can keep close contact with each other to maintain both requirements-change awareness and peripheral awareness. A contributor can also seek specialists within this network for technical assistance regarding the requirement he is working on.

A requirements-centred team is continually expanding. A contributor external to the team may contribute valuable expertise to the development of the requirement and therefore become involved with that requirement. A project plan may also neglect to include every member of the team who works on the requirement.

### 2.2. Visualizing a Requirements-centred Team

We propose that a requirements-centred team can be represented as a *social network* that represents collaborative communication paths among contributors that are involved with the implementation of requirement. A social network is a structure where a node represents an individual, and each directed edge connecting two individual nodes indicates communication originating from one individual directed to another individual. This structure can be represented visually as a graph called a *social-network diagram*. A *requirements-centred-social network* (RCSN) is a social network that displays only contributors who are involved with the implementation of the same requirement.

The RCSN captures and displays relevant information to provide awareness on the current activities and individuals working on that requirement. The following information needs to be displayed:

- individual roles, e.g. designer, programmer, tester, manager, bystander etc, with access to additional information on their activities related to the requirement;
- the direction of the communication flow;
- the amount of information flowing from one individual to another, details on the information exchanged should be easily accessible; and
- the recency (time frame) of the information exchange.

This information may be rendered as a visual graph as follows:

- individual roles can be shown using different node shapes and colour;
- directed edges connecting individual nodes indicates the presence and direction of the communication flow;
- edge width may represent the number of communication activities;
- edge colour can be used to show the type of communication medium used (e.g. email, chat);
- node/edge positioning can be used to give a cue on the recency on the information exchange – more recent exchanges should be positioned closer to the locus of the display (similarly to the Babble tool [6]) with more temporal distant communications located at the edges of the display; and finally
- the user can access more information on the individual roles and information exchanges by brushing over or clicking on the nodes and edges in the graph.

An initial RCSN can be generated using information from task assignments in a project plan when it is available. If a project plan is not available, a simple diagram can be manually created to indicate the presence of a requirement and the key people assigned to the project. As the requirement is implemented, a contributor is added to the RCSN based on communication patterns among contributors. If a contributor consults with a programmer about the selected requirement, then the programmer is added to the network, and an edge connects the two contributors. This allows tolerance for inaccurate project plans, and further supports how a network can be updated in the absence of any planning documentation. For example, if a project plan forgets to involve the technical writer, but the technical writer communicates by E-mail with the requirements analyst about this requirement, the RCSN is updated to include the technical writer and his connection with the requirements analyst. In this way, a requirements-centred team represents a *changing network of emergent relationships*.

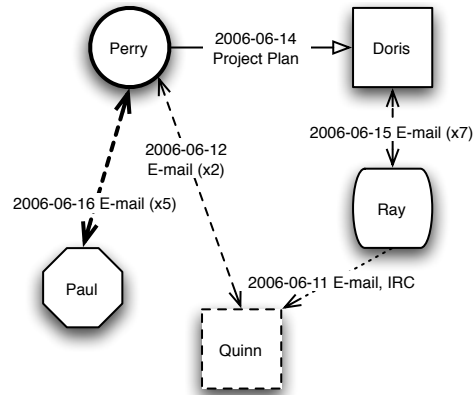
To generate the RCSN, we can analyze early requirements documentation and extract key words, possibly by utilizing a requirements template. The detection of requirement-specific communications can also be done through the mining of project repositories such as CVS and Bugzilla, email logs and chat histories. The system will monitor communication among contributors, and will scan the content of a communication message to determine if it is related to a particular requirement. If the message is related to the requirement, the system adds missing contributors to an existing RCSN.

By tracking communication patterns among contributors and automatically updating an RCSN, one can discover with whom contributors contact when implementing a requirement and maintain a record of expertise regarding this requirement. By observing the RCSN, a project manager may become more knowledgeable about who is involved in the development of the requirement, and may become more aware of a contributor's current work and skill set. A participant in the project, whether he is a manager or a contributor, can easily see who is working on a particular requirement, and can easily seek information from an appropriate expert if required. Moreover, a manager or participant is encouraged to further explore information on the status and history of the requirement's implementation by accessing this information through the social network diagram.

The RCSN diagram can be used to improve both requirements-change awareness and general awareness in the project. It can facilitate communication among contributors by providing contact information for each contributor who is working on the same requirement. A contributor can also view the diagram to observe each team member's involvement in the requirement, and therefore be aware of what the team member is doing. A requirements engineer can use the RCSN diagram to retrieve a list of everyone involved in the project so that he can easily send messages that are relevant to the entire team, such as requirements-change notifications. A project manager can use the network to identify where there may be a lack of communication between two contributors who should be working together and improve the communication infrastructure to rectify the issues.

### 2.3. Sample Visualization

An example of an RCSN diagram is shown in Figure 1. We see the communication paths among a hypothetical team working on the same requirement. Doris, a document writer, exchanges E-mail regularly with Ray, a requirements analyst. This suggests that the documentation requires a lot of advice from the requirements domain. Ray also uses IRC and E-mail to provide requirements information to Quinn, who might be writing test cases at this time. There is a miss-



**Figure 1. An example of a requirement-based-social-network diagram for a single requirement**

ing link between Perry, the project manager, and Ray, which may suggest that Perry is not very aware of requirements issues. Paul, the programmer, communicates with Perry, but may not be aware of the other members of the team.

By viewing RCSN diagrams, we can get an at-a-glance indication of what communication goes on about this requirement, and take measures to check if each member of the team is utilizing each other's skills. A diagram that has many missing links may indicate that there is a collaboration problem and that the team members are not aware of each other. We can also use the diagram to ensure that everyone is notified of a requirements change.

### 2.4. Applications to a Software Development Environment

The visualization mechanism can be integrated into a development environment that collects information in a non-intrusive manner. The diagram can be used to view information about collaboration patterns, can provide information about a contributor's expertise, facilitate communication by providing contact information, and form the framework for an automated requirements-change-notification system.

If communication regularly occurs between a team member in the initial RCSN, and a team member who was included into the RCSN at a later time, then this may be indicative that a team member in the organization had critical knowledge that was missing in the initial RCSN. Alternatively, if there is a significant lack of communication among team members in the initial RCSN, then this may be indicative of a lack of coordination and awareness within team members, and may require intervention from a project manager.

Another use of the RCSN would be to provide a recom-

mended list of contributors that should be contacted. The RCSN may be able to, for example, provide contact information to a contributor so that he has easy access to others who are working on similar tasks.

The concept of an RCSN can evolve to include multiple requirements that are dependent on one another. If multiple requirements are represented in an RCSN, then we can also draw a diagram that displays dependencies in addition to a diagram that displays communication patterns.

Finally, the information in the RCSN may be used to develop an automated awareness notification system. If the requirement changes, an automatic notification can be sent to everyone in the network, even those who are peripherally involved and may have been forgotten in a project plan. The history of an RCSN can be stored so that a contributor viewing it has an at-a-glance view of what has changed since the last time the contributor looked at the diagram. A contributor is more aware of the additional team members in the network and the new communication paths between these team members.

### 3. Challenges

To build a requirement-based social network, we must be able to monitor and detect how contributors communicate among one another. We must ensure that as much communication is tracked as possible, and that the RCSN is automatically updated. An important challenge is determining which communications are relevant to a particular requirement. Natural language processing (of bugs, documents, etc.), machine learning, or task monitoring [8] can be explored as options to determine if certain communications are related to a specific requirement. To limit the search, we can scan a requirements document for keywords in order to generate a project vocabulary that is used throughout development. We can expand the scan to include other artifacts such as bugs and code that may be related to the requirements in order to discover additional vocabulary terms.

Moreover, some communications are never recorded and therefore cannot be detected. For example, face-to-face communication makes up a significant amount of communication among contributors [7]. In addition to adding contributors to the network, we may want to remove a contributor from the network in the case that someone's contribution is obsolete. Determining the point of decay for removing a node or edge from the network is an area for future research. As the approach will not provide a perfect solution, it needs to therefore support manual insertions and deletions.

There are also challenges that are specific to how the information is displayed. Developers are highly protective of their limited screen space and will not want to give up some of it for a display which is used only for the sake of awareness. Therefore, we need to conduct research into how a

compact display can be used or revealed in a non-intrusive manner to provide awareness when it is needed. With the increasing adoption of multiple displays, this awareness screen may be more suited to a secondary display that is always visible. Nevertheless, visual cues will be needed to draw attention to changes as they occur. Indeed, we need to conduct more research to determine whether the display should be updated automatically or manually by the user when changes occur.

Finally, we need to investigate privacy concerns that developers may have with revealing information about their communication patterns.

### 4. Related Work

De Souza, et al [4], automatically generate a social network based on ownership of modules in code. They use a procedure call graph and source code repository information to create a network of social dependencies that they call a *social call graph*. Ariadne [12] is an Eclipse plugin that displays the generated social call graph. The social call graph is meant to facilitate communication between members of a team who may need to integrate components or negotiate the use of application program interfaces (APIs) with each other. Although Ariadne generates a social network diagram, our approach focuses on communication among team members, whereas De Souza's approach looks at source code dependencies through function calls only.

Sengupta, et al, present EGRET [10], a collaboration tool for global software development. Although EGRET contains some visualization views, they are limited to requirements traceability, and a list of online users. Our method proposes a more extensive visualization that concentrates on communication among contributors about a requirement.

Jazz [2] is a collaboration tool designed to facilitate communication among developers. Jazz provides a mechanism to manually add a member to a list of contacts, but does not visualize communication among contributors.

Storey, et al. [11] surveyed tools that are designed to support awareness in software development. Every awareness tool in this survey explore visualizations at the source code or module level, and do not concentrate on collaboration patterns among developers.

The RCSN is different from the previously-explained approaches because it (a) groups every contributor who works on the same requirement, even if these contributors are distributed among different teams, and (b) uses communication information rather than source code to build the expanding network. The RCSN expands by analyzing communication regarding a requirement.

Social-network analysis has been used by organizational theorists for many years, and has recently gained recogni-

tion as a method for identifying collaboration patterns in software development. Crowston and Howison use social-network analysis to study organizational structures within many open-source projects [3]. Madey, et al., use social network analysis to display which open-source developers are part of which projects [9]. The RCSN that we describe displays a social-network diagram, but we concentrate on building the network dynamically and centre the network on a single requirement so that each contributor in the network is aware of each other contributor. By limiting the diagram to display only those working on the same requirement, we effectively display only what is relevant to the viewer.

## 5. Research Agenda

We plan to develop a prototype that will generate an RCSN automatically. We will need to determine how to automatically generate an RCSN from communication patterns among contributors. We plan on using natural language processing techniques to mine information in project planning documents (for example to identify the initial composition of the RCSN, or updated information during the project) and machine learning techniques to maintain an evolving network based on the collaboration patterns around requirement development. Once we can generate RCSNs automatically, we must evaluate the tool to ensure that it increases requirements change awareness and general awareness within a software development organization. The current version of the prototype is developed as an Eclipse plugin and mines information from Bugzilla, which acts as a repository for information on contributor allocation to bugs as well as threaded discussions with respect to a work item. While we were successful to visualizing the networks of individuals that work on the same bug and those involved in its discussion, together with those involved in dependent bugs, we are now working on algorithms to characterize the collaboration patterns and visualize strength of communication lines to represent semantics of this communication.

Another goal of our research is to discover a way to incorporate more than just communication patterns into the RCSN. We would like to find a good way to use information from software artifacts, such as design documents, source code, and bug reports, to fine-tune the RCSN. By using additional information sources, we may be able to more accurately determine if the work that a contributor is doing is related to a requirement. A user of the network may identify a contributor's role, such as programmer, based on the artifacts that the contributor creates and the communication topics that the contributor discusses with others.

We believe these are some of the challenges we identified so far, and look forward to learn about other ones, as well as possible ways to address these, at the workshop.

## References

- [1] 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 *CSCW '06: Proceedings of the 2006 conference on Computer-supported cooperative work, November 4–8, 2006*. (To appear.).
- [2] L.-T. Cheng, S. Hupfer, S. Ross, and J. Patterson. Jazzing up eclipse with collaborative tools. In *Eclipse '03: Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange*, pages 45–49, New York, NY, USA, 2003. ACM Press.
- [3] K. Crowston and James Howison. The social structure of free and open source software development. *First Monday*, 10(2), February 2005. [http://firstmonday.org/issues/issue10\\_2/crowston/index.html](http://firstmonday.org/issues/issue10_2/crowston/index.html).
- [4] C. R. B. de Souza, D. Redmiles, L.-T. Cheng, D. Millen, and J. Patterson. Sometimes you need to see through walls: a field study of application programming interfaces. In *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 63–71, 2004.
- [5] K. Ehrlich and K. Chang. Leveraging expertise in global software teams: Going outside boundaries. In *International Conference on Global Software Engineering*, 2006. (To appear).
- [6] T. Erickson, D. N. Smith, W. Kellogg, M. R. Laff, J. T. Richards, and E. Bradner. Socially translucent systems: Social proxies, persistent conversation, and the design of 'babble'. In *Human Factors in Computing Systems: The Proceedings of CHI '99.*, 1999.
- [7] L. G. Izquierdo. A case study of feature-based awareness in a commercial software team and implications for design of collaborative tools. Master's thesis, University of Victoria, 2006. (To appear.).
- [8] M. Kersten and G. C. Murphy. Mylar: a degree-of-interest model for ideas. In *AOSD '05: Proceedings of the 4th international conference on Aspect-oriented software development*, pages 159–168, New York, NY, USA, 2005. ACM Press.
- [9] G. Madey, V. Freeh, and R. Tynan. *Free/Open Source Software Development*, chapter Modeling the F/OSS Community: A Quantitative Investigation, in *Free/Open Source Software Development*. Idea Publishing, 2004.
- [10] B. Sengupta, S. Chandra, and V. Sinha. Enabling collaboration in distributed requirements management. *IEEE Software*, 2006. (To appear.).
- [11] 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 *SoftVis '05: Proceedings of the 2005 ACM symposium on Software visualization*, pages 193–202, New York, NY, USA, 2005. ACM Press.
- [12] E. Trainer, S. Quirk, C. de Souza, and D. Redmiles. Bridging the Gap between Technical and Social Dependencies with Ariadne. In *Eclipse Technology Exchange (eTX) Workshop*, 2004.