# Mining Task-Based Social Networks to Explore Collaboration in Software Teams

**Timo Wolf, Adrian Schröter, Daniela Damian, Lucas D. Panjer, and Thanh H.D. Nguyen,** *University of Victoria*

The authors describe a repository-independent approach to mining task-based communication in social networks. They propose practical applications that utilize their approach to directly support development projects.

Suppose you're a software team manager who's responsible for delivering a software product by a specific date, and your team uses a code integration system (referred to as a *build* in IBM Rational Jazz and in this article) to integrate its work before delivery. When the build fails, your team needs to spend extra time diagnosing the integration issue and reworking code. As the manager, you suspect that your team failed to communicate about a code dependency, which broke the build. Your team needs to quickly disseminate information about its interdependent work to achieve a successful integration build. How can you understand your team's communication? Social-network analysis can give you insight into the team's communication patterns that might have caused the build's failure.

Current and timely knowledge of your team's social network, whether you're a project manager, a team leader, or a developer, can be important in many situations, not just with broken builds. You can't always tell exactly who the project experts and central communicators are in the development environment. The distributed nature of software development, compounded with the typically high turnover that goes with outsourcing work, only adds to this problem. Highly interdependent teams must often function across organizational and geographical boundaries. This distribution causes significant challenges to maintaining effective team communication. By examining the team's social networks, the team leader can identify collaboration and communication problems, and project newcomers can identify the experts and active communicators.

However, constructing an explicit social-network representation within an organization isn't trivial. Communication is the key to constructing an organization's social network. People in software projects communicate through diverse channels, some of which aren't easily recordable—for example, face-to-face and telephone conversations. So, software project repositories such as bug databases, source-code repositories, and automated build systems provide rich sources for mining developers' communication. The recorded communication artifacts must be translated into meaningful conversations about tasks of interest. How do you leverage developers' communication related to a specific collaborative task? How can you construct

 0740-7459/09/$25.00 © 2009 IEEE

a social network of people who collaborate on a task that interests you?

Recently, software engineering researchers have used social networks to study collaboration in software teams and have mined data from different repositories, such as software and email archives.[1–3] The social networks developed in these works are, however, difficult to compare, having been constructed for different research purposes. In this article, we describe a systematic approach for mining large software repositories to generate social networks that use task-based communication between developers. This approach is independent of any specific repository and applicable in any project that stores collaborative tasks and related communication information. We also describe our experiences in using this approach in our research and discuss practical deployment implications. When we used our approach to mine the IBM Rational Jazz project's software repository, we discovered that properties of developer social networks can predict integration build results. We also found that the large Jazz team experienced fewer delays in communication than expected because of their highly connected project-wide social network with effective information distribution among seven geographic sites.

## Our Approach

Our failed-build example illustrates our approach mining large repositories to construct social networks that can help to solve team collaboration problems. We use a graph that consists of nodes connected by edges to represent a social network. Figure 1 (on page 60) shows two examples of our task-based social networks' step-by-step construction. To explore the communication for Failed Build 1 (see Figure 1b), we start with the project-wide network, which is composed of artifacts such as source-code changes, emails, or documentation and is related to a task, such as bug 123 (see Figure 1a). Everyone who communicated about such an artifact or its respective task is included in the network and connected to a task that relates to the artifact.

We filter the people from all teams who've contributed code to this build (represented by blue-colored circles). Then, we filter the completed tasks for the build, represented by white squares. Using the task and team-member information, we connect the people for whom we've recorded task-related communications to complete the social network. In this case, we use comments on tasks as communication records, represented as dashed lines between team members and tasks in the figure.

By drawing a solid line (in this case, the blue line) between the filtered people, a missing communication connection between Cathrin and Eve becomes apparent. Cathrin was working on the GUI API task, but she never commented on the GUI test task, and vice versa for Eve. Because of this break in communication, an important dependency between these two tasks might not have been communicated or resolved, thus causing the build failure.

### Communication Brokers

When direct communication between project members isn't possible—for example, because of geographic distance or language barriers—communication brokers can provide the communication link. Figure 1c demonstrates how you can use a social network to find a communication broker between two project members.

Suppose, again, that you're the manager of a software team. Helga, a developer in your project, complains that she needs information from Adam, but Adam hasn't responded to information requests. Helga needs this information to complete her work on bug 222. You suspect that, because of time zone differences, Helga and Adam are rarely working at the same time and have trouble communicating effectively. Social-network analysis can identify other people in the project who could broker communication between Helga and Adam.

To construct a social network for this application, we take all the project's collaborative tasks, (colored green in the Figure 1a). Then, we filter the project members, keeping those who've contributed to the collaborative tasks. Finally, using the recorded task-based communication, we connect the project members to create a social network for the entire project. By visualizing this social network, we see that either Gina or Eve is a good candidate to broker communication between Helga and Adam. Choosing to use either Gina or Eve as a communication broker could be done based on their geographic location with respect to Adam.

These two examples illustrate how you can mine the same repository for two different collaboration scopes of interest to construct two different social networks. An overview of software repository mining approaches to generate developer networks is outlined in the "Constructing Developer Networks" sidebar (on page 61).

### Elements of Task-Based Social Networks

Our approach is repository- and tool-independent, and you can apply it to any repository that provides information about people, tasks, technical artifacts, and communication. This includes work, issue, or

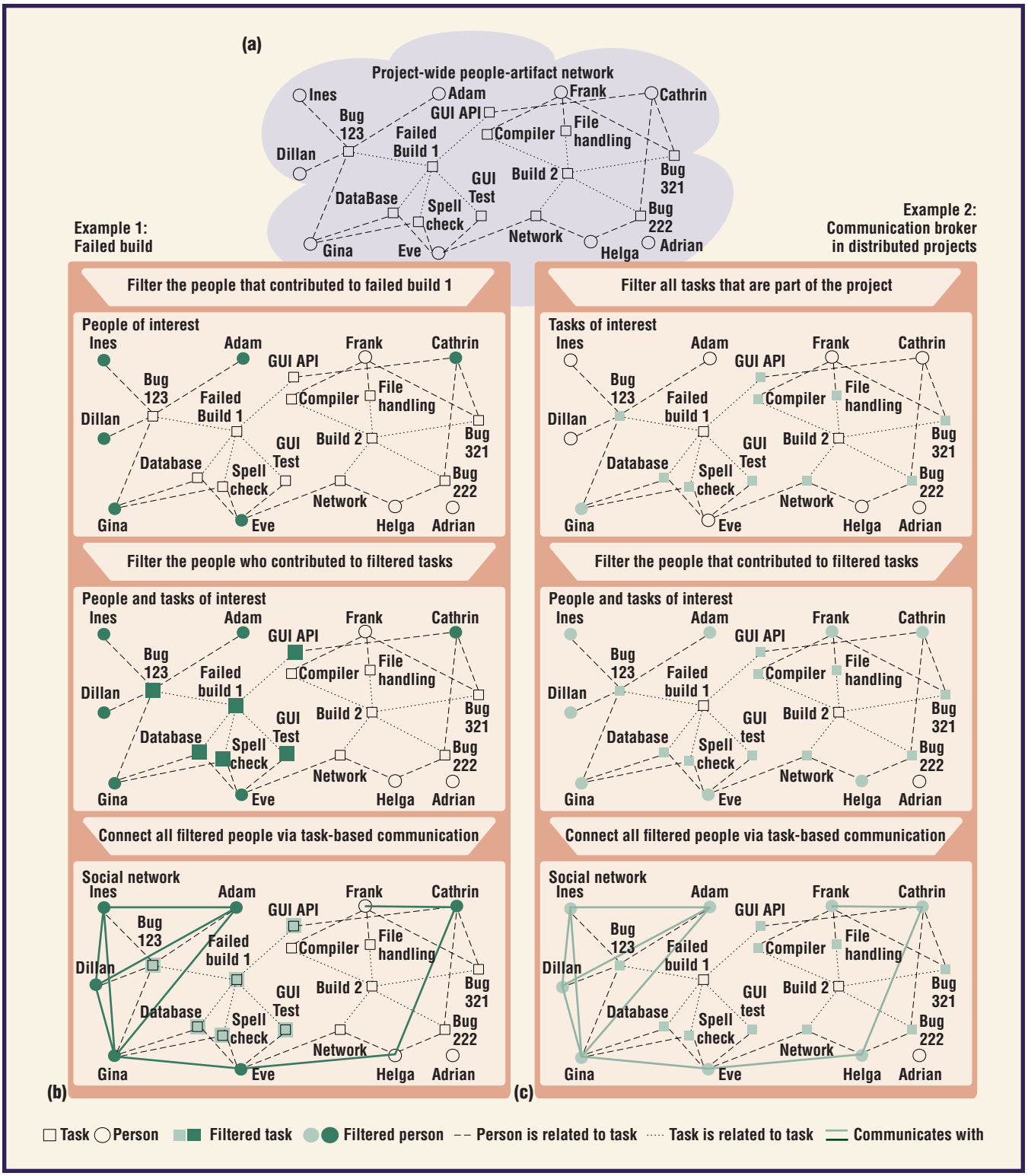


**Figure 1. Social-network construction examples in our approach: (a) project-wide communication, (b) failed build 1, and (c) a communication broker in distributed projects. This figure shows two examples of constructing a task-based social network by filtering people and tasks and connecting two or more people if task-related communication exists.**

change management repositories, such as Bugzilla or IBM Rational ClearQuest; source-code management systems, such as CVS or IBM Rational ClearCase; and communication repositories, such as email archives.

We construct and analyze social networks within

a collaboration scope. In the example of failed build 1, the collaboration scope is the communication of the failed build's contributors. Other examples include how people collaborate when working on a critical task, in a particular geographical location, or in a functional team.

Three critical elements must be mined from software development repositories to construct task-based social networks in a collaboration scope.

***Project members.*** The software project's members can be developers, testers, project managers, requirements analysts, or clients. Project members, such as Cathrin and Eve, become nodes in the social network.

***Collaborative tasks.*** These are the units of work that the project's members must collaborate and communicate. Examples include resolving Bug 123 or implementing the GUI API. More generally, we also consider implementing feature requests and requirements as collaborative tasks.

***Task-related communication.*** The unique information that team members exchange while collaborating lets us build task-based social networks. In our example in Figure 1, dashed black lines represent task-related communication such as a comment on bug 123 or an email or chat message about the GUI API. We use task-related communication to create the edges between project members in the social networks.

## Constructing Social Networks

We use three steps to construct a social network within a collaboration scope: project-member filtering to identify nodes in the network, collaborative-task filtering to use as the context for collaboration, and connecting project members to identify edges in the network. Each filtering step includes criteria that reduce the set of project members or collaborative tasks from the entire project. After we filter collaborative tasks and team members, we use recorded task-related communication to connect the nodes in the social network.

***Project-member filtering.*** We identify the project members who meet the collaboration scope's specified criteria to determine the set of nodes that we'll include in the social network. In our example, we restrict the team members to those who contributed to the failed build, such as Adam, Eve, and Cathrin (colored green in Figure 1a). In addition, you can add other constraints to the criteria to further nar-

## Constructing Developer Networks: An Overview

Developers have constructed technical-based developer networks using source-code repositories by leveraging code's technical dependencies (see Figure A1).[1–4] Two developers are connected in a network if both change the same file, module, or project. Researchers build project-wide networks[4–8] without a specific task focus (see Figure A2). In these networks, people are linked using any project-related communication. They can mine communication from repositories, such as email archives and issue trackers. We mine social networks that evolve around collaborative tasks, such as fixing a bug or implementing a feature (see Figure A3). We mine the communication that's directly related to a certain task and thus build fine-grained networks[8,9]—similar to other approaches.[10–12]

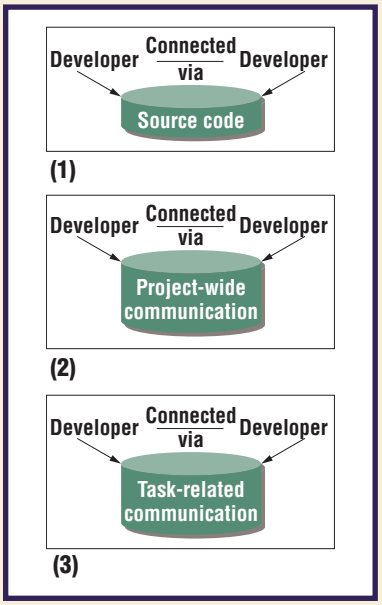


**Figure A. How to build a developer's network. There are three kinds: (1) technical-based, (2) project-wide, and (3) task-based networks.**

## References


1. M. Ohira et al., "Accelerating Cross-Project Knowledge Collaboration Using Collaborative Filtering and Social Networks," *Proc. 2nd Int'l Workshop Mining Software Repositories* (MSR 05), ACM Press, 2005, pp. 1–5.
2. M. Pinzger, N. Nagappan, and B. Murphy, "Can Developer Social Networks Predict Failures?" *Proc. Foundation Software Eng.*, ACM Press, 2008, pp. 2–12.
3. A. Meneely et al., "Predicting Failures with Developer Networks and Social-Network Analysis," *Proc. ACM SIGSOFT Foundations of Software Eng.*, ACM Press, 2008, pp. 13–23.
4. J. Goecks and E.D. Mynatt, "Leveraging Social Networks for Information Sharing," *Proc. Conf. Computer Supported Collaborative Work*, ACM Press, 2004, pp. 81–84.
5. G. Valetto et al., "Using Software Repositories to Investigate Sociotechnical Congruence in Development Projects," *Proc. 4th Int'l Workshop Mining Software Repositories* (MSR 07), ACM Press, 2007, pp. 25–35.
6. C. Bird et al., "Mining Social Networks," *Proc. 3rd Int'l Workshop Mining Software Repositories* (MSR 06), ACM Press, 2006, pp. 137–143.
7. M. Cataldo and J.D. Herbsleb, "Communication Patterns in Geographically Distributed Software Development and Engineers' Contributions to the Development Effort," *Proc. Int'l Workshop Cooperative and Human Aspects Software Eng.*, ACM Press, 2008, pp. 25–28.
8. T. Nguyen, T. Wolf, and D. Damian, "Global Software Development and Delay: Does Distance Still Matter?" *Proc. Int'l Conf. Global Software Eng.*, IEEE CS Press, 2008, pp. 45–54.
9. T. Wolf et al., *Communication, Coordination, and Integration*, tech. report DCS-322-IR, Computer Science Dept., Univ. of Victoria, 2008.
10. M. Cataldo et al., "Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools," *Proc. Conf. Computer Supported Cooperative Work* (CSCW 06), ACM Press, 2006, pp. 353–362.
11. J.D. Herbsleb and A. Mockus, "Formulation and Preliminary Test of an Empirical Theory of Coordination in Software Engineering," *Proc. European Software Eng. Conf. and ACM SIGSOFT Symp. Foundations of Software Eng.*, ACM Press, 2003, pp. pp. 112–121.
12. K. Ehrlich et al., "An Analysis of Congruence Gaps and Their Effect on Distributed Software Development," *Socio-Technical Congruence Workshop* at the *30th Int'l Conf. Software Eng.* (ICSE 08), IEEE CS Press, 2008.

**Our elements for constructing a social-network map to Jazz artifacts.**

row which project members you'll include, such as temporal constraints on when team members communicate about tasks.

***Collaborative-task filtering.*** Similar to project-member filtering, we use the criteria from the collaboration scope to select collaborative tasks, which will later provide the communication context for connecting project members. In our example, we restrict the tasks to those included in failed build 1, such as the GUI API, Bug 123, and GUI test. You can base the filtering criteria on collaborative tasks' properties, such as task priority or assigned team. Again, temporal constraints are often useful criteria, such as selecting all development tasks contributed since the last build.

***Connecting project members.*** To connect project members and create edges in the social network, we leverage recorded communication between the project members in the filtered collaborative tasks. For example, Gina and Ines have both commented on bug 123, so we create an edge between them in the social network. Our approach to constructing task-based social networks also lets us include directed and weighted edges. Directed edges can represent the direction of communication, such as email sent from one team member to another. We can use weighted edges to represent the volume of communication, such as the number of emails sent. Although the filtering steps are independent, the order in which they're applied can affect node and edge composition in the resulting social network.

## Social-Network Analysis with Jazz

We applied our approach to mine and construct social networks with the IBM Rational Jazz project in several research studies. IBM Rational is building Jazz[4] as a scalable and extensible team collaboration platform for integrating development work across task, build, source-code, and planning-management activities. Jazz is developed by a globally distributed team that uses it to manage its own work. The Jazz platform uses a client-server architecture, where the server is the central data repository that stores data for its components. The repository is accessible using a Web-based or an Eclipse-based client interface.

Our elements for constructing a social-network map to Jazz artifacts. First, project members are contributors in Jazz, and personal information, such as each contributor's name and email address, as well as project-related information, such as team affiliations, are available. Second, collaborative tasks are *work items* in Jazz, which represent the basic unit of work and can describe many types of tasks, such as bug reports, modification requests, or development tasks. Work items have a comment-based conversation, a list of observing subscribers, and other attributes, such as the item's creation date, description, and owner. Finally, task-related communication maps to *comments* on work items in Jazz, in which reading and writing comments on the work items is the main collaboration mechanism, as developers use comments to debate and discuss decisions. A comment thread forms a conversation that's attached to the work item; each comment has a creation date, comment text, and an authoring contributor.

### Data Mining Tools for Jazz

To extract the elements we needed to construct and analyze social networks, we implemented several data mining and social-network analysis tools. To extract data of interest, we developed a plug-in for the Eclipse-based Jazz client that used the provided Java API to query and retrieve the desired data from the repository.

The Jazz development project has a large Jazz-based repository containing more than 40,000 work items, 150 contributors, and 5,000 build results. We needed to collect data from the live development repository without affecting the Jazz development team's server performance and potentially disrupting their work. To meet this goal, we designed our data extraction tool to be minimally invasive and to use incremental queries—extracting small portions of the data set at a time—to minimize performance degradation.

Furthermore, querying and processing the extracted data to construct social networks are additional challenges because these actions are data- and time-intensive. For example, extracting and processing all work items from the repository took several hours. Thus, it's not feasible to extract and process all the information that you need to construct social networks in real time. For this reason, we import the data of interest into a reporting and analysis database. This approach lets us use a large proportion of the data in the Jazz team's repository for social-network construction and analysis without adversely affecting their development environment.

To process the data from the database, we used the Java Universal Network/Graph (JUNG) framework to construct and visualize the task-based social networks and to compute social-network analysis measures. Our tools also generate data sets for

use as inputs to other tools, such as statistical analysis using the R language or social-network analysis using Ucinet.

## Research Studies Using Social-Network Analysis

We used the mined task-based social networks in two research studies, which mapped to the more general build failure[5] and communication broker[6] examples provided earlier.

***Communication structures to predict build failures.*** In this study, we were interested in investigating whether properties of an integration team's social network have any relationship to the integration outcome. Builds are frequent and very important in the Jazz project. The continuous integration process requires regular daily and weekly integrations of each team's work into an assembled product. A build can succeed or fail because of compilation, testing, or other integration errors.

To integrate in Jazz, the development team members commit source-code changes (change sets) to a team-based branch. Each change set is associated with the work item that describes the change it implements, and contributors comment on the work items to discuss the changes. This study's collaboration scope is the communication between contributors who've delivered change sets to each build.

To construct the social networks for build 1 at time $t_1$ (as illustrated in Figure 2), we followed the same steps we outlined in the failed-build example we described earlier: we filtered the contributors that delivered change sets between $t_0$ and $t_1$ and identified all work items that were associated with code change sets in build 1. To connect contributors in the social network, we added a directed edge between each pair of contributors who had either commented or subscribed to a common work item since the last build. The weight on directed edges represents the number of comments on the shared work items.

After we constructed the social networks for the five teams' integration builds (between 48 and 60 builds for each team), we computed different social-network measurements, such as centrality, betweenness, and density,[7] and investigated their relationship to integration outcomes. Although none of these measures can independently predict the build outcome, when used in combination, they're more powerful. We developed a predictive model by training a Bayesian classifier that was able to accurately predict failed-build results. Table 1 shows the predictions' recall and precision values using measures of the social-network structure for the five teams. To validate our model for each team, we used leave-one-out cross validation, which trains a model for each set of data points that's one size smaller than the full set, and then we predict the data point that isn't in the training set. Study details are available in the technical report *Communication, Coordination, and Integration*.[5]

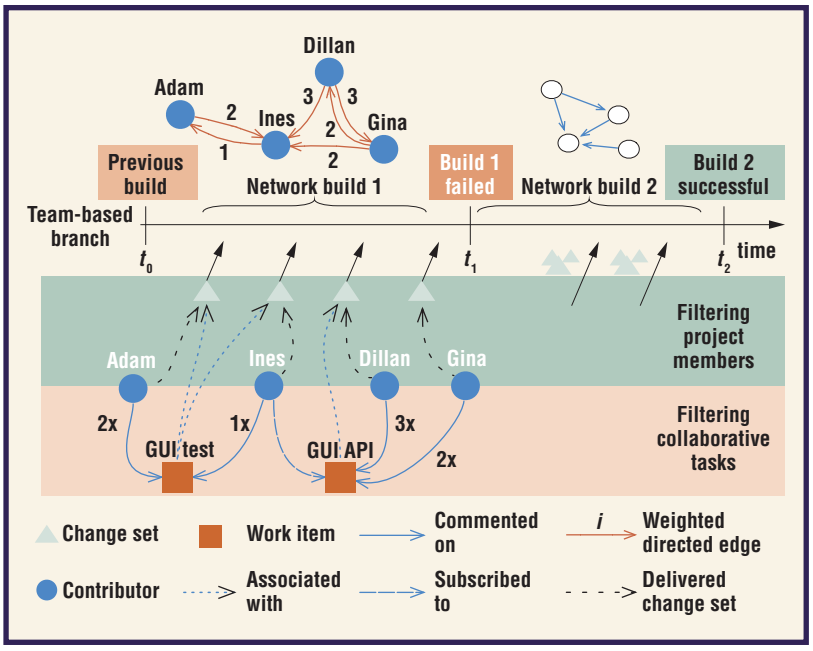


**Figure 2. Constructing social networks for predicting build failures. Code changes identify the contributors used in the build-related social network. Contributors who comment on the same work items associated with the code changes are connected in the social network.**

**Table 1**

**Prediction results for the five Jazz teams**

| | Teams | | | | | Standard deviation (percent) |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | |
| Recall (%) | 55 | 75 | 62 | 66 | 74 | 8.38 |
| Precision (%) | 52 | 50 | 75 | 76 | 66 | 1.34 |

***Communication structures in a large distributed project.*** In this second study, we were interested in the geographically distributed Jazz development team's project-wide communication structure.[6] A quantitative analysis of response time for task-communication and task-resolution times revealed a lower-than-expected impact of distance on these factors. We explored the project's social networks for useful insights that might explain this effect. Specifically, we were interested in identifying the social networks' properties, such as cohesiveness,

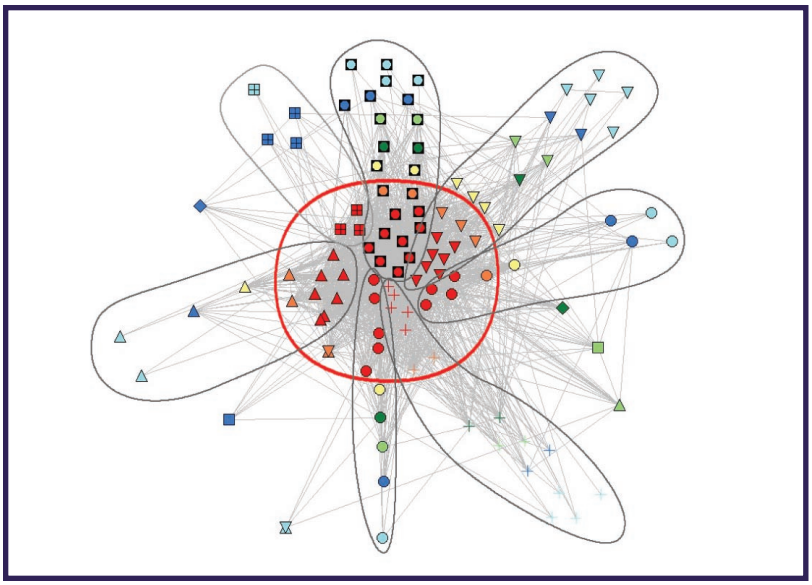

**Figure 3. The project-wide communication-based social network is constructed on the basis of all work items. Participants contributing to the work items are added, and two contributors are connected when they comment on at least one common work item.**

that were related to the results of communication delay and distributed collaboration.

This study's collaboration scope is the entire project, so it includes all collaborative tasks and all contributing project members. In this case, connecting project members is the key step in creating a project-wide social network.

To construct this social network, we followed the steps for the communication broker, described earlier. Figure 3 illustrates the resulting social network; ovals indicate the teams at each geographic site. We used the *k-core* and *core-periphery* tests[7] to analyze the network structure's properties. These tests let us see whether we could identify multiple cores in the network or whether the network showed a star-like structure in which a single core would mediate communication with the developers on the network's periphery.

The results indicate that the project-wide team collaborates as a cohesive team with one large core, as opposed to many loosely connected clusters. The colors in the figure illustrate how close a team member is to the core of the whole project. In red, we show a core of active developers (60 of 112 project members), where each developer communicates with at least 25 other developers from the core. The other colors, from yellow to light blue, indicate various lower degrees of communication in the project. Furthermore, using the *group degree centrality*[7] measure, we also found that each geographic location has roughly equal centrality and uses the people in the core to stay connected to the rest of the large team. This means that project members communicate well with project members at the other geographic locations. This could explain why distance has an insignificant effect on communication-response and task-resolution time in the large distributed team.

We learned that collaboration tools and practices can help overcome communication delays resulting from global distribution of team members. From our experience with the Jazz team, best practices such as prioritizing offsite requests and tools that integrate development, project management, and communication play a major role in achieving these results.

## Practical Implications for Deployment

Our approach to mining and constructing social networks, as well as our studies' results, have several implications for software practitioners.

***Scalable mining and analysis tools.*** The Jazz repository was under a constant load because of the global distribution of the more than 140 team members. By incrementally mining and using a secondary reporting database—or data warehouse—we reduced load and performance problems on the mined software repositories. Our approach is useful for practitioners who must not impact the performance of their team's repositories. We can store extracted data, constructed networks, and network analysis measures in the data warehouse to avoid extracting or recomputing them multiple times. For example, the social network using the set of work items and communication around each build can be stored in the data warehouse and is directly accessible for any number of further analyses. Both the data warehouse and client applications can conduct computationally intensive analyses and predictions using the data warehouse. Once we store task-based social networks and network-analysis measures in a data warehouse, visualization of social networks and further analysis becomes efficient and unintrusive to repository users.

***Team awareness through social-network visualization.*** Developers working on interdependent tasks can benefit from visualizing their social network. The network information can include developers' contact information and availability as well as a list of other tasks they're working on. This information is particularly important for newcomers to a project who lack project-specific expertise, such as who's been involved in particular tasks or architectural decisions. We're currently conducting a case study to identify which information is valuable to support developers and how that information is best conveyed visually.

***Outcome prediction using social-network analysis.*** Another social-network analysis application is to predict project outcomes, such as build results, on the basis of information about team communication behavior. Practitioners can build tools that they embed in the development environment that inform team members about the health of upcoming builds. An awareness notification system that uses a build failure prediction model could indicate whether the current communication patterns are likely to result in a failed build.

We can build a predictive model from social networks constructed with data that has been mined from task and task-communication repositories in four steps:

1. Use the initialization predictive models, which show the social networks and outcomes for all existing builds. Then store the data on social networks, build outcomes, and predictive models in a data warehouse for later use.
2. Construct a social network for an upcoming build using our approach, as described earlier.
3. Predict the upcoming build's outcome by calculating network analysis measures for the constructed social network and using them as input to the predictive model. The model then predicts an outcome for the upcoming build, and at this point, a manager could make proactive adjustments to attempt to prevent a predicted build failure.
4. Update the predictive model after the upcoming build has been completed by including the social network, network measures, and outcome of the latest build. The model can then be used again to make an outcome prediction for the next upcoming build, starting with step 2.

If the system indicates a build failure, further analysis could identify communication deficiencies, which would prompt managers to rectify the problem before the build occurs.

***Effective management through social-network analysis.*** Beyond visualizations, management can use computed social-network measures to help prevent failed builds. Our research shows that the quality of project integrations depends on the quality of developers' and team members' communication. We can prevent failures by monitoring and affecting positive communication behavior. A group of software engineers can bring different levels of expertise and knowledge to collaborative tasks. Positive team performance depends not only on the information available to the developers and the distribution of knowledge within the team, but also on the communication structure that facilitates knowledge dissemination within the team. Although we have no evidence that individual measures of communication structure can predict integration failure, these measures can be used to identify problematic communication behavior that will likely result in failed integrations. Actions that management can take to prevent failures for a particular team include

- computing social-network measures or running predictive models early in the project; in our Jazz study, the predictive model performed well even with the first 25 percent of the team communication data (that is, first quarter of project timeline); this can signal problematic communication behavior early in the project.
- examining the team's communication structure to identify problems in communicating project-specific technical dependencies. A low-density network with high communication needs is an example of a problematic network communication structure because it's possible that there are too many missing communication links between developers who have technical dependencies on other developers.[2]
- improving communication procedures by adjusting communication and knowledge management processes or project tools, to address any problems that were found when examining the communication structure—if communication links are broken or missing, the project manager can assign communication brokers.
- improving knowledge management procedures by increasing the awareness of these coordination needs as well as developers' areas of expertise in the current project—project managers can disseminate this information through regular project meetings or more adequate documentation of dependencies in project-specific knowledge repositories.

Project managers can adjust these actions to meet their specific project needs. We feel they will find these tactics very useful in facilitating communication between project team members, which in turn will help to streamline the overall outcome.

**The quality of project integrations depends on the quality of developers' and team members' communication.**

Task-based social-network mining and analysis enables you, as a participant in a software project, to tap into a vast pool of otherwise inaccessible communication

## About the Authors

**Timo Wolf** is an engineer at Siemens Corporate Technology and was a postdoctoral researcher in the Software Engineering Global Interaction Lab (Segal) at the Department of Computer Science at the University of Victoria. His research interests include coordination and communication in globally distributed development projects, social-network analysis, traceability, and tool support for distributed development projects. Wolf received his PhD in software development from Techniche Universität München. Contact him at timowolf@siemens.com.

**Adrian Schröter** is a PhD candidate in computer science at the University of Victoria. His research interests cover software development in distributed teams, software quality, and mining software repositories. Schröter received his MSc in software engineering from Saarland University. Contact him at schadr@uvic.ca.

**Daniela Damian** is an associate professor in the Department of Computer Science at the University of Victoria, where she leads research in the Software Engineering Global Interaction Laboratory (Segal). Her research interests include software engineering, computer-supported cooperative work, and human-computer interaction. Damian has studied and consulted on the practices of collaborative and global software development in large organizations such as Unisys, Dell, IBM, and Siemens. Contact her at danielad@cs.uvic.ca.

**Lucas D. Panjer** is a consulting software engineer. His research interests include software engineering processes, mining software repositories, information visualization, and human-computer interaction. Panjer has an MSc in computer science from the University of Victoria. Contact him at lucas@panjer.org.

**Thanh H.D. Nguyen** a master's candidate in the Department of Computer Science at the University of Victoria. Nguyen's research interests include collaborative and distributed software development, development tools, and collaboration and communication in software teams Nguyen is a member of the IEEE and the ACM. Contact him at duythanhg@uvic.ca.

information. Our approach has been applied to two scenarios with the Jazz project illustrating its applicability and gleaning new insights into the social patterns of that team. The use of social networks in software engineering is relatively unexplored and holds much promise for future applications.

## References

1. C. Bird et al., "Mining Email Social Networks," *Proc. Int'l Workshop Mining Software Repositories* (MSR 06), ACM Press, 2006, pp. 137–143.
2. K. Ehrlich et al., "An Analysis of Congruence Gaps and Their Effect on Distributed Software Development," *Socio-Technical Congruence Workshop* at the *30th Int'l Conf. Software Eng.* (ICSE 08), IEEE CS Press, 2008; http://docs.google.com/View?id=dhncd3jd_89fpszqcx.
3. J.D. Herbsleb and A. Mockus, "Formulation and Preliminary Test of an Empirical Theory of Coordination in Software Engineering," *Proc. 9th European Software Eng. Conf.* (ESEC/FSE-11), ACM Press, 2003, p. 138.
4. R. Frost, "Jazz and the Eclipse Way of Collaboration," *IEEE Software*, vol. 24, no. 6, 2007, pp. 114–117.
5. T. . Wolf et al., "Predicting Build Failures using Social Network Analysis on Developer Communication," *Proc. 31st Int'l Conf. Software Eng.* (ICSE 09), preprint, May 2009.
6. T. Nguyen, T. Wolf, and D. Damian, "Global Software Development and Delay: Does Distance Still Matter?" *Proc. 3rd Int'l Conf. Global Software Eng.* (ICGSE 08), IEEE CS Press, 2008, pp. 45–54.
7. S. Wasserman and K. Faust, *Social-Network Analysis: Methods and Applications*, Cambridge Univ. Press, 1994.