Mining Task-based Social Networks to Explore Collaboration in Software Teams

Timo Wolf, Adrian Schröter, Daniela Damian, Lucas D. Panjer, Thanh H. D. Nguyen
Software Engineering Global interAction Lab (SEGAL)
Department of Computer Science
University of Victoria
Victoria, British Columbia, Canada
{timowolf, schadr, danielad, ldp, duythanh}@uvic.ca

Suppose you are the manager of a software team and are responsible for delivering a software product on a specific date. Your team uses a build system to integrate their 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 communicate in timely manner to propagate information about their interdependent work to achieve a successful integration build. How can you understand the communication of your team? Social network analysis can give you insight into the communication patterns of your team that may have been the cause of the build failure.

Current and timely knowledge of the social network of people in your project, whether you are a project manager, a team leader, or a developer is important in many situations beyond broken builds. Knowledge of project experts and central communicators is information that is often invisible in the development environment. The distributed nature of software development, compounded with the typical high turnover in outsourcing relationships, only adds to this problem. Highly interdependent teams often need to function across organizational and geographic boundaries and face significant challenges to maintain awareness and effectively communicate with their team. Examination of social networks can identify collaboration problems such as missing communication links between interdependent team members. Since newcomers specifically lack historical project information, they may benefit from insights into the project's social networks that expose expertise and active communicators.

However, constructing an explicit representation of social networks within an organization is not trivial. Communications play a main role in social networks within organizations. The difficulty stems from the fact that people in software projects communicate through diverse channels some of which are not easily recordable. Given the difficulty of capturing and recording face-to-face and telephone conversations, 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 that is *related* to a specific collaborative task? How can you construct a social network of people that collaborate on a task that is of interest to you?

Recent research in software engineering has used social networks to study collaboration in software teams and mined data from different repositories, such as software and email archives [1, 4, 2]. The social networks developed in these works are, however, difficult to compare, having been constructed for different research purposes. The research contribution of this paper is a systematic approach to mine large software repositories to generate social networks that use task-based communication between developers. This approach is independent of any specific repository and can be applied 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 implications for deployment. When applied to mine the software repository of the IBM Rational Jazz project, the described approach allowed us to discover that properties of developer social networks can be used to predict integration build results. As well, we found that the large Jazz team experienced less delay in communication than expected due to the distributed nature of the project. In addition, they exhibited a highly connected project-wide social network with effective information distribution among seven geographic sites.

1 Our Approach

To illustrate our approach of mining large repositories to construct social networks that can help to solve team col-

laboration problems, we use the example of a failed build. A social network is represented as a graph that consist of nodes connected by edges. In our approach, the nodes represent people and edges represent task-related communication between these people.

Figure 1 shows the step-by-step construction of our task-based social networks. To explore the communication for Failed Build 1 (see left column), we start with the project-wide people-artifact network. The project-wide people-artifact network is composed of artifacts, such as source code changes, emails, or documentation, that can be related to a task such as Bug 123. Everyone that communicated about such an artifact or its respective task is included in the network and connected to the task, that relates to the artifact.

We *filter the people* from all teams who have contributed code to this build. These people are Adam, Ines, Dillan, Gina, Eve, and Cathrin (coloured blue). Then we *filter the tasks* that were completed for this build, which are GUI API, Bug 123, DataBase, Spell Check, and GUI Test. Using the task and team member information, we complete the social network by connecting the people for which we have recorded task-related communication. In this case we use comments on tasks as communication records, represented as dashed lines between team members and tasks in the figure.

By constructing and visualizing the task-based social network, represented as blue lines in the bottom panel of the figure, it becomes apparent that there is a missing communication connection between Cathrin and Eve. Cathrin was working on the GUI API task, but never commented on the GUI Test task, and vice versa for Eve. There may have been an important dependency between these two tasks that was not communicated or resolved. You might conclude that this may have been part of the problem that caused this build failure.

1.1 Elements of Task-based Social Networks

Having informally described how social networks can be constructed from project data, we now provide a high level description of how we conceptualize and construct social networks for software teams. Our approach is repository and tool independent and can be applied to any repositories that provide information about people, tasks, technical artifacts, and communication. This include work, issue, or change management repositories, such as Bugzilla or IBM Rational ClearQuest™; or source code management systems, such as CVS or IBM Rational ClearCase™; or even communication repositories such as email archives.

We construct and analyze social networks within a *collaboration scope* of interest. In this example, around Failed Build 1, the collaboration scope is the communication of the contributors to the failed build. Other examples include the collaboration of people working on a critical task, in a particular geographical location, or in a functional team such as testing.

There are three critical elements that are necessary to construct task-based social networks for a collaboration scope and that need to be mined from software development repositories:

Project Members are people who work on the software project. These project 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 are units of work within the project that project members need to collaborate and communicate about. Examples of collaborative tasks include resolving Bug 123 or implementing the GUI API. More generally, implementing feature requests and requirements can also be considered collaborative tasks.

Task-related Communication is the information exchanged while completing a collaborative task and is the unique information that allows us to build task-based social networks. In our example, dashed black lines represent task-related communication such as a comment on Bug 123, or an email or chat message about GUI API. Task-related communication is used to create the edges between project members in the social networks.

1.2 Constructing Social Networks:

The following three steps are used to construct a social network within a collaboration scope: filtering of project members to identify nodes in the network, filtering collaborative tasks to use as the context for collaboration, and connecting project members to identify edges in the network. Each filtering step includes criteria and reduces the set of project members or collaborative tasks from the entire project. After collaborative tasks and team members are filtered, recorded task-related communication is used to connect the nodes in the social network.

Filtering Project Members: To determine the set of nodes to be included in the social network we identify those project members that meet the criteria specified in the collaboration scope. In our example, we restrict the team members to those who contributed to the failed build, such as Adam, Eve, and Cathrin (coloured blue in Figure 3). In addition, other constraints, such as temporal constraints on

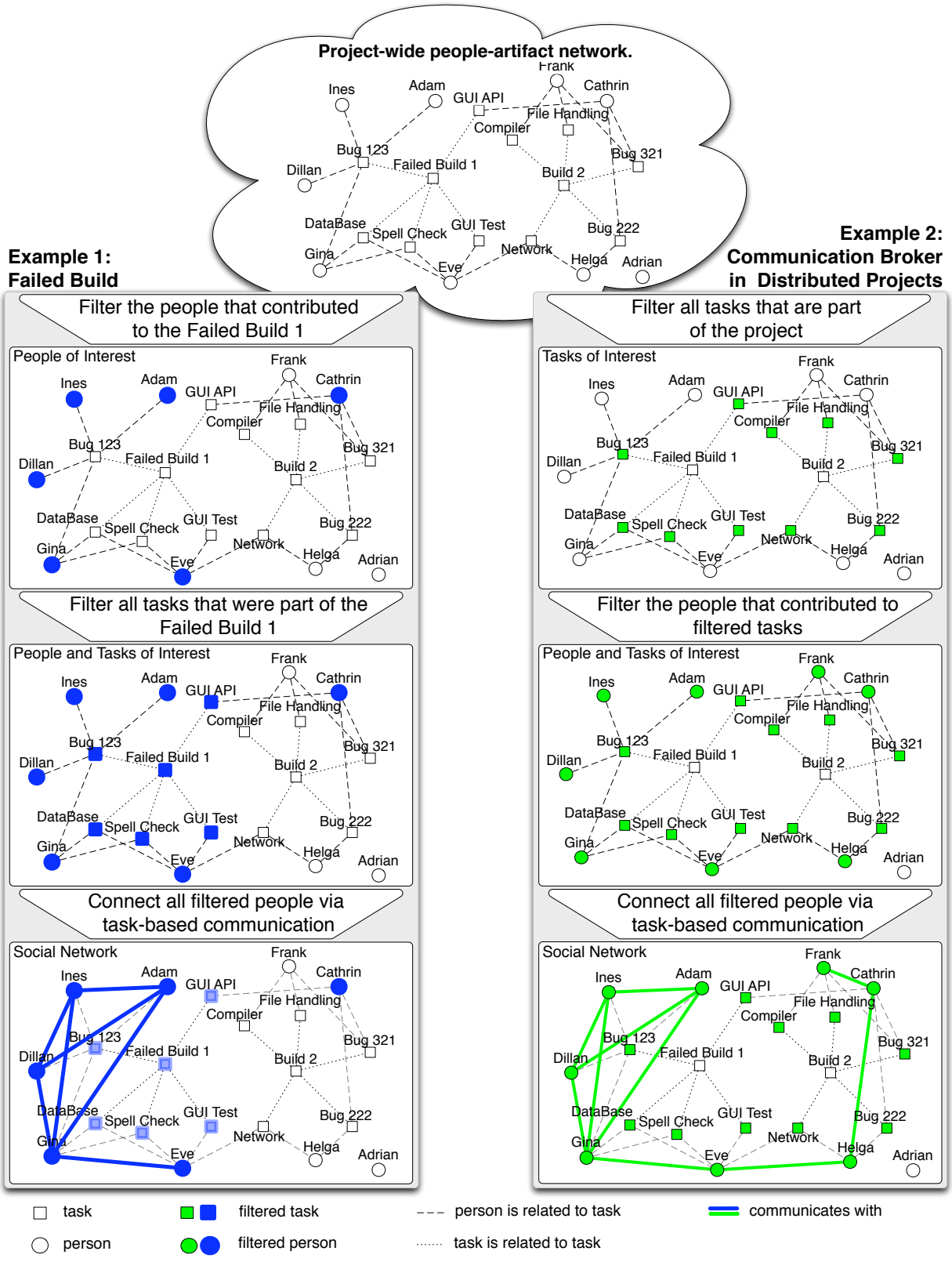


Figure 1. Social network construction examples in our approach

when team members communicated about a task, can be added to further reduce the included set of project members.

Filtering Collaborative Tasks: Similar to the filtering of project members, we use the criteria from the collaboration scope to select collaborative tasks. The collaborative tasks provide the communication context used to connect project members in the following step. In our example, we restrict the tasks to those included in Failed Build 1, such as GUI API, Bug 123, and GUI Test. The filtering criteria can be based on properties of collaborative tasks, 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, creating 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 and so we create an edge between them in the social network. Our approach to constructing task-based social networks also enables the inclusion of directed and weighted edges. Directed edges can represent the direction of communication such as email sent from one team member to another. Weighted edges can be used to represent the volume of communication such as the number of emails sent.

While the filtering steps are independent, the order in which they are applied can affect the composition of nodes and edges in the resulting social network.

1.3 A Second Example – Communication Brokers

Here we provide a second example, applying project member and task filters in a different order with differing criteria. This example demonstrates how social networks can be used to find communication brokers between two project members. This example is illustrated in the right column of Figure 1.

Suppose, again, that you are the manager of a software team. Helga, a developer in your project, complains that she needs information from Adam, but Adam has not responded to information requests. Helga needs this information to complete her work on Bug 222. You suspect that, due to time zone differences between offices, 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 that may be able to broker communication between Helga and Adam.

To construct a social network for this application, we filter the collaborative tasks in the project keeping all tasks (coloured green in the right column of Figure 1). Then we filter the project members, keeping those that have contributed to those 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 Gina and/or Eve are good candidates 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 the same repository could be mined for two different collaboration scopes of interest to construct two different social networks. An overview of software repository mining approaches to generate developer networks are outlined in the Constructing Developer Networks sidebar.

2 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. First, we describe how Jazz concepts and artifacts were mapped to the concepts used in our general approach. Then, we describe our data mining tools, followed by two research studies conducted using data extracted from the Jazz development project.

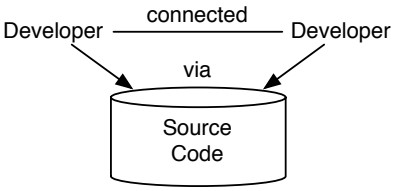
IBM Rational is building Jazz [3] as a scalable and extensible team collaboration platform for integrating development work across task, build, source code, and planning management activities. Jazz is developed by a globally distributed team that uses Jazz to manage its own work. The Jazz platform uses a client-server architecture, where the server is the central data repository that stores data for Jazz components. The repository is accessible using a web-based client interface or an Eclipse-based client. Our elements for constructing social network map to Jazz artifacts as follows:

Project Members are *contributors* in Jazz. Personal information, such as the name and email address of each contributor, as well as project related information, such as team affiliations are available.

Collaborative Tasks are *work items* in Jazz. Work items represent the basic unit of work in Jazz 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, a creation date, a description, and an owner.

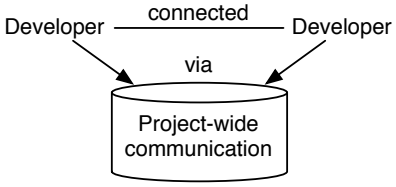
Task-related Communication is *comments* on work items in Jazz. Reading and writing comments on work items

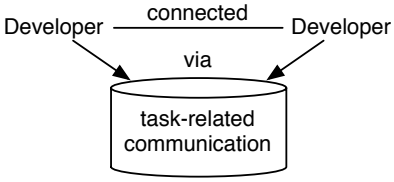
Constructing Developer Networks: An Overview



Technical-based Networks:
Source code repositories were used to construct developer networks by leveraging technical dependencies from code [1,2,3,4]. Two developers are connected in a network if both changed the same file, module, or project.

Project-wide Networks:
Networks on a project-wide level[4,5,6,7,8] are build without a specific task focus. In these networks people are linked used any project-related communication. Communication can be mined from repositories, such as email archives and issue trackers.





Task-based Networks:
We mine social networks that evolve around collaborative tasks, such as fixing a bug or implementing a feature. Similar to other approaches [10,11,12], we mine the communication directly related to a certain task, and thus build fine-grained networks. [8,9]

[1] Ohira, M., Ohsugi, N., Ohaka, T., Matsumoto, K., Accelerating Cross-Project Knowledge Collaboration Using Collaborative Filtering and Social Networks, *In Proc. of the 2nd International Workshop on Mining Software Repositories, 2005.*

[2] Pinzger, M., Nagappan, N., Murphy, B., Can Developer Social Networks Predict Failures?, *In Proc. of the Foundation of Software Engineering, 2008.*

[3] Meneely, A., Williams, L., Snipes, W., Osborne, J., Predicting Failures with Developer Networks and Social Network Analysis, *In Proc. of Foundation of Software Engineering, 2008.*

[4] Goecks, J. and Mynatt, E. D., Leveraging Social Networks for Information Sharing, *In Proc. of the Conference of Computer Supported Collaborative Work, 2004.*

[5] Valetto, G., Helander, M., Ehrlich, K., Chulani, S., Wegman, M., Williams, C., Using Software Repositories to Investigate Socio-technical Congruence in Development Projects, *In Proc. of the 4th International Workshop on Mining Software Repositories, 2007.*

[6] Bird, C., Gourley, A., Devanbu, P., Gertz, M., Swaminathan, A., Mining Social Networks, *In Proc. of the 3rd International Workshop on Mining Software Repositories, 2006.*

[7] Cataldo, M., Herbsleb, J. D., Communication Patterns in Geographically Distributed Software Development and Engineers' Contributions to the Development Effort, *In Proc. of the International Workshop on Cooperative and Human Aspects of Software Engineering, 2008.*

[8] Nguyen, T., Wolf, T., Damian, D., Global software development and delay: Does distance still matter? *In Proc. of the International Conference on Global Software Engineering 2008.*

[9] Wolf, T., Schröter, A., Damian,D., Nguyen, T., Communication, Coordination and Integration. *Technical Report DCS-322-IR at the University of Victoria 2008.*

[10] Cataldo, M., Wagstrom, P., Herbsleb, J. and Carley, K. (2006). Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools. In Proceedings of the Conference on Computer Supported Cooperative Work (CSCW'06)

[11] Herbsleb, J.D., & Mockus, A. (2003). Formulation and preliminary test of an empirical theory of coordination in software engineering. In Proceedings, European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering, Helsinki, Finland, September 1-5, pp. 112-121.

[12] Ehrlich, K, Helander, M,, Valetto, G. Davies, S. and Williams, C. (2008). An Analysis of Congruence Gaps and Their Effect on Distributed Software Development. Socio-Technical Congruence Workshop at ICSE conference, Leipzig, Germany, May 2008.

is the main collaboration mechanism in Jazz as developers use comments to debate and discuss decisions. A thread of comments forms a conversation that is attached the work item. Each comment has a creation date, comment text, and an authoring contributor.

2.1 Data Mining Tools for Jazz

In order to extract the elements 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, poten-

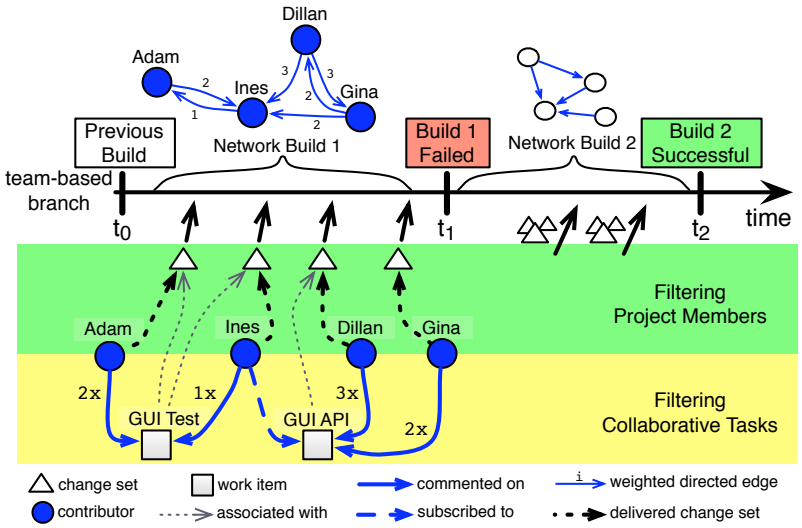


Figure 2. Construction of social networks for build failure prediction

tially 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.

Further, the querying and processing of the extracted data to construct social networks is an additional challenge because it is data and time intensive. For example, extracting and processing all work items from the repository took several hours. Thus, it is not feasible to extract and process all of the information needed to construct social networks in real-time. For this reason, we import the data of interest into a reporting and analysis database. This approach allowed us to 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 Framework (JUNG) 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.

2.2 Research Studies Using Social Network Analysis

We used the mined task-based social networks in two of our research studies that map to the more general build failure [7] and communication broker [5] 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 with the integration outcome. Code integrations (referred to as builds in Jazz) 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 due to compilation, testing, or other integration errors.

To integrate in Jazz, the members of a development team 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. As such, the *collaboration scope* of this study is the communication of contributors who have delivered change sets that were included in each build.

To construct the social networks for Build 1 at time t_1 (as illustrated in Figure 2), we followed the steps outlined in our Failed Build example 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 add a directed edge between each pair of contributors who have 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 each integration build of the five teams (between 48 and 60 builds for each team), we computed different social network mea-

	Teams					Std.Dev.
Recall	55%	75%	62%	66%	74%	8.38%
Precision	52%	50%	75%	76%	66%	1.34%

Table 1. Prediction results for the five teams

surements, such as *centrality*, *betweenness*, and *density* [6] and investigated their relationship to integration outcomes. Although none of these measures can independently predict the build outcome, when used in combination they are more powerful. We developed a predictive model by training a Bayesian classifier that was able to accurately predict failed build results. The recall and precision values of the predictions using measures of social network structure is shown in Table 1 for the five teams. In order to validate our model for each of the five teams we used *leave one out cross validation*, which trains a model for each set of data points that is in size one smaller then the full set, and then predicts the data point that is not in the training set. Study details are available in technical report [7].

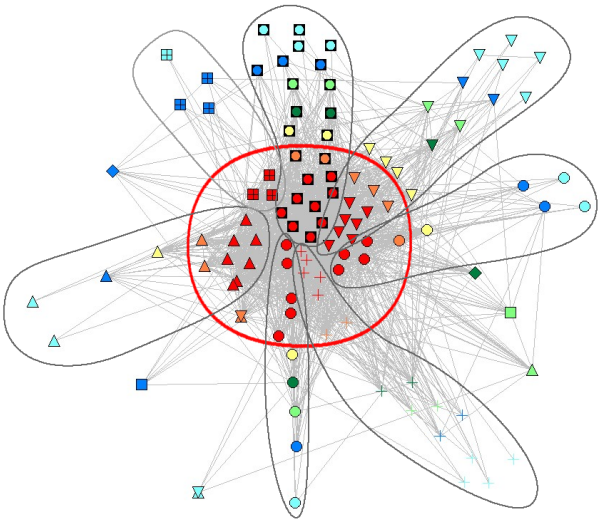


Figure 3. Project-wide communication-based social network

Communication Structures in a Large Distributed Project

In this second study [5] we were interested in the project-wide communication structure of the geographically distributed Jazz development team. 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 properties of the social networks, such as cohesiveness of the distributed teams, that were related to the results of communication delay and distributed collaboration.

The *collaboration scope* of this study is the entire project, and thus includes all collaborative tasks, and all contributing project members. In this case, connecting project members is the key step to create a project-wide social network.

To construct the project-wide social network we followed the steps as exemplified in Section 1.3 (the communication broker example).

The resulting social network is illustrated in Figure 3 in which ovals indicate the teams at each geographic site. We used the *k-core* and *core-periphery* tests [6] to analyze the properties of the network structure. These tests allowed us to test whether multiple cores can be identified in the network, or the network shows a star-like structure in which a single core mediates communication with the developers on the periphery of the network.

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 other 25 developers from the core. The other colors, from yellow to light blue, indicate different lower degrees of communication in the project. Further, using the *group degree centrality* [6] measure, we also found that each geographic location has roughly equal centrality and used 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 each other geographic location. This may explain why distance had an insignificant effect on communication response and task resolution time in the large distributed team.

In summary, we learned from this study that communication delay as a result of distribution in global teams can be overcome with recent collaboration tools and practices. From our experience with the Jazz team, best practices such as prioritizing off site requests and tools that integrate development, project management and communication play a major role in achieving these results.

3 Practical Implications for Deployment

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

Scalable Mining and Analysis Tools

By incrementally mining and using a secondary reporting

database (or data warehouse), we can reduce 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. This has been proven useful during our extraction of information from the Jazz repository, which was under constant load due to the global distribution of the more than 140 Jazz team members. Extracted data, constructed networks, and network analysis measures can be stored 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. Computationally intensive analysis and predictions can be conducted using the data warehouse or by a client accessing the data warehouse. Once task-based social networks and network analysis measures are stored in a data warehouse, visualization of social networks and further analysis becomes efficient and non-intrusive to the users of repositories.

Team Awareness through Social Network Visualization Developers working on interdependent tasks can benefit from the visualization of the social network of the team members who contributed to the interdependent tasks. The information in the network can include developer contact and availability, as well as a list of other tasks that they are working on. This information is particularly important for newcomers to a project, who lack project specific expertise, such as who has been involved with a particular task, or in architectural decisions. We are currently conducting a case study to identify which information is valuable to support developers and what visualization conveys them most effectively.

Outcome Prediction Using Social Network Analysis Another application for social network analysis is to predict project outcomes such as build results based on information about team communication behaviour. Practitioners can build tools that are embedded in the development environment that inform developers 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.

Building a predictive model from social networks constructed with data mined from task and task-communication repositories could be done with the following four steps:

1. **Initialization Predictive Models:** Initialize the predictive model using the social networks and outcomes for all existing builds. Store the data on social networks, build outcomes, and predictive model in a data warehouse for later use.
2. **Construct Social Network for Upcoming Build:** Construct the social network for the upcoming build as described in our approach.
3. **Predict the Upcoming Build Outcome:** Calculate network analysis measures for the constructed social network and use them as input to the predictive model. The model then predicts an outcome for the upcoming build. At this point, a manager could make proactive adjustments to attempt to prevent a predicted build failure.
4. **Update Predictive Model:** Finally, after the upcoming build has been completed, update the predictive model to include 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 and prompt managers to rectify the problem before the build occurs, as described next.

Effective Management through Social Network Analysis Beyond visualizations, computed social network measures can be used by management as an aid in preventing failed builds. Our research shows that the quality of communication does matter in the quality of integrations. Monitoring and affecting communication behavior is a strategy to prevent failures. Software engineers differ in the expertise and knowledge they bring to collaborative tasks. Team performance depends not only the information available to 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 in identifying problematic communication behavior likely to result in failed integrations. Actions that management can take to prevent failures for a particular team include:

1. **Compute Social Network measures or Run Predictive Model early in the project:** In our study of Jazz, the predictive model performed well even with the first 25% of the team communication data (i.e. first quarter of project timeline). If this is true in other development environments, practitioners should use this information very early in the project. If the model predicts error, see next step. Otherwise, computing social network measures such as density, betweeness and centrality early in the project is an alternative option that could signal problematic communication behavior in the project.

2. **Examine team communication structure:** Identify the presence of patterns of communication that can, when analyzed in light of project specific technical dependencies, be problematic. An example of problematic communication structure indicated by social network measures is that of a team with high coordination needs before an integration but with a communication network of low density. It is possible that there are too many missing communication links between developers that have technical dependencies [2], or there is an overall lack of communication in the team. Another example is of a team with several distributed subteams that need to coordinate and in which a number of developers have high betweeness but the subteams themselves have little communication across distance. This may indicate that there are clusters of developers who should communicate directly, and that the subteams are only loosely connected and have communication brokers that may become bottlenecks.

3. **Improve communication or knowledge management procedures:** Adjust communication and knowledge management processes or tools in the project, to address specific situations identified at previous step. If communication links between developers with coordination needs are missing, assigning communication brokers so that these developers can better coordinate interdependent work is an option. Equally valuable, increasing the awareness of these coordination needs as well as developers' expertise areas in the current project could be done through regular project meetings or more adequate documentation of these dependencies in project specific knowledge repositories. In other situations, if communication across distances is relying on information brokers, managers should consider supporting alternate points of contact in the clusters identified as relying on these brokers, or enforce documentation of information that becomes available to everyone in the project. Both these options help mitigate the risk of information brokers becoming unavailable in the project.

Despite our intention to be as specific as possible, these recommendations for action should consider and be applied in the particular context of each project. Managers should use the insights obtained from the examination of communication structures in their project as a starting point in their further analysis of communication in relation to the particular technical and organizational characteristics of their project.

4 Conclusions

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 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, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks. In *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*, pages 137–143, New York, NY, USA, 2006. ACM.

[2] K. Ehrlich, M. Helander, G. Valetto, S. Davies, and C. Williams. An Analysis of Congruence Gaps and Their Effect on Distributed Software Development. In *Socio-Technical Congruence Workshop at the ICSE conference*, May 2008.

[3] R. Frost. Jazz and the Eclipse Way of Collaboration. *IEEE Software*, 24(06):114–117, 2007.

[4] J. D. Herbsleb and A. Mockus. Formulation and preliminary test of an empirical theory of coordination in software engineering. In *ESEC/FSE-11: Proceedings of the 9th European Software Engineering Conference held jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 138–137, New York, NY, USA, 2003. ACM.

[5] T. Nguyen, T. Wolf, and D. Damian. Global software development and delay: Does distance still matter? In *Proceedings of the 3rd International Conference on Global Software Engineering (ICGSE 2008)*. IEEE Computer Society, August 2008.

[6] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[7] T. Wolf, A. Schröter, D. Damian, and T. Nguyen. Communication, Coordination and Integration. Technical Report DCS-322-IR, University of Victoria, June 2008.