Association for Information Systems
AIS Electronic Library (AISeL)

ECIS 2007 Proceedings

European Conference on Information Systems (ECIS)

2007

Mapping Social Network to Software Architecture to Detect Structure Clashes in Agile Software Development

Chintan Amrit
University of Twente, Netherlands, c.amrit@utwente.nl

Jos van Hillegersberg
University of Twente, j.vanhillegersberg@utwente.nl

MAPPING SOCIAL NETWORK TO SOFTWARE ARCHITECTURE TO DETECT STRUCTURE CLASHES IN AGILE SOFTWARE DEVELOPMENT

Chintan Amrit, University of Twente, P.O. Box 217, 7500 AE, Enschede, Netherlands, c.amrit@utwente.nl

Jos van Hillegersberg, University of Twente, P.O. Box 217, 7500 AE, Enschede, Netherlands, j.vanHillegersberg@utwente.nl

Abstract

Software development is rarely an individual effort and generally involves teams of developers collaborating together in order to generate reliable code. Such collaborations require proper communication and regular coordination among the team members. In addition, coordination is required to sort out problems due to technical dependencies that exist when components of one part of the architecture requires services or data input from components of another part of the architecture. The dynamic allocation of the different tasks to people results in various socio-technical structure clashes (STSCs). These STSCs become more pronounced in an Agile Software Development environment and managerial intervention is constantly required to alleviate problems due to STSCs. In this paper we provide a method to detect these STSCs in a longitudinal fashion with the help of a tool that we are developing. We test this method in a case study of a software company and show how such structure clashes can be detected by analyzing the social network (along with the betweenness centrality index) in relation to the task dependencies due to the software architecture.

Keywords: Agile Software Development, Social Network, Software Architecture.

1 INTRODUCTION

Coordination is an important part of the software development process. Traditional approaches to coordinating software development have focussed on development of new and enhanced tools, modularisation (both technical and managerial), formal procedures involving tools and documents to control communication among personnel (Kraut and Streeter 1995; Crowston 1997; Andres and Zmud 2001). However, such coordination mechanisms don't work in an agile software development environment that encourages very short development and delivery cycles. Small Companies tend to prefer informal communication among team members and customers over more formal approaches to coordination. In such a scenario it is essential to see to it that the necessary communication takes place. One way to see this is through identifying the dependencies that drive communication. In our research we have observed that the technological dependencies drive the need for coordination.

The technological artefacts not only represent the essential constraints and needs but also represent social practice by reflecting the working arrangements, division of labour etc. This discussion were broached by Melvin Conway (1965) in what has come to be known as Conways's law (Conway 1968). While Conway showed that the structure of the code should mirror the structure of the team that designed it, Parnas recognized that the process of decomposing the software into smaller pieces was not only a technical division but also a division of labour among the individuals (Parnas 1972).

In our research we refer to the clash between the social and technical dependencies as socio-technical structure clashes (hereafter referred to as STSC), that deals with the social dependencies that are lacking and coordination problems due to technical dependencies between the software components

that the developers are working on them (Amrit, Hillegersberg et al. 2004; de Souza, Redmiles et al. 2004).

There is very little literature available that directly addresses the problem of detection of STSCs. Most of the literature focuses on the problem at the level of software requirements, design and implementation. Coplien and Schmidt (1995) describe the problem (pattern no. 15) as" preserving architectural vision through implementation" (Coplien and Schmidt 1995). The solution that they suggest is that architects should also participate in the implementation process. Though this solution sounds simple, it is not a solution to the coordination problems that arise when there is a gap between the designed and the implemented process model. Another related solution at level of software is by Murphy, Notkin and Sullivan (2001). They develop an approach that helps the engineer use a high-level model of the structure of an existing software system as a lens through which to see the system's source code. The tool they design computes a software-reflexion model that shows where the engineer's high-level model agrees with and where it differs from a model of the source(Murphy, Notkin et al. 2001). This approach of using a software reflexion model is good for addressing the gap between design and implementation at the software level but it fails to address the problems in the social side. For example, task allocation problems that could further lead to delays at the software process level are not addressed.
The concept of a Design Structure Matrix (Eppinger 2001) tries to highlight the inherent structure of a design by examining the dependencies that exist between its component elements in a symmetric matrix. Recent studies using the Design Structure Matrix (DSM) provide insights into the relation between product architecture and organizational structure. Sosa et al. (2004) show instances of misalignment between design and team interactions are more likely to occur across organizational and system boundaries (Sosa, Eppinger et al. 2004). The focus of the DSM method is on the task matrix, so the actual social network (Wasserman and Faust 1994) of the employees is ignored.

de Souza et al. (2004) suggest that Application Program Interfaces (APIs) rather than act as a means for better collaboration actually constrains possible collaboration. They also argue that the source-code of the software can act as an important resource to identify social dependencies. As dependencies between software modules can suggest social relations that need to be built among software developers in order to facilitate the integration process. Wagstrom and Herbsleb (2006) describe an automated system which can be used to identify future modes of interaction and coordination based on analyzing the software code from the Central Versioning System (CVS).

By looking only at the source code (de Souza, Redmiles et al. 2004; Wagstrom and Herbsleb 2006), one can at best predict what future coordination should be like but not locate actual coordination problems.

In our research we try to find real communication problems by actually analyzing the social network of the developers and comparing it with the software architecture. We have devised a technique to detect STSCs by mapping Social Networks on the Software Architecture. By analyzing the social network in relation to the task dependencies due to the software architecture, we can derive conclusions on STSCs. Further we show how just by the examination of the social network, one can identify some STSCs, through a study of the network's betweenness centrality. We outline this technique in a case study of a software company called MENDIX. We do this with a help of a software tool we have developed called TESNA that can display different socio technical networks and their dependencies.

The rest of the paper is structured as follows; section 2 describes what we mean by STSCs, section 3 describes the research site and the methods that we have used to conduct our research, section 4 describes the system architecture of the software project under study, section 5 explains the betweenness centrality and describes the feedback from the CTO, section 6 contains the discussion and section 7 the conclusion and future work.

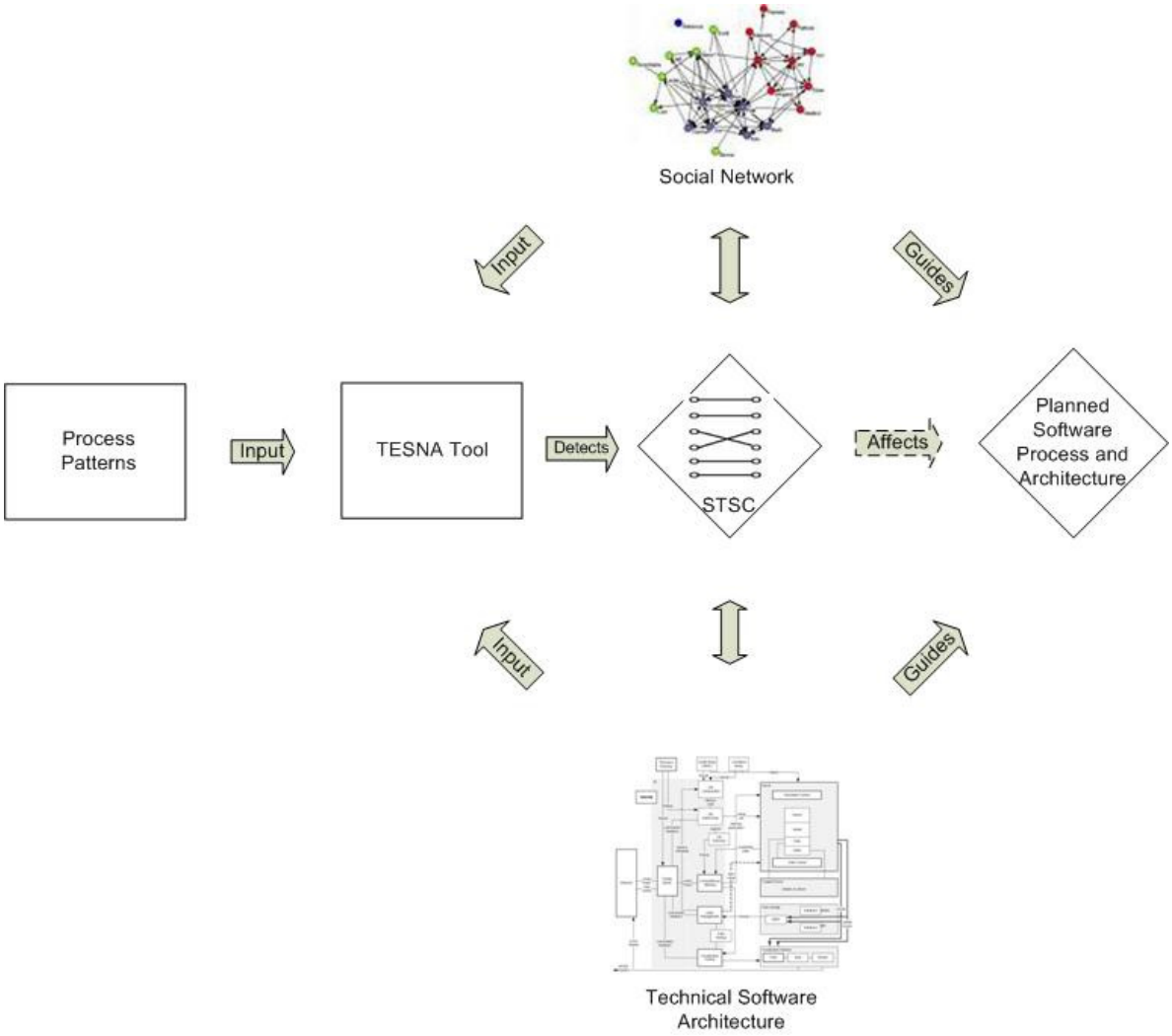

Figure 1. Planned Software Process refactoring using Socio-Technical Structure Clashes

2 SOCIO-TECHNICAL STRUCTURE CLASHES

An STSC occurs when there exists a Process Pattern that indicates that the social network of the software development team does not match the technical dependencies within the software architecture under development

These STSCs are thus indicative of coordination problems in a software development organization. Some of these problems (or STSCs) concerning development activities have been tackled by Coplien et al. and are called Process Patterns (Coplien 1994; Coplien and Harrison 2004).

However, research done by Coplien et al suffers from the following problems:

- these patterns are hard to implement
- there is very little research on their empirical validation

The reason why the patterns are hard to implement is that the problems addressed by the patterns are hard to detect, as purely manual techniques are labour intensive. A regular detection of STSCs becomes more essential in an Agile Software Development environment where it's very easy to lose oversight of the development process. Manager intervention becomes essential in such an agile environment in order to keep the development process in check

Figure 1 represents the focus of this paper, which is to provide a Method based on our tool called TESNA (TEchnical and Social Network Analysis) that can improve the planned software process. STSCs are based on Process Patterns and represent the coordination problems in the social network when the dependencies in the technical architecture are not met. The Method consists of several steps. First, input from the Social Network as well as the Software Architecture is taken, and the tool TESNA provides a visual description of the networks and metrics, based on Process Patterns. The next step of the method lies in the detection of STSCs with the help of the visualizations that can help the manager in reengineering the informal design process model in order to improve project planning (Fig. 1). Currently the tool supports both manual (qualitative) as well as automatic (quantitative) detection of STSCs, but for the purpose of this case study we used qualitative STSC detection methods for reasons described in section 4. For the purpose of this paper we selected two process patterns, one based on what has come to be known as Conway's Law (Conway 1968), and the other based on betweenness centrality (Hossain, Wu et al. 2006).

3 RESEARCH SITE AND METHODS

The research methodology used is one of Design Science Research (Hevner, March et al. 2004), we design a tool called TESNA (short for TEchnical and Social Network Analysis) and use it through case study (Li, Zhou et al. 2005) in a software development company. The different steps as described by Hevner et.al. (Hevner, March et al. 2004) were carried out and as explained in section 5 the evaluation was done with the help of the feedback from the CTO of the software company.

Our case study was conducted in a software company called MENDIX, who are developing a large middleware product. The company MENDIX is one of the fastest growing companies in The Netherlands and are listed FEM Magazine as one of the top 25 brilliant software start-ups in The Netherlands (Bueters 2006). One version of the middleware product was already released at the time of the study.

The reason behind choosing this case was that MENDIX followed an agile software development methodology, we were interested in whether a small company like MENDIX has STSCs and if we could detect them.

The system architecture consists of a client system, a Work flow server and a Modelling server (Fig 2.). The project staff included 15 full time personnel; 8 full time developers, 2 project leaders, 2 project managers, 2 sales people and one system administrator. The personnel were divided into 3 teams, with 3 developers, one project leader and one project manager for the client system, 3 developers for the Modelling Server and 3 developers and one project manger for the workflow server.

The data was collected in fall 2006 over a period of 3 months, though participant observation, interviews and gathering work related documents. Among the documents observed were the chat logs, which were stored in XML format. The logs of chat transcripts for 4 weeks, each week evenly distributed in the 3 month period, were analysed with the help our software tool, TESNA.

All the main developers, project leaders and project mangers were interviewed. Among the questions asked in the interview were; who they discuss work related subjects with (advice, discussion and work flow), how much data was exchanged per communication, and, what the mode of communication was. It was ascertained that almost all technical communication was done through online chat. This was because they had a dedicated Jabber chat server (Adams and Adams 2001) running for the company

(which eliminated wastage of time due to external chats), and it was considered more efficient than face to face communication for the developers. The primary ties in the social networks analysed from the chat log corresponded with those that the interviewees had themselves provided. Further, through participant observation of the software developers (in 6 separate visits lasting a day each) it was ascertained that almost all technical communication was through the online chat.

The data was analysed and then discussed with the CTO of the company, who doubled as a project manager (Roald in Fig 2.). With the help of our software tool TESNA, the social networks for four different weeks (each with cumulative chat data over the period of a week) of only the developers and project leads/mangers were constructed. The chat records were parsed and displayed as social networks by TESNA with the chat ids, as labels for our nodes in the social network. Though most of the first names corresponded with their actual first names, the only exception was Max whose name is listed as mnemisj.

The tool TESNA is currently capable of analysing and displaying the dependencies in the software call graphs (the software is read from the Concurrent Versioning System, or CVS), it can also display the authorship information of the software modules (also read from the CVS) and for the social side of the software development it can construct and analyse software metrics from XML logs of chat messages (the chat server being Jabber). The tool can also display the different metrics of the social network over a period of time. We used this option to analyse the betweenness centrality of the social networks over the period under study.

We took the data back to the CTO once it was displayed and analysed. In this way we could ascertain whether our technique was really useful to the CTO.

4 TECHNICAL SOFTWARE ARCHITECTURE

Most of the literature on socio technical dependencies (de Souza, Redmiles et al. 2004; Wagstrom and Herbsleb 2006) focuses on gathering the dependencies from the recently modified source code (from CVS). We tried this approach by analysing the source code of the company MENDIX with the help of our tool. We realised that as the company is small most of the dependencies in each section of the architecture (client, xml server, and modeller server) were satisfied by the communication among the developers working on them. Also, knowledge of the technology used in the particular platform (Java, JavaScript or Delphi) was an essential prerequisite for a developer to be working in that part of the project. Due to this fundamental skill requirement we noticed that developers seldom worked on projects or changed code other than their own assigned part of the Architecture. As each developer worked in only specific parts of the code, and architecture, there were workflow dependencies between the colleagues due to the architecture. The dependencies due to the XML input and output between the client/server and the servers couldn't be detected by only analysing the call graph and function call dependencies (Valverde, Cancho et al. 2002). Thus, we realised that a quantitative analysis of the source code of the software product isn't very helpful in analysing the dependencies for a small corporate company like MENDIX.

We analysed the Socio-technical Structure clashes between the actual social network of the employees involved in developed and the technical artefacts that they develop with the help of the software architecture. The software architecture of the software product produced at MENDIX along with the task allocation of the software developers and project leads of MENDIX is represented by Fig 2. This diagram gives a sense of the dependencies as a result of task allocations related to the Software Architecture of the system. The XML interchange indicates that there exists an input and output dependency between the Server, the XML server and the Client System.

The cumulative chat logs were analysed over a period of a week and converted into a social network of the developers and project leaders with the help of our tool (we use JUNG to display and calculate metrics (Madadhain, Fisher et al. 2005)). The social network (Wasserman and Faust 1994) was represented with labels, and the strength of each link was determined by the number of chat messages

exchanged. The black links were drawn for the maximum number of chat messages exchanged, and dotted links if the number of chat messages was less than half of the maximum that week. The system architecture was then superimposed on the social networks in order to ease the detection of STSCs. We also calculated the degree and betweenness centrality (Freeman 1977) of the nodes and plotted a graph showing its variation over the 3 month period. The resultant diagram was shown to the CTO for his input.

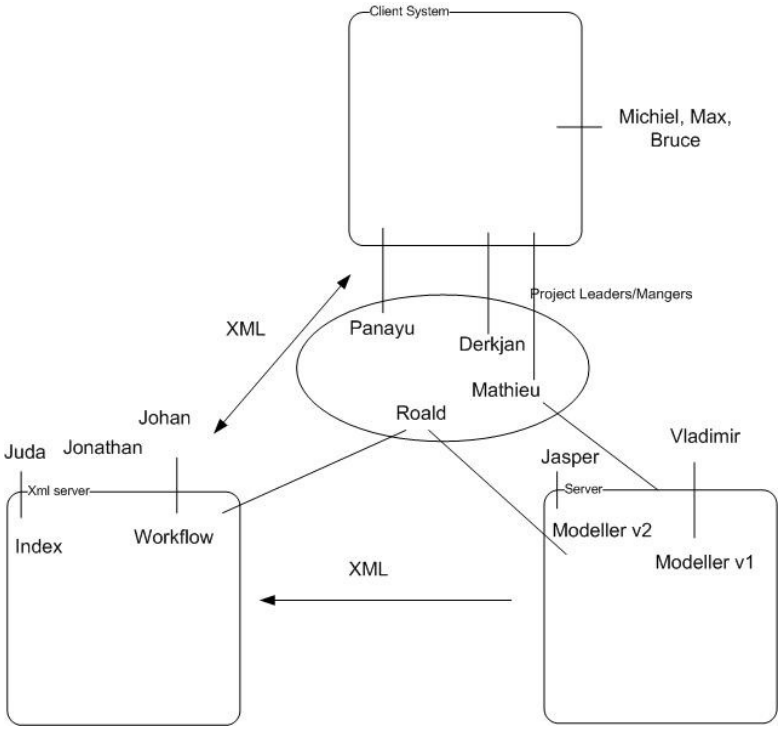

Figure 2. The Software Architecture along with the task responsibilities

5 CENTRALITY AND CTO FEEDBACK

Centrality index gives us an idea of the potential importance, influence and prominence of an actor in a network (Freeman 1978). Betweenness refers to the frequency with which a node falls between pairs of other nodes in the network. In other words, betweenness centrality is a measure of, "*the degree that each stands between others, passes messages and thereby gains a sense of importance in contributing to a solution, .. , the greater the betweenness, the greater his or her sense of participation and potency*" (Freeman 1977). In terms of coordination, betweenness maybe the most appropriate measure of centrality as it provides a measure of the influential control of each node (employee) on the whole networks (Scott 2000). This is the reason we used betweenness centrality to analyse potential STSCs.

During our interview of the CTO we asked the CTO to grade 15 different Process Patterns (from Coplien's paper (Coplien 1994)), in a scale of 1 to 10. The highest grade he gave was for Conways law pattern (Conway 1968), in his words

"*it is very important that the organization of the company is according to the architecture and I want all the developers to communicate to resolve their problems*".

So, the CTO was quite pleased when we showed our tool which maps the social networks to the software architecture of his company's product.

When asked how he would expect the social network of the developers and project leads in his company to look, the CTO said

"*I would expect Vladimir, Jonathan and Michiel to be central, as they are the Gurus in the work they do..*

and no one knows the functioning of the server, xml server and the client better than them"

The social network from week I (Fig 3.) was interesting as the CTO immediately spotted a STSC, which was the missing link between Jonathan and Juda, both of whom are developers for the XML server (Fig2.). The CTO found a puzzling link was between Mathieu and Johan, which he later realized was because they had to do a presentation together that week.

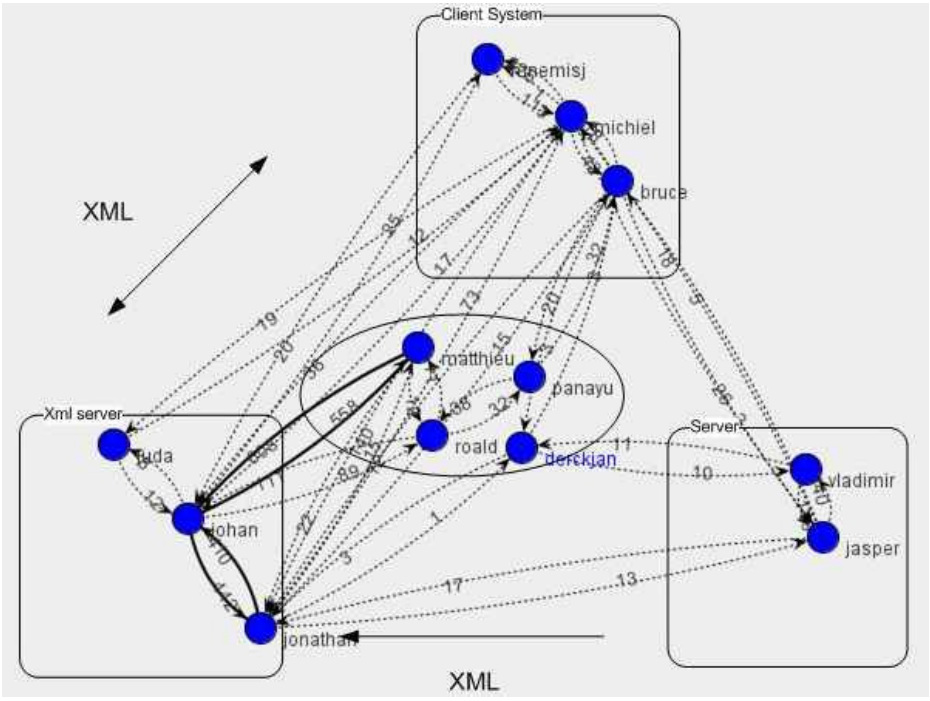

Figure 3. The social network mapped onto the Software Architecture for week I

The CTO found the social network from week II (Fig 4.) more reasonable than week I, even though there was no connection to Johan (who was away doing field studies). The three central players were Jasper, Michiel and Jonathan which was what he had expected according to the tasks and results in that week. He found that there was little communication with Derkjan (who is the project manager for the client part of the architecture Fig 2.), which he found strange as there was some trouble with the client that week.

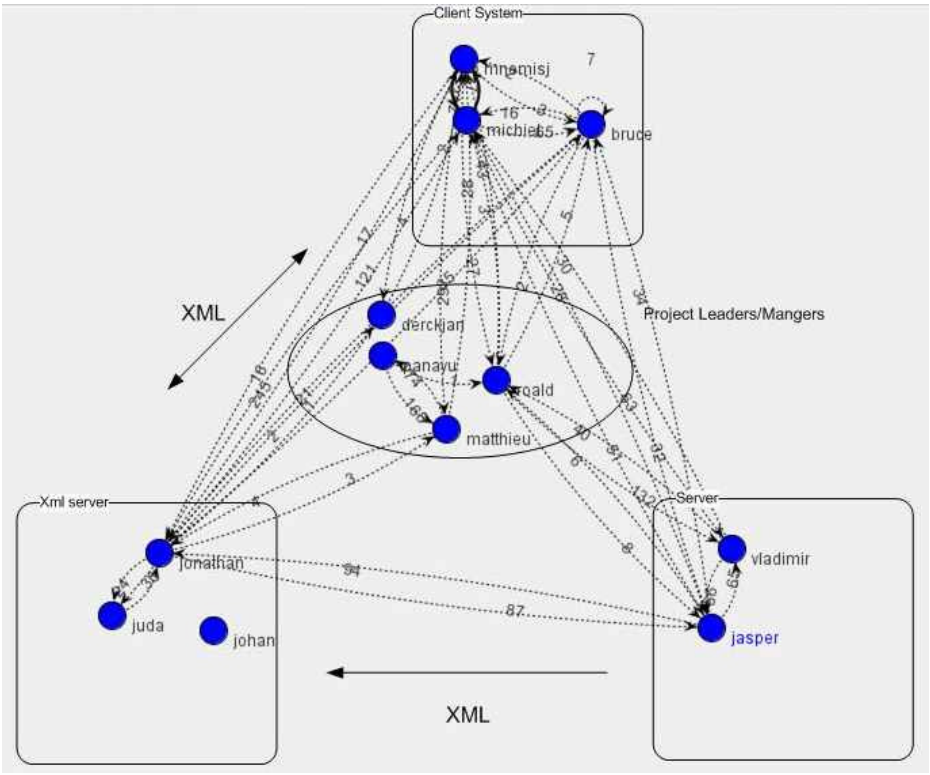

Figure 4. The social Network mapped onto the Software Architecture for week II

Week III (Fig 5.) was interesting as many of the employees were on vacation, and the CTO was interested in how the employees communicated. There was no communication between Jasper and Michiel, as Jasper was supposed to work on the Client that week. This could be an indication of a potential problem (or STSC). Also, the CTO found the fact that Mnemisj (Max) was central quite surprising.

In week IV the fact that Michiel was not communicating was surprising as the deadlines were near and it would have been important that he spoke with his fellow client developers. The reason behind Panayu and Mathieu (having high out-degree) being central was that there was a product shipment on week IV which caused the project leaders to play a more central role. The strong link between Jonathan and Mathieu was quite strange according to the CTO as they wouldn't have the need to communicate on technical problems. The fact that Bruce had a central role seemed quite strange to the CTO, while the CTO was quite surprised that Derkjan wasn't communicating much in the week with the shipment deadline.

6 DISCUSSION

The change in the betweenness centrality index (Freeman 1977) over the 3 month period can give us an idea of how the most important employee (or the employee who handles most of the communication) in the network changes depending on the tasks at hand. On observing Fig 7, we see that the employees who are important to each part of the software architecture (or the gurus as CTO called them) namely, Jonathan, Michiel and Vladimir were very central during the period around the first week. This period, we realize was exclusively for software development where their expertise was very much (as the CTO named them as the experts in their domain) in demand by fellow developers and project leaders. However, as the project moves towards delivery of the product we find the core

developers taking a more passive role in the network while the non-core developers like Jasper, Bruce and Max (mnemisj) as well as the system integration experts take a more central role. This can be explained by the fact that a greater amount of integration and front end work is required near the delivery deadline.

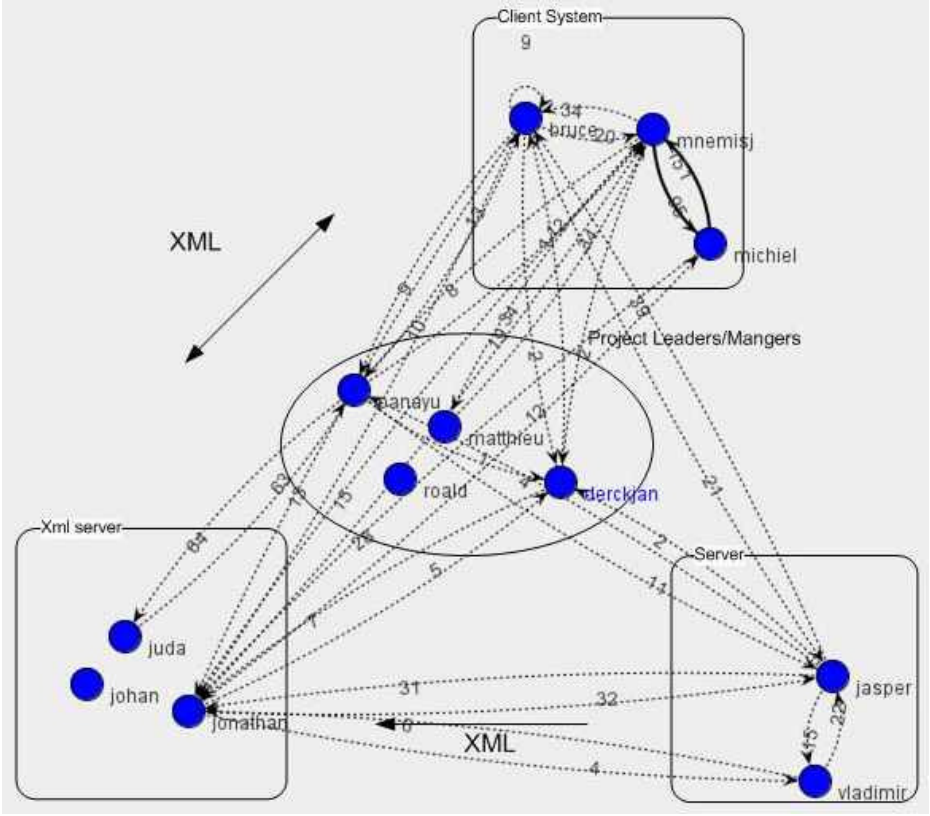

Figure 5. The social Network mapped onto the Software Architecture for week III

We also notice that the project leaders and managers (Panayu and Derkjan) assume a more central role when the project is nearer to the deadline for delivery to the customer (week IV). This movement to a more central role is required by the project leaders and managers in order to be able to control all the possible contingencies that might crop up near the delivery time. This display of the variation of betweenness centrality index of the social network can also help a Manager in recognizing STSCs relevant to different stages in the agile software process. When a person is too central for the wrong reasons, i.e. when a developer is taking responsibility to communicate with the rest of the team, then such a scenario could be a cause for a structure clash. For example, the CTO was surprised that Max (mnemisj) had a central role in the week III when not much work was required at the client side, he was also surprised that Bruce was central in week IV. There is also cause for concern (potential STSC) when two employees working on (or managing) the same part of the architecture (that is being modified) are not communicating with each other, for example Michiel and Derkjan were not communicating in any of the weeks under observation.

7 CONCLUSION AND FUTURE WORK

Agile software development can prove to be very difficult to manage in the wake of constantly changing requirements and short planning horizons. In such a scenario it's very difficult and labour intensive for the project manager to keep track of communication problems, or STSCs (Wagstrom and Herbsleb 2006). Identifying the problem areas related to STSCs can particularly prove difficult when multiple people are responsible for various tasks and when the task requirements keep changing in an agile software development environment

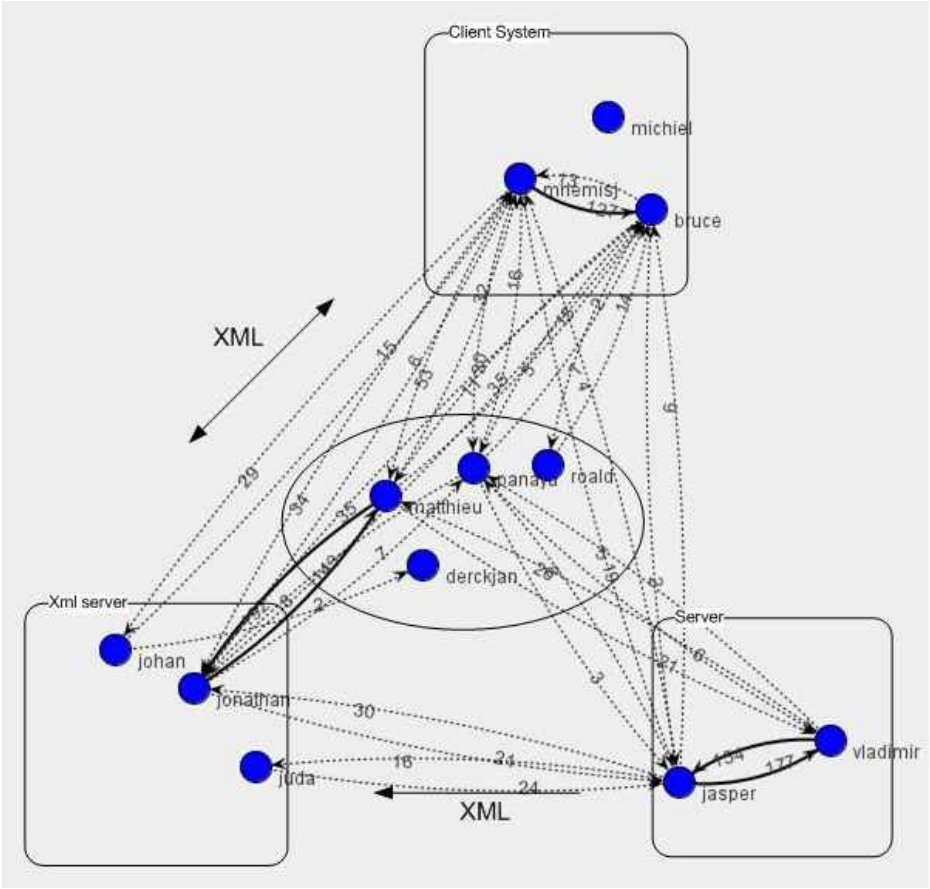

Figure 6. The social Network mapped onto the Software Architecture for week IV

We have tried to show how we can reduce this management overhead by our technique of identifying structure clashes through the analysis of the social network in relation to the task dependencies due to the software architecture.

By mapping the social networks onto the software architecture we not only see how the developers and project leaders communicate but also their task responsibilities arising from the software architecture. Further, we also calculate the betweenness centrality to identify who the most important person is, from the network point of view (Freeman 1977; Freeman 1978; Hossain, Wu et al. 2006). The display of the change in the betweenness centrality can give us an indication of how the most important person in the network changes with the change in task focus, and gives us another opportunity to identify STSCs. We think this technique is useful for the project manager who aims to improve the designed project plan (Fig 2.) and helps him/her facilitate the necessary communication.

Though one would expect to find STSCs in large software projects, we were surprised with the presence of STSCs even in a small company like MENDIX. This was further reiterated by the fact that the CTO wanted our tool in order to better manage the development process.

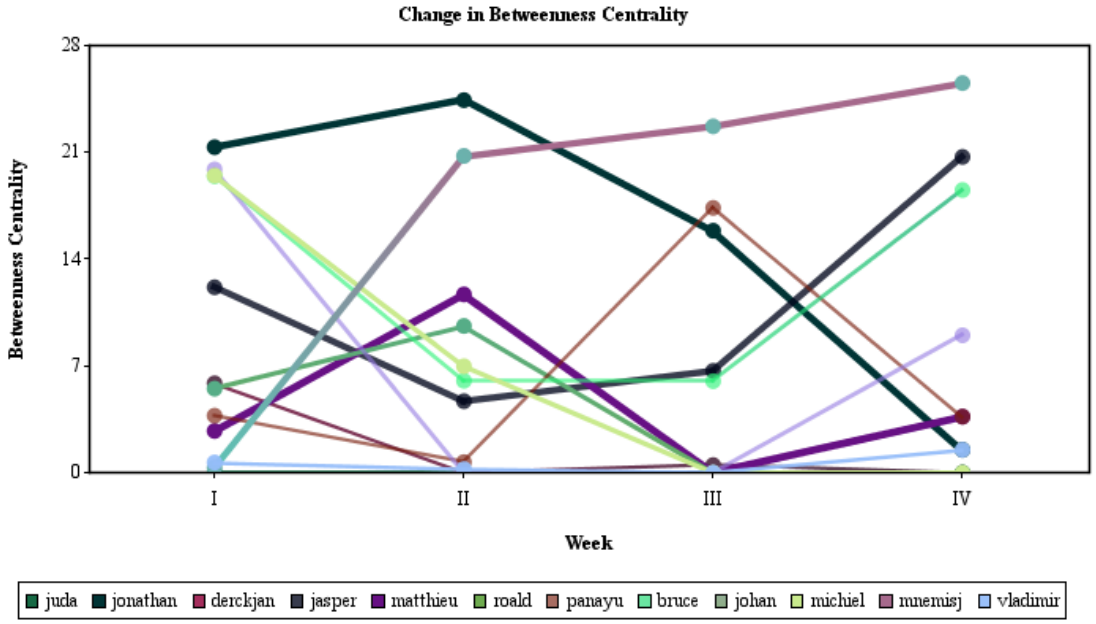

Figure 7. The change in the betweenness centrality over the four weeks

We realise that not all software companies would almost exclusively use chat as used in Mendix. In order to overcome this difficulty we plan to keep a track of e-mails (where allowed) as well as meetings and also get feedback on ego-networks with the help of a questionnaire.

In future, we plan to detect technical dependencies at different levels, for example at the code level (de Souza, Redmiles et al. 2004; Wagstrom and Herbsleb 2006), those at the level of the architecture and those at the level of work flow (Ven, Delbecq et al. 1976). Through the investigation of different dependencies we can gain insight into different possible STSCs. Dependencies due to the code structure we have found are more applicable to large software development organizations. We plan to conduct further case studies to study the presence of STSCs in large software development organizations. We have also started to investigate open source software development, to see the differences between corporate STSCs and Free/Libre open source STSCs.

References

Adams, D. J. and D. J. Adams (2001). Programming Jabber: Extending Xml Messaging, O'Reilly Associates, Inc.

Amrit, C., J. Hillegersberg, et al. (2004). A Social Network approach to Software Development. In CSCW'04 Workshop on Social Networks.

Andres, H. P. and R. W. Zmud (2001). "A Contingency Approach to Software Project Coordination." Journal of Management Information Systems Vol. 18(Issue 3): p41.

Bueters, P. (2006). Tech 25: Tech is back. FEM Business.

Conway, M. (1968). How do Committees Invent. Datamation. 14: 28-31.

Coplien, J., O (1994). A Development Process Generative Pattern Language. Proceedings of PLoP/94. Monticello, Il.: 1--33.
Coplien, J., O. and N. Harrison, B. (2004). Organizational Patterns of Agile Software Development. Upper Saddle River, NJ, USA.
Coplien, J. O. and D. C. Schmidt (1995). Pattern languages of program design. New York, NY, USA.
Crowston, K. (1997). "A Coordination Theory Approach to Organizational Process Design." Organization Science 8(2): 157-175.
de Souza, C., R. B., D. Redmiles, et al. (2004). Sometimes you need to see through walls: a field study of application programming interfaces. CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work. New York, NY, USA: 63--71.
Eppinger, S. D. (2001). "Innovation at the Speed of Information." Harvard Business Review Vol. 79(1): 149-158.
Freeman, L. C. (1977). "A Set of Measures of Centrality Based on Betweenness." Sociometry 40(1): 35-41.
Freeman, L. C. (1978). "Centrality in social networks conceptual clarification." Social Networks 1(3): 215-239.
Hevner, A., R., S. March, T., et al. (2004). Design Science in Information Systems Research. MIS Quarterly. 28.
Hossain, L., A. Wu, et al. (2006). Actor centrality correlates to project based coordination. Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work. Banff, Alberta, Canada, ACM Press.
Kraut, R., E. and L. Streeter, A. (1995). Coordination in software development. Commun. ACM. New York, NY, USA. 38: 69--81.
Li, B., Y. Zhou, et al. (2005). Matrix-based component dependence representation and its applications in software quality assurance. ACM SIGPLAN Notices. New York, NY, USA. 40: 29--36.
Madadhain, J. O., D. Fisher, et al. (2005). The JUNG (Java Universal Network/Graph) Framework. Technical Report UCI-ICS 03-17 University of California, Irvine.
Murphy, G., C. , D. Notkin, et al. (2001). Software Reflexion Models: Bridging the Gap between Design and Implementation. IEEE Trans. Softw. Eng. Piscataway, NJ, USA. 27: 364-380.
Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. Commun. ACM. New York, NY, USA. 15: 1053--1058.
Scott, J. (2000). Social Network Analysis: a handbook, Sage Publications Inc.
Sosa, M. E., S. D. Eppinger, et al. (2004). "The Misalignment of Product Architecture and Organizational Structure in Complex Product Development." J Manage. Sci. 50(12): 1674-1689.
Valverde, S., R. F. Cancho, et al. (2002). Scale-free networks from optimal design. Europhys. Lett. 17, avenue du Hoggar - Parc d'Activités de Courtaboeuf - BP 112 - F-91944 Les Ulis Cedex A - France. 60: 512-517.
Ven, A. H. V. D., A. L. Delbecq, et al. (1976). "Determinants of Coordination Modes within Organizations." American Sociological Review 41(2): 322-338.
Wagstrom, P. and J. Herbsleb, D. (2006). Dependency forecasting in the distributed agile organization. Commun. ACM. New York, NY, USA. 49: 55--56.
Wasserman, S. and K. Faust (1994). Social Network Analysis: methods and applications, Cambridge University Press.