TESNA: A Tool for Detecting Coordination Problems

Chintan Amrit, Jos van Hillegersberg

Department of IS&CM
University of Twente
{c.amrit, j.vanHillegersberg}@utwente.nl

Abstract

Detecting problems in coordination can prove to be very difficult. This is especially true in large globally distributed environments where the Software Development can quickly go out of the Project Manager’s control. In this paper we outline a methodology to analyse the socio-technical coordination structures. We also show how this can be made easier with the help of a tool called TESNA that we have developed.

1. Introduction

Coordination in large scale software development is a very difficult [1, 2]. The coordination problem is further multiplied in a globally distributed scenario [3]. Some of these coordination problems can be solved by following best practices. An example of such a best practice is what has come to be known as Conway’s Law [4]. Conway [5] states, organizations which design systems are constrained to produce designs which are copies of the communication structure of these organizations. Since Conway, researchers have invented various more detailed patterns which describe the preferred relationships between team communication structure (the social network) and technical software architecture [6]. We call such patterns Socio-Technical Patterns [7]. In addition to the classic law by Conway, various Socio-Technical patterns, including those from Coplien et al. [6] have been documented. However, these patterns have not been extensively validated empirically and can be hard to implement. The lack of empirical validation makes it complex for the project manager to decide on which Socio-Technical patterns to apply to his project. We provide the possibility to detect Socio-Technical Coordination problems (that we call STSCs) and also validate such Socio-Technical Patterns with the help of the TESNA tool and method we have developed. In this paper we show how the method and tool of TESNA works in more detail.

2. TESNA Method

As defined in [7] an STSC or Socio-Technical Structure Clash occurs if and when a Socio-Technical Pattern exists that applies to the actual social network of the software development team and the technical dependencies within the software architecture under development. STSCs are indicative of socio-technical coordination problems in a software development organization. Some of these problems (or STSCs) concerning development activities have been collected and described by Coplien et al. [6] including a set of what they call Process Patterns to deal with some of these coordination problems. As the term process patterns is also used in business process management and workflow, we prefer to use the term Socio-Technical Patterns to refer to those patterns involving problems related to both the social and technical aspects of the software process.

We claim that continuous and early detection of STSCs can help project managers in monitoring the software development process and enable the managers to take actions whenever a STSC occurs.

Figure 1 represents the overview of the method behind TESNA. Our motivation is that when implementation and monitoring of patterns is enhanced, empirical validation of patterns will also become feasible. We provide a Method and Tool called TESNA that can improve the system development by regularly monitoring the software development project and alerting in case of a STSC.
The Project Manager (who is implicitly present in Figure 1) decides which STSC to look for in Technical Software and Social Network diagrams that is shown with the help of TESNA. TESNA makes it easy for the Manager to detect STSCs and he/she can then decide if the problem is severe enough to warrant a change in the Planned Software Process and Architecture. For this tool and method to work, we need a data structure for storing the Technical dependencies as well as the social network connections. We accomplish this by using what are known as Dependency Structure Matrices.

3. Dependency Structure Matrix Overview

Since the concept of the Design Structure Matrix was first proposed by Steward [8, 9], Dependency Structure Matrices have been used in engineering literature to represent the dependency between tasks. A DSM highlights the inherent structure of a design by examining the dependencies between its component elements in a square matrix [8, 10].

Morelli and Eppinger (1995) describe a way comparing the predicted and actual communication in an organization [11]. Sosa Eppinger et.al.(2002)[12] describe factors that influence the frequency of communication and choice of media in a geographically distributed development organization. In a different study Eppinger and Rowles (2003) [13] compare the DSM formed through the interaction of system components with the DSM of the technical interactions among team members. Sosa, Eppinger and Rowles (2004) [14], highlight the factors that impact the misalignment of the product and the organizational structures.

Figure 1 shows an example of a simple DSM. The letters A-E, on both axis of the matrix, represent tasks. An ‘x’ in location (a,b) of the matrix means that the task of row a depends on the task in column b. Dependencies below the gray diagonal represent ‘feed forward information’, while tasks above the diagonal represent feedback, for example, task E gives feedback on task C. In this example, tasks A and B depend on each other.

![Figure 1. Socio-Technical Structure Clashes and the planned Software Process](image1.png)

### Figure 2. Example of a DSM

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
MacCormack et.al. (2006) [15] compare the DSMs of a commercial and a pure open source project and show how the structure of the code in the projects reflects the organizational structure that created it, much like what Conway said in his paper [5]. More recently Li et al. [16] use dependency matrices to analyze dependencies between components in a CBS. While Cataldo et.al. [17] show how DSMs can be used to predict coordination in a software development organization and then they compare the predicted coordination DSM with the actual communication DSM.

4. TESNA Tool Functionality

The tool TESNA consists of three modules namely the Social Structure, Technical Structure and the Socio-Technical Structure analysis modules. The tool uses the DSM data structure to analyse each of the different structures and

4.1 Social Structure Analysis

To analyse the Social Structures TESNA can construct and analyse metrics from logs of chat messages (from a chat server like Jabber). Moreover, TESNA displays the different metrics of the social network over a period of time. We have used this option to detect the Betweenness centrality match pattern [7] by calculating the betweenness centrality of the social networks over the period under study.

TESNA can mine bug tracking websites (like Mantis) to gather data on the social thread of responses for each bug posted. We have used this feature on a corporate case (eMaxx discussed below) that we are currently studying.

In order to display the social network got through mining these repositories TESNA uses libraries from the Java Universal Network/Graph Framework (JUNG) [18], an open source library widely used by Network researchers. The display of the social network from the eMaxx case is shown in Fig 3. Here, each of the nodes represents a member of the social network (whose name is indicated by the label next to the node) and the thickness of the line between the nodes represents the number of messages exchanged between the people represented by the nodes. The more the number of messages are the thicker the line gets.

4.2 Technical Structure Analysis

To analyse the Technical Structures TESNA can read the source code file and construct the call graph of the software. At present, the tool supports reading java code files (jar files) to determine the technical dependencies between the different components or modules of the software. TESNA again uses JUNG [18] to display the call graph of the software as shown in Figure 4.

Figure 4 represents the Call Graph or the dependency graph of an open source project called Jython. Each red node represents one java class object of Jython. As this Call Graph is already quite complex, we don’t display the names of the class objects and instead use the tool tip if the user hovers above interesting areas of the Call Graph. We will show later how we reduce this complexity further by clustering the Call Graph.
4.3 Socio-Technical Structure Analysis

TESNA can mine version control systems like CVS and SVN and find out the Socio-Technical Dependencies (the people working on the different parts of the software). TESNA uses JUNG to display the developer code network as shown in Figure 5.

Figure 5: The developer code Socio-Technical Network of Jython

The red nodes in Figure 5 represent the software class objects that the developers, represented by the blue nodes have last modified. The names of the developers are displayed by the labels next the nodes. This graphic representation uses the normal bipartite graph functionality of JUNG. So, the links between the class objects are not displayed. Such a complex graph can provide us with limited information, for example, which developer modified how many files. Using the tool tips of the red class objects one can find out the names of the class objects and in-turn find out which developer modified which class object. As we will show later TESNA reduces the complexity of this graph by clustering the class objects and then displaying the developers working on the different clusters. We further combine the network of dependencies of the class objects with the network of the developers working on the different class objects (as described in the Chapter on the DSM approach). We thus come up with the graph of the developer dependencies. Figure 6 shows the developer dependencies of the Jython project. The red nodes represent the developers working on the different modules of Jython and the directed links represent the dependency, for example bedronis is dependent on bckfnn and vice versa.

Figure 6: The developer dependency graph of Jython

TESNA displays the people dependencies (Figure 6) based on whether people are working on the same or dependent modules [17] and can compare this with the social network of the developers in order to detect Conway’s Law STSC [7].

Figure 7: The Clustered Socio-Technical Network of Jython

5. The TESNA Visualization

Large graphs can cause problems of usability and discern-ability. Though, large graphs can give an indication of the overall structure or that of some location within it, in general the display of large graphs makes them difficult to comprehend. It follows that it is easier to comprehend and perform a detailed analysis of graph structures when the size of the graph is small [19]. In response to the need to make the graphs especially Figures 4 and 5 more understandable we cluster the graphs. For clustering
we use the algorithm by Fernandez[20] and used by MacCormack et al. [15]. We cluster the graphs into a fixed number of 9 clusters according to the golden 7 plus or minus 2 rule for human comprehension [21]. The clustering is done with the help of a DSM Clustering tool that we have developed [22]. Figure 7 is a representation of the Jython call graph clustered into 9 clusters along with the developers who have modified classes in different clusters. We also show the dependencies that exist among the clusters. Such a clustered display can enable a more easy detection of different STSCs [23].

6. Discussion

The visualizations created by TESNA help the Project Manager in identifying STSCs. Once the STSCs are identified the Project Manager can decide whether the current development process needs to be changed. We have tested this methodology in multiple case studies. Among the case studies, we have conducted two case studies in a corporate environment namely Mendix [7] and eMaxx (forthcoming publication). We have also conducted multiple case studies studying Open Source STSCs [23] and have got a few interesting insights into how STSCs in the open source environment differs from Corporate environment.

7. Other TOOLS

There are a few tools available to display the social network as well as the social call graph. Augur is a visualization tool that supports distributed software development process by creating visual representations of both the software artefacts and the software development activities [24]. de Souza et al. [25] are developing a tool called Ariadne that checks dependency relationships between software call graphs and developers. Also there is a tool under development for forecasting dependencies between developers in a Dynamic/iterative environment [17]. The limitation of these tools is that they check for only one particular STSC, namely the Conway’s Law STSC.

REFERENCES


