Using Software Repositories to Investigate Socio-technical Congruence in Development Projects

Giuseppe Valetto, Mary Helander, Kate Ehrlich, Sunita Chulani, Mark Wegman, Clay Williams
IBM T.J. Watson Research Center
{gvaletto, helandm, katee, sunita_chulani, wegman, clayw}@us.ibm.com

Abstract

We propose a quantitative measure of socio-technical congruence as an indicator of the performance of an organization in carrying out a software development project. We show how the information necessary to implement that measure can be mined from commonly used software repositories, and we describe how socio-technical congruence can be computed based on that information.

1. Introduction

One of the reasons why software development is inherently complex is because it is a socio-technical endeavor. We borrow the term socio-technical from the organizational literature, to indicate an inextricable connection between social and personal behavior and technical work. Any non-trivial software development occurs within an intensively collaborative process, in which technical prowess must go hand in hand with the efficient coordination and management of a large number of inter-personal interactions across the development organization. Furthermore, those social and technical dimensions are not orthogonal. It has been recognized that the structure of a software product and the layout of the development organization working on that product correlate (Conway’s law [1]); Parnas [2] has also observed that the subdivision of development responsibility tends to induce the modularization of a software product at least as strongly as its functional decomposition.

Therefore, an important step to improve productivity and quality is to be able to measure and understand how people are organized and interact with one another when they develop software. Given the mutual influence of the social and technical dimensions of software development, though, those social studies must be contextualized with respect to the technical work being done.

One way to investigate those issues is to measure if there is a good fit (or congruence) between the coordination structure mandated by the technical work units (tasks) on the one hand, and the actual social organization (as expressed for example by the communication paths observed among its members) on the other hand. Some studies, such as [3] suggest that high degrees of such socio-technical congruence are beneficial for software development performance.

Whereas those works assume the observability of inter-personal interactions within a software development organization, and – more importantly - a task-centered view of the development process, the tools most commonly employed in today’s practice are artifact-centered. Therefore, we introduce a different measure of socio-technical congruence, which compares the structure of the software organization directly to that of the software product. Our choice can be intuitively motivated as follows: development of software artifacts that somehow depend on each other is likely to require coordination between the people responsible for those artifacts (their stakeholders). Conversely, lack of coordination across stakeholders of dependent artifacts may be a telltale of problems in the development process. That intuition is also backed by quantitative studies, such as [4], which have found correlation between artifact dependencies and communication frequency among stakeholders involved with those artifacts.

Our measure of socio-technical congruence is derivable from data kept in software repositories commonly supporting the development process. This paper discusses how it can be defined and quantitatively measured on the basis of that data.

2. Socio-technical software networks

Researchers have recently turned their attention to the issue of social networking in, since it enables the study of the actual organizational structure of software
projects (for an overview, see Xu et al. [10]). Social networks are at times, combined with views of the software product [7] [8], to form socio-technical software networks, upon which various analyses, can be carried out. To compute congruence, we use such a network, which combines three types of information (see Figure 1):

- communication/collaboration interactions between stakeholders which are mapped as directed arcs between nodes laying on plane P, for People;
- inter-relationships between artifacts, which are mapped as directed arcs between nodes laying on plane S, for Software;
- “work relationships”, which are mapped as directed arcs that cut across from plane P to plane S, and account for the work done by stakeholders on artifacts during the course of the project.

![Figure 1: A socio-technical software network.](image)

It is possible to construct a network like the one in Figure 1 by using the data maintained in various software repositories. Several works, such as CVSAnalY [6] or Augur [9], have shown how the nodes in plan P and S, plus the arcs connecting those plans can be readily mined from arguably the most commonly used software repositories, that is, source control and management systems, such as ClearCase, CVS, Subversion, and others. The metadata associated to each commit record, such as the ID for the stakeholder committing the change, the modified file(s), the timestamp, etc. can be employed to that end.

The arcs in plan S can also be derived from the source management system, by extracting and post-processing coherent software configurations, as shown for example by Kenyon [11]. By running static analysis upon the source code files, one can draw the dependencies between the corresponding compilation units (such as function invocation, field access, inheritance, containment, etc.).

To represent the arcs in plane P, which signify collaboration between stakeholders, we propose to use other repositories, such as developers’ mailing lists and bulletin boards, which can provide reliable traces of collaboration. With an appropriate tool, like that in [5], communications could even be monitored and integrated into the network as they happen. In case other data sources are not available, an approximation can still be extracted from source management systems. A basic approach is drawing collaboration arcs between all stakeholders that have ever worked on the same artifact(s). For better accuracy, one can use a time window: that is, draw an arc only if stakeholders have worked on some common artifact within that window.

We also propose to enrich arcs with attributes, such as weight (e.g. the number of commits to the same artifact by the same stakeholder, or the number of dependencies of a source file onto another), confidence (specifically for person-to-person arcs, representing the likelihood that two persons have in fact collaborated¹), size of contribution, timestamp, etc. With that data at our disposal, we can obtain a rich and reliable representation of the network. On its basis we can compute a measure of socio-technical congruence with the technique detailed below in Section 3.

### 3. Measuring socio-technical congruence

In this section, we mathematically formalize socio-technical congruence based on the topology of the network described in Section 2.

Let \( G_P = (P, A_P) \) denote the digraph of people and their relationships, where the node set is \( P \) and the arc set is \( A_P \). Arcs in the arc set \( A_P \) correspond to relationships between directed pairs of developers in set \( P \). For example, \((i,j) \in A_P \) may mean that person \( i \) has communicated with person \( j \), where \( i,j \in P \). Note that in our model, we use a directed edge (arc) to reflect a past history of communication. However, in interpreting our graphs, we assume that those communication relationships between people are always reciprocal.

Let \( G_S = (S, A_S) \) denote the digraph representing software artifacts as nodes (set \( S \)) and their relationships (denoted by arc set \( A_S \)), and let \( J \) denote a set of “joins” – i.e., arcs connecting developers in \( P \) to artifacts in \( S \). In interpreting set \( A_S \), we assume arc direction implies dependency, for example invocation

---

¹ The value for the time window can be pre-set or can be computed separately for each single project, for instance based on frequency and distribution of multiple-users commits.

² The likelihood of collaboration by two stakeholders on a given artifact could for example “decay” by considering the time elapsed between commits on the artifact by the stakeholders.
of one artifact by another. Similarly, directions implied by arcs in \( J \) imply an affiliation or involvement. For example, in Figure 1, \((7,e)\) could indicate that person 7 has an assigned role, such as owner, to artifact \( e \), and \((10,k)\) could indicate that person 10 has a history of committing changes to artifact \( k \).

To measure congruence, we first consider a proportion of relationships between software artifacts (e.g., arcs in \( A_p \)) that are mirrored in \( A_p \), where detection of mirroring is facilitated by the join set \( J \). That is, if \((i,l)\) is in \( A_p \) and \((k,i)\) and \((h,l)\) are in \( J \), then the arc \((i,l)\) is “arc mirrored” if these two conditions are true: (a) \( k \) and \( h \) are distinct nodes in \( P \), and (b) either arcs \((k,h)\) or \((h,k)\) (or both) are in \( A_p \). The number of arc actually mirrored divided by the number of times arcs in \( A_p \) could be mirrored in \( A_p \) provides a proportion that reflects one aspect of congruence, as is illustrated in Figure 2a. Figure 2b illustrates a second aspect of congruence we refer to as a “node tie”, that is, when two distinct developers (here, \( h \) and \( k \)) both have some relationship to the same artifact (i.e. node \( i \in S \)), then we expect to see a relationship between \( h \) and \( k \).

Socio-technical congruence based on arc mirroring between graphs \( G_p=(P,A_p) \) and \( G_i=(S,A_j) \) with join set \( J \) is given by:

\[
C(G_p,G_i,J) = \kappa / \gamma
\]

defined when \( \gamma > 0 \), where \( \kappa \) and \( \gamma \) are computed by the following algorithm:

**step 1:** set \( \kappa, \gamma = 0 \)

**step 2:** For each arc \((i,l)\) in \( A_p \):

**step 2a:** Let \( J_i = \) all arcs in \( J \) incident on node \( i \):
- let \( J_j = \) all arcs in \( J \) incident on node \( j \).

**step 2b:** For each \((a,i)\) in \( J_i \) and \((b,l)\) in \( J_j \):
- If \( a \neq b \), then \( \gamma = \gamma + 1 \).
- If either \((a,b) \in A_p \) or \((b,a) \in A_p \), then \( \kappa = \kappa + 1 \).

A similar computation finds socio-technical congruence based on node ties. As is the convention for measurements of congruence (see [12]), the quantification results in a value between zero and one, where a value closer to one indicates high congruence and a value closer to zero indicates low congruence.

As an example, congruence measured based on arc mirroring for Figure 1 is 0.2. There are six mirrorings in \( A_p \); \((b,d)\) by \((1,4)\) and \((4,1)\); \((i,h)\) by \((5,6)\), \((i,h)\) by \((6,9)\); \((k,l)\) by \((11,10)\), \((k,l)\) by \((10,12)\) and \((12,10)\); and \((n,o)\) by \((4,5)\). There are 24 cases where a mirroring is missing – for example, for arc \((b,e)\), persons 1 and 7 are related to artifacts \( b \) and \( e \) respectively, but there is no arc between 1 and 7 in \( P \). Similarly for arc \((b,e)\), and persons 2 and 7. The congruence measurement based on node ties is instead 0.571429. That is, software artifacts \( b \), \( i \) and \( n \) each have developers who are not related to each other; and artifacts \( h \), \( l \) and \( n \) each have at least one pair of developers who are related.

**Figure 2:** Congruence measurement concepts.

Note that both definitions of socio-technical congruence allow for global (i.e., over the whole network), as well as local measurement (i.e., over a region of interest, by restricting consideration to any subset of \( A_p \) or \( S \)). Note also that the two congruence measurements may be considered separately or combined, e.g. as a weighted combination.

Several natural extensions are also possible, for example by considering additional information through node and arc attributes, such as weights or confidence levels. The measurements are also easily adaptable to consider super nodes (i.e. aggregated nodes that represent a number of nodes), to enable working at different levels of granularity in the software project, e.g. compilation unit, module, subsystem, and the organization, e.g. individual, pair, team, department. Such adaptability is critical for scaling the analysis to consider very large projects.

4. Applications and Future Work

Our concept of socio-technical congruence describes the degree of alignment between social relationships and software relationships. Therefore a global congruence measurement provides a quick index of how well the organization is actually aligned with the planned sub-division of responsibility in the project. These measures can be made available to project leads, so that they can better govern the software development process and organization.

Local values of congruence can also provide more insight: for example, they can be used to detect certain process tasks or system areas in which the collaborative effort is likely to be struggling, and
implement remedies, such as, modify responsibilities or the organization layout to facilitate communication flow in the affected areas.

Examining the *evolution* of the congruence value over time is also useful, for example for auditing purposes.

In order to validate the significance of socio-technical congruence, it is however necessary to correlate it to other software characteristics, and see whether it can be used as a predictor of those properties. We are currently working with quality metrics, such as defect density and frequency of modification requests (MRs), which can also be mined from commonly used software repositories, such as ClearQuest, Bugzilla and other MR tracking databases. If socio-technical congruence is an indicator of how well inter-dependent development tasks are performed, we expect to see, for example, more defects where congruence is low, in particular defects related to improper interfacing between software components.

Investigation on socio-technical congruence can progress in a number of other directions. In this initial proposal we have focused only on source code artifacts, mainly because the mining of management systems is well understood and, as such, can provide high-quality, fine-grained information for our purposes. That, however, limits the scope of congruence as a metric to the implementation phase of a software project. However, other kinds of relationships (such as traceability, i.e. how artifact derive from – as opposed to depend on - others) could in principle be used to expand socio-technical networks to span the whole software life cycle. That would enable using socio-technical congruence as an earlier indicator of performance. One outstanding difficulty is the scarcity of tools in the current practice that track and record this kind of relationships, and, as a consequence, the difficulty in mining the necessary information reliably at all phases of the development process.

Finally, we are also interested in understanding the best way to incorporate socio-technical networks and congruence into development and management tools. This will empower concerned stakeholders to make more informed project decisions.

5. References


