Balancing the Value and Risk of Socio-Technical Congruence

Giuseppe Valetto
Drexel University
3141 Chestnut Street
Philadelphia, PA 19104, USA
+1 215 895-2669
valetto@cs.drexel.edu

Sunita Chulani
IBM Research
650 Harry Road
San Jose, CA, 95148 USA
+1 408 927-1767
Sunita_Chulani@us.ibm.com

Clay Williams
IBM Research
19 Skyline Drive
Hawthorne, NY, 10532 USA
+1 914 784-7457
clayw@us.ibm.com

ABSTRACT
If socio-technical congruence can bring about benefits - such as better performance - to a software development project, then actions that improve the level of congruence have a potential to increase the overall project value. However, those actions may also incur a cost, and bring additional risks into the project. We propose a number of alternative actions for increasing socio-technical congruence, based on the representation of congruence gaps in the socio-technical graph of the project. We also describe the costs and risks corresponding to those actions, to highlight how they represent options for the governance of the project, each of which impacts a different aspect, ranging from architectural decomposition, to team work practices, to project scheduling.

Keywords
Congruence, estimation, real options, risk and value, software development organizations, software development governance.

1. INTRODUCTION
Socio-technical congruence is a measure that assesses the “fit” between the structure of a software system and the structure of the organization that develops it. As such, it provides a means for empirical verification of Conway’s law [7], and it describes to which extent it is respected in a given software development effort. Conway’s thesis is that any piece of software reflects the organizational structure that produced it. As a consequence of that thesis, Conway postulated that, if an organization can align itself to closely reflect the software design, it is likely to be more efficient in performing its development tasks. Recent work by Cataldo et al. [6] has shown how to define and measure socio-technical congruence quantitatively for software development projects. Cataldo et al have also conducted a study that shows initial statistical evidence that in software development higher socio-technical congruence is indeed correlated to better performance. In that study, task performance was measured as completion time for change requests.

If congruence could be consistently linked to these or other improvements in software development efficiency (like time-to-market, knowledge sharing, and utilization of human resources), we could say that any decision by a software development organization that improves the congruence measure has a potential to increase the value of the project being undertaken by that organization (e.g. shorten the time, reduce cost of development, improve quality, etc.). However, it is possible that such a decision also has an impact on the project’s cost, or – more generally speaking - some form of risk. Maximizing the value benefits of increased congruence, while at the same time minimizing the inherent risk levels, becomes a balancing act that has significant implications for the governance of the development effort.

Starting from the above observation, this paper examines various alternative decisions that can impact the level of socio-technical congruence in a software project; describes different ways in which those decisions can impact value as well as risk; and proposes ways to estimate their relative benefits for the project.

2. GAPS AND CONGRUENCE
Valetto et al. [12] have extended the work by Cataldo et al. on socio-technical congruence. They have devised a graph theoretic algorithm that measures congruence in a manner that is mathematically equivalent, but also allows to examine the graph describing the socio-technical system [10] made up by the organization and the software product. Such a graph comprises of dependencies among the modular elements in the software product (a software graph); the personal interactions among members of the development team (a social network); and the work actions carried out by the developers on the software, which connect the above mentioned software and social components of the overall graph. The graph is analyzed by an algorithm, to rank the relationships in the social network of developers that provide the most important contributions to the overall measure of congruence. Conversely, it can also identify and rank gaps, i.e. communication / coordination relationships in the organization that would most increase the measure of congruence, but are missing. The purpose of this kind of graph analysis is to highlight the critical coordination paths within the development organization.

Figure 1: a gap in the socio-technical graph.
In Figure 1, we show an example of a congruence gap. In the bottom part of that figure – the software plane $G_s$ - two software artifacts $(S_a, S_b)$ are depicted, with a dependency relationship from $S_a$ to $S_b$. In the top part – the organizational plane $G_p$ - two organizational entities $(P_i, P_j)$ are depicted. The arcs connecting the two planes indicate development tasks, i.e., work that is being done by $P_i$ on $S_a$, and by $P_j$ on $S_b$, respectively. Notice that we do not make here any assumptions on the granularity of the organization and software elements. Figure 1 could show two individual developers working on single compilation units, or two departments in the same organization working on two subsystems, or two federated software development organizations working on different but connected software systems. The absence of an arc connecting $P_i$ and $P_j$ in Figure 1 represents a congruence gap: it denotes that there is no communication going on between $P_i$ and $P_j$, even as they carry out tasks on dependent software artifacts, and hence may need to coordinate their work. If an arc were instead present, that would signify that a communication channel of some sort is established between $P_i$ and $P_j$, which can potentially be used for their coordination.\(^1\)

Eliminating the gap in Figure 1 would increase the congruence of the socio-technical graph. Given that the higher the ranking of the gap, the higher its contribution to congruence, and assuming, in accord with the discussion in Section 1, that higher congruence has a beneficial effect on the project, we can postulate that the elimination of the highest ranked gaps can bring substantial benefit to the value of the project. However, as we will see in Section 3, gap elimination also has a cost and can bring some risk into the project. It is therefore important to examine the options for and the trade-offs of gap elimination.

3. RISK AND VALUE OF ELIMINATING GAPS

We discuss below the main options that can be exercised in addressing a coordination gap to improve socio-technical congruence. Those options map to different perspectives on the governance of a software development effort.

3.1 Closing the gap: augmenting coordination

Referring to the example in Figure 1, one obvious way to eliminate a gap is to close it, that is, make sure that communication and coordination flows between $P_i$ and $P_j$ about their inter-dependent tasks. Depending on the nature of $P_i$ and $P_j$, that can be achieved in many different ways. If $P_i$ and $P_j$ are individual co-workers, who are co-located, it may be sufficient to advise them that they are required to work together. If they are not co-located, it may be necessary to introduce $P_i$ and $P_j$ to each other and put in place mechanisms and tools that enable and encourage long-distance collaboration (that may include for instance organizing virtual or in-person face-to-face meetings to kick start their collaboration). If $P_i$ and $P_j$ are instead separate sub-teams, it may be necessary to appoint an element of each team to act as a broker, that is, a conduit and a facilitator for communications across the two sub-teams. A variety of other ways exist to augment coordination by enabling direct communication, which can effectively close the gap: they depend on the specific situation, and the nature of the project, process and organization.

Also, as discussed for instance by Ehrlich et al \cite{9}, there are ways to close the gap that do not require direct interaction between $P_i$ and $P_j$. That may involve, for example, another organizational entity that can play a brokerage role between $P_i$ and $P_j$. Notice that all of those solutions still cause communication and coordination overheads – although possibly less visible than in the case of direct communication - which impact all organizational entities involved.

All communication-based remedies to the gap come therefore with a price tag. First of all, as discussed by Brooks \cite{5}, augmenting communication and coordination among team members is an overhead that increases costs, as it tends to decrease the individual productivity of the personnel involved. On top of that basic cost, there are other factors to consider, due for example to the logistic support that may be necessary to establish and maintain the communication channel between $P_i$ and $P_j$ as long as necessary. Therefore, eliminating a congruence gap by closing it may produce a net benefit in terms of project value only when the valued gained due to increased congruence is likely to offset all the additional costs sustained because of that communication channel. The ranking of gaps helps the management of the project focus on closing those gaps that bring the highest congruence improvement to the socio-technical structure, while managing the related risks.

To reinforce the idea that communication does indeed provide value, we notice how, in many existing cost estimation models, the effort (or cost) of development is a function of several parameters including organizational communication. For example, COCOMO II \cite{4}, which is a widely used parametric estimation model, has a parameter called “Team Cohesion” that accounts for the type of interactions between team members, i.e. seamless interactions vs. cooperative interactions vs. very difficult interactions. Teams with seamless interactions have higher productivity, i.e. lower costs, compared to teams with very difficult interactions, i.e. effort increases by a power of 5.

3.2 Eliminating the gap by refactoring the software

Closing the gap is not the only option available. The same increase in congruence could occur by refactoring the software in a way so as to make the dependency between $S_a$ and $S_b$ disappear.

\(^1\) Notice that - although we omit that level of detail from the rest of this discussion for the sake of simplicity – the presence of an arc in the graph is better described as a stochastic variable, instead of a boolean one, since it indicates a probability that the two nodes $P_i$ and $P_j$ are connected.
dependency between I_b and S_b, and a dependency from S_a to I_b. However, this solution decouples the development tasks undertaken by P_i and P_j, and – as such - increases congruence.

This refactoring-centered option is a classic way in which the interplay between process and product (tasks and software artifacts) is resolved in software development projects. Parnas [11] has eloquently described it when he observed that modularization is used to divide responsibilities within the development team as much as to conquer system complexity. Work on the formalization of modularization by means of Design Structure Matrices and Design Rules by Baldwin and Clark [3], and recent steps towards the extension to the modeling of the organizational level by Avritzer et al. [2] discuss techniques for and effects of this option.

Software estimation techniques, in particular those tuned on change estimation, can help in evaluating the cost of this refactoring option. For instance, in COCOMO II, the reuse model can be used to determine the cost improvement for refactoring, by analyzing the percentage of change in the design, coding and integration of the system. Notice that this cost is assumed by the development organization as a one-time cost, with minimal or no ongoing maintenance (in the case of effective based solution discussed in Section 3.1 where the cost for increased communication is an ongoing cost, which must be sustained until at least one of the two inter-dependent tasks is completed. Besides development costs, other risk factors may exist: for example de Souza et al. [8] have noticed how the introduction of APIs to partition work and responsibilities can at times hinder some kinds of useful collaboration paths in the development organization.

### 3.3 Eliminating the gap by re-arranging the project schedule

Another way that can be used to eliminate, or at least alleviate, the effects of a gap in the socio-technical graph is to re-arrange the tasks represented by the pairs P_i/S_a and P_j/S_b. In case those two tasks occur simultaneously, coordination between P_i and P_j is likely to become a necessity. If those two tasks could be scheduled sequentially, instead, an interaction between the organizational entities in charge of the tasks may become unnecessary. For instance, in our example - given the direction of the software dependency between S_a and S_b – task P_i/S_a could precede task P_j/S_b. Such a re-scheduling scenario is represented in Figure 3, in which the dotted arc between P_i and S_b signifies the postponement of the corresponding development task, which removes the gap, and hence increases congruence.

Notice that, in some cases, interaction between P_i and P_j may still need to occur later on, once the P_i/S_a task is picked back up, but might then be limited to one-shot or sporadic consultation of P_j by P_i. Again, contrast that with the ongoing coordination required for the duration of the inter-dependent tasks, in case they are taken up in parallel.

This solution is also quite commonly seen in software development projects. For example, it is often spontaneously chosen by developers, whenever they realize they are working on the same file (hence their tasks will require merging of potentially conflicting changes at some later point), or on tightly coupled modules, (hence their tasks require dealing with “a moving target” in terms of the nature of the dependencies between those modules). At times, the simplest solution for one of those developers, if time and schedule permit, is to abandon that particular task and work on something unrelated, until that conflict is resolved.

Notice how another equivalent way to eliminate the gap by acting on task arcs would be by changing task assignments, as opposed to schedule. In our example, assigning both tasks to the same organizational entity (e.g. P_i) would also eliminate the gap, possibly even more effectively than rescheduling the P_j/S_b task, since it would eliminate the possibility that a coordination need arise down the road.

![Figure 3: Re-scheduling of a development task.](image)

In general, to change the assignment and scheduling of tasks is a typical project management decision. In a case like the one discussed, the benefit to the project brought about by the elimination of the gap through this kind of decision is evident, in that it reduces the overhead of inter-task coordination, as well as inter-personal communication. However, the risks associated with this option are also pretty evident: re-arranging tasks from parallel to sequential has potential ripple effects, which can impact the overall time line of the project. In fact, scheduling of development tasks is one of the most delicate project management activities, for that very reason. In our simple example, those ripple effects are not visible, as we focus on a local view of the congruence gap, and choose not to consider the neighbor nodes and arcs in the socio-technical graph.

Similarly, also the resolution of a gap by means of task re-assignment has associated risks, mainly related to overloading of the organizational units that take up those additional duties. That accumulation and centralization of work can also have a ripple effect on the socio-technical graph. In our example, in case P_i takes up the development of S_a, in the place of P_j, then P_j may get involved in additional task conflicts, which can occur with respect to all currently running tasks that involve artifacts that are dependent on S_a.

### 3.4 Evaluating options

Sections 3.1 to 3.3 propose several techniques for resolving gaps and increasing socio-technical congruence. They analyze the specific costs and risks of each proposed solution, which impact diverse aspects of software project governance.

In case a software development project has among its goals the optimization of socio-technical congruence (because, for example, it is known to be correlated with some value factor of strategic importance to the development organization), the techniques we have described, become alternatives that can be taken to reach that goal. Therefore, methods like real options analysis [1] could be employed to objectively balance value against risks and embrace certain decisions, rather than others.
A real option is the right, but not the obligation, to undertake a
decision to achieve some business goal. Just as investors in
financial markets are willing to pay a price to buy call and put
options to reduce the risk or increase the value of their portfolios,
decision-makers in a software project may be willing to pay a
price in terms of costs and schedule to augment the value of the
project through socio-technical congruence. The options in this
case are the potential ways to eliminate gaps, at least the most
significant, highly-ranked ones. Once the options are available –
and quantifiable – the project decision-makers can choose which
option (or mix thereof) provides the best balance for the project,
given its business goals.

4. CONCLUSIONS AND FUTURE WORK
Understanding and being able to evaluate the cost and other risks
and side effects of the options that a project has to improve its
socio-technical congruence can help an organization recognize the
most effective trade-offs for reaching some of its value goals.

We have presented a qualitative discussion of the options that a
development organization has in trying to increase socio-technical
congruence. Those options highlight how Conway’s Law impacts
multiple critical areas of the software development practice,
ranging from modular design, to team work, to project
management. As a consequence, the analysis of risk/benefit trade-
offs, when it comes to congruence, is bound to involve at the same
time multiple software engineering and governance perspectives,
from software and change estimation, to team coordination, to
process planning and execution.

We have briefly hinted at how arcs and gaps should be considered
in stochastic terms. The values and costs of the options discussed
in Section 3 may also be represented probabilistically, with a
nominal value and a variance range. Variance represents the
magnitude of the uncertainty associated to each stochastic
variable: it can be considered itself akin to risk, and must be taken
properly in account. For example, arcs in the socio-technical
graph with higher variance are riskier than those with lower
variance and need to be treated differently. In this respect, one
question that must be addressed through experimental activities is
what level of probability is sufficient to determine a gap with
sufficient confidence. Similarly, when the organization estimates
the cost of the various gap-elimination options, it needs to
appropriately consider the variance associated with each
estimation, and treat it as an inherent risk for that option.

This preliminary, qualitative analysis of the options and trade-offs
that are faced by decision-makers during a software project, when
dealing with socio-technical congruence, opens up the possibility
of formal as well as experimental scrutiny. A framework for
the formalization of the options could be set up by using estimation
techniques, to associate formulas for cost and risk to each option,
coupled with the evaluation of gap elimination benefits.
Simulations can also take place based on the choice of a given
option across the project, or a combination of options for the
highest-ranked gaps in the project.

We are also considering some scientific experiments to observe
how the various options discussed in this paper have an impact of
socio-technical congruence. Such experimental work can be
carried out by observing how decisions that have an effect on
socio-technical congruence - since they deal with coordination

5. ACKNOWLEDGMENTS
Our heartfelt thanks to Kate Ehrlich and Mary Helander at IBM
Watson research center, for all the work and discussions on gaps
in socio-technical networks. We would also like to thank Patrick
Wagstrom, with Carnegie Mellon University, and Andre van der
Hoek, with University of California at Irvine, for early discussions
on gap remediation in software projects.

6. REFERENCES
[1] Antikarov, V., and Copeland, T. Real Options: A
Implications of Software Architecture in a Global Software
Development Project. In Proc. 7th Working IEEE/IFIP
Conference on Software Architecture (WICSA 2008)
Engineering, 20th Anniversary Edition. Reading, MA:
Addison-Wesley, 1995.
K.M. Identification of coordination requirements: Implications for the design of collaboration and awareness
[7] Conway, M.E. "How Do Committees invent?" Datamation,
[8] de Souza, C. R., Redmiles, D., Cheng, L., Millen, D., and
Patterson, J., Sometimes you need to see through walls: a
field study of application programming interfaces. In Proc.
[9] Ehrlich, K., Helander, M.E., Valetto, G., Davies, S., and
Williams, C., An Analysis of Congruence Gaps and Their
Effect on Distributed Software Development, submitted to
the ICSE Workshop on Socio-Technical Congruence (STC
[11] Parnas, D.L. On the criteria to be used in decomposing
systems into modules. Communications of the ACM, 15, 12
(1972), 1053-1058.
[12] Valetto, G., Helander, M.E., Ehrlich, K., Chulani, S.,
Wegman, M.N., and Williams, C. Using Software
Repositories to Investigate Socio-technical Congruence in

REFERENCES
ACKNOWLEDGMENTS
5. ACKNOWLEDGMENTS
Our heartfelt thanks to Kate Ehrlich and Mary Helander at IBM
Watson research center, for all the work and discussions on gaps
in socio-technical networks. We would also like to thank Patrick
Wagstrom, with Carnegie Mellon University, and Andre van der
Hoek, with University of California at Irvine, for early discussions
on gap remediation in software projects.

6. REFERENCES
[1] Antikarov, V., and Copeland, T. Real Options: A
Implications of Software Architecture in a Global Software
Development Project. In Proc. 7th Working IEEE/IFIP
Conference on Software Architecture (WICSA 2008)
Engineering, 20th Anniversary Edition. Reading, MA:
Addison-Wesley, 1995.
K.M. Identification of coordination requirements: Implications for the design of collaboration and awareness
[7] Conway, M.E. "How Do Committees invent?" Datamation,
[8] de Souza, C. R., Redmiles, D., Cheng, L., Millen, D., and
Patterson, J., Sometimes you need to see through walls: a
field study of application programming interfaces. In Proc.
[9] Ehrlich, K., Helander, M.E., Valetto, G., Davies, S., and
Williams, C., An Analysis of Congruence Gaps and Their
Effect on Distributed Software Development, submitted to
the ICSE Workshop on Socio-Technical Congruence (STC
[11] Parnas, D.L. On the criteria to be used in decomposing
systems into modules. Communications of the ACM, 15, 12
(1972), 1053-1058.
[12] Valetto, G., Helander, M.E., Ehrlich, K., Chulani, S.,
Wegman, M.N., and Williams, C. Using Software
Repositories to Investigate Socio-technical Congruence in