# Challenges in Measuring, Understanding, and Achieving Social-Technical Congruence

**Anita Sarma, Jim Herbsleb, and André van der Hoek***

April 03, 2008
CMU-ISR-08-106

Institute for Software Research
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

*André van der Hoek is a Professor at University of California, Irvine, CA.

## Abstract

Congruence, the state in which a software development organization harbors sufficient coordination capabilities to meet the coordination demands of the technical products under development, is increasingly recognized as critically important to the performance of an organization. To date, it has been shown that a variety of states of *in*congruence may exist in an organization, with possibly serious negative effects on product quality, development progress, cost, and so on. Exactly how to achieve congruence, or knowing what steps to take to achieve congruence, is less understood. In this paper, we introduce a series of key challenges that we believe must be comprehensively addressed in order for congruence research to result in well-understood approaches, tactics, and tools – so these can be infused in the day-to-day practices of development organizations to improve their coordination capabilities with better aligned social and technical structures.

This effort is partially funded by the National Science Foundation under grant number IIS-0534775, IIS-0329090, and the Software Industry Center and its sponsors, particularly the Alfred P. Sloan Foundation. Effort also supported by a 2007 Jazz Faculty Grant. The views and conclusions are those of the authors and do not reflect the opinions of any sponsoring organizations/agencies.

**Keywords:** Software development, Congruence, Socio-technical Congruence

# 1 INTRODUCTION

Any software development project has its mismatches between the actual actions people take to coordinate their efforts and the optimal actions they should be undertaking to have a maximally effective collaborative experience. While these mismatches stem from choices that individuals make, these choices are to a large extent constrained and influenced by the context in which the individuals make these choices. The nature of this context is both social (e.g., organizational processes, team structures, conventions) as well as technical (e.g., artifacts that are produced, tools used for their production, and interdependencies among artifacts). A key question that arises, then, is how an organization can best align their social and technical structures so that "optimal coordination actions" can take place, in other words, how can the organization achieve a congruent state in which individuals make the right coordination choices that lead to an overall effective and efficient coordination effort?
This paper addresses this question, particularly raising considerations regarding the key research and practical challenges that arise when studying and working with the concept of congruence. The paper deliberately does not attempt to provide answers, in order to provide a clean slate of issues that we hope can serve as an unbiased and comprehensive research agenda for the broader field.

We first, in Section 2, consider congruence itself, outlining some properties that notably influence our collective ability to study and work with the concept of congruence. The next three sections raise and discuss issues and challenges that arise when attempting to *measure*, *understand*, and *achieve* social-technical congruence, respectively. We conclude the paper with a summary of our main points.

# 2 WHAT IS CONGRUENCE?

Congruence refers to a state in which an organization has sufficiently aligned their coordination capabilities to meet the coordination demands of the technical products under its development [1]. In interpreting this definition, a number of properties of congruence stand out.

Congruence represents a state. This state represents a fit between the coordination requirements that are put forth by a given development situation and the actual coordination actions that are ultimately afforded by the current organizational structures. This state captures a particular moment in time, with the actions and needs constrained by the social and technical context at that exact time.

Congruence is descriptive. An organization may have put all the right structures in place, but, in the end, the individuals that actually perform the work cannot be forced to make optimal decisions. For instance, even though tools can be and indeed are being constructed that suggest particular "good" actions to take next, these tools cannot predict all of the consequences of these actions. Even if these steps are taken, they may therefore not lead to the "next optimal state of congruence". Congruence, then, is only descriptive of a certain state in which an organization finds itself at some moment in time.

Congruence is dynamic. With the passing of time, and the actions of individuals during that time, the social and technical structures that define the context of work change. New code dependencies arise in the software, existing dependencies may change, and social structures such as team membership evolve. Each of these, in fact, can change rapidly and suddenly. A

congruent state in which an organization may find itself at one point in time may therefore no longer be congruent at a later time, and vice versa.

Congruence is multi-dimensional. Congruence is not only a matter of which developer(s) should talk to which other developer(s) because of currently existing task dependencies. Certainly, this is an important part of congruence, but numerous additional dimensions exist. Expertise, additional stakeholders, implicit communication through comments left in code or tools, and work patterns, to name a few, all play an equally important role in congruence. There are as many dimensions to congruence as there are ways to coordinate work.

Congruence can be defined at multiple levels. While ultimately it is the choices of individuals and teams that influence the degree of congruence an organization exhibits, the level at which one would like to optimize congruence is fundamentally that of the organization. Nevertheless, the degree to which an organization is congruent is likely to depend on how individuals and teams are congruent. It is therefore necessary to treat congruence at multiple levels, from individuals, subteams, and teams all the way up to the entire organization.

Congruence involves tradeoffs. It is surely possible for an organization to achieve congruence in one dimension or at one particular level, and it is probably useful for it to strive for that. On the other hand, being congruent in one dimension or at one level may by definition lead to incongruence in another dimension or level. For instance, individuals and teams may have to work in patterns that are sub-optimal to them, but contribute to a global optimum at the organizational level. Congruence, thus, has to concurrently address all possible dimensions and the tradeoffs among them.

# 3 MEASURING CONGRUENCE

A necessary step towards addressing congruence involves measuring congruence. What factors must be looked at to begin building an understanding of congruence, or, alternatively put, what information must be collected to reliably be able to answer whether or to what extent an organization is congruent? A derivative question, of course, is how this information can be collected?

Fundamentally, this reduces to understanding what the current set of coordination affordances is and what the current set of coordination needs is. Both sides of this comparison may be quite complex, multiply determined and dynamic in nature, consequently detection and measurement is challenging. This presents research questions for empirical studies of congruence and its effects, as well as pragmatic questions for building tools that make practical use of measurement of, and computations on, congruence. Coordination requirements, for example, may arise equally so from interactions of features at the requirements level, from dependencies among components and their interfaces at the architecture and design levels, and from functional, data, and semantic dependencies at the code level. Coordination capabilities, by the same token, may arise from position in the organization (e.g., same team), geographic location, shared external representations, communication in various forms, shared mental models, and probably many other sources as well. These factors may be moderated by project state as well, including such things as time pressure, team size, communication barriers, and domain knowledge.

To make a more informed and probably more useful assessment of an organization's state of congruence, it generally may be needed to not just look at a single state at a given moment in time, but to also examine historical trends. Trends can shed light on how an organization's

requirements and affordances have been in congruence (or not; or partially), to what degree, and whether its state of congruence has improved or deteriorated over time. Improving an organization's state of congruence is a process, and such a process requires time as well as careful monitoring, measuring, and interpreting of coordination information over time to ensure progress is as desired.

A broad variety of information may be useful:

- *Team structures* (but one should be aware that individuals are generally part of multiple teams, team structures change frequently, etc.);
- *Communication that takes place directly* from person to person or team to team, including instant messages, e-mails, and ideally also any voice conversations – though this may prove to be impossible;
- *Communication that takes place indirectly* through tools, i.e., messages left for one another, bug reports that contain directives, work items, etc.;
- *Processes and any standard work practices that may be prescribed*. These provide a context and mechanism for coordination, but fail to capture and may actually hinder the informal coordination actions that take place;
- *Actual coordination actions that people take with tools*, i.e., checking in and checking out artifacts, opening a bug report, changing its status, etc. Both prescribed processes and actual processes followed must be collected to understand congruence;
- *Tacit knowledge that individuals possess* regarding the current state of the product(s) under development, processes, team structures, and so on. While difficult to assess, the actual understandings that individuals have of the current context critically influence their actions;
- *Technical artifacts,* particularly such aspects as overall code structure, dependencies among code elements of various natures and granularities, authorship, etc.;
- *Locality of people and artifacts*, as well as any movement in locality;
- *Task assignments that are made over time*, together with priorities, task dependencies, deadlines, etc.;
- *Changes to work procedures, patterns, and processes*. Such changes may indicate problems that arose with respect to coordination and product development. They, in turn, of course could also lead to future such problems.

Many other kinds of information may be relevant, depending on the dimensions of congruence in which one is interested. It should also be observed that frequently there exists auxiliary information that is of a more subtle nature yet also is relevant. For instance, if a developer started on a code change, but abandoned it after noting a conflict with other work in progress, this is critical information to capture.

We do not necessarily partition the above in information pertaining to "coordination needs" and information pertaining to "coordination actions". Much information in reality evidences both of these, perhaps even at the same time (e.g., a code modification can represent a means to resolve one coordination need while creating other needs, filing a bug report represents both an explicit coordination action as well as a need to be addressed).

Not all of the information needed to measure congruence is explicitly kept. Measuring congruence, then, requires extraction and computation over existing types of data, as well as

(likely) additional instrumentation of software development practices. Acquiring data for measurement can take several forms:

- *Mining software repositories.* This is a standard approach to: (a) obtain historical data that is explicitly stored, and (b) reconstruct information that is not explicitly kept but may be appropriately derived. Determining who is a likely expert on a certain piece of code is a canonical example of the latter. A major caveat exists with this approach, however. These reconstructions, by virtue of looking at a past of incomplete information, can be difficult to make accurate. One always will have to complement automatic reconstruction with methods to validate whether the interpretations are correct, and remain so over time.
- *Instrumentation*. This approach adds monitors and extra data capture devices to the software development process, for instance to capture more detailed data on how developers interact with their immediate environment. This approach can work well for some kinds of information, but unfortunately is not suited for all. It is unlikely, for instance, that one could ever capture all communication that takes place between developers; a significant portion will always be lost, regardless of how much instrumentation is put in place.
- *Surveys, interviews, observation, etc*. This approach relies on manual, labor-intensive methods to collect more subjective and detailed information such as attitudes, opinions, and rich descriptions. While tedious to perform, this method has the benefit of being able to access information otherwise not available. It is particularly important to understand a broader context than what is evidenced in artifacts and actions alone, and as such provides an important balance with respect to the automated methods. As with the automated methods, though, care must be taken in how data is collected and interpreted.

The points we have made thus far consider obtaining information, but in measuring congruence, a second and equally difficult challenge arises in how to assimilate the raw information that is collected into useful metrics that indicate a degree of congruence for a given situation. This touches upon the area of metrics: how to turn the potentially vast streams of highly varied information into useful measures of the exact properties of congruence one wants to study? The necessity of always having to make tradeoffs in the presence of highly dynamic factors strongly influence this issue. Multiple forms of dependencies must generally be addressed in a single or small set of measures, which is compounded by the fact that coordination in and of itself is difficult to quantify as it is. Carefully defining the level of congruence to be studied and identifying the associated dimensions that play a role will help guide towards potential metrics that may be useful, but the actual articulation of such metrics remains a serious challenge – particularly in light of the dynamic nature of congruence.

Overall, the primary objectives underlying a research agenda into *measuring* congruence can be summarized as follows:

- Knowing what information to collect, particularly in light of the need to understand the broader context.
- Finding the best approach to collecting this information, directly if possible, indirectly if needed.
- Understanding appropriate ways of computing congruence measures from available data.

# 4 UNDERSTANDING CONGRUENCE

Two fundamental tasks are involved in understanding congruence. One is the scientific task of determining the effects of congruence, while the other is understanding how to detect particular states of congruence (or incongruence) and provide appropriate advice, practices, or tool assistance.

*Science of congruence.* Understanding the effects of congruence is a complex task, since there are many ways that coordination needs can be measured, many ways that coordination capabilities can be measured, and many ways congruence can be computed. Finally, there are many different effects of congruence that one might be interested in, beyond the traditional bottom-line impacts on quality, effort, and development time.

In addition to these basic questions about congruence and its effects, there are important questions about tradeoffs. If teams have worked together extensively in the past, can the resultant "shared mental model" permit them to reduce their need for communication, or other coordination activities such as daily builds and code inspections?

There may also be environmental factors – attributes of the organization, the product, or the tools – that will hinder or facilitate coordination. For example, if a design is novel and uncertain, hence likely to change, the coordination problem seems likely to be much more difficult. If there are very stringent performance requirements leading to the need for a highly optimized design and implementation, the coordination needs are again likely to increase.

Finally, we need theories of congruence in order to drive the research and allow us to make sense of the full set of results. Without theory to tie them together, results of particular investigations remain just data points. Theory provides the larger pattern into which they fit, and which explains why the results turned out as they did. This is especially important to separate incidental observations and influences from core "truths". Highly differing and oft unique external factors may dominate general trends and perhaps more actionable factors. As a research community, we must find ways in which to carefully calibrate our studies, so that the results can be placed in their appropriate context for comparison to one another.

*Using congruence to improve project performance.* In order to make use of the findings, there must be ways of recognizing particular states of congruence and taking appropriate action. This is quite a different challenge from measurement for scientific purposes, since there are real-time constraints on when the data and computational results must be available to be useful. Another issue is how to intuitively visualize (in)congruence so that it is useful for the manager or the developer and fluidly helps them in their day-to-day practices. Finally, there is the additional challenge of understanding what particular actions make sense in particular situations. The scientific results describe relations among things that happen in general, but many forces are at work in concrete situations. One must resolve the problem of how to give useful concrete advice in complex, dynamic contexts.

Throughout, a number of factors complicate the work – whether the work is scientific in nature or has the practical aim of improving an organization's ability to effectively coordinate.

First is the issue of side effects. By making progress into achieving congruence in one dimension, there often exist secondary and undesired side effects. An excellent example is provided by Ye, Yamamoto, and Nakakoji, who observe that providing support for finding experts on certain pieces of code may lead to the undesirable effect of the same expert being bogged down with question after question [2]. Their solution addresses this issue by spreading out the cognitive load of answering questions. Another such example would be a policy that preferentially constructs teams from individuals that have worked together in the past in order to take advantage of shared mental models. While this might assist coordination in the short run, it

might eventually reduce flexibility by limiting the kinds of technical and domain knowledge individuals acquire, with potentially very serious negative long-term effects. As research into congruence progresses, undoubtedly, many more examples of side effects will be discovered.

This point closely relates to the multiple levels at which congruence may be studied. Exactly how congruence at the level of an individual influences congruence at the level of the team or even the entire organization is an open research question.

Second is the challenge that comes from the advent of new technology, the facing of new situations, and organizational changes. What may be considered acceptably congruent today may not be acceptable tomorrow. An example is configuration management, the use of which provided for a long time what many considered a close-to-optimal solution [3, 4]. By leveraging workspaces, more work is coordinated through the use of formalized reconciliation points (i.e., check-ins and merges of parallel changes). Any conflicts that arise are considered a necessary cost to achieving more parallel work [5]. With the advent of workspace awareness tools, which break traditional workspace isolation by notifying developers of relevant parallel work (i.e., parallel work that may lead to future conflicts), a "higher level" of congruence can be achieved, outdating traditional configuration management solutions. This is just one technological example of a new context in which congruence is to be achieved, but many examples of sometimes drastic changes in context exist that influence the state of congruence of an organization in many different ways.

A final complicating factor is that any advance in our ability to understand congruence is going to depend on the ways in which we can investigate the situations that present themselves. Metrics certainly are a critical component of building such an understanding, but to understand complex situations we will also need innovative visualizations that help summarize potentially vast amounts of information and distill that information back to critical patterns. The traditional visualization in this regard has been the social-network graph, but already some evidence exists that congruence between social and technical factors is better presented in other forms, for instance in a grid that enables detection of congruence of individuals and teams with respect to a given componentized code base [6]. Beyond metrics, thus, we expect significant future work in the area of congruence to focus on finding and designing novel visualizations that help both researchers and practitioners in properly diagnosing and recognizing situations of (in)congruence.

We hypothesize too that several canonical "styles" of congruence may emerge. In previous work [7], we identified several layers of coordination maturity, each layer defining an incrementally powerful coordination paradigm that defines how organizations principally operate at that layer. We believe that congruent forms of optimal coordination strategies will emerge for each coordination paradigm, forms that are necessarily limited by the organizational and social structures imposed by each type of paradigm. Ideally, each of the resulting "styles" contains a number of standard patterns that characterize how an organization in this style effectively operates when it is in a congruent state, and other patterns that will be indicative of it not being in a congruent state.

As a final caveat, we observe once more the fundamentally organizational nature of congruence. Much initial research has and will continue to take place in the context of a single product – its versioning and bug tracking archive, its team(s), and its overall coordination strategies and state of affairs. Most organizations develop multiple products, and its people will be part of multiple teams across those products. Care, thus, must be taken to understand the limitations that are set forth when studying a single product or team. Such studies must be accompanied with a careful

analysis of how its lessons may or may not be general (i.e., a careful articulation of the relevant threads to the validity).

Overall, the research agenda for *understanding* congruence can be summarized as follows:

- Building appropriate theories of congruence.
- Relating congruence as it exists at multiple levels, from individual developer up to the entire organization, or even across multiple organizations.
- Understanding paradigms of congruence, i.e., how sets of practices fit together to comprise coordination solutions with characteristic styles or flavors.
- Understanding the effects, and side effects, of practical interventions aimed at achieving congruence.
- Designing metrics and visualizations that help in understanding given situations.

# 5 ACHIEVING CONGRUENCE

Upon being in a state of incongruence, there are two basic strategies for working towards achieving congruence: lowering demand for coordination, and increasing coordination capacity. Lowering demand might involve approaches such as restructuring the code at hand to have fewer dependencies between parts of the product where coordination capacity between responsible parties is low, to provide training to the entire team on certain otherwise unfamiliar aspects of the product so that expertise is spread more evenly, or performing more upfront work to ensure interfaces and requirements are fully worked out and stable. Similarly, increasing coordination capacity can involve a wide variety of approaches, as in buying or building new tools for communication, coordination, or awareness, providing training on culture and communication, or providing shared displays of up-to-date project information.

We note that lowering demand or increasing capacity are not necessarily independent of one another. Consider an increased focus on requirements gathering and up-front design activity. While it may lead to lower coordination needs during the coding phase in terms of coder-to-coder coordination, it at the same time may lead to an increase in coordination needs in terms of coder-to-designer coordination. We must thus be careful to look at all the tradeoffs of these kinds of solutions – congruence is multi-dimensional.

In addition to looking at whether strategies lower demand for coordination or increase coordination capacity, we can also break down strategies as social versus technical. Some of the strategies mentioned in the previous paragraphs are distinctly social in nature, involving adjustments in what one would call the "people factors" that create an important part of the organizational context in which coordination will take place. Restructuring teams, providing training, instilling modern processes, team building activities, and establishing a more (or less) frequent meeting schedule are examples of social strategies.

Technical strategies fall into two categories: (1) reducing coordination needs, typically by reducing dependencies in the code, and (2) using tools that better support the coordination processes that are needed. Minimizing dependencies is certainly a goal throughout the design and implementation of a software system, but it is generally a difficult proposition to restructure code to have fewer dependencies on a regular basis (refactoring notwithstanding, for the purpose of maintaining congruence, drastic restructurings may need to take place regularly). Tools that better support coordination, thus, seem to have gotten more attention to date and play a major role in establishing a portfolio of techniques in an organization [8, 9].

An interesting point with respect to tool development is that most tools are constructed to respond to problems with previous tools; that is, new tools will optimize some needs, but necessarily lead to other needs that are met in less than ideal ways. Interruption management tools, for instance, emerged in response to the creation of tools that helped people communicate in a more instant and direct manner (particularly IM). We consider the incremental layering of new tools on previous tools a sign of progress and deeper understandings of the nature of congruence.

At times, though, it will be necessary to take a step back and understand whether the assumptions underneath certain coordination tools still hold. Configuration management is an excellent example in this regard. At this moment in time, it is still considered one of the essential tools (and it may be so for a long time). On the other hand, we have some informal accounts of the use of screen-sharing software to see each other's workspaces and coding activities, which seems to indicate that there are work practices that do not mesh well with the insulation provided by workspaces. Similarly, the ready adoption by the students in our project classes of Google Docs as a bug tracking tool or even for coding (despite the availability of state-of-the-art tools as alternatives to use) seems to also show a preference that is shifting towards very open development practices. The exact forces at work are subject to further study; we simply want to point out here that not all improvements will be incremental in nature but some will generate rather radical breaks with established practices.

Overall, then, we summarize the research agenda for *achieving* congruence as:

- Understanding what strategies are available.
- Knowing how each strategy influences congruence along the various dimensions and levels – positively and negatively.
- Understanding the cost of each strategy.
- How to build and maintain portfolios of strategies to be applied – and keep those current.
- Performing all of these in the face of coordination capabilities and requirements that are highly volatile, resulting in a need for mechanisms that can effectively provide and maintain up-to-date information upon which to make decisions.

# 6 CONCLUSIONS

In this paper, we have provided an overview of what we believe are some key challenges that must be comprehensively addressed in order for congruence research to result in theories, approaches, tactics, and tools that can be repeatedly and reliably inserted in today's development practices. We particularly have discussed a broad variety of issues with respect to *measuring*, *understanding*, and *achieving* congruence. Naturally, these are all related and one cannot and should not treat these issues in isolation.

We have deliberately included a broad definition and discussion of congruence to take into consideration the fact that coordination is essentially everywhere. Whenever there is a need for coordination, congruence captures how well the organizational and collaborative arrangements among people are capable of satisfying these needs. Congruence attempts to place measures, understandings, and actions onto the degree of match between need and capability.

A particular theme running throughout the entire paper concerns the fundamental nature of congruence. The fact that it represents a state, is descriptive only, is both dynamic and multi-dimensional, involves tradeoffs, and can be defined at multiple levels directly influences all

aspects of the discussion we have had. It is these characteristics that make the research agenda that we have laid out one that is quite ambitious. To make progress, the community will need to make numerous small steps, bring results together, repeat and extend studies, address individual aspects of congruence one by one before attempting to combine multiple aspects, and slowly but surely build towards a common core of knowledge. We hope this paper helps in outlining some possible directions along which this knowledge should be built.

We wish to add a final word on education. Congruence, in many ways, is a fundamental concern in software engineering – it underlies many of the words of wisdom we preach in our courses, from the very introductory course on software engineering to advanced capstone courses involving real-world customers or students engaging in open source or distributed projects. Bringing the lessons learned in the research on congruence into the classroom, particularly the studies that highlight the problems that arise when coordination is incongruent and the potential approaches to overcoming these problems, stand to enhance the preparedness of our students significantly.

# 7 REFERENCES


1. Cataldo, M., et al. 2006. Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools. ACM Conference on Computer Supported Cooperative Work. p. 353-362.
2. Ye, Y., Y. Yamamoto, and K. Nakakoji. 2007. A Socio-Technical Framework for Supporting Programmers, 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. p. 351-360.
3. Estublier, J., et al.,2005. Impact of Software Engineering Research on the Practice of Software Configuration Management. ACM Transactions of Software Engineering and Methodology, 14(4): p. 1-48.
4. Estublier, J., 2000. Software Configuration Management: A Roadmap, The Future of Software Engineering, A. Finkelstein, Editor. p. 281-289.
5. Mens, T.,2002. A State-of-the-Art Survey on Software Merging. IEEE Transactions on Software Engineering, 28(5): p. 449-462.
6. Trainer, E., et al. 2005. Bridging the Gap between Technical and Social Dependencies with Ariadne. OOPSLA Workshop on Eclipse Technology eXchange. p. 26-30.
7. Sarma, A., A Survey of Collaborative Tools in Software Development. 2005, University of California Irvine, ISR Technical Report UCI-ISR-05-3. p. 68.
8. Storey, M.-A., D. Cubranic, and D.M. German. 2005. On the Use of Visualization to Support Awareness of Human Activities in Software Development: A Survey and a Framework. ACM Symposium on Software Visualization. p. 193-202.
9. Biehl, J., et al. 2007. FASTDash: A Visual Dashboard for Fostering Awareness in Software Teams. SIGCHI Conference on Human Factors in computing systems. p. 1313-1322.

**Institute for Software Research • Carnegie Mellon University • 5000 Forbes Avenue • Pittsburgh, PA 15213-3890**