Using Development Experience to Calculate Congruence

Anita Sarma and Jim Herbsleb

April 03, 2008
CMU-ISR-08-105

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

Abstract

Coordination congruence has been defined as the match between coordination requirements and the actual coordination behavior of a team, where requirements are calculated based on underlying task dependencies and behavior based on communication patterns. In this paper we propose to expand the notion of congruence in two distinct ways. First, we use the concept of shared mental model as a determinant of coordination behavior, where shared mental model is defined as the common conceptualization of artifacts, tasks, and team members shared among developers who have worked together in the past. Second, we create a measure of expertise congruence that determines the match between the expertise that is required and that which is allocated to a project. We also present some issues that need careful investigation as we expand the notion of congruence.

This effort is partially funded by the National Science Foundation under grant number 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 Engineering, Coordination, Development experience, Congruence

1 INTRODUCTION

Software development involves a large number of developers working together with a large, common set of highly interdependent artifacts. The underlying technical dependencies among software artifacts create social dependencies among team members, which necessitate coordination in the team. However, identifying with who should one coordinate and regarding what is nontrivial. Despite the use of sophisticated coordination tools (e.g., configuration management system, issue trackers, and communication portals), a majority of teams typically subsist on suboptimal coordination. Ethnographical studies have shown the lack of proper coordination can be a major cause of software defects and problems [1, 2].

Cataldo et al., [1] have put forth the concept of coordination congruence, which calculates how well an organization's coordination patterns match their coordination requirements. Such a measure brings the attention of an organization to the state of their coordination patterns and if incongruence exists, they can take steps to reduce the gaps. Cataldo et al., compute coordination requirements of a team based on the underlying task dependencies in the project. These dependencies are then compared with the actual coordination behavior of the team to provide a measure of coordination congruence. In their work, they primarily determined coordination behavior from the communication patterns in the team.

The need for coordination arises generally because of the following reasons: (1) interdependent tasks, (2) conflicting changes to the code base, (3) common resources for the team, and (3) experience of team members. While Cataldo had specifically investigated congruence as a match between coordination requirements (because of interdependent tasks) and coordination behavior (represented by communication patterns), here, we expand the concept of congruence to consider the experience of individual developers – both their work experience on software artifacts and their knowledge of skill sets and experiences of their team members. More specifically, we propose the investigation of past experiences of developers to create a measure of: (1) shared mental models as a determinant of coordination behavior and use this measure to calculate coordination congruence, and (2) expertise congruence to determine the match between expertise that is required and which is currently allocated to a team.

The first contribution of the paper is the concept of the use of shared mental models as a determinant of coordination behavior. Past empirical work has found that when developers have worked together in the past they possess a shared understanding of the nature of software artifacts and their fellow team members' skills, which facilitates coordination amongst them [3]. Building on this research, we propose to identify shared mental models among developers based on their past experience and use this information as a measure for coordination behavior, which can then be used to calculate coordination congruence in a team. Possible sources of data for information of when developers have worked together in the past can be coding experiences (e.g., version control logs or Modification Request (MR) records) or design experiences (e.g., design records or team meeting logs)..

The second contribution of the paper is the concept of expertise congruence. Possessing the "right" knowledge to implement a task is a critical factor and when an individual lacks the knowledge they have to seek others who have the requisite knowledge. Developers in a project have been shown to spend a significant portion of their coordination activities in seeking (or providing) help. Not surprisingly, there exists a significant body of work on

expertise recommenders (e.g., EEL [4], Expertise Browser [5], Hipikat [6]), these systems focus on recommending experts to individual developers who need help in their particular task. Our work differs from these tools, as it is our goal to identify how well the expertise allocated to a project matches with the expertise required for the project. The required expertise for a project can be determined by its constituent software artifacts and the past experience of developers in those or related artifacts. A measure of expertise congruence can be determined by comparing the required expertise and the allocated expertise to a project.

Research into congruence is in its initial stages and as we begin to expand the concept of congruence to include other coordination behaviors and development metrics, two important research questions need to be addressed. First, how should we present congruence information to make it useful to developers or managers? Second, how can we validate whether the different congruence measures that we are proposing have an actual effect on the productivity of a team. In this paper, we briefly discuss the necessity for answering these questions and some possible solutions.

The rest of the paper is arranged as follows. Section 2 discusses shared mental model as a form of coordination behavior. Section 3 introduces the concept of expertise congruence followed by a discussion of possible next steps in Section 4. We present our concluding remarks in Section 5.

2 Coordination Congruence

Coordination is critical to software development. However, despite the use of the state-of-art coordination technology and spending significant amounts of time and efforts, in many cases teams perform with suboptimal coordination. Prior work has shown that mismatches between the coordination requirements and behaviors of a team can lead to lower productivity and increased costs [1].

Cataldo et al. introduced a novel methodology for computing coordination congruence – a measure of the "fit" between the coordination requirements and the actual coordination patterns of a team [1]. More specifically, they identify the set of dependencies among tasks to identify which sets of individuals should coordinate with whom. They created a set of two matrices. The first matrix – *Task Assignments*, a people by task matrix with elements $T_A[i,j]$ representing assignment of an individual to a particular task. $T_A[i,j]$ can either be a 1 or 0 to indicate whether developer i is assigned to task j). Following the same approach, they created a set of dependencies among tasks as a square matrix – *Task Dependencies*, with elements $T_D[i,j]$ can either be a 0 or 1 to indicate dependency between task i and task j based on the underlying dependencies between software artifacts. Interdependency is computed when artifacts have been edited together in the past. These matrices can then be used to create a *Coordination Requirements (C_R)* matrix that determines which pairs of developer should coordinate based on the underlying (logical) task dependencies.

Cataldo et al., then use data on communication patterns – primarily chat messages or log messages left in the Modification Request (MR) – to create a matrix of actual Coordination Behavior (C_B). The coordination requirements and coordination behavior matrices are then compared to identify the coordination congruence of the team. That is, how well the coordination behavior matches the coordination requirements.

2.1 Shared Mental Model

Traditionally coordination in a team is determined by the extent of communication, however, studies have shown there are other ways of coordination through team cognition mechanisms, such as shared mental models or shared work familiarity [2, 7, 8]. Particularly, past empirical work has shown that developers possessing shared team knowledge have a positive effect on team coordination, especially in distributed settings [9, 10], presumably because these experiences have allowed developers to develop a shared mental model (from now referred to as SMM) of the software and of the tasks involved in working on it. We, therefore, expand the concept of coordination behavior from just communication patterns to the use of shared mental models for calculating coordination congruence.

Consider an example scenario, where two teams are working together to implement a new version of a software system. Further, let us assume that Alice and Bob from Team-A and Team-B, respectively, have worked together in the past on an earlier version of the system. As a result of their past experience, both Alice and Bob are aware of an interface issue in the software, where in spite of a module functionality that is assigned to Team-B, can be ideally implemented by Team-A. During the reimplementation of that interface for the new version, Bob, because of his past experience, can now simply assign the implementation of the interface to Team-A (say Alice, because she possesses the requisite skill set) and Alice also aware of the quirks of the system accepts the task. However, if Alice and Bob had not worked in the past, the communication required to reassign the task between the team would have been much greater. In this example, the shared mental model of the system and the team members' expertise as shared between Alice and Bob affects their coordination behavior.

Shared mental models can be defined as a conceptualization of the work, the people performing the tasks, and the individual's own place in the process [3, 10]. Such knowledge can be created when team members work or train together on similar tasks. Shared mental models are also developed when individuals shared a common team experiences and organizational familiarity. We note that different kind of activities can create shared mental models. As a first step towards investigating shared mental models, we consider developers to possess a shared understanding of the structure and the complexity of the software when they have worked together in the past on the same or similar project. The hypothesis is that because of the shared understanding of the project they can coordinate better.

Towards this goal, we create a Shared Mental Model matrix where each element SMM[i,j] represents the condition when developer i has worked with developer j in the past (in implementing a MR) and, therefore, have had an opportunity to develop a common mental model[1]. Elements SMM[i,j] can be either (1) a 1 or 0 to show a dependency or (2) a weighted measure that represents the number of projects, artifacts, or work items in which the two individuals have worked together in the past. Weights can also be used to represent the recency of their collaboration.

This shared mental model matrix can then be used in two ways. First, the shared mental model can be used as a determinant of coordination behavior and then compared with the coordination requirement matrix (as computed using Cataldo et al.'s methodology of using

[1] A similar SMM matrix can be created by identifying developers who have worked on the same artifact within a specified window of time.

underlying task dependencies) to compute coordination congruence. Second, the shared mental model matrix can be compared with the actual communication patterns in the organization as evidenced by chat messages or discussion threads in MRs to understand the nature of communication patterns when developers share common mental models as opposed to when they do not have a common reference base.

Following such an approach raises several interesting questions.

1. What activities should be considered to create the SMM matrix? A first step, but a coarse measure, is to identify developers who have shared programming experience (e.g., worked together on the same MR). However, it is possible that shared mental models are created from other activities such as design planning or consulting. We shall explore other such measures to refine how and when shared mental models are created.
2. What are the effects of shared mental models on communication efforts? Past work has identified that shared mental models help developers coordinate better [3], we would like to investigate whether such models act as a substitute for actual communication. Further, in real life settings, it is possible that only a small pair of developers have worked together, in such cases we will investigate whether the effect of shared mental models is nullified or if it propagates through the team.
3. What effect does the passage of time and turnover have on shared mental model? Shared mental models are created when individuals possess a shared conceptualization of the structure of the system and how each developer fits in the project. With the passage of time, it is intuitive that this mental model deteriorates and becomes inaccurate. How rapidly does deterioration happen, and how can we address it?
4. Do shared mental models enable efficient coordination across distributed teams? Coordination costs across geographically distributed team members are much higher than collocated teams because of the lack of shared context. We will investigate whether shared mental models can create the needed context and thereby alleviate some of the problems with distributed development.

3 Expertise Congruence

Expertise can be considered to be either the knowledge of a particular technology, tool, or domain, or the knowledge about software artifacts in the project. Obtaining the right expertise to support the implementation or design is a critical factor to the success of a team. Research has shown that developers often spend a significant amount of their time either locating or collaborating with experts [6]. However, finding the right expertise in a team is often not easy and depends on an individual's knowledge of their team members activities.

Prior work on recommender systems rely on the past development efforts to create a measure of expertise. Ackerman and Halverson [11] in their study observed that past experience was the primary criterion that developers ordinarily used to determine expertise. More specifically, developers often referred to the change versioning system to identify developers who had experience with a particular file, generally assuming that the last person to have changed an artifact was most likely the 'expert".

The primary goal of existing work on expertise recommendation systems (e.g., EEL [4], ExpertiseBrowser [5], Hipikat [6]) is to help a developer locate expertise for their particular task. Here, we propose a slightly different concept. Instead of recommending experts for individuals who need assistance in their particular task, we compute the match between the expertise that is allocated and the expertise that is required for the project. By doing so, managers can analyze if there exists any incongruence (or gaps) in their team and take appropriate measures. For example, knowledge about a wide gap in the expertise in a team can prompt the manager to adjust the team composition to include more experts when possible. This in turn can then reduce the amount of time developers spend in looking for expertise outside the team. Alternatively, managers would be able to track whether their developers are becoming too highly specialized; meaning that they have little breadth of knowledge of the system, and that there is little knowledge redundancy to mitigate the risk of expert loss. A caveat is that expertise congruence should not be considered as a substitution for coordination, but as a complimentary concept [7].
We propose to extend the concept of "congruence measure" presented by Cataldo et al. to create a measure for expertise congruence. More specifically, we compare the level of expertise of a MR team assigned to a particular MR to the level of expertise required for that MR. To do this, we look at the past development experience of individual team members with individual software artifacts that have to be modified for the MR in question. This assessment of a set of current MRs then provides feedback on the congruence of team assignment on the project management level.

To describe the relationship between the MR, software artifacts and developers, we introduce a set of following matrices: (1) Experience matrix, (2) Task Allocation matrix, and (3) MR matrix. "Experience matrix", with elements E[i,j] represents the past experience of developer i in implementing or modifying artifact j in the change management repository. E[i,j] can, for example, represent the number of lines of code that a developer has written for a particular software artifact. The "Task Allocation matrix", with elements T[i,j] represents the current allocation of developers to particular MRs. T[i,j] matrix can be either 0 or 1 depending on whether developer j is allocated to MR i. Finally, we create the "MR matrix", a MR by artifact matrix, which represents the software files that need modification for a particular MR (cell ij can be 0 or 1 based on the fact whether MR i contains artifact j).

Multiplying E[i,j] and T[i,j] provides the "Expertise matrix", where each cell represents the aggregate experience of a MR team i with software artifact (file) j in the project. The numbers in the Expertise matrix are to be compared with the desired level of experience (this level can be set separately for different artifacts and/or MR, as preferred by the organization). The Expertise matrix can then be rewritten in a binary form, where 1s would correspond to MR artifacts for which the required level of expertise has been met, and 0s otherwise. The Expertise matrix elements corresponding to artifacts that are not to be modified in a particular MR are set to zero. Expertise congruence can then be determined as a ratio of the number of 1s in the Expertise matrix to the number of 1s in the MR matrix. For example, if the team assigned to a particular MR has expertise on 5 out of the 10 files in which the MR requires modification, we determine the expertise congruence to be 0.5. Congruence represents the proportion of expertise requirements that were satisfied and hence its value falls between 0 and 1.

We note that on first appearance it seems that preferentially creating a team of experts (e.g., experience on a project in the past or skill set) with a high "expertise congruence" is a good solution. However, while such a policy would assist coordination and increase productivity of the team in the short term, in the long term such a policy would suffer from two setbacks. First, it would create a team with a high level of redundancy reflecting a low diversity in the team's skill set, which might lead to a lower productivity of the team [12]. Second, it would lead to developers repeatedly working on the same project, which might make the task unchallenging and lead to employee turnover. Additionally, such a preferential policy for only assigning experts to a team might be infeasible because of changes in geographic locations and employee turnover.

We also note that the ability to evaluate the expertise congruence can be potentially useful for team allocation instead of just retrospective analysis. However, allocation of team members to a project does not merely involve selecting developers who have the most experience, but is a complex task that requires, in addition to expertise congruence, optimization of additional variables such as team load and priority of the project as compared to other ongoing projects to which a developer might be currently assigned.

4 Discussion

Currently, research on congruence is in its infancy and research potentials in this area abounds. Thus far, the concept of congruence has been used for retrospective analysis of a project to understand and validate the concept of coordination congruence. Here we have discussed two distinct ways (shared mental models as a determinant of coordination behavior and expertise congruence) in which to expand congruence into new directions. For the purposes of this paper, we have relied heavily on the concept of MR.to determine a developers' past experience (both for creating shared mental models and as a determinant of expertise). As part of future research, we need to investigate other activities and archives to support our hypothesis. Further, we have broadly used the notion of lines of code produced as a determinant of experience. We note that such a metric may not always accurately reflect a developers' expertise or the amount of interest vested in a project. We will investigate other sources that provide more accurate measure or are complementary to the lines of code measure.

While, it is important to extend the concept of congruence from the state of current research [1, 13] to investigate different coordination congruence measures, it is imperative to facilitate the adoption of this research by the software industry. Towards this goal, it is critical that we (1) validate whether these congruence measures are actually useful in real life and (2) provide intuitive ways for managers or developers to understand congruence and take appropriate steps.

To facilitate the adoption of congruence in real-life projects, we must first demonstrate the feasibility and use of congruence through retrospective analysis of development archives (code repositories, communication logs, version logs), and surveys to identify the coordination requirements and behaviors in a team.

Currently, congruence is measured as a number between 0 and 1. While this is an accurate measure of the state of congruence, it is non intuitive and does not help the user in understanding where the incongruence lies and what steps must one take to close the gap. We believe that visualizations that intuitively display: (1) interdependencies among developers, (2) communication patterns among the team, and (3) gaps in the network which lead to incongruence are useful and can provide assistance in making the team more congruent.

5 CONCLUSIONS

We proposed extending the concept of coordination congruence to include past experience of developers. More specifically, we proposed the use the shared mental model of developers who have worked in the past as a determinant of coordination behavior and proposed expertise congruence which provides a measure of the match between the expertise required and that which is actually allocated to a team. Finally, we discuss future directions for our research as well as for congruence research in general.

6 REFERENCES

[1] Cataldo, M., et al. 2006. Identification of Coordination Requirements: Implications for the Design of Collaboration and Awareness Tools. in ACM Conference on Computer Supported Cooperative Work. Banff, Alberta, Canada. p. 353-362

[2] Curtis, B., H. Krasner., and N. Iscoe.,1988. A Field Study of the Software Design Process for Large Systems. Communications of the ACM, 31(11): p. 1268-1287.

[3] Espinosa, J.A., et al. 2002. Shared Mental Models, Familiarity, and Coordination: A Multi-Method Study of Distributed Software Teams. in International Conference in Information Systems. Barcelona, Spain. p.425-433.

[4] Minto, S. and G.C. Murphy, Recommending Emergent Teams, in Fourth International Workshop on Mining Software Repositories. 2007. p. 5.

[5] Mockus, A. and J. Herbsleb. 2002. Expertise Browser: A Quantitative Approach to Identifying Expertise. International Conference on Software Engineering. Orlando, FL. p. 503-512.

[6] Cubranic, D., et al.,2005. Hipikat: A Project Memory for Software Development IEEE Transactions on Software Engineering, 31(6): p. 446-465.

[7] Faraj, S. and L. Sproull,2000. Coordinating Expertise in Software Development Teams. Management Science, 46(12): p. 1554-1568.

[8] Walz, D.B., J.J. Elam, and B. Curtis,1993. Inside a Software Design Team: Knowledge Acquisition, Sharing, and Integration. Communications of the ACM, 36(10): p. 63-67.

[9] Espinosa, A., et al.,2007. Team Knowledge and Coordination in Geographically Distributed Software Development. Journal of Management Information Systems, 24(1): p. 5-12.

[10] Espinosa, A., et al.,2007. Familiarity, Complexity and Team Performance in Geographically Distributed Software Development. Organization Science, 18: p. 613-630.

[11] Ackerman, M.S. and C.A. Halverson,2000. Reexamining organizational memory. Communications of the ACM, 43(1): p. 58-64.

[12] Dattero, R., S.D. Galup, and J. Quan,2007. The knowledge audit: Meta-Matrix analysis. Knowledge Management Research & Practice, 5(3): p. 212-221.

[13] Valetto, G., et al., Using Software Repositories to Investigate Socio-technical Congruence in Development Projects, in Workshop on Mining Software Repositories. 2007. p. 27.

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