2006-580: ASYNCHRONOUS COLLABORATION: ACHIEVING SHARED UNDERSTANDING BEYOND THE FIRST 100 METERS

Raghvinder Sangwan, Pennsylvania State University
Dr. Raghvinder S. Sangwan is an Assistant Professor of Information Science in the School of Graduate Professional Studies at the Pennsylvania State University. He currently teaches software engineering to professional graduate students from Fortune 500 companies such as Boeing, Lockheed Martin, Merck, Siemens, and Unisys. Dr. Sangwan is a Consulting Member of Technical Staff at Siemens Corporate Research, with over seven years of experience in software engineering research and development. Formerly, he was a lead architect at Siemens Medical, where he was involved in a distributed development project with over a thousand developers in six countries across three continents. Dr. Sangwan holds a Ph.D. in Computer and Information Sciences from Temple University.

Kathryn Jablokow, Pennsylvania State University-Great Valley
Dr. Kathryn W. Jablokow is an Associate Professor of Mechanical Engineering at the Great Valley School of Graduate Professional Studies, Pennsylvania State University. A graduate of The Ohio State University (Ph.D., Electrical Engineering, 1989), Dr. Jablokow’s teaching and research interests include robotics, control systems, and problem solving in science and engineering. She is the author (under the name K. W. Lilly) of Efficient Dynamic Simulation of Robotic Mechanisms (Kluwer), an Associate Editor of IEEE’s Robotics and Automation Magazine, and a member of the Executive Committee of ASME’s Technology and Society Division. Dr. Jablokow has developed four courses based on Adaption-Innovation theory at the graduate level and is currently investigating the relationship between cognitive style and invention.

Matt Bass, Siemens Corporate Research
Matthew Bass is currently a Member of the Technical Staff for the Software Architecture Program at Siemens Corporate Research. In this role, Matt conducts research and consults with Siemens business units in the areas of Geographically Distributed Software Development and Software Architecture. Prior to coming to SCR, Matt was a Member of the Technical Staff at Carnegie Mellon's Software Engineering Institute. Matt holds a Masters of Software Engineering from Carnegie Mellon University and a Bachelors of Science in Computer Science from the University of Pittsburgh.

Dan Paulish, Siemens Corporate Research
Dr. Dan Paulish has over 20 years of experience in software project management, and he has lectured internationally on software project management, software process improvement methods, and measurement. Dr. Paulish has worked as the Software Architecture R&D Program Manager at Siemens Corporate Research for the past 9 years. Prior to this, he worked in the areas of software measurement and software process improvement methods for software projects. Dr. Paulish is a co-author of Software Metrics: A Practitioner's Guide to Improved Product Development, published by IEEE Press, and the author of Architecture-Centric Software Project Management, published by Addison Wesley. Dr. Paulish holds a Ph.D. in Electrical Engineering from the Polytechnic University in Brooklyn, NY.
Asynchronous collaboration: achieving shared understanding beyond the first 100 meters

Abstract

Motivated by a real-world example from the domain of software product development, we discuss some of the key factors that impact shared understanding among collaborating teams in general, along with specific implications of those factors for asynchronous collaboration in particular. Shared understanding is viewed through the lens of Kirton’s Adaption-Innovation theory, a powerful framework for understanding problem solving that provides insights on the creative behavior of individuals and the convergence and divergence of collaborating teams. Proposed research directions are suggested for the future, and implications of this work for engineering education are discussed as well.

1. Introduction

It is well known that effective communication plays a key role in the performance of product development teams. Researchers have shown, for example, that well-coordinated teams demonstrate a higher level of overall performance, especially when their tasks are interrelated and compactly situated. Achieving the desired levels of coordination among geographically distributed teams can be extremely challenging, however, due to the negative impact that increased distance has on communication. Research shows that a mere 100 meters of separation results in a significant drop in communication between team personnel. Beyond this point, it becomes almost irrelevant whether collaborators are located in two different buildings, cities, countries, or continents: communication is greatly degraded and team performance suffers serious setbacks in all these contexts.

These negative effects are even more pronounced when the teams are located several time zones apart with minimal overlap in working hours. Under these conditions, collaborators must rely heavily on asynchronous interactions (i.e., different individuals providing input to the task at different times); if the work is closely related, and the teams are not communicating well, problems can escalate quickly. This paper will concentrate on the role that shared understanding plays within this wider context, with a special focus on some of the cognitive factors related to the successful resolution of differences in shared understanding and suggestions for ways in which these factors might best be managed when collaborators are far apart.

In response to the many critical issues involved in asynchronous collaboration, project managers often attempt to minimize cross-site communication by allocating relatively independent assignments to different sites. While this is a reasonable approach, the aim is difficult to achieve, especially in the case of software product development. Creating loosely coupled software components that can be developed independently by separate teams is challenging due to the complex nature of the dependencies that exist among the components of a software product; in this context, a “dependency” is defined as any aspect of a component that relies on or provides something to another component or aspect of the system. That “something” could be a particular data construct, a semantic behavior, a timing behavior, or a resource behavior, among others. To manage these many dependencies effectively, the technical teams involved must collaborate, no
matter where they are (respectively) located. To ensure the overall success of the project, their communication must be facilitated and managed as they work on these closely interrelated tasks.

One key aspect of team communication that is critical to the success of a distributed project is the shared understanding of the original problem to be solved\textsuperscript{22, 23}. As previous research demonstrates, shared understanding is a function of many variables related to process, outcome, and the problem solvers themselves\textsuperscript{25}. In this paper, we will view the concept of shared understanding through the lens of Kirton’s problem solving theory\textsuperscript{17} in order to shed light on some of the key factors involved and the relationships between them. In particular, we will focus on problem solving level and problem solving style as fundamental cognitive variables in collaborative problem solving (at any distance), the impact that gaps in these variables (i.e., gaps between what is required and what is being deployed) can have on shared understanding (in particular) and problem solving performance (in general), and the management of these two dimensions of cognitive diversity in the context of teams engaged in asynchronous collaboration.

We begin (in Section 2) with the specific real-world example from software product development that motivated us to explore the issues described above. In Section 3, we discuss key aspects of Kirton’s problem solving theory, including cognitive level, cognitive style, and the Paradox of Structure as it relates to problem solving diversity in teams; in addition, we link Kirton’s work with the contributions of others by examining the impact that gaps in cognitive level and/or style can have on shared understanding. This is followed (in Section 4) by a return to the motivating example of Section 2 with new insights gained from the application of Kirton’s theory. Implications for asynchronous collaborative problem solving are discussed in Section 5, including some proposed directions for future research. Section 6 contains a discussion of the implications this work has for the future of engineering education in general, while some final comments and conclusions are presented in Section 7.

2. A Motivating Example from Software Product Development: The Global Studio Project

Siemens Corporate Research (SCR), in collaboration with six universities, across four continents (Carnegie Mellon, USA; Monmouth University, USA; Pontifical Catholic University, Brazil; Technical University of Munich, Germany; University of Limerick, Ireland; International Institute of Information Technology, India) is currently conducting a multi-year experiment to gain a better understanding of the issues surrounding and the impact of various practices in managing globally distributed software development projects. The primary technical goal of this Global Studio Project (GSP) is to develop a unified management station (called MSLite) for building automation systems (i.e., systems that automatically control the internal functions of buildings, such as heating, ventilation, air conditioning, access, and lighting), while simultaneously exploring best practices in asynchronous, globally-distributed problem solving.

The intended users of the MSLite system are facility managers who need to operate the many systems required to support building functions. Since there are a large number of these systems, a Field System Simulator (FSS) is used during software product development to simulate the building automation domain. An FSS configuration file is used to create the initial configurations of these systems, including their structure and the initial values of their various properties. For example, a system that monitors air conditioning on some floor of a building may have several temperature sensors at various locations, and their threshold values for how to regulate the
Temperature may be set to the desired levels within the FSS configuration file. Figure 1 illustrates this broad functional context of the MSLite system.

![Figure 1: Broad functional context of the MSLite System](image)

In order to distribute the development of the MSLite system to the six student teams listed above, its architecture must be defined and then used to derive relatively independent tasks, which are then allocated to the different teams. The system architecture can be described using several different views; the view most relevant for work allocation is the module view, which shows software modules and their dependencies. A software module is a unit of implementation that defines a coherent piece of functionality. Figure 2 shows a high level module view for MSLite.

![Figure 2: High level module view for MSLite](image)

One can continue to drill down into each module and add more detail, showing the sub-modules and their dependencies as well. These modules (and sub-modules) can then be assigned to the individual teams for development. The intent is to add enough detail so that it is clear to each team which modules they must rely on for implementing their piece of the project and which
other modules rely, in turn, on them. The module view, however, only shows static structural dependencies; it is also useful to create a components and connectors view of a system, which illustrates the dynamic dependencies at runtime. As an example, Figure 3 shows a components and connectors view for MSLite.

![Figure 3: Components and connectors view of the MSLite System](image)

Here, a component is an independently deployable unit of functionality, and connectors are mechanisms of communication among components. A component is created through a combination of one or more software modules. As seen from Figures 2 and 3, a one-to-one
mapping between the modules in the module view and the components in the components and connectors view does not exist directly. Yet this mapping must be defined so that each team knows clearly which components are affected by which modules; obviously, this is a complex and challenging task. Together, the views enable the dependency analysis that is used to determine the level of coordination and collaboration required among the teams working on different aspects of a system. In addition, one must also understand other factors, such as timing and resource dependencies among the components, to get a complete and accurate picture of the system.

Figure 4: The organizational structure for the Global Studio Project (GSP)

Once dependencies are defined (and hopefully, well understood), the software modules are allocated to the teams for development. As shown in Figure 4, the teams in the GSP are organized in terms of technical and administrative functions using a “hub and spoke” model. The “hub” is a central team with seven distinct roles and is located at Siemens Corporate Research in Princeton, New Jersey (USA). Most roles, except for the project manager and the supplier
manager, are filled by student interns who come from one or more of the six universities listed earlier; the project and supplier managers are both full-time Siemens employees. The project manager is responsible for the entire multi-team project, while the supplier manager serves as the project manager for each remote team. The “spokes” are remote student teams with five distinct roles, as shown. Each team is limited to 8 to 10 members; a single team member may play several roles, or a single role may be filled by several team members.

The central team has many responsibilities: eliciting system requirements, formulating a definition of the problem, creating the software architecture that defines the gross structure of the system, distributing development among the student teams, integrating the elements as they are developed, performing integration and system testing, administratively managing the project, and defining the overarching software development process. In turn, the remote student teams design, develop, and test the code modules and/or subsystems corresponding to the elements allocated to them by the central team.

As can be seen from the description of the GSP thus far, there are a number of issues that make software development in such a distributed environment particularly challenging. First, the dependencies among work units must be well understood if the distributed teams are to develop them efficiently and effectively. A significant amount of communication and negotiation has to occur between the developers and the users of the software modules to arrive at a good understanding of the final product. Since most software problems are initially ill-defined, incomplete, and uncertain, this can be very difficult. As a result, there is a great need for reliable and efficient information-sharing mechanisms, careful and complete documentation, and effective collaboration on units of work in order to create a shared understanding of the problem to be solved. Work assignments for remote teams have to be carefully crafted, taking into account the skill level of the team members and the functional coupling among software units among other factors.

In the GSP experiment, we have found these issues manifested in various ways, including:

- **Misunderstandings of requirements and architecture**: Requirements and architecture, when not clearly specified, led teams to make assumptions in lieu of missing or incomplete information in order to make progress on the work assigned to them. Often the assumptions were incorrect, causing serious misunderstandings.

- **Disruptions in schedule**: When components of a system depend on each other, many critical paths exist where certain parts of the system have to be implemented before others. When teams whose work fell in these critical paths “slipped” their schedules, it had a cascading effect, leading to disruptions in the overall schedule for the development of the system and an extensive re-planning effort. Another related issue was the sheer number of inquiries from remote team members to the central team, overloading them with questions of clarification. The central team became a bottleneck, affecting productivity and, in turn, delaying the schedule. The remote teams sought clarification even when work packages delivered to them by the central team consisted of well-written specifications. The purpose of many inquiries turned out not to be an issue of clarification, but rather, an attempt by the remote team members to synchronize their understanding of the problem with that of the central team.
Demoralized teams: When teams had to sit idle because the parts of the system their work relied on had not been completed, or if they had to do substantial rework because the requirements and architecture of the system were either not communicated properly or were misunderstood, they quickly became demoralized.

All these challenges point to difficulties in assessing and managing the shared understanding of the original problem (or a sub-problem within it), whether related to the problem’s definition or issues of scheduling. In the next section, we offer a new perspective on shared understanding by way of problem solving theory, with an aim of providing new insights that might direct future research on this important topic.

3. Shared Understanding through the Lens of Adaption-Innovation Theory

Researchers have suggested a number of models for shared understanding and developed a variety of systems for measuring and monitoring it. Some of these models and approaches are quite general6, 9, 20, 25, 33, while others are more specific3, 13, 19, 29, 30. Each of these contributions offers some insight into the nature of this complex concept, but even in combination, they do not explain all the subtleties involved. In this paper, we suggest that examining shared understanding through the lens of Kirton’s Adaption-Innovation theory17 can help fill in some of the gaps and shed light on collaborative problem solving in a broader context as well.

Adaption-Innovation theory is well established and has been highly validated in practice for 30 years, with over 500 journal articles and 90 graduate theses devoted to its development and application28. To date, most team applications have occurred in corporate environments, with somewhat limited dissemination of the results. A few exceptions are Diana’s work with R&D personnel10, Hammerschmidt’s investigation of management teams12, Keller’s study of R&D project groups16, and Buffinton, et al.’s study of teams of engineering and management students5.

One aim of our work here is to increase the visibility of Adaption-Innovation theory and the value it provides in real-world application. In particular, Adaption-Innovation theory helps explain the convergence and divergence of problem solving teams, whether they are working asynchronously or not. In this section, we offer a brief summary of some key concepts from A-I theory, synthesized with previous research in shared understanding and asynchronous collaboration in order to clarify the relationships between them. We begin with some basic assumptions and definitions related to problem solving and problem solvers in general.

3.1. The Distinction between Level and Style

At a fundamental level, all humans use the same basic problem solving process11, 17, 31, 34. Each individual carries out the steps of this process differently, however, using diverse cognitive capacities and unique collections of stored mental resources, and doing so in different characteristic ways or styles. One of the most significant contributions of Adaption-Innovation theory is the clear distinction it provides between the level and the style of problem solving, or more generally, between cognitive level and cognitive style. Other key problem solving variables identified by Kirton include the motives that drive individuals to solve particular problems and their individual perceptions of opportunity that give rise to these problems17. In this paper, we will focus on cognitive level and cognitive style, with slightly more attention given to the latter due to its relative unfamiliarity in the engineering community.
Cognitive level refers to the combination of an individual’s innate potential capacity and his accumulated manifest capacity. Potential cognitive level provides a measure of how much a person can know and ultimately helps to define the upper limit of the individual’s problem solving ability. It may be measured in terms of intelligence (e.g., I.Q.), talent, or the degree of complexity a person can manage, among others. Manifest cognitive level provides a measure of how much a person already knows and/or has experienced. This information is used as a personal “in-house” resource when problem solving, accessed and increased through memory and learning, respectively. It might be assessed in terms of acquired knowledge, experience, skills, or levels of attainment, in addition to other means.

From a general standpoint, the definition and role of cognitive level in the problem solving process are understood quite well; in assigning problem solvers to a particular task (such as software development, for example), we routinely consider their natural limits (potential level), as well as their current skills, abilities, and experience (manifest level). Both of these dimensions of cognitive level will serve as limits on the shared understanding of any concept, including a specific problem.

Kirton defines cognitive style as the “strategic, stable, characteristic, preferred manner in which people respond to and seek to bring about change,” including the solution of problems. At the core of Adaption-Innovation cognitive style is the way in which a person prefers to manage structure, with the different style preferences stretching along a continuum that ranges from strongly adaptive on one end to strongly innovative on the other. In general, individuals who are more adaptive prefer more structure as they solve problems, with more of this structure consensually agreed. In contrast, more innovative problem solvers prefer less structure when problem solving and are less concerned about acting in accordance with the structures (e.g., assumptions, models, architectures) that currently exist.

In general, then, more adaptive individuals prefer to approach problems from within the given frame of reference and strive to produce solutions that are “better” rather than “different.” The value of these individuals is clear, since they tend to be the experts of the prevailing system and are dedicated to its maintenance and efficiency. In other words, they are especially good at fine-tuning the current rules and procedures in order to make them (and the prevailing system) operate as effectively and efficiently as possible. The more innovative, on the other hand, tend to detach a given problem from its customary frame of reference and search for solutions that are typically seen as “different,” they may or may not be “better.” One way of summarizing this basic difference in cognitive style is to say that individuals who are more adaptive prefer to solve problems using the current “rules”, while more innovative individuals tend to solve (the same) problems despite them.

These differences in cognitive style produce distinctive patterns of behavior, which are particularly important when groups of individuals solve problems collaboratively. Problem solving is commonly considered to include the following stages: problem definition, data collection, idea generation, evaluation of solutions, and final solution implementation. With regard to these stages, more adaptive problem solvers generally accept problems as they have been defined, along with any agreed-upon constraints. In collecting data, they strive to be meticulous in their searching and favor information and perspectives that are closely related to the original problem structure. When generating ideas, adaptive individuals will typically offer a
sufficient number of novel and creative solutions which are relevant, readily acceptable, and aimed at improvements on the current system or product. These solutions are often easier to implement than solutions generated by a more innovative person because they fit more readily into the prevailing system’s structure. When evaluating and implementing solutions, the more adaptive problem solver looks for a quick resolution to the problem that will limit disruption and immediately increase efficiency. More innovative problem solvers, on the other hand, are more likely to reject the original, generally accepted definition of a problem and redefine or reframe it. It may be difficult to relate this new view of the problem to its original definition, but it may also bring new insight. In collecting data, the more innovative tend to look outside the original problem structure for different perspectives, which they import into the solution process, often spanning multiple domains. When generating ideas, innovative individuals may proliferate novel and creative ideas, some of which are not generally acceptable or may not appear relevant to the problem. When evaluating and implementing solutions, the more innovative problem solver is less concerned with immediate efficiency and the risk of system disruption; he tends to look for potential long-term gains instead.

3.2. The Impact of Level and Style on Shared Understanding

Shared understanding can be defined in terms of (shared) mental models, that is, cognitive models that provide an organized framework through and within which the current situation or problem can be made meaningful. Shared mental models are the outcome of experience and knowledge, both of which are shaped by an individual’s potential level, preferred style, and prevailing motives. In other words, what a person knows and experiences (manifest level) is a function of how much he can know (potential level), how he prefers to learn about it (cognitive style), and what drives him to learn it in the first place (motive). As discussed by Kirton and others, one practical outcome of the variations in level and style is the different views of problems and their solutions that result – in short, insight into the preferred ways people understand and seek to bring about change. Each cognitive level and style has its own advantages and disadvantages in the context of a particular problem.

With regard to level, for example, a person of great expertise in a specific domain can be indispensable when a particularly difficult problem in that domain arises. That individual’s high level may also give him an advantage in the understanding and resolution of other problems that share similar (but not necessarily identical) characteristics and/or domains. But an excess of expertise – when it is not needed – can also be a disadvantage, particularly if the “expert” is unable to recognize the differences that do exist between his specific domain of expertise and the one in which the current problem resides, or if he reads more into the problem than was intended or required. “More” is not always “better” in problem solving; there are times when a novice or relative newcomer to a particular field can solve certain problems more effectively than the resident experts.

In addition, it may be difficult for a person of extensive experience or high potential level to communicate their understanding of a problem to those who are less informed or able, unless he is particularly skilled when it comes to educating others first. In general, though, “enough” of the relevant domains of expertise, knowledge, and skill must be represented in a problem solving team (or acquired along the way), as and when they are needed, in order to meet the
requirements of the problem at hand. These include the development of a shared understanding of how that problem is defined and which solutions will be acceptable in resolving it.

Similar statements can be made with regard to cognitive style and its impact on shared understanding of a problem. As an example, consider the problem definition stage: for a more adaptive individual, the chief aim of this stage is to define the problem as explicitly as possible in a way that is acceptable to authority and to other members of the team as well. If a problem definition is supplied at the outset, a more adaptive person is more likely to accept this definition as fixed, clarify any questions he has about details, and endeavor to move forward to the next stage of problem solving. If any part of the problem is not well defined, the more adaptive person is more likely to be frustrated by the ambiguity and may have difficulty moving beyond this stage until such questions are resolved. More innovative individuals see the problem definition stage as a chance to explore different problem formulations, and they are generally more comfortable with ambiguity in the definition of a problem. Innovators often delight in defining and redefining problems using new perspectives, even when a problem statement is supplied. This can lead to difficulties for themselves and others, however, if they have not learned to discipline their behavior and converge on a suitable definition in a reasonable time.

3.3. Problem A, Problem B, and the Paradox of Structure

With brief descriptions of cognitive level and cognitive style in hand, and some basic observations on how they impact shared understanding, we will now consider several more key concepts from Kirton’s theory of problem solving that relate to collaboration. First, we note once again that the difficulty and complexity of the problems engineers must solve today make collaboration a necessity. No one individual has all the required cognitive levels (e.g., knowledge, experience, skill) at his disposal to solve such problems, nor is any one person readily able to bring all the required styles of problem solving to bear. A diversity of problem solvers is necessary to solve a diversity of problems, as Kirton notes17, including the potentially wide diversity of sub-problems that underpin any major problem solving effort.

It is easy to imagine the difficulties that can arise when individuals of different levels and styles come together to solve a problem; most engineers (indeed, most humans) are all too familiar with them, in fact. Kirton summarizes the situation succinctly in terms of “Problem A” and “Problem B”17. Problem A is the original problem that a group has come together to solve; its resolution is their principal (and in the case of software development, the assigned) aim. When the group forms, however, they automatically inherit an additional problem (Problem B), that is, the management of their own diversity, which may include differences in level, style, motive, and perceptions of opportunity, among others. Handling Problem B effectively is generally neither trivial nor simple, but it is absolutely necessary if Problem A is to be solved successfully.

The existence of Problem B presents an interesting paradox for the problem solvers involved: the diversity of level and style (among other variables) that can create so many challenges in collaboration may be the very diversity required to solve the problem at hand! Kirton refers to such dilemmas as examples of the Paradox of Structure17, i.e., that notion that all structures – be they cognitive, physical, social, or otherwise – both enable and limit at the same time. In support of this concept, Carlile7 notes that gaps in manifest cognitive level (specifically, knowledge and skill) between individuals and/or groups are desirable and necessary in order to meet the goals of the problem solving team as a whole; however, these same knowledge and skill structures also
act as limits. Carlile concludes⁷: “Knowledge is both a source of and a barrier to innovation” (where “innovation” in this case refers to the act of new product development rather than a particular cognitive style).

Similar observations have been made with respect to a team’s diversity of cognitive style. Jablokow and Booth⁸, for example, note the paradoxical influence of cognitive gaps in style between collaborating groups of engineers engaged in new product development within integrated organizations. In particular, they observe that mean differences in style between maintenance and production engineers (more likely to be more adaptive, on average) and R&D engineers (more likely to be the more innovative group, on average) are both necessary and problematic.

In summary, Kirton notes that successful teams spend more time on Problem A than on Problem B by learning how to make the best use of each individual’s unique problem solving contributions when and as they are needed, while minimizing friction between problem solvers of diverse characteristics through healthy doses of understanding and mutual respect⁹. In other words, successful teams make the best use of their particular cognitive structure by identifying and managing the balance between its enabling and limiting features (the paradox of their particular structure), all with the most effective resolution of the current problem in mind.

4. **Return to the Motivating Example**

Returning to the example described in Section 2, in which globally distributed teams of student problem solvers are attempting to collaborate effectively on the development of a highly complex software system, we now consider the links between the observed challenges and the insights into collaborative problem solving which Adaption-Innovation theory provides. To support our discussion, we note first that a number of different types of data have been collected in the Global Studio Project (GSP), although none of them have been formally analyzed thus far. These data include the following:

- **Skill level**: technical skill levels of all team members
- **Team histories**: who has previously worked with whom and for how long
- **Communication data**: who communicated with whom (and how); whether the communications were related to solving a particular “problem”; the effectiveness of those communications in solving that particular problem; whether communications were related to simple data exchange, etc.
- **Task-related data**: the tasks on which each individual spends his time (and how much time); evaluations of work descriptions/task definitions provided by the central team

In addition, the project manager for each team (a role played by the supplier manager within the central team) reports his observations of each student team back to the central team as milestones of the project are reached. There is no formal tracking of each team’s problem solving process between the completions of those milestones, however, which can lead to great difficulties when software components from one team do not meet the requirements anticipated by another team.

We note first that the data collected thus far provide some notion of the problem solving level of the individuals making up each team (both initially and as the project progresses); skill levels,
experience within a particular team, and the time spent on a particular task can all be linked to problem solving level. In addition, patterns of communication and (to some extent) their effectiveness in resolving specific problems have been recorded; these do not provide a direct measure of either level or style, but if examined more closely, they might well reveal variations in both.

Using Pascual’s framework for mental models, the data currently being recorded in the Global Studio Project (GSP) relate to certain aspects of “teamwork” and “taskwork”, respectively, but they do not provide a complete picture of either one. In particular, direct measures of problem solving style and the direct detailed tracking of the problem solving process of each team are noticeably absent. Before these processes can be tracked, however, a firm understanding of the cognitive processes involved must be acquired; only then can we understand and best decide how to track any individual’s progress in relation to that of others (both in the team and across teams) within the same overall system.

5. Implications for Asynchronous Collaborative Problem Solving

Managing the diversity of level and style which inevitably results in and across collaborating teams in the interests of shared understanding and effective communication is a significant challenge and one that is exacerbated by separations in time and space. As Mulder, et al. note, for example, synchronous settings are better for establishing shared understanding of a problem (“convergence”), while asynchronous settings are more effective for information exchange (“conveyance”). Using these observations and problem solving theory as a backdrop, what can we suggest in general regarding the measurement, monitoring, and management of shared understanding under asynchronous conditions?

First, if potentially clashing differences in cognitive level and style (Problem B) are to be managed in support of understanding (and completing) the original task (Problem A), their direct measure at the outset of a distributed project will be required. Effective measures of level are generally well understood and available (e.g., familiarity with a particular subject, skill level, experience); validated measures of cognitive style are also available. For example, the Kirton Adaption-Innovation Inventory (or KAI) is used to assess one’s style preference along the Adaption-Innovation continuum. The KAI has been highly validated across cultures and performs its function neatly and compactly; we have found its use to be very effective in improving the performance of problem solving teams. Other measures of cognitive style, including the MBTI, might be used as well. In general, assessing the cognitive level and style of the members of globally distributed teams should not be problematic beyond the need to choose the appropriate instruments, administer them, and process the results, all assuming that the necessary resources of time and money are available.

Second, the impact of level and style throughout the problem solving process of each team must be observed and tracked. To do so will require a model of the problem solving process itself, and this model must take both synchronous and asynchronous interactions into account. As mentioned earlier, several general models of the problem solving process exist, but none of these explicitly address synchronous vs. asynchronous interactions as part of their respective frameworks; here, there is much work to be done. Once an appropriate process model has been created and validated, mechanisms will have to be designed to record the impact of level and style on each of its stages. The current practice of recording “communication data” in
the Global Studio Project (see Section 4) is one example of such a tracking mechanism, but more rigorous (and better understood) practices are needed. Here, the tracking mechanisms themselves may require both synchronous and asynchronous elements; deeper study of this task is also required.

Third, as the impact of level and style on each stage of the problem solving process (within and across teams) is tracked and analyzed, mechanisms for revising the process must be incorporated, so ineffective practices can be changed, and changes in the problem can be addressed effectively. Challenges in collaborative problem solving can arise because of cognitive gaps within or between teams (Person to Person), but they can also arise as a result of gaps between the resources of the available problem solvers and the resources required to solve the problem at hand (Person to Problem)\(^{15}\). The impact of both kinds of gaps (in both level and style) must be monitored and managed throughout the problem solving effort.

In conclusion, it is apparent that all members of every team – but especially the team leaders – need to understand the process of problem solving and its progression, as well as some of the other key concepts discussed here, in order to implement the suggestions presented above. The necessary key concepts, such as the Paradox of Structure, Problem A and Problem B, and the distinction between cognitive level and cognitive style, apply throughout the problem solving process and across technical domains (and disciplines!). The added investment on the part of every team member is well worth the time and effort, however. As Kirton notes\(^{17}\), problem solving is the key to life – the knowledge and experience acquired in this “professional” context will be equally valuable in other contexts as well.

5.1. Research Propositions and Future Work

The primary aim of this paper is not to suggest a comprehensive final solution to the “problem” of shared understanding for asynchronous collaborating teams. Rather, our goal is to offer enhancements to the existing framework for future study of this critical issue and the factors that influence it. In particular, we believe that Adaption-Innovation (A-I) theory provides a robust framework for understanding and investigating some of the key variables involved in shared understanding. As a result, and with the motivating example of Section 2 in mind, we are currently pursuing the following general directions in our research:

- Development of a problem solving process model for globally distributed technical projects that explicitly includes synchronous and asynchronous elements and interactions;

- Design of mechanisms and assessments for tracking the impact of cognitive level and cognitive style on each stage of such a problem solving process;

- Design of mechanisms for revising this distributed problem solving process to incorporate feedback and to respond to changes in the problem (as they relate to gaps in level and style).

As these general tasks are completed, we will be applying them to the Global Studio Project for vetting and validation. This will require the assessment of cognitive level and style across all the teams involved, a step which is currently underway.
6. Further Implications: the Future of Engineering Education

Many people within the engineering education community have argued that engineers need to understand more than engineering subjects alone; “communication”, “teamwork”, and “problem solving” have been identified as problematic areas for engineering graduates for the past decade or more. The issues raised in this paper – i.e., the complexity of today’s technical problems, the increasing number of globally distributed projects – support this argument. What is, perhaps, unique about our presentation is the emphasis we place on the rigor and depth of understanding of cognition and problem solving required by all members of every team – not just the main administrative team or the team leaders. Problem solving leadership, as noted by Kirton\textsuperscript{17}, is a social role, which may be assumed by any member of a team as required in order to solve the current problem. As such, it is imperative that each team member is equally well prepared to assume the lead where and when they are called upon to do so.

6.1. A Formal Problem Solving Module at Penn State University

As an example of formal problem solving education in an engineering context, a 3-course module in problem solving is currently offered at Penn State University’s Great Valley School of Graduate Professional Studies. Briefly, the first course in this module focuses on the individual problem solver, helping students gain insights about their own problem solving characteristics (including cognitive level and cognitive style) and how they are related to (and impact) the systems (i.e., structures) with which they deal. The second course expands the domain of interest to problem solving teams, addressing such issues as cognitive gap, Problem A/Problem B, and coping behavior, which allows a problem solver to behave in ways that do not match his preferences (and which cost the individual extra effort). The third course focuses on problem solving leadership and the facilitation of problem solving teams through knowledge and application of problem solving theory. The Penn State Problem Solving Module has been very successful thus far. Over 50 students per year (on average) attend the first course, and all three courses are being considered as cross-cutting electives in Penn State Great Valley’s Management and Education Divisions. Further details about these courses will be provided in a future publication.

7. Final Comments and Conclusions

Educating engineers to be better problem solvers is a goal with which all engineering educators (and many practitioners in industry) are familiar. The challenges associated with meeting this goal continue to mount, however, as problems (including those within academia itself) become larger, more difficult, and more complex. The need to distribute such problems among a diversity of problem solvers is implicit, and the management of this diversity adds to the overall levels of difficulty and complexity involved.

How should the engineering profession respond to these growing challenges? We argue that engineers need to know more than they used to, and about different things. In particular, engineers need to understand the (cognitive) problem solving process in more depth, including the key variables that underlie individual differences in behavior and the impact these variables can have on the products engineers create. In addition, issues of culture must be considered, as global distribution of projects becomes more and more prevalent, and asynchronous interactions become the norm. In the past, many admonitions to engineers to broaden their education to include the social and behavioral sciences were based on the view that engineers would benefit
socially from an understanding and appreciation of other disciplines. Now, it appears, such knowledge and experience will be necessary in order to succeed technically as well.

References


---

i Note: “he” is used to indicate “an individual” in this work; please read he/she, his/her, etc. throughout, as appropriate.

ii The KAI must be administered by a certificated practitioner; a worldwide network of these practitioners has been established through Kirtan’s Occupational Research Centre in the United Kingdom (see [www.kaicentre.com](http://www.kaicentre.com) for more information).