# Reconciling Perspectives:

## How People Manage the Process of Software Development

Steve Adolph
Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada
stevea@ece.ubc.ca

Philippe Kruchten
Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada
pbk@ece.ubc.ca

***Abstract*—Social factors are the significant cost drivers for software development and in this field study we develop a grounded theory of how people manage the process of software development. We discovered the main concern of those involved in the process of software development is getting the job done and to get the job done, people engage in a constant process of Reconciling Perspectives. Reconciling Perspectives is a four-stage process to drive the convergence of the different points of view or perspectives individuals have of a software project. What this theory reveals is the importance of individuals' ability to reach out and engage in negotiations to the success of a software project**.

***Keywords: software development process, software team, shared mental model, mental model convergence, software method, Scrum, grounded theory***

## I. INTRODUCTION

Numerous studies demonstrate that individual ability and team social factors are significant cost drivers for a software engineering projects, often swamping all other factors [1-5]. Caper Jones' data show that high management and staff experience contribute 120% to productivity while effective methods/process contribute only 35% [6]. Cost drivers associated with personal factors from COCOMO model "reflect the strong influence of personal capability on software productivity" [7]. Boehm concluded:

> "*Personnel attributes and human relations activities provide by far the largest source of opportunity for improving software productivity*"

If social factors are the biggest cost drivers and explain variances in the productivity of software development teams, then research that helps us identify and understand social processes in software engineering should yield significant benefit to the industry. This paper reports the results of our research to understand these cost drivers by constructing a grounded theory of how people actually manage the process of software development.

This study makes two contributions, the first is our findings, and the second is an in-depth analysis of our experience using grounded theory [8]. Grounded theories require voluminous description to explain the categories and the relationships between the categories. Given the conference paper page limit we can only describe the core concept and provide only some insight into a few closely related concepts. In this paper the *conceptual elements of the theory* are highlighted with an underlined italicized font. For example, the conceptual element Bunkering is highlighted as *Bunkering*.

Section 2 is an outline of our study and a brief description of our use of grounded theory. Section 3 is our theory and section 4 is a follow-up discussion, comparing our results to known theories. Section 5 summarizes our results and recommendations.

## II. OUR STUDY

### *A. Motivation*

Methodologists argue the benefits of following a software development methodology and there are studies that demonstrate a positive correlation between software development method adoption and team effectiveness in terms of product quality and productivity [9-11]. However, other industry data demonstrate software development method's effect on productivity is limited [6]. Furthermore, the contribution of software methods to team effectiveness is frequently questioned [12-14]. Our own anecdotal experience based on 25 plus years of personal industry experience suggests very low rates of methodology adoption, especially at the team and individual level.

What is going on here? Is it possible low rates of software methods adoption are a strong indicator that software practitioners do not believe their needs are addressed by software methods? While there are anecdotes about methodology usage or lack of usage, there is a gap in our empirical knowledge about the interface between software developers and methods. An empirical study, with a

product of a substantive theory that explains how people manage the software development process, could diminish that gap. We chose a qualitative approach for our research because we were interested in understanding the issues that are relevant to software engineers. We chose Grounded Theory as our method because we were interested in generating theory.

### *B. Grounded Theory*

Grounded theory is a qualitative research method that discovers theory from data and is useful for discovering behavioral patterns that shape social processes as people interact together in groups [15] . The goal of this grounded theory is to understand the action in a substantive area from the point of view of the actors involved [16]. Grounded Theory is best for answering questions of the form "what is going on here?" Schreiber and Noerager [17] argued it is useful for when we want to learn how people manage their lives in the context of a problematic situation and about the process of how people understand and deal with what is happening to them. An account of a phenomenon is developed that identifies the major categories, their relationships, and the context and process, thus providing a theory of the phenomenon that is much more than a descriptive account [18].

Co-discoverers Barney Glaser and Anselm Strauss, called the method "grounded" because a theory is systematically generated from a broad array of data through a rigorous process of constant comparison. Grounded theory is different from the dominant logico-deductive methods of inquiry because rather than develop a theory and then systematically seek out evidence to verify it, grounded theory researchers gather data and systematically generate a mid-level substantive theory derived directly from the data. We described our application of and experience with classical grounded theory in [8].

### *C. Grounded Theory and Literature Reviews*

Glaser strongly encouraged grounded theorists to conduct their literature review after their theory has emerged to avoid undue influence by extant theory on the emerging theory [19]. This should not be taken as ban on conducting a literature review prior to study and we conducted our literature review in two phases: the first phase framed the problem we were exploring and led to formation of our research question. Some of our prior research into software method adoption and use is cited in section II.A.

We conducted the second phase of our literature review after our grounded theory of *Reconciling Perspectives* emerged and compared extant theory to our grounded theory.

### *D. Research Question*

Grounded Theory is a method that permits researchers to discover an actual problem that exists for the participants in a substantive area rather than what professional researchers may believe is the participants' problem. In a Grounded Theory study, the researcher works with a general area of interest rather than with a specific problem [20]. The opportunity to discover what was truly on the mind of our participants appealed to us.

For our study, we worked with a broadly defined problem statement of "*how do people manage the process of software development?*" While we have some preconceived ideas about potential problems, (e.g., coordination, communication, adaptation), we left our problem statement open so that we could determine if any of our preconceived problems mattered to software developers. Had we gone into the field with a rigorously defined research problem we likely would never have learned the process of *Reconciling Perspectives* was a way people *Got the Job Done*.

### *E. Data Collection and Analysis*

We engaged in a long-term field study collecting and analyzing data over the period of 12 months from February 2009 through January 2010. We were able to collect field data from three different software development organizations: an onsite customer support field office with 7 staff, a small product company with approximately 150 employees, and a software research and development center for large multi-national software product vendor with over 1000 employees locally. Subjects varied in age and experience from new hires less than two years out of school to experienced individuals with 25 years experience. Most subjects were directly involved with the creation and delivery of software, and had job titles such as project manager, software development manager, business analyst, quality assurance engineer, and developer.

We collected data using both semi-structured interviews and participant observation. Participant observation was a mainstream data collection method for us and not just a supplement to our interviews because it allowed us to observe what people did rather than just what they believed. As the study progressed, observed events generated interview questions. In all we conducted some 20 interviews and spent some 42 days over the year observing participants at work. Interviews were digitally recorded and then transcribed, and observation field notes were written in lab notebooks.

We followed grounded theory practice of coding and analyzing our data as it was collected. It often took two to three days of effort to code and analyze the data. Insights we had while coding the data were captured in memos. As

new data was collected it was compared to existing concepts and what we learned from the analysis we used to adapt our interview and observation protocols. We used Atlassian *Confluence* (a Wiki) to manage the data, and memos.

### *F. Validity*

We can frame the issue of validity and rigor in qualitative research with two questions:

1) Is the story expressed in the theory a true story and not a fabrication?
2) Is the theory a good story, one people will find interesting, adds to the known body knowledge and is useful for informing policy?

Lincoln and Guba [21] answer these questions by defining the frequently cited criteria of trustworthiness for naturalistic inquiry, in terms of confirmability, dependability/auditability, credibility, and transferability. To satisfy the requirements for confirmability, dependability, and auditability, we maintained logs of our interviews and observations along with a large bank of both theory and process memos. For confirmability this theory was reviewed with the participants to verify they agree with our interpretation of the data. Finally the theory was presented to those who were not part of this study to verify its transferability.

## III. RESULTS

### *A. The Software Development Eco-System*

Software development occurs in the context of a greater organizational eco-system where an *Acquirer* has a *Job* they wish done and a *Supplier* who has at least some of the ability to perform that *Job* creates *Work Product*s that realize the *Job*. We borrow the term eco-system from Jim Highsmith [22] because it accurately describes the software development context in the organizations we observed. Software development takes place in a confusing and chaotic mix of formal and informal software methods, corporate policies, competing interests, personal agendas, personality types and formal and informal relationships. In ecological terms, there is a great deal of bio-diversity and organizations are anything but a monoculture. Figure 1 gives an overview of the elements of our theory.

### *B. Getting the Job Done*

The central concern of everyone interviewed and observed during this study is *Getting the Job Done*, that is, delivering the best *Work Product*s they knew how create. A major source of frustration were the impediments to *Getting the Job Done*.

*"Um for myself particularly the concern is ah being blocked to something. I can't do, I can't achieve by myself. I know that supplying this would be sort of would hamper progress (unclear). Sometimes maybe to the point where the project is completely halted for a long period of time until it's resolved. And so I think that's my main concern is um is really just not being able to do the work that ah that needs to be done"* site 1, subject 3 lines 205:209

*Getting the Job Done* means creating working software that:

- Satisfies the software team member's need to see the end and achieve a feeling of accomplishment
- Appeases the customer or even make the customer happy,
- Satisfies the team member's desire to minimize technical debt

### *C. Perspectives*

Throughout this study, and anecdotally throughout the authors' industrial careers, we discovered a frequent and difficult problem in software development is "getting everyone on the same page" Everyone seems to have a different point of view or *Perspective* about what the *Job* is and the process for delivering the *Job*. Much like how members of a choir must all sing from the same sheet of music, everyone participating in the software project must also see the *Job* the same way for the *Job* to get done (*Getting the Job Done*). The inability to get everyone on the same page is a significant impediment to *Getting the Job Done*.

*"And the biggest problem you have is—I can think of one case recently where we had the team lead here and—one of the managers here and a team lead at the (unclear) team kind of discussing this bug and the three of us were discussing it in lieu for a week only to realize that all of us had completely different interpretations of it….So we weren't even on the same page on it. So it leaves more room for miscommunication which is—not a good problem in software*." Site 2, subject 3, lines 260:268

A *Perspective* is an individual's point of view and the term perspective emerged from our data. What we are referring to as a perspective seems functionally comparable to what much of the psychology literature refers to as a mental model [23, 24]. Shared mental models are ''knowledge structures held by members of a team that enable them to form accurate explanations and expectations for the task, and in turn, to coordinate their actions and adapt their behavior to demands of [their unique domain]'' [25]. Further Firore and Salas assert, "*Members of effective teams possessed a shared set of knowledge that facilitates their interactions. In particular highly effective teams must hold compatible knowledge structures about a variety of facets of team tasks*". Cronin described

mismatches between mental models as *"…representational gaps, inconsistencies between individual's definition of the team's problem, limit both of these processes making it more difficult for team members to integrate one another's information and increasing the likelihood of conflic*t" [26]

### *D. Reconciling Perspectives*

*Reconciling Perspectives* is the basic social structural process people managing the process of software development use to resolve *Perspective Mismatches* and remove impediments created by *Perspective Mismatches* to *Getting the Job Done*. This is an iterative and recursive process. Iterative because gaps in knowledge may require repeated applications of the process to fully reconcile the *Perspective* and recursive because reconciling a *Perspective Mismatch* may lead to the discovery of another mismatch that must be resolved before the original *Perspective Mismatch* can be resolved.

| Reconciling Perspectives | | | |
|---|---|---|---|
| Converging | | Validating | |
| Reaching Out | Negotiating Consensus *Translation Broadening Scouting* | Bunkering *Cutoff Exposure* | Accepting |

Figure 1 –Key elements of our emergent theory

In *Reconciling Perspectives* there are two clear stages that account for variations in behavior, *Converging* and *Validating*. During *Converging* individuals recognize the existence of a *Perspective Mismatch* between themselves and others that is impeding *Getting the Job Done* and *Reach Out* to engage others to create a consensual shared *Perspective* or *Consensual Agreement* to resolve the *Perspective Mismatch* by *Negotiating Consensus*. While the *Consensual Agreement* removes an impediment to moving forward it is only tentative because while a *Consensual Agreement* may have been reached, the parties, are only assuming they have a better understanding of how the others see the problem and its solution. The process is not complete until it is validated (*Validating*) by creating the *Work Product* during *Bunkering* and having it accepted (*Accepting*) by the *Acquirer*.

#### *1) Dimensions*

We observed the *Scale*, *Formality*, and, *Cycle Time* of this process varies from the large scale where acquiring organizations engaged with supplier organizations through formal mandated and structured meetings, to the small scale and ad hoc that may begin with one individual stepping up to another and asking, "can we talk?" The process cycle time may be short with only a few hours elapsing between *Reaching Out* and *Accepting*, or may require months until a *Work Product* is accepted.

#### *2) Conditions*

The success of the process is strongly related to the personal traits of those participating in the process of software development. In all the projects observed during this study, a common success driver was the presence of an individual or group of individuals who had the *Personal Strength* to initiate this process. While *Personal Strength* is partly an innate trait of the individual such as being an introvert or extrovert, it is also related to their level of *Experience*. Experience individuals tended to have developed many *Communications Channels* they utilize to help them detect when something is amiss. One characteristic of more experienced individuals are the informal social networks they have created to check in with.

### *E. Converging*

*Converging* is the stage where individuals recognize a *Perspective Mismatch* is impeding *Getting the Job Done* and reaches out (*Reaching Out*) to engage others in negotiations (*Negotiating Consensus*) to converge their *Perspectives* sufficiently and create a *Consensual Agreement.* The *Consensual Agreement* removes the impediment to *Getting the Job Done*. The *Consensual Agreement* is only a hypothesis because it is reached when individuals conversationally agree they are on the same page. The use of facilitation tools such as whiteboard sketches, simulations or code walkthroughs, were employed frequently during this stage. While the *Converging* stage ends when every one agrees that they understand what the *Job* is, the process is not complete until the consensus is validated (*Validating*) by the delivery of a *Work Product*. It's a bit like the old sage "the proof is in the pudding"

What we observed and referred to as *Converging* is somewhat similar to what Sara McComb [27] describes as "mental model convergence," where team members' mental models evolve into a shared mental model, through their interactions and observations of each other.

#### *1) Reaching Out*

*Reconciling Perspectives* begins when individuals believe *Getting the Job Done* is impeded by a *Perspective Mismatch* and *Reaches Out* and invites others to engage with them to resolve the mismatch. *Perspective Mismatches* are discovered when individuals critically reflect on what they believe they are hearing in communications with others (verbal or written) and realize their understanding or expectation of the *Job* does not match the content of those communications. *Reaching*

*Out* may occur in an ad hoc manner when an individual takes the initiative to question something for them which does not sound right.

*"Mostly because I find any issue especially let's say a bug fix which doesn't look like it's super huge—when you need you know more that you know 2 days worth of email discussion about it—something's off. Either the thing is way bigger and actually requires refactoring or you're not understanding one another"* Site 2, subject 3, lines 270:273

It was not enough for an individual to recognize the existence of *Perspective Mismatch* because an individual could simply recognize there was a problem between how they saw the *Job* and how others saw the *Job* and yet not take action. Once the mismatch is detected the individual must *Reach Out* to engage the other parties if the process of *Reconciling Perspectives* is to start.

*Reaching Out* transitions to *Negotiating Consensus* when the other parties agree to participate in the dialog. This may be as simple as agreeing to the question "can I talk to you about this" to calling a formal meeting at a process gate.

*2) Negotiating Consensus*

Negotiation is the activity that stands out in software development. It seems everyone is always negotiating: stakeholders and developers negotiate features and budget, development teams negotiate resources, team members negotiate task assignments, and developers negotiate APIs and allocation of behaviors to modules. One interesting observation made during this study, was no one definitively distinguished between the activities of planning, requirements gathering, analysis, and design during the interviews. All were described as negotiation.

*Negotiating Consensus* is a dialog between *Acquirers* and *Suppliers* intended to converge their expectations for what the *Job* is about and create a *Consensual Agreement* that will enable them to remove impediments to *Getting the Job Done*. The *Consensual Agreement* sets the *Acquirer's* expectations for what *Work Products* they can expect to receive, and sets the *Supplier's* expectations for what *Work Products* they are committing to deliver. Many colloquially view negotiation as process for making trade-offs and while this is part of negotiation, it is also a discovery process for both *Acquirer* and *Supplier* enabling them to discover better informed trade-offs.

Negotiation occurs at all levels and on all scales, from CTOs, analysts, and product managers negotiating product features with customers, to individual developers negotiating an interface. Software development is an ongoing multi-level negotiation. *Negotiating Consensus* may be a short, single phase, informal, simple agreement characterized by this stylized conversation heard over and over again between software developers:

| | |
|---|---|
| Dev 1 (Acquirer): | "Hey can we talk?" |
| Dev 2 (Supplier): | "Sure" |
| Dev 1 (Acquirer): | "When we talked about this interface before I assumed you were giving the context, now I find out that's not the case. Can you pass me the handle the context?" |
| Dev 2 (Supplier): | "Sure" |

Or *Negotiating Consensus* may be a long, protracted and multi-phased, cycling repeatedly through many cycles, of negotiating a preliminary *Consensual Agreement* and then *Validating* the agreement to generate the information necessary to improve future decisions.

We observed three broad strategies for *Negotiating Consensus* based on what we observed as the general approach people took to resolving their *Perspective Mismatch*: *Translation*, *Broadening*, and *Scouting*. *Translation* is a straightforward situation where *Perspectives* are well aligned but the *Acquirer* and *Supplier* are simply using different terminology to express themselves. It was easy to overhear these conversation conclude successfully with "…oh now I see what you mean!"

"*when we talk story points to our customers, they don't like it because it's um abstract and amorphous to them. They don't ah – what they want is predictability. So, for them you know they don't care whether we're saying, "Well, this is a 200 story point project." What they care is um how much does the story point cost and how long does it take to build it? And how much stuff can I get for it? So, that's the way we present it to the customer*" Site 1, subject 1, lines 92:100

The second strategy for *Negotiating Consensus* is *Broadening* where an individual attempts to broaden their understanding of the *Job* by trying to understand the other's point of view. A common source of impediment was software developers viewing the *Job* from a technical perspective and not fully appreciating the business impact of their decisions or *Acquirers* not understanding the technical consequences of their demands – for example understanding how a request may increase technical debt. The ability or desire to *Broaden* their *Perspectives* certainly appeared to be a strategy commonly employed by more experience individuals.

The final strategy is *Scouting*. In both *Translation* and *Broadening*, one or both parties had sufficient knowledge

to understand the *Job*, but could not express it in a way the other understood or cared about. *Scouting* becomes the *Negotiating Consensus* strategy when neither the *Acquirer* nor the *Supplier* had sufficient knowledge to explain their view to the other. The negotiation becomes blocked because neither side has sufficient information to answer the other's questions. The purpose of *Scouting* is to discover the information necessary to help the *Acquirer* and *Supplier* converge their *Perspectives* and make a decision that removes the impediment to *Getting the Job Done*.

A common source of failure for projects was *Scouting Blowout* where *Scouting* continued with out end and *Consensual Agreement* was not reached and no decision is explicitly made on how to remove the impediment. The *Supplier* either continued to make perceived progress to *Getting the Job Done* by working with their *Perspective* of the *Job* or remained stuck in *Scouting* and failed to deliver the *Job*. Both situations lead to unwanted *Surprise*. During a retrospective of a near failed project, we observed exemplars of these situations where a project appeared to have "*wandered through the deser*t" for six months without creating a true *Consensual Agreement*.

The ability to *Compromise* or the cultivation of a *Flexible Response* facilitates the *Negotiating Consensus* process. If the opportunity for trade-offs is constrained then *Negotiating Consensus* degenerates into a situation where either the *Acquirer* or *Supplier* simply imposes an agreement on the other. In these situations the power of *Consensual Agreement* to resolve the *Perspective Mismatch* is weakened and even nullified because participants may acquiesce to the terms imposed by the other simply to move ahead in some assumed direction.

*Converging* transitions to *Validating* when a *Consensual Agreement* on what the *Job* is and what *Work Products* are needed is created. The *Consensual Agreement* is based on a shared assumption that all participants have a working consensual shared *Perspective* of the *Job*. The development process can now move ahead.

*F. Validating*

The *Consensual Agreement* made during *Converging* is only a hypothesis that must be tested and is only validated when the Suppliers create a *Work Product* that is accepted by *Acquirer*. There are two stages to *Validating*, *Bunkering* and *Accepting*. *Bunkering*, is a quiet stage during which the *Suppliers* create *Work Products* that satisfy negotiated *Job* requirements. What strongly characterizes this stage is the drop in conversation between *Acquirer* and *Supplier*. *Suppliers* preferred minimal external interference while they focus on creating the *Work Product*s. *Accepting* is the final stage of the process where the *Suppliers* present the *Work Products* to the *Acquirers* as evidence they worked from the same *Perspective*.

*1) Bunkering*

Whereas during *Converging* individuals actively engage in conversations during *Reaching Out* and *Negotiating Consensus*, *Bunkering* is a "quiet" stage. Using the knowledge gained from the *Consensual Agreement* created during *Converging*, *Suppliers* create the *Work Products* they believe will satisfy the *Acquirer's Job* requirements. Further conversation is often seen at best as distracting during this stage and at worse, as interference. *Bunkering* is characterized by a distinct drop in conversation levels and interactions between *Acquirers* and *Suppliers*. This enforced "quiet" is built into software methods like Scrum which all teams participating in this study claimed they were using. However, it also seems part of the culture regardless of the software method used. Once a *Consensual Agreement* is reached, construction of a *Work Product* should proceed relatively undisturbed.

*Bunkering* can put the *Getting the Job Done* at risk because of the potential to suppress *Reaching Out*. During *Bunkering* an individual may encounter an inconsistency between how they understood the *Consensual Agreement* and the results they are getting while creating the *Job Work Product*s and may be reluctant to start *Reaching Out* while in the *Bunkering* stage. Some software methods like Scrum attempt to mitigate this situation by mandating a daily check-in to verify no one is blocked by an impediment. This is an example where the software method helps create the opportunity for initiating the *Reconciling Perspectives* process.

*"So yes we've definitely had the time boxes blown out—by people just digging into things and then not making the progress we expect and sometimes we don't spot that in time."* Site 2, subject 4, lines 56:58

We observed two *Bunkering* extremes: *Cut-Off* and *Exposure*. During *Bunkering* people could completely cut themselves off from all conversations and other sources of information that may reveal to them a *Perspective Mismatch*. This is sometimes referred to as "going dark" [28]. At the other extreme is *Exposure*, where an individual or group listens to all conversations and therefore cannot focus on *Getting the Job Done*.

*Bunkering* transitioned to *Accepting* when *Suppliers* presented the job *Work Products* to the *Acquirers* for their acceptance.

*2) Accepting*

During *Accepting*, the *Supplier* presents their work to the *Acquirer* to solicit their approval. *Accepting* may be as simple as the *Acquirer* informally stating, "Ok it's good"

or may be a formal ceremony for transitioning and *Accepting* a *Work Product*. Only when the *Work Product* is accepted is there objective evidence all involved were likely operating from the same *Perspective*.

A negative result during *Accepting* is *Surprise* where the *Work Product* delivered by the *Supplier* does not meet with the *Acquirer's* expectations. We observed many situations in which lengthy *Bunkering* stages directly influenced *Surprise*. Long cycle *Reconciling Perspectives* processes therefore have a greater potential to result in *Surprise* or larger *Surprises* than shorter cycle *Reconciling Perspectives* processes. An example of larger *Surprise* occurred at one site where *Supplier* and *Acquirer* teams effectively cut each other off for over six months. When the *Work Products* were presented there were near catastrophic *Surprises*.

## IV. Discussion

### A. Mental Model Convergence, Knowledge Communities, Knowledge Creation, and Reconciling Perspectives

Elements of *Reconciling Perspectives* describe a process similar to that of Mental Model Convergence [27] where close interactions between individual team members facilitates the convergence of their disparate mental models to form a common understanding of a shared task. Convergence of mental models explains how individuals socialize themselves into teams and during our study we observed situations that suggested this process occurring within teams. However, more of our data described interactions individuals had with other individuals from outside their team and from other functional groups. Much of our interview data describes interactions between software developers and analysts and project managers.

Boland and Tenkasi characterized knowledge intensive organizations as composed of knowledge communities with highly specialized technologies and knowledge domains. Communication within a community that strengthens and develops localized knowledge is a "perspective making" process and communication that improves a communities ability takes the knowledge of other communities into account is "perspective taking" [29]. If we characterize software developers, analysts, and project managers as belonging to different knowledge communities then much of data we collected on their interactions could be characterized in Boland's and Tenkasi's terms as "perspective taking".

Perspective making and perspective taking are knowledge creation processes and knowledge based firms realize competitive advantage by individuals' ability to share and utilize their distinct knowledge. *Reconciling Perspectives* then becomes a knowledge creation process where individuals are Perspective Taking during the *Negotiating Consensus* stage. The negotiating strategy of *Scouting* clearly adds to the organization's body of knowledge. The negotiating strategies of *Translating* and *Broadening* also create organizational knowledge by making knowledge that is known in one community accessible and useful to another.

A differentiating characteristic of *Reconciling Perspectives* is the process only starts if individuals recognize a *Perspective Mismatch* and are willing to do something about it. Organizational and personal barriers that prevent individuals from *Reaching Out*, impede this knowledge creation process [30].

### B. Bunkering and Cognitive Divergence

Levesque's et al [31] challenges the prediction that team members mental models converge over time. Their study discovered a decline in shared mental models over time which they related to a decrease in interaction. Lévesque suggested that increasing specialization among team members contributed to the decline in interaction.

The *Bunkering* phase of *Reconciling Perspectives* may also explain this decrease. In many situations once a *Consensual Agreement* had been reached, team members reduced their interactions with others to work undisturbed at *Getting the Job Done*. Levesque's findings suggest *Bunkering* may lead to cognitive divergence or *Perspective Mismatch* between team members. With long cycle time methods, this divergence can widen enough to risk *Surprise* during *Accepting*.

### C. Agile Methods and Reconciling Perspectives

Scrum is an example of a management method that explicitly attempts to facilitate the *Reconciling Perspective* process. Scrum's prescribed meetings, sprint planning, daily stand-ups, sprint review, and sprint retrospective are all opportunities for those participating in a project to check-in and if necessary start *Reaching Out*. The Scrum planning meeting is an opportunity to *Negotiate Consensus* if a *Perspective Mismatch* is detected between the Product Owner (*Acquirer*) and delivery team (*Supplier*). Once the sprint begins, the delivery team focuses on *Getting the Job Done*, without being *Exposed* to interfering change (*Bunkering*). Team members are encouraged to regularly check-in with the team to prevent the *Bunkering* stage from disconnecting them from others (*Cut-Off*). When the *Work Products* for a story (*Job*) are complete, it is presented to the Product Owner for acceptance (*Accepting)*. Scrum's short cycle time of two to four weeks mitigates *Surprise* during *Validating*.

We can use *Reconciling Perspectives* to explain failed Scrum projects. During this study we heard many stories of failed Scrum projects where the common thread was teams cut themselves off and did not did not detect the *Perspective Mismatches*, or did not start the *Reaching Out* process. They went "through the motions" but did not follow the intent of Scrum. A frequent condition cited for an effective Scrum team is a "strong" and available Product Owner. Our observations support this condition, but we also observed situations where other strong team members stepped into a vacuum created by a disengaged Product Owner. We would broaden this Scrum success condition to say a necessary condition for success with Scrum is at least one strong individual who is engaged such that they can readily detect *Perspective Mismatches* and have the strength to *Reach Out* when they do.

### D. Enhancing Software Team Performance

Shared mental model theory predicts higher team performance when the members can converge their mental models. This suggests enhancing the ability of those engaged in software development to reach out (*Reaching Out)* and negotiate (*Negotiating Consensus)* when they encounter impediments will enhance software team performance. While individual traits such as extroversion and introversion will pre-dispose individual behavior, there is precedence suggesting training people how to reach out and how to negotiate can improve software team performance [32]. The aviation industry is an example of one industry where team training enhanced safe operation of aircraft [33].

## V. Summary

Our study explains how people manage the process of software developing using the process of *Reconciling Perspectives* to remove impediments created by *Perspective Mismatches* to *Getting the Job Done.* Learning the central concern of our subjects and developing a theory that explains how they resolve that concern gives us a key to explore extant theory to find solutions to software engineering problems. The grounded theory of *Reconciling Perspectives* shares elements with theories of mental model convergence and organizational knowledge creation.

The theory of *Reconciling Perspectives* holds few surprises for any reader of popular organizational theorists. Grounded theory is not about creating surprising new theories; rather it is about understanding what is happening to those experiencing a phenomenon and understanding it from their point of view. What did take us a little by surprise is most study participants framed conversations associated with scheduling, requirements management, and design all as negotiations. This suggests negotiation is as a dominant activity in the minds of those engaged in the process of software development.

From our observations during this study we can draw several conclusions and make recommendations:

1) A necessary condition for the success of a software project is at least one individual who is sufficiently engaged they can detect *Perspective Mismatches*, and has the personal strength to reach out and initiate the *Reconciling Perspectives* process.
2) The health of a software project and its probability of success can be measured by the level of conversations where people are *Reaching Out* and *Negotiating Consensus*.
3) Our results suggest team training and development of principled negotiation skills for software developers has good potential for improving team performance.
4) Finally this study adds weight to the argument that qualitative research methods are effective for creating software engineering knowledge. We demonstrated the utility of Grounded Theory as a software engineering research method, enabling us to see what is important to those whose lives we are trying to improve

## VI. Acknowledgements

The authors wish to express their thanks to Agile Alliance, the Scrum Alliance, and the Eclipse Foundation for their support of this research. Our further thanks to our reviewers, and especially the anonymous reviewer who introduced us to Boland's and Tenkasi's paper.

## References

[1] B. W. Boehm, "Software Engineering Economics," *IEEE Transaction on Software Engineering,* vol. 10, pp. 4-21, January 1984.

[2] A. Cockburn, "Agile Software Development Joins the "Would-Be" Crowd," *Cutter IT Journal,* vol. 15, January 2002.

[3] W. Curtis, H. Krasner, V. Shen, and N. Iscoe, "On building software process models under the lamppost," presented at the Proceedings of the 9th international conference on Software Engineering, Monterey, California, United States, 1987.

[4] T. Lister and T. DeMarco, *Peopleware: Productive Projects and Teams*. New York: Dorset House, 1987.

[5] S. Sawyer and P. J. Guinan, "Software development: processes and performance," *IBM Syst. J.,* vol. 37, pp. 552-569, 1998.

[6] C. Jones, *Software Assessments, Benchmarks, and Best Practices*. Boston: Addison-Wesley, 2000.

[7] B. W. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. W. Selby, "Cost models for future software life cycle processes: COCOMO 2.0 " *Annals of Software Engineering,* vol. 1, pp. 57-94, 1995.

[8] S. Adolph, W. Hall, and P. Kruchten, "Using grounded theory to study the experience of software development," *Empirical Software Engineering,* pp. 1-27, DOI: 10.1007/s10664-010-9152-6 2011.

[9] M. Diaz and J. Sligo, "How software process improvement helped Motorola," *Software, IEEE,* vol. 14, pp. 75-81, 1997.

[10] D. E. Harter, M. S. Krishnan, and S. A. Slaughter, "Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development," *Management Science*, vol. 46, pp. 451-466, 2000.

[11] M. S. Krishnan, C. H. Kriebel, S. Kekre, and T. Mukhopadhyay, "An Empirical Analysis of Productivity and Quality in Software Products," *Management Science*, vol. 46, pp. 745-759, 2000.

[12] A. Cockburn, "People and Methodologies in Software Development," Doctor Philosophiae, Faculty of Mathematics and Natural Sciences, University of Oslo, Oslo, 2003.

[13] T. Dyba, N. B. Moe, and E. Arisholm, "Measuring software methodology usage: challenges of conceptualization and operationalization," *Proc. of International Symposium on Empirical Software Engineering*, 2005, p. 11

[14] B. Fitzgerald, "An empirical investigation into the adoption of systems development methodologies," *Information & Management*, vol. 34, pp. 317-328, 1998.

[15] B. G. Glaser and A. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research.* Chicago, Illinois: Aldine, 1967.

[16] B. G. Glaser, *Doing Grounded Theory: Issues and Discussions.* Mill Valley, California: Sociology Press, 1998.

[17] R. S. Schreiber and P. N. Stern, *Using Grounded Theory in Nursing*. New York: Springer Publishing Company, 2001.

[18] P. H. Becker, "Common Pitfalls in Published Grounded Theory Research," *Qual Health Res*, vol. 3, pp. 254-260, May 1, 1993.

[19] B. G. Glaser, *Theoretical Sensitivity*. Mill Valley, California: Sociology Press, 1978.

[20] A. M. McCallin, "Designing a grounded theory study: some practicalities," *Nursing in Critical Care*, vol. 8, pp. 203-208, 2003.

[21] Y. S. Lincoln and E. G. Guba, *Naturalistic Inquiry*. Newbury Park: Sage, 1985.

[22] J. Highsmith, *Agile Software Development Ecosystems*, Boston: Addison Wesley, 2002.

[23] W. B. Rouse, J. A. Cannon-Bowers, and E. Salas, "The role of mental models in team performance in complex systems," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 22, pp. 1296-1308, 1992.

[24] P. N. Johnson-Laird, *Mental Models: Towards a Cognitive Science of Language, Inference, and Conciousness*. Cambridge: Cambridge University Press, 1983.

[25] J. A. Cannon-Bowers, E. Salas, and S. Converse, "Shared Mental Models in Expert Team Decision Making," in *Current Issues in Individual and Group Decision Making*, J. Castellan, Ed., Mawah, NJ: Lawrence Erlbaum Associates, 1993.

[26] M. Cronin and L. Weingart, "Representational Gaps, Information Processing, and Conflict in Functionally Diverse Teams," *Academy of Management Review*, vol. 32(3) pp. 761-773, 2007.

[27] S. McComb, "Mental model convergence: The shift from being an individual to being a team member," *Multi-Level Issues in Organizations and Time*, vol. 6, 2007,

[28] J. McCarthy and D. Gilbert, *Dynamics of Software Development*, Redmond: Microsoft Press, 1995.

[29] R. J. Boland, Jr. and R. V. Tenkasi, "Perspective Making and Perspective Taking in Communities of Knowing," *Organization Science*, vol. 6, pp. 350-372, 1995.

[30] D. Dougherty, "Interpretive Barriers to Successful Product Innovation in Large Firms," *Organization Science*, vol. 3, pp. 179-202, 1992.

[31] L. L. Levesque, J. M. Wilson, and D. R. Wholey, "Cognitive divergence and shared mental models in software development project teams," *Journal of Organizational Behavior*, vol. 22, pp. 135-144, 2001.

[32] J. E. Mathieu, T. S. Heffner, G. F. Goodwin, E. Salas, and J. A. Cannon-Bowers, "The Influence of Shared Mental Models on Team Process and Performance," *Journal of Applied Psychology*, vol. 85, pp. 273-283, 2000.

[33] E. L. Wiener, B. G. Kanki, and R. L. Helmreich, Eds., *Cockpit Resource Management*. San Diego California: Academic Press, 1993