The Journal of Systems and Software 85 (2012) 1269–1286

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Reconciling perspectives: A grounded theory of how people manage the process of software development

Steve Adolph [a,*], Philippe Kruchten [a], Wendy Hall [b]

[a] Electrical and Computer Engineering, University of British Columbia, Vancouver, Canada
[b] School of Nursing, University of British Columbia, Vancouver, Canada

ARTICLE INFO

Article history:
Received 1 September 2011
Received in revised form 27 January 2012
Accepted 31 January 2012
Available online 8 February 2012

Keywords:
Software engineering
Software team
Agile manifesto agile software development
Scrum
Shared mental model
Grounded theory

ABSTRACT

Social factors are significant cost drivers for the process of software development. In this field study we generate a grounded theory of how people manage the process of software development. The main concern of engineers involved in the process of software development is getting the job done. To get the job done, people engage in a four-stage process of Reconciling Perspectives. Reconciling Perspectives represents an attempt to converge individuals' points of view or perspectives about a software project. The process emphasizes the importance of individuals' abilities to both reach out and engage in negotiations and create shelter from environmental noise to bring a software project to fruition.

1. Introduction

Software development is a risky and expensive proposition. Talking with software developers quickly illustrates the quagmire that software development can become. Late delivery, defective products, cost overruns, and frustrated staff and stake holders are some of the debris resulting from a failed or less than successful project. This is expensive debris considering that global software development is a 1.6 trillion US dollar industry (Bartels et al., 2006). Improving the productivity of software development teams and the quality of delivered software will result in significant economic returns.

Numerous studies have demonstrated that individual abilities and team social factors are significant cost drivers for software engineering projects, often swamping all other factors (Boehm, 1984; Cockburn, 2002; Cockburn and Highsmith, 2001; Curtis et al., 1987; Jones, 2000; Lister and DeMarco, 1987; Sawyer and Guinan, 1998). Caper Jones (2000) data demonstrate that high levels of management and staff experience contribute 120% to productivity while effective methods and processes contribute only 35%. Cost drivers associated with personal factors from the COCOMO model "reflect the strong influence of personal capability on software productivity" (Boehm et al., 1995, p. 86). Earlier, Boehm concluded:

> "Personnel attributes and human relations activities provide by far the largest source of opportunity for improving software productivity" (Boehm, 1984)

If social factors are the biggest cost drivers, and explain variance in the productivity of software development teams, research studies that help us identify and understand social processes in software engineering should yield significant benefit to the industry. In the past, most software engineering research has focused on tools and production methods, which have limited ability to account for and manage the variance in software projects (Glass et al., 2002; Sawyer and Guinan, 1998; Shaw, 2003; Sjoeberg et al., 2005; Zannier et al., 2006). Shaw's (2003) analysis of papers accepted by the International Conference on Software Engineering (ICSE) shows the majority of accepted papers to be

> "...reports [of] an improved method or means of developing software that is, of designing, implementing, evolving, maintaining, or otherwise operating on the software system itself. Papers addressing these questions dominate both the submitted and the accepted papers" (Shaw, 2003, p. 727).

In the past, procedures or techniques and tools and notation accounted for the majority of the accepted papers (69%), while less than 10% of the accepted papers described empirical or qualitative models. Another survey of the software engineering research

* Corresponding author.
E-mail addresses: stevea@ece.ubc.ca, steve@wsaconsulting.com (S. Adolph), pbk@ece.ubc.ca (P. Kruchten), Wendy.Hall@nursing.ubc.ca (W. Hall).

0164-1212/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2012.01.059

literature (Glass et al., 2002) reported a very small percentage (in many cases less than 1%) of research papers explored organizational issues or employed research methods useful in understanding social behavior.

The software engineering research agenda is changing, with more researchers investigating the influence that personal attributes and human relationships have on software projects (Dittrich et al., 2007). The Agile Manifesto and the Agile software development movement placed a spotlight on the importance of social interactions with the first article of the Manifesto for Agile Software Development:

"Individuals and interactions over processes and tools"

The emphasis on individuals and interactions motivated research into social interaction (Chong, 2005; Hoda et al., 2010; Moe et al., 2008; Robinson and Sharp, 2005; Whitworth and Biddle, 2007).

However, we believe the state of software engineering research can still be summed up in Curtis et al.'s (1987) retelling of the 13th century story of a man who, after losing his keys, crosses the street to search under a lamppost because the light is much better there. This should not come as a surprise to us because we are engineers and not sociologists. Engineers—especially at the research level—are engineers because they enjoy solving technical problems. Developing tools and methods is what we are educated to do, and what we enjoy doing; however, if we are to better understand the factors that have the greatest influence on software development performance, our studies must also examine social processes.

This paper is an extended revision of our Agile 2011 conference paper (Adolph and Kruchten, 2011). For this research project, we had the opportunity to study a number of Agile software development teams in order to understand the variability in the productivity of software development by creating a substantive theory of how people actually do, or "manage" as we have referred to it, the process of software development. Our study makes two contributions: the first is our findings, and the second is our experience using grounded theory (Glaser and Strauss, 1967). 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*. In Section 2 we present an outline of our study and a description of our use of the grounded theory method. In Section 3 we present our theory, and in Section 4 we provide a discussion of our results within the context of existing theories. Section 5 summarizes our results and recommendations.

2. Our study

2.1. Motivation

The study we conducted was aimed at understanding the social processes that influence software team performance and the effects of software methods on those processes. 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 (Diaz and Sligo, 1997; Harter et al., 2000). On the other hand, industry data demonstrate that the effect of software methods on software development productivity is limited (Jones, 2000; Sawyer and Guinan, 1998). Furthermore, the contribution of software methods to team effectiveness is frequently questioned (Cockburn, 2003; Dyba et al., 2005; Fitzgerald, 1998).

What is going on here? Is it possible that low rates of software methods adoption are a strong indicator that software practitioners do not believe their needs are being addressed by software methods? While there are anecdotes about methodology usage, or lack thereof, there is a gap in our empirical knowledge about the interface between software developers and methods. A grounded theory study, producing a substantive theory that explains how people manage the software development process, could diminish that gap. Anecdotes about software development, unlike theory, are not useful to guide policy. Questions arise, such as: "What are the needs of those involved in the process of software development?" 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 that explains how engineers manage the problem of quality software development.

2.2. Grounded theory

Grounded theory is a qualitative research method wherein researchers construct theory from data. It is useful for explaining behavioral patterns that shape social processes as people interact in groups (Glaser and Strauss, 1967). The goal of a grounded theory study is to understand the action in a substantive area from the point of view of the actors involved (Glaser, 1998). Grounded theory is best for answering questions of the form: "What is going on here?" Schreiber and Stern (2001) argued that grounded theory 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 explanation of a phenomenon is developed that identifies major categories, their relationships, context, and process; a grounded theory of a phenomenon is much more than a descriptive account (Becker, 1993).

Co-discoverers Barney Glaser and Anselm Strauss named 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 beginning with a theory and systematically seeking evidence to verify it, grounded theory researchers gather data and systematically generate a mid-range substantive theory grounded on that data.

Grounded theory is like Agile software development in the sense that it is deceptively simple conceptually, yet rigorous and disciplined in practice. Fig. 1 is reproduced from Adolph et al. (2011) and shows how we visualize the process of grounded theory.

(A) A researcher begins collecting data on a phenomenon of interest, and analyzes the data by searching for patterns of incidents to indicate concepts. Concepts are the building blocks of a grounded theory, and conceptualization is one of its distinguishing traits. While Glaser tends to use the terms "concept" and "category" interchangeably, we chose to think of categories as an aggregation of concepts.

(B) The theoretical properties of a category are developed by comparing incidents in incoming data with previous incidents in the same category. During the analysis, the "core category" is developed. The core category captures the most variation in the data (Glaser, 1978) while addressing the main concern of the study participants. The process of generating categories and their properties continues until the categories become "saturated"; that is, when further collection of data does not add any new properties to the existing categories, the incidents are said to be interchangeable.

(C) After saturation, the substantive theory is compared to theories described in the literature. The literature search is deferred to late in data collection to avoid forcing pre-conceived concepts on the substantive theory being developed from the data.

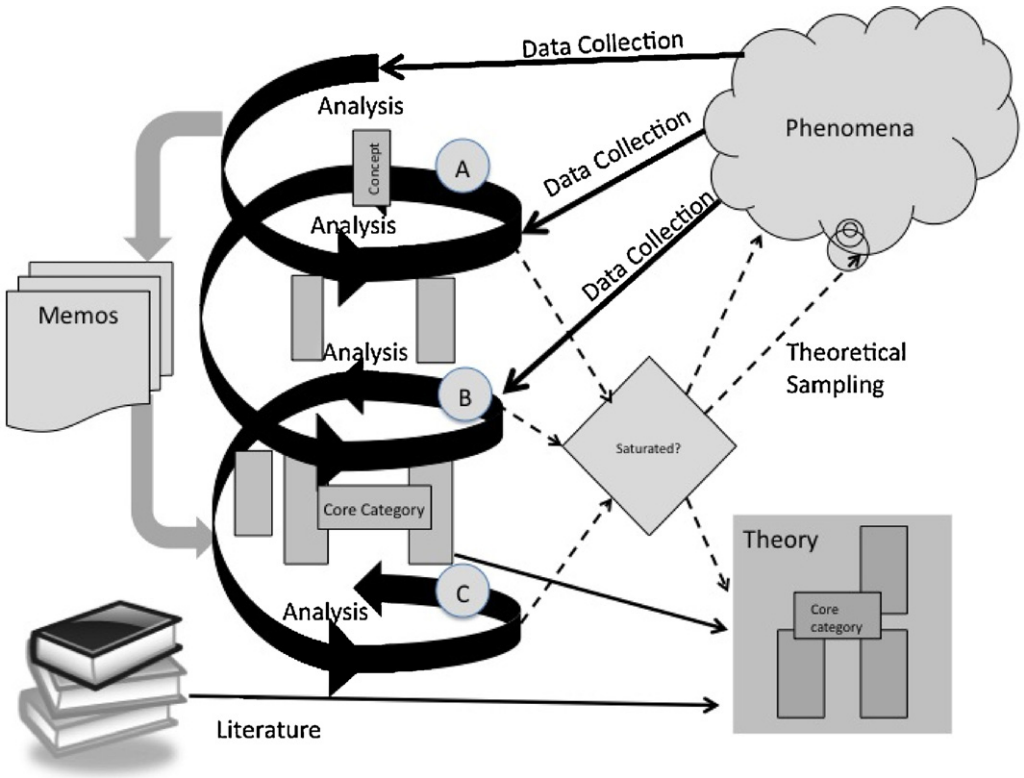

Fig. 1. The grounded theory method (Adolph et al., 2011).

(D) Throughout the process, the researcher writes memos capturing his or her thoughts and analytic processes; the memos support the emerging concepts, categories, and their relationships.

From this overview, one could regard producing a grounded theory as straightforward, mechanical, and rigorous: a process in which the researcher makes easy progress. We found it messy, tedious, and difficult. Generating a grounded theory required creativity and theoretical sensitivity on the part of the researchers to develop the theory. Theory development most certainly did not progress in a straightforward linear manner. As we describe in Section 2.5, it involved many false leaps, blind alleys, and dead ends. Nonetheless, it was an extremely rewarding experience that enabled us to explain the experience of software engineering from a very different perspective.

Hoda et al. (2010) and Whitworth and Biddle (2007) have demonstrated the utility of grounded theory for studying Agile software development. Hoda generated a theory for organizing self-organizing teams and, while Whitworth did not completely generate a theory, her observations were conceptualized and connected to Social Identity Theory. Section 2.5 provides some description of how we applied grounded theory in our study; we describe our experience with classical grounded theory more thoroughly in Adolph et al. (2011).

2.3. Grounded theory and literature reviews

Glaser (1978) strongly encouraged grounded theorists to conduct their literature reviews after their theory has emerged to avoid undue influence by extant theory on emerging theory; however, his advice should not be interpreted as ban on conducting a literature review prior to undertaking a study, because any study must be situated in the context of current work. We conducted our literature review in two phases.

The first phase framed how the problem we were exploring led to the formation of our research question. The review gave us confidence there was value in conducting this research. Some of our prior research into software method adoption and use is cited in our motivation. The second phase of our literature review was undertaken after we constructed our grounded theory entitled *Reconciling Perspectives* by comparing extant theory to our grounded theory. This comparison is described in the discussion section of this paper.

2.4. Research question

The grounded theory method permits researchers to identify an actual problem that exists for participants in a substantive area rather than beginning with 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 until a problem is identified (McCallin, 2003). The opportunity to determine 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?*" Our question did not address Agile methods specifically because we were interested in creating a model of how people actually develop software, with an eye to discovering gaps between that model and current software methods. 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 ideas mattered to software developers. Had we gone into the field with a rigorously defined research problem, we likely would never have learned that the process of *Reconciling Perspectives* was a way people have of *Getting the Job Done*.

2.5. Data collection and analysis

2.5.1. Research sites

Field research means going into the field and immersing oneself in the environment (Emerson et al., 1995). We were fortunate to have good personal connections with many software engineering organizations, meaning that we did not have to conduct "ambush" interviews (opportunistically inviting anyone we met in a hallway for an interview). These strong connections also created opportunities for the primary researcher to be on site for extended periods of time. The primary researcher used his personal connections with principals in each of the test sites to invite these sites to participate in this study. While some (Mulhall, 2003) have expressed concern about selecting field sites simply because they are accessible to the researcher, when the choice is between ambush interviews at a conference and an opportunity for an extended engagement, accessibility wins. After agreement was reached between the researchers and participants, a formal letter of invitation was sent to the organization in accordance with University research ethics guidelines.

We engaged in the field study from February 2009 through January 2010, collecting data (using semi-structured interviews and participant observation) and analyzing data over the 12-month period. Participant observation was a major data collection method rather than a supplement to our interviews because it allowed the primary researcher to observe what people did rather than having them comment on what they believed they did. In total, over the year, the primary researcher conducted 20 interviews and spent some 42 days observing participants at work. We were able to collect field data from the three different software development organizations listed in Table 1.

After gaining access to a site, the principal investigator made a concerted effort to maintain a regular visitation schedule, usually once a week for local sites and once a month for out-of-town sites. There were times when we collected data faster than we could analyze it. We sometimes stopped the visitation schedule until adequate analysis had occurred because grounded theory requires concurrent data collection and analysis. Unfortunately, the result of such stoppages was that we had difficulty in re-engaging the sites. Once a researcher is out of sight, he or she is out of mind, especially for busy software development teams. In addition, Site 1 abruptly dropped out of the study part way through due to circumstances unrelated to the study. The loss of Site 1 was somewhat mitigated by the product customization teams at Site 2 that were often deployed like Site 1 at a customer site.

While most teams participating in this study claimed to follow Scrum (Schwaber and Beedle, 2002), the reality was that all were operating in a mixed-methods environment. The Site 1 team was embedded within a non-Agile organization. The product development teams at Site 2 were using Scrum, while their product support counterparts tended to only follow elements of the Scrum method. Teams at Site 3 who participated in this study followed Scrum but also worked within the context of a formal stage gated software life cycle. At the enterprise level, Site 3 was starting an Agile transition.

2.5.2. Interviews

Research ethics did not permit us to directly recruit interview subjects. When a site agreed to participate in the study, we scheduled a kick-off meeting with the participant groups to explain the study and, specifically, the ethics of informed consent. People were then invited to contact the primary researcher, if they wished to directly participate in the study. Some 53 individuals were available to interview from all three sites according to our count of signed informed consent forms. Much to our delight, people responded enthusiastically to our invitations, and the only real problem with interviewing was finding time in a busy individual's schedule and a free conference room. Study participants varied in age and experience from front line developers 5 years out of school to experienced, seasoned individuals with 19 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. Table 2 lists the individuals who participated in study interviews. All study participants were drawn from the engineering side of the organization and but, unfortunately, the closest we came to interviewing the business side of the organization was a project manager and a business analyst.

On average, the primary researcher spent approximately 1 h formally interviewing each subject. With three of the subjects, we were able to conduct a formal secondary or follow-up interview to clarify and probe deeper into new concepts that emerged from their interviews. With most of the other subjects, follow-up interviews were much more ad hoc, often conducted as casual conversations in the subject's work area, after which the primary researcher quickly jotted down field notes. All interviews, with the exception of the follow-up interviews, started with the question, "*Tell me in your own words how you create software here at site x*." Subsequent interview questions were guided by the subject's answer to this question. Subjects were often asked to provide stories of both successful and less-than-successful projects in which they had participated. As the study progressed, events from participant observation generated further interview questions. Interviews were digitally recorded and then transcribed. Observation field notes were written in lab notebooks and then later scanned.

2.5.3. Participant observation

Our observation data was collected either by observing meetings or by simply sitting quietly in a cubicle and observing what was taking place around us. The second style of observation was more passive, and consisted of simply being on site and watching people's day-to-day interactions. The primary researcher was frequently invited to regularly scheduled status and planning meetings, as well as to ad hoc problem solving meetings. The researcher watched people asking questions, observed their patterns of movement, and noted who the "go to" people were. There were also many opportunities for informal conversations between the researcher and the study participants. The settings provided opportunities to observe interactions between people as they stood in front of large "information radiators" and conducted ad hoc problem solving meetings and negotiations. These observations were all captured in our field notes.

Because participant observation allowed observations of what people actually did, it served as an important mainstream data collection method. We considered the participant observation invaluable because it demonstrated firsthand how participants interacted. The primary researcher was able to observe phenomena that would not have been uncovered in interviews, from frustration with the unreasonable demands of an important client to the intense negotiation of a sprint planning meeting. He was able to observe the individuals who served as a group's "go to" people and observed how they put aside their tasks to help others. He observed spontaneous problem-solving meetings, and numerous NERF[1] Dart wars. Participant observation also helped us gather some data on the business side of the organization either through informal conversations or by observing representatives of the business during status meetings.

[1] NERF is an official trademark of Hasbro Corporation.

Table 1
Study site description.

Site 1	An onsite customer support field office with 7 staff providing operational support and customized enhancements for a large quasi-government entity. The field support team followed Scrum while the quasi-government organization followed a waterfall-based software development lifecycle.
Site 2	A small e-commerce product company with approximately 150 employees. This company had two distinct development streams: one that focused on developing and evolving the core e-commerce engine, and a second stream which customized the engine for clients. The product development stream used Scrum as the base of their software development method. The product customization teams followed some Scrum practices at the team level, but mostly followed a waterfall-based software development lifecycle. Three teams from product development and one team from professional services participated in this study.
Site 3	A software research and development center for a large multi-national software product vendor with over 1250 employees locally (350 are in engineering). One group of approximately 40 people participated in this study. While the organization officially followed a corporate software development lifecycle, three teams in the study group were actively following Scrum.

2.5.4. Document analysis

Two of the organizations (Site 1 and Site 2) made their internal documentation, project data, and status reports available to us. At Site 1, we had unlimited access to their online bulletin board and were able to review their process manuals and weekly status reports, as well as their corporate organization charts. At Site 2, for the product development team, most project data such as organization charts, project time lines, and release plans where posted up on the large whiteboards that covered the walls. At Site 3, we were not given access to internal company documentation. Document analysis helped us understand the context of the organization (e.g., the organizational charts), and the event history derived from project plans and status reports. From these, we were able to create questions to ask interview subjects about what happened at those events.

2.5.5. Sampling

Grounded theory does not use statistical sampling because the population under study is that set of concepts that constitute the phenomena rather than individuals experiencing the phenomena. The sampling strategy must promote the development of sufficient concepts to support a conceptually dense grounded theory. Grounded theory employs theoretical sampling where the analyst "*jointly collects, codes, and analyzes his data and decides what data to collect next and where to find them in order to develop his theory as it emerges*" (Glaser, 1978, p. 36). In other words, the analytic process and developing concepts provide the direction for the investigator to find further incidents to develop the properties and dimensions of the concepts. Theoretical sampling is driven by the emerging categories and hypotheses; therefore, it is an ongoing process that cannot be pre-determined. Theoretical sampling supports the development of an emerging theory that is tightly integrated.

We found starting the sampling process to be problematic—somewhat analogous to the bootstrap problem. The evolving theory guides the sampling process but, without any data to analyze, where do you start? We "booted up" the process using "judgmental" (Marshall, 1996) sampling to recruit our first interview candidates. Our goal was to begin with a variety of views, and frame the topic by asking both line developers and managers from different companies to tell us their stories about how they managed the process of software development. This first phase generated many, many open codes capturing how people practiced software development. Clustering the codes quickly saturated (no new data) the categories directly related to how people create software. At a conceptual level, people pretty much create software the same way; however, we were interested in how people manage the process of software development.

2.5.6. Analysis

We followed the grounded theory practice of coding and analyzing our data as it was collected (Glaser, 1992). As new data were collected, they were compared to existing concepts. We used what we learned from the analysis to adapt our interview and observation protocols. Insights we had while coding the data and clustering the codes were captured in memos. We used Atlassian *Confluence* (a Wiki) to manage the data and memos.

There are three coding phases in classical grounded theory: open coding, selective coding, and theoretical coding. Open coding (also referred to as substantive coding) generates concepts from the data that will become the building blocks for the theory (Glaser, 1992). The process of doing grounded theory is based on a concept-indicator model of constant comparisons of incidents or indicators to incidents (indicators) (Glaser and Strauss, 1967). Indicators are actual data, such as behavioral actions and events observed or described in documents and in interviews; they are often captured in the words of the participants (Strauss, 1987, p. 25). An indicator may be a word, a phrase, a sentence, or a paragraph in the data being analyzed. Concepts or categories are building blocks for a grounded theory. They can be behaviors, or factors affecting behaviors, which

Table 2
Interview subject description.

Subject Id	Role	Experience	Education
Site 1 Subject 1	Branch/Project Manager	13 years	BA
Site 1 Subject 2	Developer	5 years	MSc Computing Science
Site 1 Subject 3	Developer	7 years	BSc Computing Science
Site 2 subject 1	Team Leader/Mentor/Developer	19 years	
Site 2 Subject 3	Business Analyst	5 years	BASc, Systems Engineering
Site 2 Subject 4	Software Development Manager	10 years	MSC Software Engineering
Site 2 Subject 5	Team Lead/Mentor/Developer	6 years	MSC Mechanical Engineering
Site 3 Subject 1	Team Lead/Developer	8 years	BMath
Site 3 Subject 2	Team Lead/Developer	13 years	
Site 3 Subject 3	Software Development Manager	13 years	BASc
Site 3 Subject 4	Architect	14 years, 5 years as a developer	Self-taught software developer
Site 3 Subject 5	Development Manager	10 years	BA English literature
Site 3 Subject 6	Lead Architect/Project Leader	16 years experience	BSc
Site 3 Subject 7	QA Manager	12 years	BASc Electrical Engineering
Site 3 Subject 8	Lead Program Manager	12 years	

help to explain to the analyst how the basic problem of the actors is resolved or processed (Glaser, 1978).

During open coding, concepts are generated by asking generative questions (Glaser, 1978, p. 57) such as "What is this data a study of?", "What category does this incident indicate?", "What is actually happening in the data?". For example, when reading the following passage in an interview transcript:

> "Otherwise, we have no control. And ah we'll never make our dates. Another example of a project that ran rampant and it ah technically was on time and on budget, but there was so many change requests that it almost tripled in budget through the process of the project. And we um at the beginning of the project they cut out all the scope and every single piece of it got put in back through the course of the project. The project was supposed to take six months; it took a year and a half. It was supposed to be under $500,000; it took up $1.5 million. Um, things like ah they didn't do a legal review until it was way just before they launched" Site 1 Subject 1

We asked "what is going on here?" A project had run rampant because the original project leadership abdicated responsibility; it was not until new leadership was put in place that control of the project was regained and the project delivered. We captured this passage as an indicator, tagging it with the in vivo code "*Otherwise we have no control*". When we reflected on "What concept does this incident indicate?" what immediately came to mind was "*Leadership*" or, in this example, the lack of it. We immediately jotted down a conceptual memo referring to leadership as something that steers a project toward completion.

More indictors were developed with further reading of the interview transcript, and each new indicator was compared to previous indicators and concepts, once again asking the questions "What is going on here?" and "What category does this incident indicate?" Throughout this process of constant comparison, new insights were recorded as memos, fleshing out the concept of *Leadership* and developing its properties and dimensions. Open coding is a tedious process, and it often took two or three days to code a 1-h interview. Field notes captured during participant observation were also coded.

Open coding generates concepts very quickly, and it is easy to become enamored with concepts that support pet or preconceived ideas and also to become overwhelmed by the number of concepts. While the concepts are exciting and interesting, they need to be integrated into a story that explains how people resolve their main concern. We recognized that the constant comparative method was invaluable for helping us prune concepts because concepts must "earn their way into the theory". Many exciting concepts simply fell by the wayside because we could not find other incidents to support them or because there was not enough variation to differentiate the concept from others we had already identified.

Selective coding involves identifying the core category that best explains how study participants resolve their central concern. Without a core category, there is no grounded theory. The outcome would be, at best, an interesting collection of concepts which support thick description. Using grounded theory, we aimed to construct a well-integrated set of hypotheses that explained how the concepts operated. Finding the center of gravity is a metaphor that for us best describes the process of selective coding and explains the role of the core category in integrating closely related concepts.

Discovering the main concern (that is: what problem people are trying to resolve) helped us construct the core category. When we directly asked people to identify their main concern, we received a variety of answers; but, as we sifted through the data, we observed a pattern of near angst with *Getting the Job Done*. We were trying to understand how people managed the process of software development to get the job done.

There were many false starts for selective coding. One of our early false starts was considering the concept of "Scouting" as our core category. "Scouting" had much appeal as a core category because it confirmed what many of us believe to be true about the process of software development: it is a wicked problem (Rittel and Webber, 1973) where a solution requires an unpredictable exploration of unknown and possible dangerous territory. "Scouting" was certainly a recurring pattern in our data; it met some of the criteria for a core category, and could be seen as a method for *Getting the Job Done*. On the other hand, the "Scouting" category did not explain most of the behaviors or events in our data. "Scouting" did not explain why people could go dark, or how experience and leadership influenced the process. Finally, it did not explain the efforts people made to "*get everyone on the same page*". As we directed our data collection to develop the properties of Scouting, we saw more and more incidents in the data that highlighted the need to manage communications between individuals and groups.

After more false starts, the concept of "*Bunkering*" began to emerge as another candidate core category. We conceptualized "*Bunkering*" as a boundary-setting process for how individuals and groups protected themselves from complexity and uncertainty by limiting their scope of concern. We invested considerable effort developing the "*Bunkering*" category because it seemed to explain all the data we were collecting; however, the resulting theory failed Glaser's quality criteria of parsimony. The theory of *Bunkering* was, in reality, many related theories, such as one describing the process of software development, another describing how people engage in a software project, and another describing the influence of leadership and personal relationships on how people engage in a software project. A factor contributing to the lack of parsimony was our lack of conceptualization. The concepts we were using to try to build the theory were too literal. We resolved this issue by asking the generative question: "What is actually happening in the data?" From this reasoning, the category "*Reconciling Perspectives*", a process that participants use to reconcile their different points of view and negotiate compromises, was constructed. It was surprising, once we constructed a core category that "fit", how quickly we could integrate the theory around it, like objects being drawn into an orbit. It made us recall Glaser's observation that the theory will integrate if we do not force our pre-conceived concepts on it.

"*Theoretical codes conceptualize how the substantive codes may relate to each other as hypotheses to be integrated into the theory*" (Glaser and Holton, 2004). Open coding breaks the data open and generates categories as the building blocks for the theory but a grounded theory is not a loose collection of categories. A grounded theory explains how people manage a problem through an integrated set of hypotheses. For example, two categories generated during open coding were "Experience" and "Reaching Out"; they may be theoretically coded as causal and associated with degrees. The more experience an individual has, the greater the likelihood he or she will reach out to others in order to find compromises.

Theoretical codes are not mutually exclusive and other theoretical code families, such as the "Strategy" family, describe how in our emerging theory individuals maneuvered others to remove impediments to "*Getting the Job Done* by *Reconciling Perspectives*". For us, theoretical codes illuminated relationships between categories and provided a vocabulary for describing those relationships.

The primary researcher collected all data and performed the detailed analysis of the data, while the other researchers provided guidance in clarifying, elaborating, and consolidating the emerging concepts. During the many false starts and blind alleys, the other researchers provided "sober second thought" to restart the analysis by simply asking generative questions of the primary researcher, such as "What is the data really indicating?" and "What

is really going on here?". Some of the most productive collaborations occurred when the other researchers suggested alternative concepts to those proposed by the primary researcher. These collaborations helped sharpen the emerging concept and pointed to a need for further data collection.

2.5.7. Validity

Researcher bias is certainly a threat to validity in any research and even more so in qualitative research because the results are our interpretation of the data. While another researcher analyzing our data may have a different interpretation of the data than ours, if researchers follow grounded theory practices, their theory must fit the data and offer the best explanation for the majority of the data. A useful metaphor may be to liken this to the work of two painters painting the same sunset. Unless we ask them to provide a hyper-realistic descriptive representation of the sunset, the painters will each interpret the scene differently and offer different insights into the nature of the sunset beyond what is possible in a descriptive photograph. One painter may choose to highlight the brilliance and beauty of the sunset, while the other may highlight the feeling of calm as another day draws to an end; however, both paintings are valid interpretations supported by the data.

The quantitative criteria for research rigor and quality, with which software engineering researchers may be familiar, do not fit with qualitative tenets. Just as we are trying to answer a different type of question with qualitative approaches, we need to follow different criteria for rigor. We can frame the grounded theory rigor issue with two questions:

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

Lincoln and Guba (1985) answered these questions by defining the frequently cited criteria of trustworthiness for naturalistic inquiry, which has four aspects: confirmability, dependability/auditability, credibility, and transferability.

- **Confirmability**: conclusions depend on subjects and conditions of the study rather than the researcher.
- **Auditability**: the study process is consistent and reasonably stable over time and between researchers.
- **Credibility**: the research findings are credible and consistent, to the people we study and to our readers. For authenticity, our findings should be related to significant elements in the research context/situation.
- **Transferability**: the findings/conclusions can be transferred to other contexts and help to derive useful theories.

The interpretive nature of qualitative research makes replication of results difficult, but we demonstrated that our results were reliable by making them auditable. We maintained logs of our interviews and observations, which included date, time, and location. We created a large bank of both theory and process memos and have offered thick descriptions of concepts and categories. Auditability and thick description supports credibility, as does data triangulation. We gathered data using different mechanisms and different types of sites. We gathered data using multiple techniques, such as interviews, participant observation, and document analysis. Data were collected from sites with different project types and different sizes of organizations.

Another threat to validity was researcher bias, where the results reflected the opinions of the researcher rather than what was indicated by the data. Researcher bias was mitigated by using the grounded theory approach to constant comparison and by the researchers explicitly stating their biases.

A significant source of bias in this study was that the data was obtained from people who mostly have what we could refer to as strong personalities, some even to the point of extroversion; these subjects commented on problems associated with other people not talking to one another. Unfortunately, we could not actively recruit into the study examples of individuals who were reluctant to talk, so our interview data was biased toward those who like to talk. This bias was partially mitigated by our observation of teams during the study period, which included those team members who were not inclined to participate in the interviews.

Another interview bias was that all participants in this study were recruited from the engineering side organization rather than the business side of the organization. While several of the senior people interviewed in this study have experience playing quasi-business roles (e.g., project manager), this remains another limitation for our study.

3. Results

Software development takes place in the context of a diverse organizational *Eco-system* where an *Acquirer* requests a *Supplier* to perform a *Job* for them. To perform a *Job*, the *Supplier* and *Acquirer* must agree on their expectations of the *Job*, including the *Work Products* that will be delivered as part of the *Job* and the schedule for their delivery. The *Supplier* creates the *Work Products* for the *Job* and delivers them to the *Acquirer* for acceptance. The diversity of the organizational *Eco-system*, with the narrow and different specializations of the *Acquirer* and *Supplier*, may lead to a *Perspective Mismatch* where those involved in the process of software development have different expectations of the *Job* in terms of *Work Products* and schedule.

We discovered that people use a process of *Reconciling Perspectives* to manage the process of software development and to resolve the problem of *Perspective Mismatch*. Their main concern is *Getting the Job Done* and the different *Perspectives* individuals have of a *Job* impede *Getting the Job Done*. *Reconciling Perspectives* is the basic social structural process people managing the process of software development use to resolve *Perspective Mismatches* and remove impediments 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 Mismatch*, 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. Fig. 2 is a schematic of the process.

In *Reconciling Perspectives*, there are two clear stages that account for variations in behavior: *Converging* and *Validating*. During *Converging*, individuals try to reach a shared *Consensual Perspective* by *Reaching Out* and *Negotiating Consensus*. After achieving a *Consensual Perspective*, the individuals validate the *Consensual Perspective* by creating the *Job Work Products* during *Bunkering* and then judging them during *Accepting*. The process has four properties:

Scope	the number of individuals in the organization affected by the *Perspective Mismatch*.
Cycle Time	the interval of time from when the *Perspective Mismatch* is detected until the reconciliation is validated.
Communications	the frequency, volume, and diversity of *Communication* between individuals in the organization.
Team Dynamics	the capability of a team to manage their *Communications*.

When engaged in the process of software development, whether as an *Acquirer* or a *Supplier*, individuals have a point of view

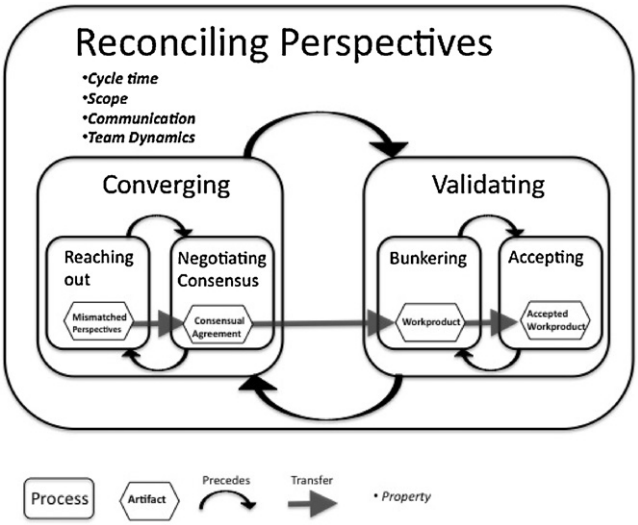

Fig. 2. Reconciling perspectives.

or *Perspective* which is how they see and understand a *Job*. When individuals reflect on *Communications* they have heard or observed regarding a software project, if their interpretation of *Communications* does not match their expectations then there is a *Perspective Mismatch* that can impede *Getting the Job Done*. This *Perspective Mismatch* can impede the project because *Suppliers* and *Acquires* may have different expectations of the value, quality, and priority of *Job* features, *Job Work Products*, and the schedule that the *Work Products* will be delivered to. When individuals recognize a *Perspective Mismatch*, they start a *Converging* process by *Reaching Out* to engage others in *Reconciling Perspectives*. If the others agree to engage, then the parties begin *Negotiating Consensus* to create a common *Perspective* or *Consensual Perspective*. A *Consensual Perspective* may be as simple as reaching an informal decision, or may be a formal signed off agreement that specifies the *Job*.

While the *Consensual Perspective* removes an impediment to moving forward, it is only tentative because the parties are each assuming that they have a better understanding of how the others see the problem and its solution. While their *Communications* may no longer indicate a *Perspective Mismatch*, the process is not complete until it is validated (*Validating*) by creating the *Work Product* during *Bunkering* and presenting it to the *Acquirer* (*Accepting*). During *Bunkering*, people prefer to focus on creating the *Job Work Products* with minimal interference.

Reconciling Perspectives is characterized by its observable properties of *Scope*, *Cycle Time,* and *Communication*. The process varies from the large scope, where acquiring organizations engage with supplier organizations through formally mandated and structured meetings, to the small scope and ad hoc that may begin with one individual seeking out another and asking: "Can we talk?" The process *Cycle Time* may be short, with only a few minutes elapsing between *Reaching Out* and *Accepting*, or may be long, requiring months until a *Work Product* is accepted. *Communication* is the frequency and diversity with which people engage with others and exchange ideas. People may be actively and frequently communicating, sharing, and exploring ideas, and negotiating features and priorities, or they may be quiet and isolated. People may only share ideas with their immediate colleagues, or they may seek diverse ideas beyond their immediate colleagues.

The outcome of the process is strongly related to *Team Dynamics* which is the team's ability to manage communications. The degree to which an individual or team is open to *Communications* is captured by the *Team Dynamic* property of *Openness to Communications*, which ranges from *Exposed* to *Cut-off*. One property of *Team Dynamics* that influences *Openness to Communications* is *Personal Strength* or the propensity of an individual to engage others. In all the projects observed during this study, a common success driver is the presence of an individual or group of individuals who have the *Personal Strength* to initiate and engage in this process. While *Personal Strength* is partly an innate trait of the individual or group, it is also related to the individual's or group's level of *Experience*. Experienced individuals tend to have developed a diverse set of *Back Channels* which they utilize to help them increase the diversity of their *Communications* and detect when something is amiss. A characteristic of more experienced individuals/groups is the increased numbers of informal social networks they create. *Personal Strength* is also the ability to prevent becoming *Exposed* by knowing when to say "no" and decline engagement offers from others that will divert attention or interfere with *Getting the Job Done*.

Managing *Communications* is a key skill for *Reconciling Perspectives* because there is a *Communications* tension between the *Converging* and *Validating* phases of the process. The *Converging* phase requires that individuals and teams be open to *Communications*, such that they have the opportunity to reflect on what they are seeing and hearing and detect *Perspective Mismatches*. Individuals and teams must be open to engaging in negotiations, and be able to communicate effectively enough to converge their *Perspectives* and reach a *Consensual Perspective*. During the *Validating* phase, though, individuals and teams must focus on *Getting the Job Done* by creating the *Job Work Products*; therefore, during *Bunkering*, they prefer to reduce *Openness to Communications* in order to minimize distractions. This works at cross purposes to the needs of *Reconciling Perspectives* where they must remain open to *Communications*. To run to one's cubicle and work in isolation creating *Job Work Products* risks *Cut-off*, where an individual or a team disconnects from *Communications* and therefore misses *Communications* indicating *Perspective Mismatches*. In contrast, not restricting *Communications* can leave an individual or team *Exposed*, wasting energy and not *Getting the Job Done*, because they are engaged in negotiations to resolve constantly shifting requirements and priorities, or being diverted to work on other *Jobs*.

The *Team Dynamic* property of *Leadership* can not only encourage inter-personal engagement but also protect individuals from becoming *Exposed*, with the resulting time wasting interference. *Leadership* from an individual's manager, or from a respected thought leader within the organization, boosts *Personal Strength* by giving individuals the confidence that "their backs are covered" When individuals or groups believe their backs are covered, or believe that they will not be sanctioned for raising issues, they are more likely to take risks and reach out. *Leadership* also plays a role in moving the process forward by actively encouraging people to raise issues, confront *Perspective Mismatches*, and make decisions to resolve them. Interview subjects used phrases like, "drum beating" and "energizing" to describe efforts to by leaders to encourage people to communicate and reach out.

Reconciling Perspectives is a fragile process that is stalled by indecision and disengagement, and *Leadership* drum beating creates a rhythm to keep the process from stalling or to restart a stalled process. Teams that were observed making progress had effective Scrum Masters who diligently maintained the rhythm of the Scrum process and worked to remove impediments that could stall the team. For many teams, the Scrum Master was a subject matter expert and was, therefore, the "go to" person when an individual was stuck for both that team and other teams. It was not unusual to see a line-up forming around some individual leaders. There were also numerous situations where the process had stalled and other

individual team members took it upon themselves to restart the process.

Effective leaders balance *Openness to Communications* between *Exposed* and *Cut-off* for the team by buffering or *Sheltering* individuals from interruptions and distractions during the *Bunkering* stage of the process by relieving others of some of their responsibility for maintaining the frequency and diversity of *Communications*. While protecting individuals from wasteful interruptions, effective leaders are also aware of issues that are sufficiently important to warrant interruptions because they indicated a *Perspective Mismatch* that affects the sheltered individuals. Some leaders do not balance *Openness to Communications* and either leave the team *Exposed* or aggressively shelter teams from the organizational *Eco-system*. We encountered several stories of leaders actually cutting their team off from the rest of the organizational *Eco-system*. In some cases, the *Cut-Off* was so extreme that one person referred to a team as a "black hole" from which nothing ever escaped.

3.1. The software development context

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 Products* that realize the *Job*. The concept of an *Eco-system* provides the context and vocabulary for understanding the theory of *Reconciling Perspectives* and is not an integral concept of the theory.

The term "eco-system" best 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. At all three sites, there was no single software development methodology used; rather, each site used a collection of methods existing within the context of a perceived interpretation of an advertised corporate method. As one subject explained:

> "Our team uses the scrum process. Um, so but unfortunately we have to fit within a larger organization that's using ah a waterfall methodology" Site 3, Subject 3

Differences between groups were further highlighted by the layout of the workspace and how the teams under study worked with other teams in the organization. For example, Site 1 involved a small Scrum team working within the bureaucracy of a large quasi-governmental agency. The workspaces for the governmental agency were uniform in color, shape, size, and furniture whereas the Scrum team was relegated to what, at best, could be called a windowless basement with office furniture that appeared to have been purchased at a variety of garage sales.

3.2. Getting the job done

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

> "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

Participants defined *Getting the Job Done* as creating working software that:

- Appeases the customer or even makes the customer happy,
- Satisfies the software team members' needs to see the end and achieve a feeling of accomplishment, and
- Satisfies the team members' desires to minimize technical debt.

Despite marketing hyperbole of "delighting customers", those participating in this study set the customer satisfaction bar low at appeasement or, more forthrightly, at "anything that keeps the heat off".

> "They feel that they're under pressure so they just have to find a way to get it to work" Site 2, Subject 5

This was such a frequently recurring pattern that we referred to it as the "heat calculation", where individuals and teams chose courses of action that least antagonized a client. In one observed status meeting, a team explicitly debated whether they would draw "less heat" by delivering a deficient product on time or by delaying the release of the product.

We did not observe such a strong concern for customer appeasement in the teams that were not customer facing. For these teams, *Getting the Job Done* was described more in terms of technical craftsmanship. Much of the frustration expressed by software developers during the interviews in *Getting the Job Done* was not only to deliver software to the *Acquirer* but to deliver quality software because all software developers participating in this study took pride in their craft and had a strong desire to create quality work.

> "I don't really enjoy solving a problem as much when I'm given just an hour to deliver something or a few hours to deliver something that I know will—if I don't spend more time it won't be as good quality as it should be." Site 2, Subject 5

3.3. Perspectives mismatches

Throughout this study we discovered that a frequent and difficult problem in software development is "getting everyone on the same page"

> "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

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*. We referred to the source of these impediments, created by differing points of view to *Getting the Job Done*, as a *Perspective Mismatch*. There are four broad groups of *Perspective Mismatches* based on the topic of the mismatch (Table 3):

The diversity of the organization *Eco-system* contributed to the *Perspective Mismatch*. There were many instances where

Table 3
Perspective mismatches topics.

Features	Different *Perspectives* regarding the behavior, scope (what features are in and what features are out), and quality of *Job* features.
Priorities	Different *Perspectives* on the order and relative importance of *Job* features with respect to other *Job* features and other *Jobs*.
Method and work products	Different *Perspectives* regarding what *Work Products* are created and when those work products are delivered.
Schedule and status	Different *Perspectives* when a *Work Product* is ready and the current quality of *Work Products*.

individuals did not understand how others in the organization associated with the software project performed their work and contributed to the *Job*. One of the justifications for a software method is that it provides a common perspective for all involved in a software project for how software will be created in the organization. Software methods are supposed to set the expectations of everyone associated with the project and get them on the same page. All organizations participating in this study did claim to have a corporate-wide method but what we saw in most projects was that few people had any interest in the corporate methodology:

> "Um, I'll be honest—as somebody who's a lot further down the chain I very rarely pay attention to that process or the milestones associated with it. I hear vague um speeches of it. You know, we're hitting EMRA and whatever, or whatever the milestone name happens to be. And there are big meetings to prep for it. But as somebody a lot further down the chain, to me, it's just dates" Site 3, Subject 1

This *Perspective Mismatch* creates friction that can impede *Getting the Job Done*, where individuals only understand their localized area:

> "I just find it creates a lot of friction because people don't—only understand their layer" Site 3, Subject 6

Perspective Mismatches were not only the result of differences of opinion on what features should be included in system, but also the result of quality trade-offs. A significant area of mismatch was priorities, what should be delivered first:

> "you know each person has their own ah priorities". Site 3, Subject 7

3.4. Converging

Converging is first stage of *Reconciling Perspectives*, where individuals recognize that a *Perspective Mismatch* is impeding *Getting the Job Done* and reach out (*Reaching Out*) to engage others in negotiations (*Negotiating Consensus*) to converge their *Perspectives* sufficiently and decide on a course of action (*Consensual Perspective*). While the *Consensual Perspective* can temporarily remove the *Perspective Mismatch* as an impediment to *Getting the Job Done*, it is only a hypothesis because it is reached by individuals agreeing that they are on the same page. The process of *Reconciling Perspectives* is not complete until the consensus is validated (*Validating*) by the delivery of a *Work Product*. It's a bit like the old expression "the proof is in the pudding".

The *Converging* stage ends when everyone agrees that they understand what the *Job* is and can decide on a course of action in the form of a *Consensual Perspective*.

3.5. Reaching out

Reconciling Perspectives begins when individuals reflect on the *Feedback* in the *Communications* they are receiving about the *Job*, believe *Getting the Job Done* is impeded by a *Perspective Mismatch*, and begin *Reaching Out*. They reach out by inviting 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 the *Feedback* in their *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 that, for them, 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 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

Without the *Communications*, there is no opportunity for *Feedback*, and without *Feedback*, there is no opportunity to reflect and determine if what one is hearing matches one's *Perspective*. One interviewee succinctly expressed the need for *Communications* when describing a challenged project:

> "But there just weren't enough conversations taking place" Site 2, Subject 3

Once an individual recognizes the possibility of a *Perspective Mismatch*, he or she needs to reach out and engage others to investigate and resolve the mismatch. *Personal Strength* is the propensity of individuals to act when they believe a *Perspective Mismatch* is impeding *Getting the Job Done*. This can be problematic if the individual's *Personal Strength* is low. Many of the interview subjects voiced the common stereotype of software developers as quiet or introverted individuals who are reluctant to reach out.

> "And maybe that's another thing. Is the—is this sort of—developers aren't the most social people at times right? So they-they don't go and speak to people about their problems necessarily. Being able to—so to some extent maybe it's a lack of confidence in the—if you're seen to ask—or people seem to think that asking for help is a sign of weakness or something" Site 2, Subject 4

Leadership contributes to *Personal Strength* and gives individuals the confidence that their back is covered when they are raising issues that may have serious consequences for the *Job*.

> "as long as you've been open with <name> and treat him with respect and integrity um he will back you up. So, you're not going to get—you're not going to get backstabbed by him. And that's a very important thing. It relaxes me and I think it goes down to you—you—he gives us support. . . So, I would say that it—it allows me to take some—some risks if I know what I'm doing is right and I am open enough about it" Site 3, Subject 8

Leadership also has the potential to shut down individuals who express concerns about *Perspective Mismatches* between them and their *Leadership*. Leaders are in a position to either support or block an individual *Reaching Out*.

> "Communication and not being afraid to express ideas and have-have discussion you know? I'll have one way of viewing something or designing something but I don't feel if—I'm afraid to say it because I'll get trampled on by somebody who's more experienced or more senior. But everyone is able to offer their view and kind of like learn—everyone can be involved in the

kind of—the learning experience. And then contribute to you know—some good quality code being delivered" Site 2, Subject 5

It is not enough for an individual to recognize a *Perspective Mismatch* and have the *Personal Strength* to reach out, because those individuals who are intended recipients of reaching out may choose not to engage. The process of *Reconciling Perspectives* cannot continue if the individuals to whom the person is reaching out will not engage in the process.

"it was very typical of projects that kind of go astray. Where people live in denial for too long. And—people who voice their concerns are kind of ignored—until it just bubbles up. I—don't think that anybody was really quiet about it. I don't think that anybody—who was being realistic or worked on it—like everybody could say well hey this thing is really not going very well. I just don't think that that was percolating all the way up" Site 2, Subject 3, follow-up

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?" or as formal as placing an agenda item on a change control meeting.

3.6. Negotiating consensus

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

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

Negotiation occurs at all levels and at all scopes, from CTOs, analysts, and product managers negotiating product features with large external 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:

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

The properties *Cycle Time* and *Scope* characterize *Negotiating Consensus.* Many negotiations, much like the one characterized in Fig. 3, are common conversations, short, and measurable in minutes if not seconds. The scope of these negotiations is very narrow and specific and conducted between individuals without much formality or structure. Other negotiations could be between informal groups, or between formally defined groups such as Scrum teams or even organizational divisions. As the scope increases, the formality of the process increases to accommodate the more complex coordination needs of the participants for established meeting times, meeting facilitation rules, agendas, action items, and follow-ups. There were times when a *Consensual Perspective* was reached during a meeting, and other times when the agreement was to continue investigating an issue. This type of negotiation was fairly common at the beginning of projects where *Acquirers* and *Suppliers* are negotiating project scope and budget. A common strategy in these situations was *Progressive Refinement*, decomposing large, poorly defined features into a set of smaller and more precisely defined requirements. Most groups participating in this study had a multi-cycle negotiation process for reaching a *Consensual Perspective* on scope and budget.

A common failure pattern was an excessively long *Cycle Time* or stalling during *Negotiating Consensus* and never reaching a timely *Consensual Perspective*. Several challenging projects that were observed or reported during this study could be explained by their never reaching an explicit *Consensual Perspective*.

"I'd-I'd much rather know it has to work. It has to work. It has to be somewhat releasable—maybe not every iteration but every month. And with 14 months that's 14 times it needs to be—and we weren't there. We-we were broken for quite a while. Which took—took—you know—the decision making capabilities away from the business—which isn't what we're about" Site 2 Subject 4

Many of the stories about out-of-control projects, such as the one above described by Site 1 Subject 1, could in part be explained as never explicitly reaching a *Consensual Perspective*. Project participants defined their expectations of the *Job* from their own mismatched perspectives. In some of these situations, we observed people stepping in and filling the *Leadership* vacuum to obtain an explicit *Consensual Perspective*. This is an example of *Leadership* drum beating, maintaining a rhythm that leads to a decision in a timely manner. We frequently observed problem-solving sessions breaking down into arguing about minutiae or hypotheticals. There were many instances of leaders forcefully driving an agenda, where the intent was not to force acceptance of the leader's choice but to make a decision in a timely manner, even if that decision was to acquire more data and meet at a later time. This was exemplified by Site 2 Subject 4's story of individuals stepping up to recover a near catastrophic project caught in what could best be called "leaderless drift":

"I don't think—without that step up in leadership—and the other team lead—once he figure it out as well stepped up as well. And without that <JOB> wouldn't—I don't think I be sitting here talking to you—not in this company anyway. I think it would have been—if I'd carried on—it-it just never would have been released. So it—stepping up—getting everyone to talk—getting everyone focused". Site 2, Subject 4

The ability to *Compromise*, or the cultivation of a *Flexible Response*, facilitates the *Negotiating Consensus* process. If the opportunity for trade-offs is constrained, *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 Perspective* to resolve the *Perspective Mismatch* is weakened, or even nullified, because participants may acquiesce to the terms imposed by the other simply to move ahead in some assumed direction. As one senior developer explained it:

"But—if you kind of attempt to have a perfectionist kind of take on your work it's a bit of a difficult thing to sometimes weigh

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 [for] the context?"
Dev 2 (Supplier): "Sure"

Fig. 3. Negotiating consensus.

those options and you know you'll say okay well I can really see how this should be working. But then you try and get that kind of developer or technical view that will say okay that's a cool idea. And it makes sense but it will take us half a year to implement. So that just gives you a more realistic perspective on maybe what—you know—you have one desire but—perhaps it's far away from reality" Site 2, Subject 3

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!" *Translation* was a strategy most often observed in informal short cycle negotiations; however, examples of *Translation* could also occur during more formal negotiations.

"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

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. Common sources of impediment were software developers viewing the *Job* from a technical perspective and not fully appreciating the business impact of their decisions, and *Acquirers* not understanding the technical consequences of their demands—for example: understanding how a request may increase technical debt or limit opportunities to grow the product. The ability or desire to *Broaden* their *Perspectives* certainly appeared to be a strategy commonly employed by more experienced individuals. One developer spoke of broadening in terms of seeing the context or "bigger picture".

"I think kind of—one important thing is the context. So if a stakeholder is only aware of their—small item—is the most important thing to them in-in the world. But if they're able to see the big picture and have an understanding of what the whole big picture is then I think that allows it." Site 3, Subject 5

The final strategy is *Scouting*. In both *Translation* and *Broadening*, one or both parties have sufficient knowledge to understand the *Job*, but cannot express it in a way that the other understands or cares about. *Scouting* becomes the *Negotiating Consensus* strategy when neither the *Acquirer* nor the *Supplier* has sufficient knowledge to explain their view to the other party. 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 without end, a *Consensual Perspective* was not reached, and no decision was explicitly made on how to remove the impediment. The *Supplier* either continued to make perceived progress toward *Getting the Job Done* by working with their *Perspective* of the *Job*, or remained stuck in *Scouting* and failed to deliver the *Job Work Products*. 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 desert*" for 6 months without creating a true *Consensual Perspective*.

Converging transitions to *Validating* when a *Consensual Perspective* on what the *Job* is and what *Work Products* are needed is created. The *Consensual Perspective* is based on a shared assumption by all participants that they have a shared *Perspective* of the *Job* and that the impediment created by the *Perspective Mismatch* has been removed. The development process can now move ahead.

3.7. Validating

The *Consensual Perspective* reached during *Converging* is a hypothesis that must be tested, and is only validated when the *Supplier* creates a *Work Product* that is accepted by the *Acquirer*. There are two stages to *Validating*: *Bunkering*, and *Accepting*. *Bunkering* is a quiet stage during which the *Supplier* focuses on creating *Work Products* that satisfy the negotiated *Job* requirements. What strongly characterizes this stage is the drop in *Communications* between *Acquirer* and *Supplier*. *Accepting* is the final stage of the process where the *Supplier* presents the *Work Products* to the *Acquirer* as evidence that they worked from the same *Perspective*.

3.8. Bunkering

Whereas during *Converging* individuals actively engage in conversations during *Reaching Out* and *Negotiating Consensus*, *Bunkering* is a "heads down" working "quiet" stage; using the knowledge gained from the *Consensual Perspective* created during *Converging*, the *Supplier* creates the *Work Products* they believe will satisfy the *Acquirer's Job* requirements. A distinct drop in *Communications* and interactions between *Acquirer* and *Supplier* occurs during *Bunkering*. This "quiet time" is built into software methods like Scrum, which most teams participating in this study claimed they were using; however, it also seems to be part of the culture, regardless of the software method used. Once a *Consensual Perspective* is reached, construction of a *Work Product* should proceed relatively undisturbed until it is done. The desired, quiet, undisturbed environment created during the *Bunkering* stage contributes to productively creating *Work Products*.

"we were in a lot of ways kind of flying under the radar... Therefore we didn't have to answer to anybody—so directly. So we got a good chunk of time running like that on our own. We got a lot of stuff done" Site 3, Subject 4

Bunkering is a boundary setting stage and the *Team Dynamic* property of *Openness to Communications* characterizes how permeable to *Communications* individuals and teams are during *Bunkering*. During *Bunkering*, people can completely cut themselves off from all conversations and other sources of information that

may reveal to them a *Perspective Mismatch*. Individuals, and often entire teams, cut themselves off from the rest of the organizational *Eco-system* and are not open to the *Communications* that may raise indicators of a *Perspective Mismatch*. *Cut-off* reduces the opportunity for the process of *Reconciling Perspectives* to begin because the opportunity to receive and reflect on *Feedback* is reduced.

> "Like two people who work together and they maybe need to integrate some stuff with the rest of the team but they kind of pigeon-hole themselves and you know only talk to each other but not really interact so much with the rest of the team members and the end of it iteration approaches and you realize that they've been working on something that's actually not—you can't integrate with the rest of the team—with their code. So—definitely kind of that isolated lack of talking to other people" Site 2, Subject 3

The opposite extreme of *Cut-off* is *Exposed*, where an individual or group engages with all *Communications* and therefore cannot focus on *Getting the Job Done*. They have far too much work in progress; they are also taking on more, or changing priorities, or escalations are constantly interrupting them. The diverse nature of the *Eco-system* means that not everyone has the same *Perspective* on work priorities and there are numerous instances of people directly approaching developers to get what they perceive as a high priority *Job* done. Individuals and teams that could not say no became, as one project manager put it, they are "their own worst enemies".

> "But you're your own worst enemy when you do that because the next thing you know, the BA's asking this developer for an estimate on this and can you do that? And oh I found this new scenario and last week" Site 1, Subject 1

The consequences of being *Exposed* were often evident in projects at the tail end of a waterfall process, when individuals and teams were often panicking to remove defects. It was common to see priorities rapidly changing in reaction to discovery of new defects, and partially finished work put aside to work on these higher priority defects. The result was that defect opening rates far exceeded defect closing rates, and the job simply did not get done.

> ". . .most of the developers were not happy about the way things were (unclear). It was it was—it got to this what we call a bug-fixing hell of a state. There was so many bugs and we were trying to fix the bugs a couple of months. There was long days when we were trying to get this working" Site 2, Subject 1

Bunkering can put *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 understand the *Consensual Perspective* and the results they are getting while creating the *Job Work Products* but they may be reluctant to start *Reaching Out* or to engage with others reaching out to them.

> "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

Finding a balance between being *Cut-off* and *Exposed* is necessary for *Reconciling Perspectives* to resolve *Perspective Mismatches* and lead to *Getting the Job Done*. Individuals and teams must remain sufficiently open to *Communications* and be able to detect *Perspective Mismatches* and yet not be *Exposed* to the rapid changes and vagaries of the organizational *Eco-system* that can interfere with *Getting the Job Done*. This need to find a balance between being *Cut-off* and *Exposed* highlights the inherent tension of the *Reconciling Perspectives* process.

> "Um ah and I think we still haven't got to a good place where we get that balance between you know short term, heads down focus, but also a sense of where the vision of things are going" Site 3, Subject 6

Leadership helped find this balance by *Sheltering* individuals or teams from the turmoil of the organizational *Eco-system*. *Leadership* frequently protected the team from exposure by standing between them and the *Eco-system*. One subject referred to this protective or buffering role as a "firewall".

> "Sometimes it's a firewall kind of role. And our (unclear) plays that role quite a bit as well—a ton. He insulates all of us from a lot of that stuff" Site 3, Subject 7

A leader in this role was able to stay open to communications and yet offer some measure of stability to those who were trying to *Get the Job Done*. How much a leader needed to stay open to communications was proportional to where the leader believed their responsibilities lay in the organizational hierarchy.

> "the higher up you move in the chain because you're expected to-to take on more—more responsibility right? So you'll only get—you only really get shielded so far in as much responsibility as somebody has given you. If you have a reasonable amount of responsibility you won't be needed—you won't need to be shielded that much" Site 2, Subject 5

Bunkering transitioned to *Accepting* when the *Supplier* presented the *Job Work Products* to the *Acquirer* for their acceptance.

3.9. Accepting

During *Accepting*, the *Supplier* presents the *Job Work Product* to the *Acquirer* and solicits their approval. *Accepting* may be as simple as the *Acquirer* informally stating, "Ok it's good" or it may be a formal ceremony for transitioning and *Accepting* a *Work Product*. Only when the *Work Product* is accepted is there objective evidence that 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 in larger *Surprises*, than shorter cycle *Reconciling Perspectives* processes. A near catastrophic *Surprise* occurred at one site where *Supplier* and *Acquirer* teams effectively cut each other off for over 6 months.

4. Discussion

This discussion of *Reconciling Perspectives* compares it to other relevant grounded theories and extant theory to both situate *Reconciling Perspectives* within the existing body of knowledge and to generate policy recommendations. The *Anticipation-eXecution-Expectation* (AxE) model (Hsieh et al., 2006) and *Organizing Self Organizing Teams* (Hoda et al., 2010) are two grounded theories that illuminate some properties of *Reconciling Perspectives*. The AxE model emerged out of a study of coordination between individuals in teams, and suggests the influence that the process of *Reconciling Perspective* has on trust relationships. *Organizing Self-Organizing* teams provides greater depth on our concept of *Leadership* and the leadership roles that individuals play.

Glaser's (1978) recommendation, to perform the literature search after the theory emerges, enables us to use the grounded theory to sensitize us to search for comparative extant theory that may further explain the situations we observed, and then use the extent theory with its known body of knowledge to guide policy decisions.

In this discussion, we compare *Reconciling Perspectives* to Mental Model Convergence (Cannon-Bowers et al., 1993; Cronin and Weingart, 2007; Firore et al., 2001; Johnson-Laird, 1983) and Contingency Theory (Burns and Stalker, 1961; Lawrence and Lorsch, 1967; Woodward, 1958).

All teams participating in this study claimed to follow an Agile software development method, specifically Scrum, although one team later claimed they had switched to Kanban. We use *Reconciling Perspectives* to explain how Scrum teams function, and also to explain some of the dysfunctions we observed during this study. Finally, based on what we have learned during this research, we offer policy recommendations for improving how people manage the process of software development.

4.1. Anticipation-eXecution—Expectation model

The Anticipation-eXecution-Expectation (AxE) model (Hsieh et al., 2006) explains coordination between individuals and teams. AxE emerged as a descriptive framework from a study of culture and the impact of intercultural dynamics on global software development. What struck the researchers were the mishaps the teams under study experienced despite using tools and methods based on Input-Process-Output (IPO) coordination models, "*The inputs, the processes and the outputs are often apparently well-defined, but still all kinds of failures occur*" (Hsieh et al., 2006, p. 9)

At the heart of the AxE model is a work product handed from party to party (e.g., from *Acquirer* to *Supplier*) that is described by the triple:

Content	defines the artifact in terms of what it is made of and how it is represented (e.g., a use case).
Quality	defines the quality of the artifact at hand-over time; for example, is it simply a verbal expression of need or is it a formal and reviewed specification?
Time	specifies the time at which the handover occurs.

We can use AxE to describe the process of software development. The *Supplier* has an *anticipated* outcome (*Perspective*) of the content, the quality, and the time they anticipate they will deliver their outcome to the *Acquirer*. The *Acquirer* has an *expected* outcome (*Perspective*) of what they believe will be the content, the quality, and the time when the *Supplier* will deliver the outcome. Lastly, there is the executed outcome: what the *Supplier* actually hands over to the *Acquirer* and the time when the delivery takes place. The deltas between the anticipated and expected outcomes are similar to the *Perspective Mismatch* in *Reconciling Perspectives*.

AxE has a three-step process or "waltz" for creating outcomes where:

1. the parties must communicate, and where their anticipated and expected values are formed (Planning);
2. the actual artifacts (work products) are handed over (Execution); and
3. the delta between the anticipated results and the executed results (the delivered results) is evaluated and communicated (Feedback).

In the AxE model, a large delta between expectations and results leads to a decline in trust between the parties. A large delta between the executed outcome and the outcome expected by the *Acquirer* is expressed as *Surprise* in *Reconciling Perspectives*. A large delta between executed and expected outcomes also results in a decline in trust between *Supplier* and *Acquirer*, whereas a small delta may increase trust.

Reconciling Perspectives extends AxE by offering a model of the *Communications* between *Acquirer* and *Supplier* for setting expectations. If the quality and volume of the *Communication* between the *Supplier* and *Acquirer* does not lead either to recognizing the difference, specifically, deltas between their anticipated and expected outcomes, or if there is insufficient *Personal Strength* to reach out or *Leadership* to encourage individuals to *Reach Out*, it is unlikely that the executed result will meet the expected outcome. The *Bunkering* stage of *Reconciling Perspective* maps to AxE's execution phase, highlighting the need for appropriately managed communications during execution.

AxE extends *Reconciling Perspectives* by highlighting the decline in trust between *Supplier* and *Acquirer* when there are significant deltas between anticipated and executed work products (*Surprise*).

4.2. Organizing self-organizing teams

Hoda et al. (2010) investigation of self-organizing teams identified six roles that team members play to facilitate the organizing of self-organizing teams:

Mentor	Guides and supports the team initially, helps them to become confident in their use of Agile methods, and encourages continued adherence to Agile practices.
Coordinator	Acts as a representative of the self-organizing Agile team to coordinate communication and change requests from customers.
Translator	Understands and translates between the business language used by customers and the technical terminology used by the team in an effort to improve communication between the two.
Champion	Champions the Agile cause with the senior management within their organization in order to gain support for the self-organizing Agile team.
Promoter	Promotes Agile with customers and attempts to secure their involvement and collaboration to support the efficient functioning of the self-organizing Agile team.
Terminator	Identifies team members threatening the proper functioning and productivity of the self-organizing Agile team and engages senior management support in removing such members from the team.

While the focus of their study was on team self-organization in companies adopting Agile software development methods, we can still contrast their study finding with ours to characterize some of the dimensions of *Leadership* that increase individual *Personal Strength*, perform drum beating, and shelter others. Several of the roles are directly related to boosting *Personal Strength*: Mentor, Champion, and Promoter may all be seen as roles that give team members confidence in the process, and confidence to reach out and attempt to engage others when they believe they have a *Perspective Mismatch*. The role of the Coordinator concurs with the role we see in *Leadership* to shelter teams from interference yet keep the team sufficiently connected to the organizational *Eco-system* such that they are not closed or *Cut-off* from *Communications*. In *Organizing Self-Organizing Teams*, both the Translator and the Promoter could also be seen as creating a rhythm, and being the "drum beater" both within the team and to the outside organization.

During our study, several subjects related to us stories of the damage teams suffered from the lack of a Terminator. These stories of project failures or dysfunctional teams were directly related to leaders who either let their teams become *Exposed* or deliberately *Cut-off* the team from the organizational *Eco-system*. In one case, the eventual removal of an ineffective leader resulted in an immediate improvement in team performance.

4.3. Mental model convergence, knowledge communities, knowledge creation, and reconciling perspectives

What we are referring to as a *Perspective* is comparable to what much of the psychology literature refers to as a mental model (Johnson-Laird, 1983). 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]*" (Cannon-Bowers et al., 1993). Furthermore, Firore and Salas (2001) assert that "*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 and Weingart (2007) describe 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 conflict*"

The *Converging* stage of *Reconciling Perspectives* concurs with what Sara McComb (2007) describes as "*mental model convergence*," where team members' mental models evolve into a shared mental model through their interactions and observations of each other. Convergence of mental models explains how individuals socialize themselves into teams and, during our study, we observed situations that suggested this process was occurring within teams. A significant difference in *Reconciling Perspectives* is that the process extends beyond the immediate team, and much of our interview data described interactions that individuals had with other individuals from other functional groups.

Levesque et al.'s study (2001) 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 this decline. Their finding supports the phenomena we observed during the *Bunkering* stage. Most software teams consist of people with a variety of skills and domain knowledge and, once they understand their sprint tasks, they often see no value in staying connected to other team members because there is little else that other team members can do to help them. In many situations, once a *Consensual Perspective* 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*.

Boland and Tenkasi (1995) characterized knowledge-intensive organizations as composed of knowledge communities that each possess highly specialized technologies and knowledge domains. Communication within a community that strengthens and develops localized knowledge is a "perspective making" process, while communication that improves a community's ability to take the knowledge of other communities into account is "perspective taking". If we characterize software developers, analysts, and project managers as belonging to different knowledge communities, then much of the data we collected on their interactions is supported by Boland's and Tenkasi's term "perspective taking".

Perspective making and perspective taking are knowledge creation processes, and knowledge-based firms realize competitive advantage by leveraging their 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 extends an 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 that 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.

4.4. Organizations as biological systems, contingency theory and reconciling perspectives

Morgan's *Images of Organization* (2006) presents a survey of models or metaphors for characterizing organizations, one of which is Ludwig von Bertalanffy's (1950) work comparing organizations to biological systems. As biological systems, organizations are modeled as a collection of interrelated sub-systems with wholes contained within wholes. Individuals are systems living in the context of a group living in the context of a department or larger division. These systems are open to their environment, an "*environment defined by organizations direct interactions with customers, competitors, suppliers, labor unions, government agencies, as well as the larger contextual or general environment*" (Morgan, 2006, p. 38). For the organization to survive, it must achieve an appropriate relationship with its surrounding environment. von Bertalanffy's work was developed by others into what is collectively known as Contingency Theory (Burns and Stalker, 1961; Lawrence and Lorsch, 1967; Woodward, 1958); one of the main themes is that there is no one best way of organizing and different approaches to management may be necessary to perform different tasks within the same organization.

This biological view of the organization is in stark contrast to the Taylorist (Taylor, 1911) scientific management philosophy that characterizes the organization as a machine bureaucracy. Burns and Stalker's (1961) studies of a variety of industries established the distinction between the Taylorist "mechanistic" approach and more contingent "organic" approaches to organizing. Organizations operating in relatively stable environments that are routine and well understood benefit from being organized in a more mechanistic hierarchical way. Innovative organizations, operating in more uncertain and turbulent environments, require a more organic approach that adapts to the environment. Lawrence and Lorsch (1967) further refine contingency theory by suggesting that styles of organization may have to vary between organizational sub-units because of the detailed characteristics of their sub-environment.

If we assume that software development occurs in uncertain and turbulent environments, organizations creating software must be organized more organically, likely with a fair degree of differentiation between the organization's internal sub-units, and these sub-units must interact and develop appropriate relationships with each other. We clearly observed this distinction between organizational sub-units during our study. *Reconciling Perspectives* is one part of the interaction between organizational sub-units, whether those sub-units are individuals, groups, or organizational divisions, because *Reconciling Perspectives* requires those sub-units to maintain an appropriate openness and connection to their environment. The system must manage its relationship to the environment to obtain and reflect on feedback, to reach out, and to negotiate. This is in contrast to closed or so-called "mechanistic systems" where the sub-systems are cut-off from the environment and communication occurs in a strict "chain of command" like style. From their study, Lawrence and Lorsch (1967) declared that an appropriate mode of organization for innovative organizations was the use of "*multi-disciplinary project teams and the appointment of personnel skilled in the art of coordination and conflict resolution*" (Lawrence and Lorsch, 1967)

4.5. Agile methods and reconciling perspectives

Reconciling Perspectives is a lens through which we can interpret the Agile Manifesto (AgileAlliance, 2001a) and Agile Principles (AgileAlliance, 2001b). It is straightforward to connect the *Converging* stage to manifesto articles of "*Individuals and interactions over processes and tools*", "*Customer collaboration over contract negotiations*" and "*Responding to change over following a plan*" In

Reconciling Perspectives, individuals interact to reach out and negotiate a *Consensual Perspective*. Customers are one type of *Acquirers* and, if we do not collaborate with our customers and use a contract to *Cut-Off* ourselves from the customers, there is minimal opportunity for early detection of *Perspective Mismatches*. The same is true with strictly following a plan that can again cut the team off from the organizational *Eco-system. Converging* a *Perspective Mismatch* means responding to change.

Connecting the *Validating* stage to the Agile Manifesto slightly blurs the interpretation because while it is straightforward to connect the Manifesto article "*Working software over comprehensive documentation*" to *Accepting* it is not straightforward to connect *Bunkering* to the Manifesto. *Reconciling Perspectives* suggests individuals and teams need quiet focused time to *Get the Job Done* which contrasts with the emphasis the Manifesto places on interaction and collaboration. The need for open communications is quite clear in our data because the root cause for many project challenges observed during this study was insufficient communication, both in volume and in diversity. However, many individuals were also challenged by too much communication and were unable to focus on *Getting the Job Done*. Too much of a good thing can be a problem.

The need for focused time is perhaps captured in the Agile Principle "*Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done*" which could be interpreted as trust the team and give them the focused time they need to *Get the Job Done*. Agile software methods such as Scrum are more explicit capturing the need for "focus" in the five Scrum values. Moreover, the intention behind the interval from the time a Sprint is planned until the Sprint Demo is that the team has the opportunity to work without interruption and change to their Sprint Backlog; the Scrum Master is tasked with protecting the team from outside interference.

All teams participating in this study claimed to be using Scrum (Schwaber and Beedle, 2002), which is a method for organizing teams to deliver a product or service. 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 sprint planning meeting is an opportunity to detect a *Perspective Mismatch* and *Negotiate Consensus* as the Scrum Product Owner (*Acquirer*) explains backlog items to the delivery team (*Supplier*). Once the delivery team commits to the sprint, then 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, they are presented to the Product Owner for acceptance (*Accepting*). Scrum's short cycle time of 2–4 weeks mitigates *Surprise* during *Validating*. The Scrum Master is a trained facilitator who owns the process, drum beating, and personal strength boosting and, along with the Product Owner, *Shelters* the team from outside interference.

During this study we heard many stories of failed Scrum projects, and we can use *Reconciling Perspectives* to explain these projects. The common thread was teams that cut themselves off and did not did not detect the *Perspective Mismatches*, or did not start the *Reaching Out* process. They went "through the Scrum motions" but did not follow the intent of Scrum. A condition cited for an effective Scrum team is a "strong" and available Product Owner, and our observations support this condition. We observed situations where other strong team members mitigated this situation by stepping into the vacuum created by an unavailable or disengaged Product Owner.

Ineffective Scrum Masters limited the effectiveness of the Scrum process by not taking ownership of the process, and by their inability to facilitate the process. Some merely "chaired" meetings and could not boost personal strength, drum beat, or shelter the team. In many situations, some Scrum Masters simply either let the team become exposed, or completely locked it down, cutting it off from the rest of the organization. While many of these Scrum Masters had Scrum Master training, the training often lacked emphasis on facilitation skills.

Another common failure was long cycle time. While most teams claimed to develop using short sprints, there were many stories and examples of sprint commitments not being reached, and work that was not completed simply being rolled over into the next sprint. In other situations, the *Work Products* demonstrated during the sprint review were incomplete; therefore, they gave a misleading impression to the product owner. The true states of the *Work Products* were not revealed until much later, usually at a significant product release milestone. The result was everyone acting as if they were operating on short cycle times and assuming the benefits of a short cycle time while, in fact, they had very long cycle times.

4.6. Enhancing software team performance

One subject succinctly summed up the source for many of the difficulties experienced in software projects:

> "But there just weren't enough conversations taking place" Site 2, Subject 3

Simply getting people to talk to one another should benefit software team performance. Shared mental model theory predicts higher team performance when the members can have their mental models converge; people's models converge when they observe and engage with each other. This suggests that 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. Lawrence and Lorsch (1967) offer a solution in their recommendation for multi-disciplinary teams and for the use of personnel skilled in the art of coordination and conflict resolution. Scrum, and specifically Scrum's Scrum Master role, may be seen as a manifestation of this recommendation. However, the role of a facilitator is not a role that anyone can assume without appropriate training, and ineffective Scrum Masters were a frequently cited as contributors to software team failure.

While individual traits such as extroversion and introversion will pre-dispose individual behavior, there is precedence from the aviation industry suggesting that training people how to reach out and negotiate can improve team performance. Cockpit Resource Management (or Crew Resource Management) (Wiener et al., 1995) is training that is given to all airline transport pilots, teaching those who have been trained as individuals how to work as a team. The program was instituted after investigating accidents blamed on "pilot error" in an effort to discover what was missing from pilot training. What was discovered was not a deficiency in so-called "stick and rudder skills" but an inability of the flight crew to function as a team.

The effectiveness of team training has lead to other professions where individuals must operate in teams, such as medicine to adopt Crew Resource Management like training. This study demonstrates a strong need for managed communications and yet our profession provides little in the way of the necessary training. For example, the Certified Scrum Master (CSM) learning objectives (ScrumAlliance, 2011) state individuals learn the responsibilities Scrum Master role including:

- serves the product owner and team,
- removes the impediments,

- coaches the Product Owner and team, and
- protects the team.

While these are necessary learning objectives for individuals and teams using Scrum and especially for those individuals stepping into the role of the Scrum Master, they are not sufficient because there is little opportunity within the CSM to teach or develop the necessary skills. From this, we can make two recommendations for improving software team performance:

1) Team training should become part of the regular engineering and software professional education, and
2) For organizations practicing Scrum, greater attention should be paid to the selection of the Scrum Masters, either
   a. select individuals for the role with demonstrated leadership and facilitation skills,
   b. provide facilitation training to stronger team members who have the respect of their peers,
   c. hire or contract individuals skilled in facilitation to serve as professional Scrum masters.

5. Summary

Our study explains how people manage the process of developing software using the process of *Reconciling Perspectives* to remove impediments created by *Perspective Mismatches* that prevent *Getting the Job Done*. The process breaks down when:

- Individuals and teams go dark (McCarthy and Gilbert, 1995) and the process simply does not start because the number of *Communications* is so low or intermittent that individuals are unable to reflect on the *Feedback* they are hearing in the communications and do not realize that there is a *Perspective Mismatch.*
- Individuals lack the *Personal Strength* to reach out; they are aware that something doesn't sound right, but choose not to engage others in an attempt to negotiate a *Consensual Perspective*.
- Others do not agree when those detecting the *Perspective Mismatch* reach out to them.
- Negotiations bog down and do not reach a *Consensual Perspective*.

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

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

(1) A necessary condition for the success of a software project is that there is at least one individual who is sufficiently engaged that they can detect *Perspective Mismatches*, and who 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 number of conversations where people are *Reaching Out* and *Negotiating Consensus*.
(3) Team training and development of principled negotiation skills for software developers has good potential for improving team performance.

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.

Acknowledgements

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

References

Adolph, S., Hall, W., Kruchten, P., 2011. Using grounded theory to study the experience of software development. Empirical Software Engineering, 1–27.
Adolph, S., Kruchten, P., 2011. Reconciling perspectives: how people manage the process of software development. Paper presented at the AGILE Conference (AGILE), August 7–13, 2011.
AgileAlliance, 2001a. Manifesto for Agile Software Development, Retrieved from http://www.agilemanifesto.org/.
AgileAlliance, 2001b. Principles of Agile Software Development.
Bartels, A., Holmes, B.J., Lo, H., 2006. US Slowdown in 2007 will Dampen the $1.6 Trillion Global IT Market. Forrester Research, Retrieved from http://www.forrester.com/Research/Document/Excerpt/0,7211,40451,00.html.
Becker, P.H., 1993. Common pitfalls in published grounded theory research. Qualitative Health Research 3 (2), 254–260.
Boehm, B.W., 1984. Software engineering economics. IEEE Transaction on Software Engineering 10 (1).
Boehm, B.W., Clark, B., Horowitz, E., Westland, C., Madachy, R., Selby, R.W., 1995. Cost models for future software life cycle processes: COCOMO 2.0. Annals of Software Engineering 1 (1).
Boland Jr., R.J., Tenkasi, R.V., 1995. Perspective making and perspective taking in communities of knowing. Organization Science 6 (4), 350–372.
Burns, T., Stalker, G.M., 1961. The Management of Innovation. Tavistok, London.
Cannon-Bowers, J.A., Salas, E., Converse, S., 1993. Shared mental models in expert team decision making. In: Castellan, J. (Ed.), Current Issues in Individual and Group Decision Making. Lawrence Erlbaum Associates, New Jersey.
Chong, J., 2005. Social behaviors on XP and non-XP teams: a comparative study. Paper presented at the Proceedings of Agile Conference, July 24–29, 2005.
Cockburn, A., 2002. Agile software development joins the would-be crowd. Cutter IT Journal 15 (1).
Cockburn, A., 2003. People and Methodologies in Software Development. University of Oslo, Oslo.
Cockburn, A., Highsmith, J., 2001. Agile software development, the people factor. Computer 34 (11), 131–133.
Cronin, M., Weingart, L., 2007. Representational gaps, information processing, and conflict in functionally diverse teams. Academy of Management Review 32 (3).
Curtis, W., Krasner, H., Shen, V., Iscoe, N., 1987. On building software process models under the lamppost. Paper presented at the Proceedings of the 9th International Conference on Software Engineering.
Diaz, M., Sligo, J., 1997. How software process improvement helped Motorola. IEEE Software 14 (5), 75–81.
Dittrich, Y., John, M., Singer, J., Tessem, B., 2007. For the special issue on qualitative software engineering research. Information and Software Technology 49 (6), 531–539.
Dyba, T., Moe, N.B., Arisholm, E., 2005. Measuring software methodology usage: challenges of conceptualization and operationalization. Empirical Software Engineering. 2005 International Symposium on, 17–18 November, 11pp., doi:10.1109/ISESE.2005.1541852.
Emerson, R., Fretz, R., Shaw, L., 1995. Writing Ethnographic Fieldnotes. University of Chicago Press, Chicago.
Firore, S., Salas, E., Cannon-Bowers, J.A., 2001. Group dynamics and shared mental model development. In: London, M. (Ed.), How People Evaluate Others in Organizations. Lawrence Erlbaum, New Jersey.
Fitzgerald, B., 1998. An empirical investigation into the adoption of systems development methodologies. Information & Management 34 (6), 317–328.
Glaser, B., Holton, J., 2004. Remodeling grounded theory. Forum Qualitative Sozialforschung/Forum: Qualitative Social Research in Nursing & Health 5 (2), Retrieved from http://www.qualitative-research.net/fqstexte/2-04/2-04glaser-e.htm.
Glaser, B.G., 1978. Theoretical Sensitivity. Sociology Press, Mill Valley, CA.
Glaser, B.G., 1992. Basics of Grounded Theory Analysis. Sociology Press, Mill Valley, CA.
Glaser, B.G., 1998. Doing Grounded Theory: Issues and Discussions. Sociology Press, Mill Valley, CA.
Glaser, B.G., Strauss, A., 1967. The Discovery of Grounded Theory: Strategies for Qualitative Research. Aldine, Chicago, IL.

Glass, R.L., Vessey, I., Ramesh, V., 2002. Research in software engineering: an analysis of the literature. Information and Software Technology 44 (8), 491–506.
Harter, D.E., Krishnan, M.S., Slaughter, S.A., 2000. Effects of process maturity on quality, cycle time, and effort in software product development. Management Science 46 (4), 451–466.
Hoda, R., Noble, J., Marshall, S., 2010. Organizing *self-organizing teams*. Paper presented at the Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering – vol. 1.
Hsieh, Y., Kruchten, P., MacGregor, E., 2006. Matching expectations—when culture wreaks havoc with global software development. Paper presented at the Work Beyond Boundaries: An International Conference on Tele-mediated Employment and its Implications for Urban Communities, Emergence, Vancouver, Canada, June 15–17, 2006.
Johnson-Laird, P.N., 1983. Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness. Cambridge University Press, Cambridge.
Jones, C., 2000. Software Assessments, Benchmarks, and Best Practices. Addison-Wesley, Boston.
Lawrence, P., Lorsch, J.W., 1967. Organization and Environment: Managing Differentiation and Integration. Harvard University, Boston.
Levesque, L.L., Wilson, J.M., Wholey, D.R., 2001. Cognitive divergence and shared mental models in software development project teams. Journal of Organizational Behavior 22 (2), 135–144.
Lincoln, Y.S., Guba, E.G., 1985. Naturalistic Inquiry. Sage, Newbury Park.
Lister, T., DeMarco, T., 1987. Peopleware: Productive Projects and Teams. Dorset House, New York.
Marshall, M., 1996. Sampling for qualitative research. Family Practice 13 (6).
McCallin, A.M., 2003. Designing a grounded theory study: some practicalities. Nursing in Critical Care 8 (5), 203–208.
McCarthy, J., Gilbert, D., 1995. Dynamics of Software Development. Microsoft Press.
DMcComb, S., 2007. Mental model convergence: the shift from being an individual to being a team member. Multi-Level Issues in Organizations and Time 6.
Moe, N.B., Dingsoyr, T., Dyba, T., 2008. Understanding self-organizing teams in agile software development. Paper presented at the 19th Australian Conference on Software Engineering, ASWEC, March 26–28, 2008.
Morgan, G., 2006. Images of Organization. Updated Edition. Sage Publications, Thousand Oaks, CA.
Mulhall, A., 2003. In the field: notes on observation in qualitative research. Journal of Advanced Nursing 41 (3), 306–313.
Rittel, H., Webber, M., 1973. Dilemmas in a general theory of planning. Policy Science 4, 155–169.
Robinson, H., Sharp, H., 2005. Organisational culture and XP: three case studies. Paper presented at the Proceedings of Agile Conference, July 24–29, 2005.
Sawyer, S., Guinan, P.J., 1998. Software development: processes and performance. IBM Systems Journal 37 (4), 552–569.
Schreiber, R.S., Stern, P.N., 2001. Using Grounded Theory in Nursing. Springer Publishing Company, New York.
Schwaber, K., Beedle, M., 2002. Agile Software Development with Scrum. Prentice Hall, Upper Saddle River, NJ.
ScrumAlliance, 2011. Certified ScrumMaster (CSM) Content Outline and Learning Objectives, Retrieved from http://www.scrumalliance.org/system/resource_files/0000/3640/CSM_Content_Outline_and_Learning_Objectives_Dec2011.pdf.
Shaw, M., 2003. Writing good software engineering research papers: minitutorial. IEEE Transactions on Software Engineering.
Sjoeberg, D.I.K., Hannay, J.E., Hansen, O., Kampenes, V.B., Karahasanovic, A., Liborg, N.K., et al., 2005. A survey of controlled experiments in software engineering. IEEE Transactions on Software Engineering 31 (9), 733–753.
Strauss, A., 1987. Qualitative Analysis for Social Scientists. Cambridge University Press, Cambridge.
Taylor, F.W., 1911. Principles of Scientific Management. Harper & Row, New York.
von Bertalanffy, L., 1950. The theory of open systems in physics and biology. Science 111.
Whitworth, E., Biddle, R., 2007. The social nature of Agile teams. Paper presented at the AGILE 2007.
Wiener, E.L., Kanki, B.G., Helmreich, R.L. (Eds.), 1995. Cockpit Resource Management. Academic Press, San Diego.
Woodward, J., 1958. Management and Technology. Her Majesty's Stationary Office, London.
Zannier, C., Melnik, G., Maurer, F., 2006. On the success of empirical studies in the international conference on software engineering. Paper presented at the Proceeding of the 28th International Conference on Software Engineering.

Steve Adolph is a PhD candidate in Electrical and Computer Engineering at the University of British Columbia and an agile coach with Rally Software. His research interests include software process engineering, requirements management, and software architecture.

Philippe Kruchten is a professor of software engineering at the University of British Columbia, and NSERC Chair in Design Engineering. He led the development of the *Rational Unified Process*.

Wendy Hall is a nursing professor in the School of Nursing at the University of British Columbia. Her research interests include research methods and parent child development.