Empir Software Eng (2011) 16:487–513
DOI 10.1007/s10664-010-9152-6

Using grounded theory to study the experience of software development

Steve Adolph · Wendy Hall · Philippe Kruchten

Published online: 27 January 2011

Guest Editors: Carolyn Seaman, Jonathan Sillito, Rafael Prikladnicki, Tore Dybå, Kari Rönkkö

Abstract Grounded Theory is a research method that generates theory from data and is useful for understanding how people resolve problems that are of concern to them. Although the method looks deceptively simple in concept, implementing Grounded Theory research can often be confusing in practice. Furthermore, despite many papers in the social science disciplines and nursing describing the use of Grounded Theory, there are very few examples and relevant guides for the software engineering researcher. This paper describes our experience using classical (i.e., Glaserian) Grounded Theory in a software engineering context and attempts to interpret the canons of classical Grounded Theory in a manner that is relevant to software engineers. We provide model to help the software engineering researchers interpret the often fuzzy definitions found in Grounded Theory texts and share our experience and lessons learned during our research. We summarize these lessons learned in a set of fifteen guidelines.

Keywords Empirical software engineering research · Grounded theory · Qualitative research · Theory generation

1 Introduction

We are engaged in a qualitative research project to develop a Grounded Theory that explains how people manage the process of software development. Grounded Theory is

S. Adolph (✉)
Electrical and Computer Engineering, University of British Columbia, Vancouver, Canada
e-mail: stevea@ece.ubc.ca

W. Hall
School of Nursing, University of British Columbia, Vancouver, Canada
e-mail: Wendy.Hall@nursing.ubc.ca

P. Kruchten
Electrical and Computer Engineering, University of British Columbia, Vancouver, Canada
e-mail: pbk@ece.ubc.ca

daunting for the software engineering researcher adventurous enough to undertake qualitative research, and can sometimes lead to less than satisfactory results (Adolph et al. 2008). In an Information Systems literature survey of articles published between 1985 and 2007, Matavire and Brown (2008) discovered 126 articles purporting the use of Grounded Theory most of which made "*minimal use of the full Grounded Theory methodology*". Part of the problem is there are few good sources of information to help software engineers learn how to apply the Grounded Theory Method. Many of the leading text books on the topic, and especially those by Grounded Theory co-inventor Barney Glaser, can describe Grounded Theory in near mystical terms and there are only a few practical examples for a novice engineering Grounded Theory researcher to follow (Martin et al. 2009; Seaman 1999; Whitworth and Biddle 2007; Hoda et al. 2010). With few exceptions (Coleman and Conner 2007; Hughes and Jones 2004; Seaman 1999; Urqhart 2001), most scholarly treatments of Grounded Theory are from the disciplines of Nursing and Sociology (Annells 1996; Benoliel 1996; Dey 1999; Schreiber and Stern 2001). These are excellent resources, but have limited relevance for software engineering researchers and do not provide the structure and guidance a software engineering researcher may require to effectively use the Grounded Theory Method.

Notwithstanding these limitations, our experience demonstrates Grounded Theory is an excellent method for studying software engineering and generating theories that are relevant to the practitioner. This paper describes our experience using Grounded Theory for software engineering research, interpretation of Grounded Theory, and recommendations for software engineering researchers considering using Grounded Theory for their research.

Our study was based on Glaserian or "classical" Grounded Theory. Many authors have tried to "pin down", clarify and even enhance Glaser's concepts (Hall and Callery 2001; Morse 2001; Schreiber and Stern 2001) often inviting a critical paragraph from Barney Glaser in one of his books (Glaser 2001). We do not claim to have the definitive interpretation of Glaser's work, but we argue we have effectively interpreted Glaser's method to make it relevant and accessible to the software engineering researcher. We have captured our interpretation in a set of flexible guidelines we found useful for conducting our study.

This paper follows an e-article written by Glaser and Holton (2004) which is an overview of Grounded Theory. While Glaser's e-article aims for a concise overview of Grounded Theory, many ideas were still loosely defined and useful examples for software engineers were lacking. Despite the later divergence between Grounded Theory co-founder's Barney Glaser and Anselm Strauss we quote Strauss in some parts of this paper because his information is sometimes the more salient of the two founders. This paper contributes a model we used to interpret Glaser's writings and the practical lessons learned from our research. Section two provides a brief description of our study and why we chose Grounded Theory. While this paper uses examples from our research to support our description of Grounded Theory, it is not an exposition of our research, which will be published at a later date. Section three is an overview of the Grounded Theory Method, and explains when Grounded Theory is an appropriate research method. Section four is our interpretation of the Grounded Theory analysis model. Section five uses our study as a case study of the many practical issues novice Grounded Theory researchers are likely to encounter. Section six explains rigor and quality guidelines for Grounded Theory. Finally, section seven summarizes our experience and lessons learned with fifteen recommendations for those using Grounded Theory for software engineering research.

2 Our Study

Software development is a risky and expensive proposition. It is only necessary to talk with software developers to quickly learn the quagmire software development sometimes becomes. Late delivery, defective products, cost overruns, and ruined careers and businesses are some of the debris left behind by a failed or less than successful project. This is expensive debris considering global software development is a 1.6 trillion US dollar industry (Bartels et al. 2006). Anything we can do to improve the productivity of software development teams and improve the quality of delivered software will result in significant economic returns.

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

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

Unfortunately most software engineering research is 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). If social factors are the biggest cost drivers and explain variances in the productivity of software development teams, research that helps us identify and understand social processes in software engineering should yield significant benefit to the industry. Yet comparatively little empirical research has been undertaken about the social processes that influence software development. Shaw's (2003) analysis of papers accepted by the International Conference on Software Engineering (ICSE) shows the majority of accepted papers

> …reports 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).

Procedure or techniques and tools and notation accounted for majority of accepted papers (69%), while less 10% of the accepted papers described empirical or qualitative models. Another survey of the software engineering research literature (Glass et al. 2002) discovered that 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.

While more researchers are investigating the influence personal attributes and human relationships have on software projects (Dittrich et al. 2007), Curtis et al. (1987)) retelling of a 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 still sums up the state of software engineering research. 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. The activity of developing tools and methods is what we are educated to do and enjoy doing. However, it is clear that if we are to better understand the factors that have the greatest influence on software development performance then our research must include research into social processes.

The Grounded Theory study we conducted was aimed at understanding the social processes that influence software team performance and the influence software methods have 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; Krishnan et al. 2000). However, other industry data demonstrate software development method's effect on productivity is limited (Jones 2000). Furthermore, the contribution of software methods to team effectiveness is frequently questioned (Alistair Cockburn 2003; Dyba et al. 2005; Fitzgerald 1998).

What is going on here? It is possible low rates of software methods adoption are a strong indicator that software practitioners do not believe their needs are addressed by software methods? While there are anecdotes about methodology usage or lack of usage, there is a gap in our empirical knowledge about the interface between software developers and methods. A Grounded Theory study, with a product of a substantive theory that explains how people manage the software development process, could diminish that gap. The many anecdotes about software development, unlike theory, are not useful to guide policy. Questions arise, such as what are the real 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.

3 What is Grounded Theory?

A Grounded Theory is a set of integrated conceptual hypotheses systematically generated to produce a theory about a substantive area (Glaser and Holton 2004). Grounded Theory is a method for generating theory from data. Glaser and Strauss (1967) proposed the method as having application for both qualitative and quantitative data. When working with qualitative data, and in contrast to other qualitative research methods, such as narrative or ethnography, Grounded Theory generates a substantive theory that explains participants' behavior as a set of integrated hypothesis.

Co-inventors Barney Glaser and Anselm Strauss called the method "grounded" because a theory is systematically obtained from a broad array of data through a rigorous process of constant comparison – it is 'grounded' in the data (Glaser 1965; Glaser and Strauss 1967). Grounded theory differs from logico-deductive methods of inquiry because, rather than develop a theory without relying on data and then systematically seek out evidence to verify the theoretical constructs, researchers using Grounded Theory set out to gather data and then systematically develop a substantive theory directly from the data (Glaser and Strauss 1967, p. pp 4). The goal of Grounded Theory is to generate concepts and categories that account for a pattern of behavior which is relevant and problematic for those involved (Glaser 1978, p. pp 78)

For many of us in the software engineering field, the term theory is a source of confusion with respect to Grounded Theory. Our backgrounds in physical sciences and mathematics tend to lead us to regard a theory as a universal truth like Einstein's theory of relativity. Social theories do not claim to be universal truths and vary in their scope and

generalizability. Grounded Theory generates a mid-level or substantive theory, which describes processes, such as those in social movements, organizations, or communities. A further challenge for engineers who are more comfortable with an objective view of reality is that when studying people we cannot ignore our personal values because personal values influence the ways people interpret reality. In Grounded Theory, however, pre-existing values and knowledge serve only as another source of data and are not privileged. Grounded Theory takes into account personal values and exposure to concepts in particular disciplines through the concept of theoretical sensitivity which Glaser defines as "the research's knowledge, understanding, and skill which foster his generation of categories and properties and increase his ability to relate them into hypotheses" (Glaser 1992).

3.1 Why Grounded Theory?

Grounded Theory is best for answering questions of the form "what is going on here?" Schreiber and Stern (2001) argued it is useful for when we want to learn how people manage their lives in the context of a problematic situation and about the process of how people understand and deal with what is happening to them through time and changing circumstances. Grounded Theory is useful for research in areas that have not been previously studied or where a new perspective might be beneficial.

The appeal of Grounded Theory to the software engineering researcher is that Grounded Theory generates theory rather than just testing it. Grounded Theory also possesses a visceral appeal for software engineering researchers because theory is generated by following a rigorous method, much of which is derived from Paul Lazerfeld's (Lazerfeld and Wagner 1958) teaching of qualitative mathematics; Barney Glaser was one of his students.

Grounded Theory is an inappropriate form of inquiry for answering a question such as "are pair programming teams more effective than individual programmers?" Grounded theory is appropriate if we are interested in explaining how pair programming unfolds by asking: "How do individuals manage the process of pair programming" The answer to such a question helps inform the creation of strategies for managing pair programming. For the purposes of our study, low rates of software method adoption suggest a disconnection between the practices software methods prescribe for managing the software process and the practices people actually execute. We wanted to understand the central concern of those involved in software development and how they resolved their central concern, with the intention of being able to inform software development policy.

3.2 Grounded Theory Overview

Grounded Theory is like agile software development in the sense that it is deceptively simple conceptually, yet rigorous and disciplined in practice. Figure 1 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 that are in turn aggregated into categories.
B. The theoretical properties of the 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 is a category that captures the most variation in the data (Glaser 1978) and addresses the main concern of the study participants. The process continues until the categories become "saturated", that is,

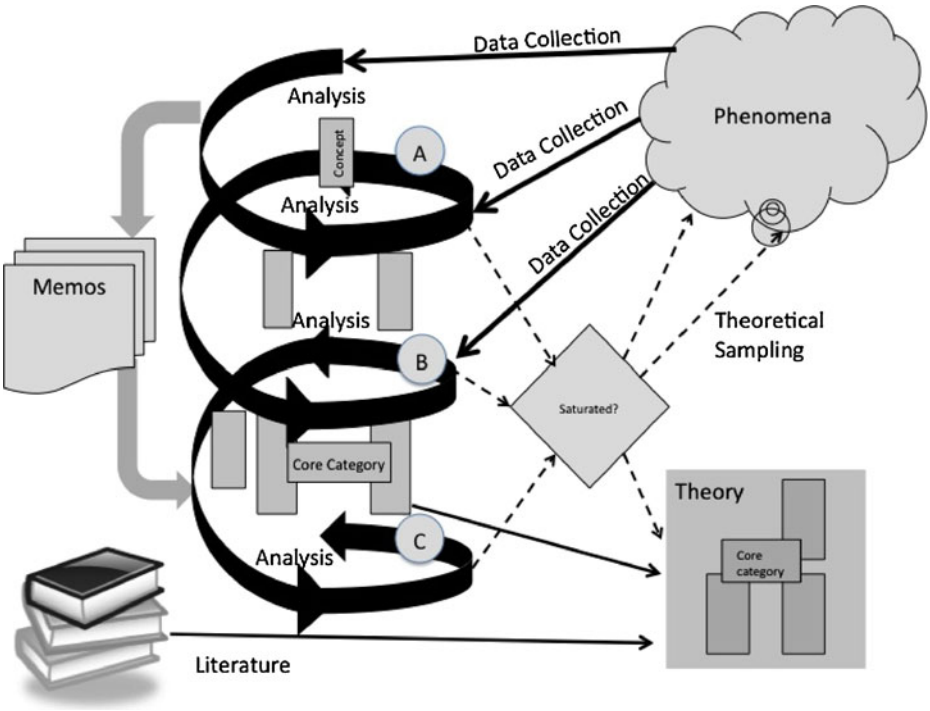

Fig. 1 The grounded theory method

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 theory is compared to theories described in the literature. The literature search is deferred to late in data collection to avoid forcing pre-conceived theories on the substantive theory being developed from the data.

Throughout the process, the researcher writes memos, capturing his or her thoughts that support the emerging concepts, categories and their relationships.

From this overview, one could regard producing a Grounded Theory as straight forward, 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 or theoretical sensitivity on the part of the researcher to develop the theory. Theory development most certainly did not progress in a straight forward linear manner. Rather, it involved many false leaps, blind alleys, and dead ends. Nonetheless, it was a very, very rewarding experience that enabled us to explain the experience of software engineering from a very different perspective.

3.3 Which Version of Grounded Theory?

What confuses most novice Grounded Theory researchers is there is not one, but at least three similar, yet distinct research methods that claim the name Grounded Theory. The original method was described in the Glaser's and Strauss's book "The Discovery of

Grounded Theory" (Glaser and Strauss 1967). Strauss later considered ways to formalize and proceduralize grounded theory and published *Basics of Qualitative Research* with Juliet Corbin in 1990 (Strauss and Corbin 1990). Later Kathy Charmaz made an effort to clarify some of the ontological and epistemological ambiguities underlying Grounded Theory by publishing *Constructing Grounded Theory* (Charmaz 2006).

There is a voluminous and sometimes shrill debate in the Grounded Theory community about which method(s) should carry the label of Grounded Theory. Most software engineering researchers using Grounded Theory tend to cite Strauss and Corbin (1998), likely preferring Strauss and Corbin's more structured and proceduralized approach to analysis. We chose the classical Glaserian approach for two pragmatic reasons: there was less likelihood of imposing pre-existing categories on the data (Glaser 1992); and there were more resources available to us in terms of mentors, seminars, and websites based on the Glaserian approach. For example, our university has a world class research nursing school which has significant experience using classical Grounded Theory.

4 The Grounded Theory Analysis Model

Engineers expect rigor and clarity in specification and an impediment to using Grounded Theory and especially classical Grounded Theory for software engineering research is the loose and sometimes near mystical descriptions of the Grounded Theory Method. Terms are frequently used interchangeably and it is expected the researcher approaching Grounded Theory is already well versed in sociological research. In this section we clarify this situation by offering our interpretation of the Grounded Theory analysis model.

4.1 Concept-Indicator Model

Grounded Theory is based on a concept-indicator model of constant comparisons of incidents or indicators to incidents (indicators) as shown in Fig. 2. Concepts are often behaviors or factors affecting behaviors, which help to explain to the analyst how the basic problem of the actors is resolved or processed (Strauss 1987, p. 33). As concepts are generated the researcher begins comparing incidents (indicators) to the emerging concept, see Fig. 3 (Glaser 1978, p. 62). Indicators are actual data, such as behavioral actions and events observed or described in documents and in interviews; they are often captured by the

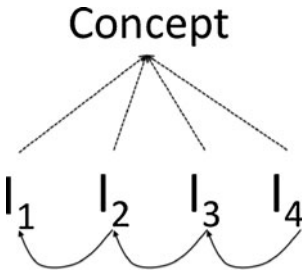

Fig. 2 Concept indicator model comparing indicator to indicator

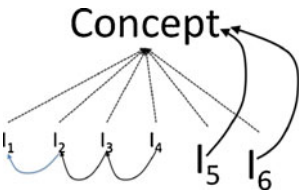

Fig. 3 Concept indicator model comparing subsequent Incidents with concept

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. Glaser, in his usual mystic writing style, never quite definitely states what a concept is; however, for the purpose of our study, we loosely regarded a concept as an abstraction suggested by the indicators. We elaborate on this topic in section 4.2.

When reading grounded theory texts, an impression can be created, particularly in Strauss and Corbin's (1998) writing, that indicators represent minute data fragments, because of the frequent admonition of line-by-line comparison and examples provided. In actuality, indicators are exactly what the term implies; they indicate some event or behavior which may be indicated by a word, a sentence, paragraph, or even a set of paragraphs. Some examples from our research are included in Table 1.

4.2 Concepts, Codes, Categories, and Properties

Concepts and categories are not well defined and often it seems the terms are used interchangeably. Glaser and Strauss originally stated a category "*stands by itself as a conceptual element of the theory*" (Glaser and Strauss 1967). Later Glaser said a category is a "*type of concept that is usually used for a higher level of abstraction*" (Glaser 1998, p. 38) and in later writings Glaser defined a concept as a "*naming of an emergent social pattern grounded in the research data*" (Glaser 2001).

Interpreting this set of fuzzy definitions for concepts and categories, we chose to operationally model a concept as a Unified Modeling Language (Booch et al. 1998) class that represents a pattern of behavior as suggested by a set of indicators. Codes were simply an evocative name for the indicator. Concepts were given names that were evocative of the patterns of behavior suggested by the indicators. We further chose to regard categories at a higher level abstraction where they aggregate a set of concepts. This relationship between concepts and categories may at first appear counter-intuitive because a category represents a higher level of abstraction than the concepts it is aggregating; however, a category is a specialization of a concept. Viewing concepts and categories through this model helps clarify our understanding of properties, what Glaser and Strauss referred to as "*a conceptual aspect or element of a category*"(Glaser and Strauss 1967). We chose to view

Table 1 Example indicators and codes

Brainstorming	*"I do some brainstorming as to what they want or what they really want"*
Tracing	*"So, I spent quite a bit of time kind of tracing through code to try to figure out what each thing you know kind of did"*
Experimenting	*"...so they were experimenting with different ways to you know track issues, um, we also had problems with the branching code and tracking you know changes to multiple (unclear) code"*

properties from the UML perspective of an "attribute". Glaser and Strauss often make reference to sub-categories which we thought of as a "whole-part" relationship between categories. Figure 4 is a low fidelity UML representation of the way we chose to reason about these relationships. Our model was intended to help us reason about these fuzzily defined relationships and is not intended to be definitive.

Concepts are the building blocks of a Grounded Theory and conceptualization is a distinguishing trait of grounded theory. The two most important properties of conceptualization for generating Grounded Theory are that concepts are abstractions of time, place, and people, and that concepts have enduring grab (Glaser 2001). To crystallize the importance of conceptualization, a favorite question of Glaser's during a seminar discussion is "*how would you conceptualize that?"* If we cannot conceptualize, we can fail to see the common threads running between incidents and concepts that can be collapsed into categories. Only conceptualization results in a parsimonious and conceptually dense theory.

Conceptualization should not be a stranger to software engineers. Conceptualization is what we routinely do to create cohesive, loosely coupled systems. We hopefully do not create completely separate designs of a system for a web interface, a windows client, and a CRM system. Rather we abstract and model all of these in terms of an external interface which is a conceptualization of the web interface, the Windows client, and the CRM system.

Naming a concept is sometimes problematic. While we tended to use in-vivo inspired codes to capture incidents (see Table 1), the codes act as indicators which cannot capture the full scope of the emerging concept. This is where Glaser again asserts the importance of the constant comparative method

> The pattern is named by constantly trying to fit words to it to best capture its imageric meaning. This constant fitting leads to a best fit name of a pattern, to wit a category or a property of a category. Validity is achieved, after much fitting of words, when the chosen one best represents the pattern (Glaser 2001)

In our study we saw the indicators "tracing through the code library", "spikes", and "JAD sessions" as suggesting the concept of "Scouting". In all of these incidents, participants were engaged in an exploration to find answers, often with many blind alleys and dead ends. "Scouting" developed as a concept that categorized items where the purpose

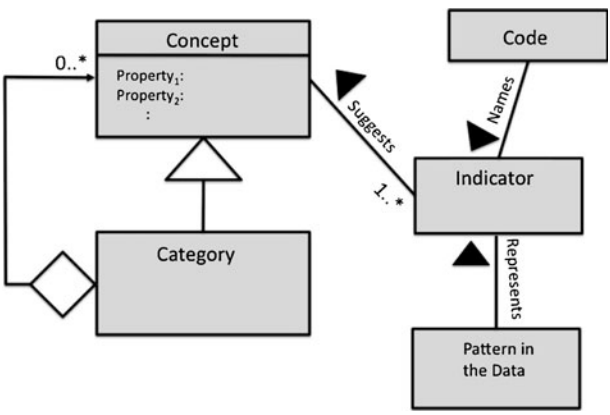

Fig. 4 UML model of indicators, concepts, and categories

was to obtain information, but there was uncertainty about the resources and time required to obtain that information. All of these activities were characterized by the excitement of discovering new knowledge, but also the frustration of managing "Scouting" in a resource limited environment. Moving from the descriptive to the conceptual helps create a dense parsimonious theory.

4.3 Constant Comparison

Constant comparison is a systematic way to generate concepts from incidents. In the words of Grounded Theory co-discover Anselm Straus (Strauss 1987, p. 25)

> Initially, indicators are compared to indicators, where the analyst is forced to confront similarities, differences and degrees of consistency of meaning among the indicators. Through this laborious and often tedious process, concepts are generated and indicators are then compared to the emerging concepts. This sharpens and clarifies the concept to achieve the best fit to the data. Meanwhile further properties of the category are generated until the codes are verified and saturated yielding nothing much new. In this model the concepts have "earned their way into the theory by systematic generation from data

Constant comparison is another distinguishing trait of Grounded Theory because Grounded Theory is much more than simply a search for patterns in the data. It is a continuous process of generation and verification that helps purge the emerging theory of preconceived concepts and requires concepts to earn their way into the theory. New data is constantly compared against previously observed data and concepts. Concepts that are not sustained by the data (single indicators or weak patterns) soon drop out. If followed rigorously, constant comparison makes Grounded Theory self-correcting — a feature that cushioned us more than a few times.

Comparison of the indicators from Table 1, specifically, "Brainstorming", "Tracing", and "Experimenting", gave rise to the concept of "Scouting", an activity to discover a path through unknown or unfamiliar territory. This is illustrated in Fig. 5. Further comparison began to develop dimensions for this concept, for example, "Brainstorming" and "Experimenting" versus "Tracing": the purpose of "Brainstorming" and "Experimenting" is to generate candidate solutions, whereas "Tracing" is forensic, trying to discover how something works and what side-effects a solution may have. This gives a qualitative range for the purpose of "Scouting", from "forensic" to "generative" (Fig. 6).

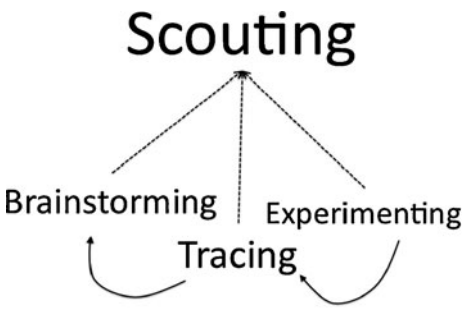

Fig. 5 Constant comparison and concept emergence

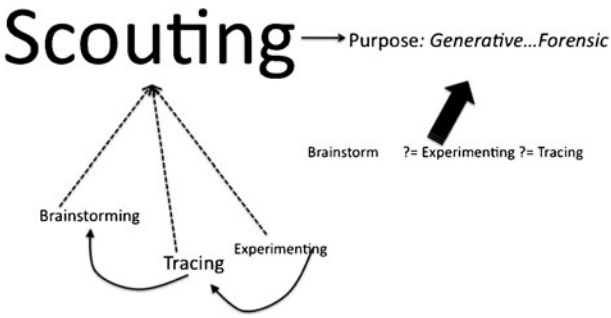

Fig. 6 Constant comparison and generation of properties

As the concept begins to emerge, subsequent indicators are compared to the concept. For example:

Workshopping	*during those workshops we don't reduce any-any scope or anything like that. So it's more of a like—okay we want to do this so what would it involve—to do this*

When we compared "Workshopping" to the existing "Scouting" concept it helped us develop the category by adding another dimension to it we named "plurality" which ranged from "individual" to "group". For example, is "Scouting" done by individuals as in "Tracing", or is it conducted as part of group exercise as in, "Brainstorming" and "Workshopping"? (Fig. 7)

Subsequent indictors further dimensionalized the "Scouting" concept to include:

- Time Continuity - whether scouting took place as a short term continuous effort or as a frequently suspended long term effort.
- Ownership – who declared the scouting effort complete, the scouter or someone else?

4.4 Saturation and Interchangeable Indicators

In comparing incident to incident when coding data, the researcher begins to see a pattern and to develop categories that fit it. In continuously comparing further incidents to concepts, the researcher starts seeing the same pattern over and over again in different ways (Glaser 1998). Saturation occurs when, as more data are collected and analyzed, nothing

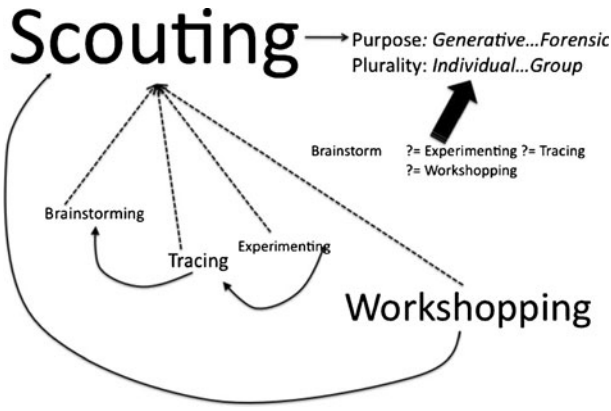

Fig. 7 Constant comparison and further elaboration of properties

new is added to a category. Saturation occurs because the incidents are interchangeable. Indicators for events and behaviors may vary in detail or be repetitious, but add up to the same thing analytically. Further collection of data does not further elaborate the category.

We discovered categories tended to saturate very quickly. With less than five interviews, we started only seeing minor variations in the indicators for "Scouting". Codes such as "JAD Session", "Walk-Through", and "Code Read", all seemed to be variations of existing indicators, and while they strengthened our confidence in the category, additional collection of data added nothing new.

4.5 Theoretical Sampling

In any research project, sampling strategies and the adequacy of the selected sample affect the validity and reliability of the study. For a Grounded Theory study, the population under study is the set of concepts that constitute the phenomena rather than individuals experiencing the phenomena. Therefore, the sampling strategy must make sufficient concepts available to develop a conceptually dense substantive theory. Grounded Theory employs theoretical sampling where the analysts "*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). The analysis process and the developing concepts direct the investigator to sample for further incidents to develop properties and dimensions.

Theoretical sampling is different from selective sampling, which uses a predetermined selection process. By sampling individuals, a researcher has the potential to repeat samples of the same concepts, without adequately sampling the concepts in the problem space to produce a Grounded Theory. According to Glaser "*taking a random sample of a set number of people does not make sense in grounded theory*" (Glaser 1998, p. 159).

Theoretical sampling is driven by emerging categories and hypotheses, and is an ongoing process that cannot be pre-determined. It is a critical practice supporting the development of an emerging theory that is tightly intertwined and supportive of the iterative and concurrent collection of data.

> By responding to the need for theoretical completeness, theoretical sampling directs the researcher to new data sources which constantly generate the parsimony and scope of the theory as it accounts for how the main concern is constantly resolved (Glaser 1998, p. 158).

We found starting the sampling process 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" sampling (Marshall 1996) 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 codes for how people practiced software development, and quickly saturated the categories directly related to how people create software. At a conceptual level, people create software much the same way.

The transition from judgmental sampling to theoretical sampling happened quickly and definitively after the first five interviews. Even during this early phase, we were always modifying our interview and observation protocols; however, it was within these first interviews we began to notice the emergence of "Uncertainty" and "Perspectives" as categories and this is when the emerging theory really began to drive our sampling process. With the categories representing the practice of software development completely saturated,

there was no point in continuing to ask people "how do you manage the process of software development". Rather, during the next phase of data collection we began to ask more about how people worked with the uncertainty of software development so we could develop the emerging "Uncertainty" and "Perspectives" categories.

4.6 Theoretical Sensitivity

Glaser describes theoretical sensitivity as "*the process of developing the insight with which a researcher comes to the research situation*" (Glaser 1978). Grounded Theory co-founder Anselm Strauss defined theoretical sensitivity as "a personal quality of the researcher. It indicates an awareness of the subtleties of meaning of data. ...[It] refers to the attribute of having insight, the ability to give meaning to data, the capacity to understand, and capability to separate the pertinent from that which isn't" (Strauss and Corbin 1990, p. 42).

Theoretical sensitivity is often referred to as a creative aspect of Grounded Theory because it involves the researcher acknowledging experience and expertise developed in the field of study. The Grounded Theory researcher requires creativity to "see" the recurring patterns of behavior and events because his or her reflections about what is happening in the data drives theory development. Theoretical sensitivity relates to the choice and appropriateness of imagery evoked by concepts and their names.

As a matter of practice, theoretical sensitivity is managing the balance between awareness and the extent to which awareness affects theorizing. A Grounded Theory researcher must develop theoretical insight and make something of that insight (Glaser and Holton 2004), but theoretical insight is one more source of data, which cannot be privileged over other sources. In other words, researchers cannot impose their preconceived ideas and biases on the emerging theory.

Theoretical sensitivity makes Grounded Theory research very much a solitary effort. The insight and perspective of the individual researcher is what gives insight into the data. We are aware of instances where individuals have collaborated to develop a Grounded Theory. Often these groups have resorted to pre-defining codes such that all individuals code data in the same way. This is forcing pre-conceived concepts on the data.

5 Grounded Theory in Practice

The Grounded Theory texts and papers did not prepare us for the difficulties and impediments we encountered during our study. In this section, we offer exemplars from our study as an experience report of doing Grounded Theory for software engineering research. Many of our lessons learned are summarized in section seven.

5.1 Getting Started

5.1.1 The Inception of the Project

Some of us had never conducted sociological research before so we followed Fred Brooks famous words "*plan to throw one away*" and threw away two studies: the First Report (Adolph et al. 2008), and an unpublished field study. Both of these early studies were small and provided the opportunity to learn, fail safely, and obtain feedback to hone our skills. For anyone considering using the Grounded Theory method for the first time, we strongly recommend a pilot study to practice and develop skills.

Furthermore, we decided to reach out beyond our engineering department and seek advice from others who are expert and have experience conducting Grounded Theory research. Grounded theory is used extensively in nursing research and our University has a world-class research Nursing school. We were pleasantly surprised and greatly appreciative by the enthusiasm others have to reach across research disciplines. If there is only one lesson to learn from our experience is important engineering research resources are available outside the engineering department.

5.1.2 The Research Problem

Grounded Theory is a method that permits researchers to discover an actual problem that exists for the participants in a substantive area rather than what professional researchers may believe is the participants' problem. Therefore, in a Grounded Theory study, the researcher works with a general area of interest rather than with a specific problem (McCallin 2003). This does not mean there is no specific problem, but as Glaser writes "*This problem and its processing will emerge in the initial stages of the research. And it will emerge if it is not derailed by what the researcher thinks is relevant beforehand and forces it on the study*" (Glaser 1998, p. 116). In commenting on problem emergence in his own study 'awareness into dying' (Barney Glaser and Strauss 1965) Glaser states: "*we had no idea that awareness was a problem that the medical team had to handle constantly and that they did so by generating awareness contexts*" (Glaser 1998, p. 119).

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

5.1.3 Ethics

Most universities have research ethics boards, which review and approve all research that involves human subjects. While the potential for grievous indiscretions made in human psychological research make it clear why such controls are necessary, those of us engaged in software engineering research may regard the ethics process as an annoyance or another bureaucratic impediment. Software developers are not an underprivileged or easily abused minority; however, Singer and Vinson (2002) explained an ethics review process is necessary even for software engineering research.

5.1.4 Tools

Contrary to Glaser's advice not to record interviews, we digitally recorded interviews, which were then transcribed. Some of us do not have the quality of personal memory required to reconstruct an interview strictly from the quick jotting of field notes. This did cause us some grief later in when we interpreted the "line by line" coding instruction literally because of the volume of data. Nonetheless, we still believe, despite Glaser's recommendation, recording and transcribing interviews is the most effective way to capture interview data.

While we did not use specific qualitative computer-based analytic tools, we did use computer support for managing our data. All interviews and field notes were transcribed into MS Word and were then stored in a Wiki. We used the MS Word "comment" feature to tag incidents and write short memos. As concepts emerged, memos describing them were written up in the Wiki and then manually linked to the source document(s).

We trialed several tools for our First Report, and during our field study and found the tools limiting because they tended to distance us from our data. Manually transcribing and linking the data between the MS Word documents and the Wiki while tedious, enabled us to immerse ourselves in the data and paced us. Tools can create the illusion that more is somehow better because the tools make it appear easier to quickly handle larger volumes of data while distancing oneself from the data. This can result in a limited or cursory analysis of large volumes of data and equating research quality with number of interviews.

5.2 Data Collection

5.2.1 Field Access

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 and did not have to conduct any "ambush" interviews. Rather, we had the opportunity to be on site for extended periods of time. Mulhall (2003) raised some concern about selecting field sites simply because they are accessible to the researcher; however, when the choice is between ambush interviews and an opportunity for an extended engagement, then accessibility wins. In the write-up of our results we clearly state one of the major reasons field sites were chosen was because we could obtain access. In the selection of potential field sites, we also used judgment to obtain some variation in the types of organizations participating in the study. We were able to collect field data from three different organizations, an onsite customer support field office, a small product company, and a software research and development center for large multi-national software product vendor.

Once we had access to a site, we 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 and stopped our visitation schedule. Unfortunately the result of such stoppages was difficulty in re-engaging the site. Once a researcher is out of sight, they are out of mind, especially for busy software development team. We found that simply being on site, even if we were not explicitly engaged in data collection helped build trust and familiarity, which made it safe for people to approach us and participate in the study. We were pleasantly surprised by the openness and candidness of many interviewees, as well as by the openness of the organizations.

5.2.2 Data Collection Methods

We collected data using both semi-structured interviews and participant observation because interview data can provide a superficial description of a phenomena and accesses only what participants say they do. Participant observation, which allowed us to observe what people did, is an important mainstream data collection method rather than a mere supplement to interviews. We considered the participant observation invaluable because it also demonstrated how participants interacted. We were able to observe phenomena that would not be uncovered in an interview, from the frustration with the unreasonable demands of an important client to intense negotiation during a planning meeting. We were

able to observe who were the group's "go to" people and how they put aside their tasks to help others, spontaneous problem solving meetings, and numerous Nerf dart wars. Without participant observation, it is likely we would have only generated a theory of how people create software and not how they managed it. Nonetheless, because Grounded Theory is intended to determine, in part, how individuals make meaning of a situation, perceptions can only come from interviews. One question that becomes quite important during the interview is asking "why?"

On average, we collected 15 h of observation data for every hour of interview time. A pattern of data collection that evolved was for us to observe a meeting, and then during an interview with someone who participated in that meeting, ask questions about their role in the meeting and their interpretation of what took place.

5.3 Analysis

Analysis develops a relationship between participants, meaning and their actions. We explored meaning and actions in the data by looking for similarities and differences within and between interview transcripts and observation field notes. While the various flavors of Grounded Theory introduce specific practices and nomenclature for analysis, we found all follow the same basic approach of fracturing and integration. Line-by-line coding fractures the data enabling the analyst to develop categories and their properties (Glaser 1992). This develops descriptive codes as labels for the indicators that are evocative of the patterns in the data so that a cluster of similar meaning units can be categorized (Glaser 1992). We developed categories and their properties and integrated them by testing relationships and thus developing the theory.

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 codes for incidents that can be clustered into concepts and categories. Selective coding involves identifying the core category, the category that explains the greatest variation in the data and around which the emerging theory is built around. Theoretical coding conceptualizes how substantive codes relate to each other as hypotheses.

5.4 Generating Concepts - Open Coding

Open coding generates concepts from the data that will become the building blocks for the theory. During open coding, concepts are generated by asking generative questions (Glaser 1978, p. 57) "What is this data a study of?", "What category does this incident indicate?", "What is actually happening in the data?"

Our first approach to open coding was to read the interview transcripts to get an overall impression – something Glaser advises against, advocating "line-by-line coding …. ensures the grounding of categories beyond impressionism" (Glaser 1978) With the first few interviews, this impressionist approach inspired the generation of numerous speculative concepts, supported by dozens of memos. Many of these memos were short jots speculating what a passage in an interview may indicate. After reading two transcripts and generating over 60 memos, we realized we were trying to force our "pet" themes on the data. Many of our early concepts were supported by only a single incident. We recognized the value of the constant comparative method in demanding concepts "earn their way into the theory". After changing the approach, many codes simply fell by the wayside. We could not find other incidents to support them or there was not enough variation to differentiate the incident from others we had identified.

We followed Glaser's advice and ensured we were carefully coding line-by-line. This does not mean we expected to label every line of data with a code. Rather, we carefully inspected every line for incidents because an incident may be discovered in a word, a line, or across many lines. When we discovered an incident we labeled it with a code using the MS Word comment feature and compared the newly discovered incident to other incidents or categories. If the incident suggested a new concept then we wrote a memo to capture our ideas for the concept and added that to our Wiki. Open coding appears simple and yet is a difficult tedious enterprise, comparing incidents to other incidents and concepts; we sometimes took nearly a week to properly code a one hour interview transcript, although some interviews were coded in about two days.

5.4.1 Discovering the Central Concern: Selective Coding

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 result of analysis is perhaps, at best, an interesting collection of themes which support thick description, but not a well integrated set of hypothesis that explain how the various categories operate. Furthermore, without a core category to subsequently focus the researcher's efforts, the researcher's energy can be dissipated into a thin inconsequential theory.

The core category is the one category among all the categories generated during analysis that, in addition to other qualities, integrates the categories and is centrally relevant. Glaser provides some 11 criteria for choosing the core category – the most important of these criteria to us is the core category be the "most connected". The core category reoccurs frequently in the data and comes to be seen as a stable pattern that is more and more related to other categories. The generation of theory occurs around the core category. According to Glaser, the first delimiting rule of Grounded theory is "only variables related to the core are included in the theory" (Glaser 1978, p. 93)

We regard the core category like a center of gravity integrating closely related categories, rather than having a loosely related set of categories that fail to explain how a problem is being managed. When the core category is selected we develop categories relevant to the core category and to drop more distant, less relevant categories. Selective coding aids theoretical sampling because it focused our subsequent data collection.

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) whose solution requires an unpredictable exploration of unknown and possible dangerous territory. "Scouting" was certainly a recurring pattern in our data, which met some of the criteria for a core category. 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*".

After several more false starts, the concept of "Bunkering" began to emerge as another candidate core category. We conceptualized "Bunkering" as the process of how an individual or even a group protected themselves from complexity and uncertainty by limiting their scope of concern and by managing their communications. We invested considerable effort developing the "Bunkering" category because Bunking 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 trying to build the theory from were too literal. We resolved this issue by re-asking the generative question "what is actually happening in the data?" There was much happening, but what was core? From this reasoning the category "Reconciling Compromise", a process that participants use to reconcile their different points of view and negotiate compromises, quickly emerged. This category explained the greatest variation in the data rather than all the data. It was surprising, once we selected 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.

5.4.2 Relating the Concepts: Theoretical Coding

"*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" which may be theoretically coded as causal and associated with degrees. The more experience an individual has, the greater the likelihood they will reach out to others find compromises.

In his book entitled "Theoretical Sensitivity" (Glaser 1978) offers analysts coding families for organizing and associating categories. At first glance, it appears these coding families contradict Glaser's repeated admonitions about forcing and pre-conception. Glaser justifies the coding families by arguing theoretical codes are not paradigms imposed on the emergent theory rather they are accessed based on cues in the data. The use of theoretical codes explicates the nature of relationships. Theoretical coding families sharpen the relationships relationship between categories because when they are implicit they are weak.

In our emerging theory there is strong sense of movement between stages. When people discover "Getting the Job Done"is impeded by a mismatch between them and their client's understanding of the job, they reach out and attempt to broaden their perspective. They generate compromises in an effort to remove the impediments and then use those compromises to "get the job done". The results are presented for feedback. From these cues, the best fit theoretical code for our emerging theory is 'process' which helped us crystallize how the theory explains a phenomena with multiple stages that occurs over time. 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". For us,theoretical codes illuminated relationships between categories and provided a vocabulary for describing these relationships.

5.5 Memoing

Memos capture, make real, and preserve the ideas emerging from an analyst's pre-conscious processing as the data are analyzed (Glaser 1998). Whenever an idea emerges about a concept or the connections between concepts, analysts should stop coding and write a memo because memoing represents the ideas the analyst is developing from the data.

Ideas are fleeting and fragile, whereas the data will always be there. Although typically based on description, memos help raise description to a theoretical level through the conceptual rendering of the material. The writing of theoretical memos is the core stage in the process of generating theory. If an analyst skips memoing, he/she is not doing grounded theory (Glaser 1978).

In our study, we took Glaser's advice on memoing "to the max" and wrote memos whenever an idea "popped" into our heads. During early open coding, we wrote a large number of what were rather speculative memos. An arbitrary word or phrase in a transcript could trigger an uncritical "aha moment" and this large number of memos produced concepts not grounded in the data and some flights of fancy. This is a healthy process because, as long as we remember that concepts must earn their way into the theory through constant comparison, such a process explicated our biases. Our memos fell into three broad categories:

1. Category Memos: As categories were developed during constant comparison, the memos captured the concepts. Initially concept memos simply served as a glossary, but as we became more mature in the process, the memos became much richer, describing categories in dimensional terms, their properties and relationships to other categories. Our theory is built from category memos.
2. Process Memos: We kept extensive process memos that served as an audit trail of our activities and decisions. Much of this paper is derived from our process memos.
3. Speculative Memos: We created speculative memos when we were inspired by an idea. These memos may not make a contribution to the theory but they captured interesting insights that emerged from the data and revealed biases.

Memos are not pretty things. They are written in the passion of the moment and can be short "jots" or long rambling prose, full of misspellings and grammatical errors. They are a tool for the analyst to raise description to the theoretical level through conceptual rendering and not for immediate sharing of ideas (Glaser 1998).

While memos are written ad hoc, they are not undisciplined. There is a need to use some caution to avoid smuggling extant theory through the back door (Glaser 1998, p. 181). In one case, we used Herbert Simon's concept of satisficing (Simon 1947) as short hand to characterize how people manage the process of software development by trying to find a reasonable solution, without wasting resources to find the best solution. The concept of satisficing does not belong in our theory because we were imposing a concept from an existing theory on the substantive theory. In the pure sense, a Grounded Theory stands on the data on which it is grounded, rather than being derived from an existing theory. There is room for insights and reflections about the theory in discussion sections of papers and dissertations.

5.6 Sorting and Writing

Analysis develops categories and the relationships among those categories that are captured and expressed in memos, which informs the theory. The analyst sorts the memos to organize them into the outline of a theory for writing and presentation. There are no pre-conceived outlines because the outline is generated through the sorting of the categories and properties in the memos into similarities, connections and conceptual orderings (Glaser and Holton 2004).

The classical approach to sorting is to dump written memos on a dining room table and manually arrange them topologically, looking for connections, and the strength of those

connections.[1] We sorted our memos using the hierarchy or tree view in our Wiki as the "table top" and then simply moved memos up and down the memo hierarchy.

5.7 Literature Search

Glaser (1978) argued that extensive reading in the substantive area, in the form of a literature review, could unduly influence Grounded Theory research through forcing extant theoretical concepts on the collection and analysis of data. Thus, to undertake an extensive review of literature before developing a core category violates a basic premise of Grounded Theory — that theory emerges from data not from extant theory. An extensive literature review could also run the risk of clouding the researcher's ability to remain open to developing a completely new core category that has not figured prominently in the research to date, thereby thwarting theoretical sensitivity (Glaser and Holton 2004). On the other hand, researchers, and particularly doctoral students, must situate their work in the context of the extant literature; Glaser's position cannot be used to avoid important claims of the significance of the work and the potential contribution.

Exhortations to delay the literature review should not be viewed as an outright ban on reviewing the literature. Traditional logico-deductive research starts with existing theory with the intention of affirming or denying a theory, or extending an existing theory. Accurately placing the proposed research within existing knowledge landscape does not preclude looking at a topic differently or exploring an area where the theoretical landscape is not well developed. The literature search could arguably be used to increase the theoretical sensitivity of the analyst.

Our literature search took place in two phases, the first was a "framing phase" that identified the general space we were interested in exploring. Our search identified empirical software engineering research papers that explicated where researchers had gone before and methodology papers that explicated how Grounded Theory was being employed in the software engineering community. The second more traditional phase of a literature search was conducted after the theory was developed and stabilized. It was intended to compare and contrast our theory with the existing knowledge landscape.

To avoid importing or comparing ideas from familiar theories when conducting Grounded Theory research, constant comparison serves as a self-correcting mechanism for the theory. Categories must "earn their way" into the theory. In our case, significant prior experience working with rapid decision making theories and shared memory and our belief that any theory would incorporate these ideas was disrupted by rigorously following the constant comparison method.

6 Grounded Theory Rigor and Quality

Qualitative research is often viewed with discomfort in software engineering because of misunderstandings regarding validation and verification. We have observed several situations in which Grounded Theory researchers were asked to "validate" their theory using quantitative mechanisms. This demonstrates a misunderstanding of Grounded Theory because unless the researcher by-passed the constant comparative method and fabricated their concepts and relationships, the theory is grounded in the data and is a valid explanation of how people solve their problems.

[1] Any researcher with a toddler at home may not find this method practical.

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

Lincoln and Guba (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. Gasson effectively compared these aspects to criteria software engineering researchers may be more familiar with in Table 2 (Gasson 2003)

Many software engineering researchers regard generalizability as a trait of a good theory because if we cannot know the scope of applicability of a theory, then what is the use of the theory? This is usually the motivator for researchers attempting to validate their Grounded Theory through statistical methods. Generalization unfortunately usually requires stripping away the very contextual nuances we are interested in revealing in qualitative research. Discovering the percentage of software developers who enjoy pair programming is not answering the question how do developers manage or experience the process of pair programming.

Transferability is a better criterion for the generalizability or utility of qualitative findings and places the onus for generalizing of the results on the reader of the research results rather than researcher. Transferability requires the researcher provide sufficient description of the context in which the results obtained so that other researchers can then compare that context to one they wish to transfer the results to. In our view, transferability is more than just enabling another researcher to transfer the results to another situation. It is the ability of a practitioner to read the results and exclaim as one did “wow that’s my life!” realizing the results have relevance for them.

Table 2 Gasson’s comparison of Lincoln and Guba’s trustworthiness criteria to traditional research rigor criteria

Issue of Concern	Positivist World View	Interpretive World View
Representativeness of findings	***Objectivity**: findings are free from researcher bias*	***Confirmability**: conclusions depend on subjects and conditions of the study rather than the researcher*
Reproducibility of findings	***Reliability**: the study findings can be replicated, independently of context, time or researcher.*	***Dependability/Auditability**: the study process is consistent and reasonably stable over time and between researchers.*
Rigor of method	***Internal validity**: a statistically- significant relationship is established, to demonstrate that certain conditions are associated with other conditions, often by "triangulation" of findings*	***Internal consistency (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.*
Generalizability of findings.	***External validity**: the researcher establishes a domain in which findings are generalizable.*	***Transferability**: how far can the findings/ conclusions be transferred to other contexts and how do they help to derive useful theories?*

Lincoln and Guba's trustworthiness criteria suggest that we did not fabricate the story and the story may have utility in other sites. Glaser's quality criteria of fit, workability, relevance, and modifiability for judging grounded theory is similar while more focused on determining if the theory fits the data and was not forced (Glaser 1998, p. 19):

- Fit is another word for validity, does the category adequately express the pattern in the data, which it purports to conceptualize?
- Workability means do the categories and the way they are related into hypotheses sufficiently account for how the main concern of the participants in a substantive area is continuously resolved?
- Relevance makes the research important because it deals with the main concerns of the participants involved. Relevance like good categories evokes instant grab – captures the attention.
- Modifiability is significant because theory is never right or wrong and just gets modified as new data is compared to it.
- Parsimony and scope, wherein the theory accounts for much variation in the data with as few categories possible (Hall and Callery 2001)

Any research process requires attention to transparency and keeping good, well organized notes and records about the research process. Can evidence of the effort and the reasoning process be presented? We employed several practices to demonstrate an audit trail:

Thick Description	Through our process memos we maintained a research diary of our activities and thoughts during this project and decision regarding choices during the analysis phase.
Member Checks	We reviewed transcription and analysis of interviews and with participants to incorporate their comments in data analysis.
Auditable	We kept our work auditable by maintaining rich, thick, and well organized descriptions – all searchable within our Wiki.
Applicability	We kept descriptions of the field sites, participant observations episodes, and of interview subjects

"Thick" description is a term that is used regularly by qualitative researchers, but what constitutes good thick description? We took some guidance from Strauss and Corbin's (1990) criteria for judging the adequacy of the research process. We used this list to inform ourselves about what we should include in our audit trail (process memos):

Criterion#1:	How was the original sample selected? On what grounds (selective sampling)?
Criterion#2:	What major categories emerged?
Criterion#3.	What were some of the events, incidents, actions, and so on that indicated some of these major categories?
Criterion#4.	On the basis of what categories did theoretical sampling proceed? That is, how did theoretical formulations guide some of the data collection?
Criterion#5:	What were some of the hypotheses pertaining to relations among categories? On what grounds were they formulated and tested?
Criterion#6:	Were there instances when hypotheses did not hold up against what was actually seen? How were the discrepancies accounted for? How did they affect the hypotheses?
Criterion#7:	How and why was the core category selected? Was the selection sudden or gradual, difficult or easy? On what grounds were the final analytic decisions made?

7 Summary

We found classical Grounded Theory an effective tool for software engineering research when we are interested in how people manage their lives in problematic situations. Regardless of how the data is analyzed, interviewing and observing people will generate volumes of data and there will be interesting and useful insights in that data. The real gold of Grounded Theory is the ability view a problem from the participants' perspective and to tell the story of that problem as a set of conceptual hypotheses. While Grounded Theory is time consuming and tedious we were able to generate a theory explaining how people manage the process of software development. We used classical Grounded Theory for our study because there is less "forcing" and we had access to classical Grounded Theory experts and resources. Based on our experience, we can summarize this paper by offering the following guidelines for other software engineering researchers considering classical Grounded Theory:

1. Reach out for help. Many software engineering researchers are struggling with sociological research while experts in sociological research and Grounded Theory may be sitting in next building. Nursing, Education and Sociology are disciplines which frequently use Grounded Theory in research.
2. Do not get caught up in the debate over which method is the "best" Grounded Theory method. The best method is the one that has the most resource and support availability. Nonetheless, be clear which method you are employing in your study (Adolph et al. 2008).
3. Do not reduce the relevance of your research to your participants or your opportunity to see situations from a different perspective by constraining your research problem with detailed questions. Work with a broadly defined problem statement that seeks to explicate a process (Adolph et al. 2008) that is relevant to your participants.
4. It is certainly easier to manage data electronically rather than physical volumes of paper. A Wiki is a useful tool for supporting qualitative data management. However, regardless of whether data is represented electronically or on paper, analysis is still very much a tedious manual process. Good analysis requires you immerse yourself in the data. Therefore avoid letting your tools come between you and your data. Do not leave your notes lying on the dining room table if you have a cat or a toddler at home.
5. The definitions of codes, concepts, categories, properties, and indicators are fuzzy in most descriptions of Grounded Theory which often confounds the software engineering researcher. We found it useful for our research to supplement these definitions with the following guidelines:

   i. A concept represents a recurring pattern of social behavior.
   ii. An indicator is data that suggests a concept.
   iii. A category is specialization of a concept that aggregates a set of concepts and is at a higher level of abstraction than the concepts it aggregates.
   iv. A property is a concept that dimensionalizes a higher level category.
6. Theoretical sampling can be bootstrapped using judgmental or purposive sampling. Invite participants with different roles or backgrounds from different sites to participate and ask them to tell you their story of how they conduct their task. As concepts and categories emerge adjust your sampling strategy to develop these categories.
7. Maintain a regular visitation schedule at field sites, even if you are not able to or not planning to collect data during the visit. It is difficult to resume data collection after

even a short hiatus. Also, a regular visitation schedule develops trust, such that will encourage people to approach you and offer their story.

8. Use participant observation as a first class data collection strategy and do not rely exclusively on interviews. Observation enables you to see how people actually conduct their lives versus how they say they conduct their lives. Furthermore, using multiple data collection strategies enhances rigor.
9. Open coding generates concepts by constantly comparing incident to incident and incident to category and is not simply a search for patterns. Rigorously compare each new indicator to previous indicators and categories and ask the generative questions, "What is this data a study of?", "What category does this incident indicate?", "What is actually happening in the data?"
10. Line by-line comparison means rigorously inspecting each line in your data searching for incidents and does not mean every line represents an incident. An incident may be derived from a word up to a set of paragraphs.
11. There is no Grounded Theory without a core category. Think of the core category as the theory's center of gravity, it is the category that is integrates the other categories and is centrally relevant. Do not worry excessively about premature selection of the core category; the constant comparison method will help quickly reveal a poor choice.
12. Open coding, selective coding, and theoretical coding are not three distinct phases of analysis. While it is necessary to generate some categories with open coding before there is an opportunity for selective and theoretical coding, these coding activities take place concurrently.
13. Memo, memo, memo! There is no Grounded Theory without memoing, therefore be bold in your memoing because memos are intended to be private. Too many memos are better than too few because even irrelevant and superfluous memos assist in explicating biases. You cannot generate a theory or provide an adequate audit trail with too few memos.
14. Rigor criteria commonly used for quantitative research are not useful for judging the quality of qualitative research. Just as qualitative research answers different questions, different criteria for rigor and quality must be employed. Guba and Lincoln defined trustworthiness as the criteria for judging the rigor of qualitative research. Glaser defined similar criteria which focused on whether the theory is truly representative of the data and not forced.
15. Conduct your literature search in 2 phases: the first phase is a broad review positioning your research within the current knowledge landscape. The second more detailed phase compares the substantive theory with existing theory. Do not engage in the second phase until you are confident you have a stable core category to avoid the risk of forcing.

If we believe that people do trump process (A. Cockburn and Highsmith 2001) then the Grounded Theory method is effective for generating software engineering theory. Grounded Theory offers a tool for investigating areas that have not been previously studied or where a new perspective might be beneficial (Schreiber and Stern 2001). While the emerging theory of "Reconciling Compromise" is not a radically new idea, it represents a central concern of the study participants that is not well addressed in the software engineering discipline. This is the exciting and powerful insight into our discipline that can be explained with the Grounded Theory method.

The challenge for software engineering researchers is we are engineers and not social scientists. Our traditional education and training does not encourage us to investigate the

human experience. Yet, if our discipline is to grow, this is exactly the type of research we must undertake because it offers the greatest opportunity for improving the software engineering discipline.

A further challenge for us in a discipline that is so accustomed to automating tedious process is that, although Grounded Theory can generate powerful results, it is a tedious manual process. It is difficult to learn and maintain the discipline required to follow the constant comparative method and not to jump to premature or pet conclusions. Reading the Grounded Theory literature there is an impression things should proceed in a lovely smooth, rapid manner. This is research. It is a "Scouting" process (see we transferred one of our ideas!) which means exploring boxed canyons, getting struck, back tracking, trying to discover a path through an unknown landscape. After all if it was not hard, the results would not be so rewarding.

Acknowledgements The authors would like to thank the Scrum Alliance and the Agile Alliance for their generous support of this research.

References

Adolph S, Hall W, Kruchten P (2008) A methodological leg to stand on: lessons learned using grounded theory to study software development. Paper presented at the Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds.

Annells M (1996) Grounded Theory Method: Philosophical Perspectives, Paradigm of Inquiry, and Postmodernism. Qual Health Res 6(3):379–393

Bartels A, Holmes BJ, 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

Benoliel JQ (1996) Grounded Theory and Nursing Knowledge. Qual Health Res 6(3):406–428

Boehm, B. W. (1984). Software Engineering Economics. IEEE Trans Softw Eng 10(1)

Boehm BW, Clark B, Horowitz E, Westland C, Madachy R, Selby RW (1995) Cost models for future software life cycle processes: COCOMO 2.0. Ann Softw Eng 1(1)

Booch G, Rumbaugh J, Jacobson I (1998) *The Unified Modeling Language User Guide*. Addison-Wesley

Charmaz K (2006) *Constructing Grounded Theory: A Practical Guide Through Qualitative Analysis*. Thousand Oaks, Sage

Cockburn A (2002) Agile Software Development Joins the "Would-Be" Crowd. Cutter IT J 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

Coleman G, Conner O (2007) Using Grounded Theory to Understand Software Process Improvement: A Study of Irish Software Product Companies. Inf Softw Technol 49(6)

Corbin J, Strauss A (1990) Grounded Theory Research: Procedures, Canons, and Evaluative Criteria. Qual Sociol 13(1):19

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

Dey I (1999) Grounding Grounded Theory: Guidelines for Qualitative Inquiry. Academic Press

Diaz M, Sligo J (1997) How software process improvement helped Motorola. Softw IEEE 14(5):75–81

Dittrich Y, John M, Singer J, Tessem B (2007) For the Special Issue on Qualitative Software Engineering Research. Inf Softw Technol 49(6):531–539

Dyba T, Moe NB, Arisholm E (2005) Measuring software methodology usage: challenges of conceptualization and operationalization.

Emerson R, Fretz R, Shaw L (1995) *Writing Ethnographic Fieldnotes*. University of Chicago Press, Chicago

Fitzgerald B (1998) An empirical investigation into the adoption of systems development methodologies. Inf Manage 34(6):317–328

Gasson S (2003) Rigor In Grounded Theory Research: An Interpretive Perspective On Generating Theory From Qualitative Field Studies. In M. Whitman & A. Woszczynski (Eds.), Handbook for Information Systems Research Hershey PA: Idea Group Publishing

Glaser BG (1965) The Constant Comparative Method of Qualitative Analysis. Soc Probl 12(4):436–445

Glaser BG (1978) *Theoretical Sensitivity*. Sociology Press, Mill Valley, California
Glaser, BG (1992) *Emergence vs Forcing: Basics of Grounded Theory Analysis*. Sociology Press, Mill Valley, California
Glaser BG (1998) *Doing Grounded Theory: Issues and Discussions*. Sociology Press, Mill Valley, California
Glaser B (2001) *The Grounded Theory Perspective: Conceptualization Contrasted with Description*. Sociology Press, Mill Valley
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, Strauss A (1965) *Awareness of Dying*. Adline, Chicago
Glaser BG, Strauss A (1967) *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Aldine, Chicago Illinois
Glass RL, Vessey I, Ramesh V (2002) Research in software engineering: an analysis of the literature. Inf Softw Technol 44(8):491–506
Hall WA, Callery P (2001) Enhancing the Rigor of Grounded Theory: Incorporating Reflexivity and Relationality. Qual Health Res 11(2):257–272
Harter DE, Krishnan MS, Slaughter SA (2000) Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development. Manage Sci 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 - Volume 1
Hughes J, Jones S (2004) Reflections on the use of Grounded Theory in Interpretive Information Systems Research. Electronic J Bus Res Methods
Jones C (2000) *Software Assessments, Benchmarks, and Best Practices*. Addison-Wesley, Boston
Krishnan MS, Kriebel CH, Kekre S, Mukhopadhyay T (2000) An Empirical Analysis of Productivity and Quality in Software Products. Manage Sci 46(6):745–759
Lazerfeld P, Wagner T (1958) *The Academic Mind: Social Scientist in a Time of Crisis*. Free Press, Glencoe, Illinois
Lincoln YS, Guba EG (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)
Martin A, Biddle R, Noble J (2009) XP Customer Practices: A Grounded Theory. Paper presented at the Agile Conference, 2009. AGILE '09
Matavire R, Brown I (2008) Investigating the use of "Grounded Theory" in information systems research. Paper presented at the Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology
McCallin AM (2003) Designing a grounded theory study: some practicalities. Nurs Crit Care 8(5):203–208
Morse J (2001) Situating Grounded Theory Within Qualitative Inquiry. In: Schreiber RS, Stern PN (eds) *Using Grounded Theory in Nursing*. Springer Publishing Company, New York
Mulhall A (2003) In the field: notes on observation in qualitative research. J Adv Nurs 41(3):306–313
Rittel H, Webber M (1973) Dilemmas in a General Theory of Planning. Policy Sci 4:155–169
Sawyer S, Guinan PJ (1998) Software development: processes and performance. IBM Syst J 37(4):552–569
Schreiber RS, Stern PN (2001) *Using Grounded Theory in Nursing*. Springer Publishing Company, New York
Seaman CB (1999) Qualitative Methods in Empirical Studies of Software Engineering Research. IEEE Trans Software Eng 25(4):557–572
Shaw M (2003) Writing Good Software Engineering Research Papers: Minitutorial. IEEE Trans Softw Eng
Simon H (1947) *Administrative Behaviour*. The Macmillan Company, New York
Singer J, Vinson NG (2002) Ethical issues in empirical studies of software engineering. Softw Eng IEEE Trans 28(12):1171–1180
Sjoeberg DIK, Hannay JE, Hansen O, Kampenes VB, Karahasanovic A, Liborg NK et al (2005) A survey of controlled experiments in software engineering. Softw Eng IEEE Trans 31(9):733–753
Strauss A (1987) *Qualitative Analysis for Social Scientists*. Cambridge University Press, Cambridge
Strauss A, Corbin J (1990) *Grounded Theory Procedure and Techniques*. Sage Publications, Basics of Qualitative Research
Strauss A, Corbin J (1998) *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage Publications, Second Edition
Urqhart C (2001) An Encounter With Grounded Theory: Tackling the Practical and Philosophical Issues. In: Trauth E (ed) *Qualitative Research in IS: Issues and Trends*. IGI Global, Hershey, PA
Whitworth E, Biddle R (2007) The Social Nature of Agile Teams. Paper presented at the AGILE 2007
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 a consultant with Rally Software. His research interests include software process engineering, requirements management, and software architecture. His Internet address is *sadolph@rallydev.com*

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. Her Internet address is *Wendy.Hall@nursing.ubc.ca*

Philippe Kruchten is a professor of software engineering at the University of British Columbia. He is the author of *Rational Unified Process: An Introduction*. His Internet address is *pbk@ece.ubc.ca*