Hindawi Publishing Corporation
Advances in Software Engineering
Volume 2013, Article ID 468021, 9 pages
http://dx.doi.org/10.1155/2013/468021

Review Article

Thematic Review and Analysis of Grounded Theory Application in Software Engineering

Omar Badreddin

University of Ottawa, Ottawa, ON, Canada K1N 6N5

Correspondence should be addressed to Omar Badreddin; obadr024@uottawa.ca

Received 5 June 2013; Accepted 4 September 2013

Academic Editor: Phillip A. Laplante

We present *metacodes*, a new concept to guide grounded theory (GT) research in software engineering. Metacodes are high level codes that can help software engineering researchers guide the data coding process. Metacodes are constructed in the course of analyzing software engineering papers that use grounded theory as a research methodology. We performed a high level analysis to discover common themes in such papers and discovered that GT had been applied primarily in three software engineering disciplines: agile development processes, geographically distributed software development, and requirements engineering. For each category, we collected and analyzed all grounded theory codes and created, following a GT analysis process, what we call *metacodes* that can be used to drive further theory building. This paper surveys the use of grounded theory in software engineering and presents an overview of successes and challenges of applying this research methodology.

1. Introduction

Grounded theory (GT) is a systematic qualitative research methodology, originating in the social sciences and emphasizing the generation of theory from qualitative data in the process of conducting research. Grounded theory, in its original form, was proposed by Glaser and Strauss [1]. However, it was not until 1993 that we could find the first significant grounded theory work applied in software engineering (SE) [2]. Since that date, more researchers have adopted the process and GT has shown promising results. There is a limited, but increasing, body of literature reporting the application of grounded theory in SE. Nevertheless, GT applications in software engineering are still very limited, most likely due to the complexities of applying GT methodology in SE. The GT methodology, we argue, requires adaptation for successful employment in the software engineering domain. The contribution of this paper is to provide what we will call *metacodes* that can be used to drive the initial coding phase of GT. The paper also provides an analysis of existing GT applications in software engineering and the characteristics of such application as exhibited in the existing literature.

This paper is organized as follows. Section 2 presents a brief history of grounded theory and its application in the software engineering arena. Section 3 presents the methodology we adopted to survey, categorize, and analyze GT coding. The subsequent three sections present a literature review and the metacodes thematically organized by the application area. We look at three areas: agile development, distributed development, and requirements engineering. The remainder of the paper presents some GT characteristics that are specific to applications in software engineering and an overview of where GT has been successful and where challenges exist in the application of GT in software engineering.

2. Background and History

Grounded theory is a systematic qualitative research methodology that emphasizes the generation of theory from data. Grounded theory operates almost in a reverse fashion to the traditional scientific method. Rather than proposing a hypothesis and gathering data to support it, data collection is pursued first without any preconceptions. Key points in the data are marked with a series of "codes," which are then grouped into similar concepts or categories. These categories become the basis of a theory. The coding process is typically performed in two steps, initial then focused coding. The categorization process is normally referred to as *axial coding*.

Grounded theory emerged as a research methodology in the 1960s, during a time when sociological research practices were particularly reliant on quantitative methodologies. In 1967, Glaser and Strauss coined the term grounded theory in their book *The Discovery of Grounded Theory* [1]. The term refers to the idea of a theory that is generated by—or grounded in—an iterative process of analysis and sampling of qualitative data gathered from concrete settings, such as interviews, participant observation, and archival research.

The roots of this methodology can be traced back to the work of Dilthey and Rickman [3] who argued against the pursuit of causal explanations at the expense of establishing understanding. Grounded theory methodology can also be traced back to the symbolic interactionist perspective of Blumer [4]. The term "symbolic interaction" refers to the peculiar and distinctive character of interaction as it takes place between human beings. The peculiarity consists in the fact that human beings interpret or "define" each other's actions instead of merely reacting to each other's actions.

Since GTs' inception in the social sciences, grounded theory has become increasingly popular in information systems as a research methodology. This is evident by the growing literature on the methodology and its applications. The first publication our team was able to identify as an application of grounded theory in the area of software engineering was the work by Calloway and Ariav [5] and Toraskar [6] in 1991. In these publications, the researchers described how they adopted grounded theory in understanding how managerial users evaluate their decision support systems.

The first international journal publication of a grounded theory application in software engineering is that of Orlikowski in 1993 [2]. In this work, the researcher presents findings of a study into the adoption of CASE tools. The researcher justified the use of grounded theory as a research methodology on the basis that it provided "a focus on contextual and processual elements as well as the action of key players associated with organizational change elements that are often omitted in IS studies."

More recently, Baskerville and Pries-Heje [7] employed grounded theory combined with action research to enhance the rigor and traceability in the theory-development part of their work. Action research is a reflective process of progressive problem solving led by individuals working with professionals to improve the way they address issues and solve problems. Other work has employed grounded theory to initiate more focused data collection activities [8].

Grounded theory applications have extended to other areas within software engineering. While the literature is limited, the most prominent discipline of grounded theory work is in software development methodologies, as evident in the quantity of published work in this discipline. Out of the 60 research papers we identified as applications of grounded theory in software engineering 25 addressed software development methodology. Other subdisciplines with significant bodies of GT research include requirements engineering and distributed software development practices.

We believe that GT is a research methodology particularly useful for software engineering research for two main reasons. First, software development is a human-intensive process. Second, software is used by humans with complex interaction and usage patterns, where quantitative evidence is nonexistent or difficult to formulate. Secondly, GT provides an effective approach for qualitative analysis of the rich data sets available in typical software engineering environments. For example, emails and chat histories provide recorded communication information possibly over an extended period of time. Such information is an important data source for GT studies that are available in typical software engineering environments.

The low and slow adoption of GT methodology in software engineering is due to a number of factors. First, GT originated in the social sciences, and since its adoption in software engineering, there has been little guidance on how to employ the methodology. Second, it is not clear what characteristics of GT need adaptation to better fit the nature of software engineering research. Many researchers in software engineering are not familiar with GT and in our experience can be skeptical of its effectiveness. In addition, as our survey highlights, the number of researchers that have reported using GT is small, which contributes a barrier to accelerating GT adoption.

3. Discussion of Sources

Surveying the application of grounded theory in software engineering turned out to be more challenging than anticipated. Grounded theory work is published in a large variety of journals and conference proceedings. A significant portion of grounded theory research can be located in journals dealing with empirical studies. Nevertheless, a growing number of grounded theory projects deal with development processes, requirements engineering, software tools, and development practices. Such work is typically published in journals not related to empirical studies. What follows is a review of the methodology used to identify candidate GT sources to ensure that we covered the full gamete of papers on the subject.

We located more than 60 published papers that explicitly reported the use of grounded theory in the analysis of their data in an area related to software engineering. While the determination of the use of grounded theory as a research methodology was relatively clear, the scope that defines what software engineering is remains more challenging. Hence, we found a thematic presentation was the most appropriate. The surveyed resources are organized under three main themes: agile development, distributed development, and requirements engineering. These three disciplines contain a major portion of the grounded theory work within software engineering.

Some grounded theory approaches recommend starting with high level codes to drive theory building [9]. This is particularly challenging due to the small amount of the literature available on the application of GT in software engineering. In order to help software engineering researchers, we collected all codes and categories that were reported in each GT application theme. We then analyzed those codes using a GT approach to create what we call *metacodes* or codes of codes. We first collected all codes and subcodes from the grounded theory papers in each theme separately. Those

codes were then analyzed, rearranged, and merged to create a final shallow hierarchy of metacodes. Each metacode is associated with tags that summarize a larger number of codes and subcodes as exhibited in the literature within a specific theme. It is our conjecture that the metacodes can be of value to future applications of GT in the software engineering themes presented in this paper. They can function as high level codes that drive theory building in these areas.

4. Grounded Theory in Agile Development Methodologies

We were able to identify 32 published papers that applied grounded theory to study software development methodologies. Of these, 9 reported studying agile methodologies.

Agile software development refers to a group of methodologies that share and promote principles such as development with short iterations, teamwork, collaboration, and process adaptability throughout the life cycle of the project [10]. The roots of agile development can be traced back to 1974 when an adaptive software development process was introduced by Edmonds [11]. However, the definition of modern agile development processes evolved in the 1990s. For example, Extreme Programming was formally introduced in 1996 [12].

Out of all surveyed papers, 9 reported research into agile methodologies using grounded theory. This number reflects the fact that agile development processes are a relatively new and evolving concept. In addition, applications of grounded theory work in software development methodology in general are limited [8]. The earliest work that reported a grounded theory methodology in an agile development process setting is that of Kahkonen et al. [13].

One of the most prominent work is that of Coleman et al. [14–16], who report on how software process and software process improvement (SPI) are applied in the practice of software development. Their study focused on a number of indigenous Irish software companies at various stages of development. In the first phase of the study, they performed four interviews in three different companies. Each interview contained 53 questions. In the second phase, they investigated 11 more companies, performing interviews of about an hour each. They initially performed focused and axial coding, which resulted in three themes and 17 core categories. The theory they present represents a form of "experience" road map illustrating some of the potential pitfalls a software product company could face and how others have avoided or resolved them. Their findings also included supporting evidence and justifications regarding the low level of adoption of CMM/CMMI and ISO 9000 by Irish software companies. They cited the cost of its implementation and maintenance, the added burden on the development efforts, and increased documentation and bureaucracy as the main factors behind the low adoption of such SPI initiatives. For example, they report that smaller companies believed SPI would negatively impact their creativity and flexibility.

Another example of the use of the grounded theory approach in an agile environment involved exploring the socio-psychological characteristics of agile teams and learning about the types of experience acquired in such software development teams [17, 18]. The findings contribute a better understanding of the link between agile practices and positive team outcomes such as motivation and cohesion.

Table 1: Metacodes for agile development methodologies.

Number	Agile development metacodes	Tags/description
1	Characteristics/practices of agile development	Communications, processes, negotiations, skills, team, commitment, management, implementation, knowledge sharing trust, software builds, team rooms, workspaces, and meetings
2	Challenges of agile development	Requirements, communications, people-oriented process, formality, and team cohesion
3	Company characteristics	Domain, number of projects, and market sector
4	Project characteristics	Duration, complexity, development sites, customer locations, and team size
5	Lessons	Tools, expertise, culture, trust, training, commitment, and resource management

4.1. Metacodes for Agile Development Methodologies. We collected codes and subcodes from the 9 studies that adopted GT to investigate agile development methodology. We constructed the metacodes by analyzing 50 codes and 206 subcodes. Metacodes and tags are summarized in Table 1.

Table 1 presents a summary of the metacodes we constructed in the agile methodology theme. Each metacode represents a large number of codes and subcodes, samples of which are presented in the rightmost column. Here we provide a description for each of the metacodes.

4.1.1. Characteristics/Practices of Agile Development. This metacode is used to group codes and subcodes that refer to a characteristic specific to an agile software development project. This includes the nature of communication within teams, knowledge sharing, and the characteristics of trust within a development team, management, and the client. It also includes team rooms and the nature of the workspaces and meetings.

4.1.2. Challenges of Agile Development. This code groups challenges in agile development related to such topics as requirements gathering activities, requirement stability, nature and frequency of changes in requirements, communications, focus on the people-oriented rather than process-oriented control, lack of formality, and lack of team cohesion.

4.1.3. Company Characteristics. Company-related codes were reported in two studies. This metacode groups tags related

to the company domain, the number of agile projects in execution and in total, and the targeted market sector.

4.1.4. Project Characteristics. This metacode represents all codes related to the agile project characteristics. This includes duration of the projects on average and individually complexity of the project as perceived and objectively the number of development sites and development team size.

4.1.5. Lessons. This metacode collects all lessons learned that are related to agile development. Lessons learned were related to the tools being utilized, the importance of expertise within the team, the role of culture in the success of projects, and the role of trust. In addition, it includes the importance of formal training and the commitment of every team member to the success of the agile activities and the importance of proactive resource management.

5. Grounded Theory and Geographically Distributed Development (GDD)

Out of our surveyed literature, we identified seven studies on Geographically Distributed Development (GDD) that used GT. GDD, also known as Distributed Software Development (DSD), has grown to be a common practice in today's industry [19]. Despite the limited number of publications, GDD seems to be a fertile discipline for grounded theory application for the following reasons.

(i) GDD has grown, and is still growing, exponentially in the last decade [20].

(ii) GDD brings about additional complexity to any development process.

(iii) There is a wealth of data sources that can be analyzed using grounded theory analysis. For example, communications in GDD are typically written communications (email, chat sessions) that can be easily recorded over an extended period of time with little effort and little disruption to existing business activities. Such data are typically absent in normal settings or require significant effort to facilitate data collection.

It is typical for grounded theory research activities to take place in real life situations, by interviewing or collecting data from real projects. However, one study [21] reported grounded theory methodology using student subjects comprising 21 virtual teams collaborating in the completion of a given task. In this study, the researcher aimed at uncovering how distributed projects are managed and executed. The study concludes with characteristics of managing a distributed project as well as proposing a model for distributed project management. A similar work [22] also utilized students in a study of distributed development using student participants. The study relied on the analysis of electronic communications collected during the performance of a distributed development task by the students.

There are situations when a surveyed GT work addressed both GDD and agile methodology at the same time. In such situations, we classified the paper under both themes, including their codes and sub-codes in the analysis and construction of metacodes in both themes. GDD becomes complex and challenging when an agile method is adopted [23]. Ramesh et al. [20] have reported a grounded theory approach that analyzes data from three different organizations, attempting to answer the question whether distributed software development can be agile. Ramesh et al. have identified a number of challenges specific to distributed agile development processes; nevertheless, they concluded that distributed and agile approaches can be combined.

Layman et al. [19] pursued a different approach. Layman et al. studied a successful distributed agile development project in the USA and the Czech Republic in an attempt to uncover the characteristics of these successful projects. They collected the data from archives of emails as well as semistructured interviews. Quantitative data (the number of source file lines for example) was supplementary to their qualitative data. Their work's main contribution is the recommendation of four success factors for a distributed XP methodology: the facilitation of communication by the management, short asynchronous communication loops, identifiable customer authority to resolve requirement related issues, and a high process visibility.

Managing requirements in a distributed development setting present unique challenges. Requirements engineering is a communication-intensive and dynamic task. When stakeholders are geographically distributed, requirement engineering tasks become even more complex. Damian and Zowghi [24] present their field study work that investigates requirements engineering challenges introduced by stakeholders' geographical distribution in a multisite organization. Their goal is to examine requirements engineering practice in global software development and formulate recommendations for improvements. In a subsequent section, we discuss grounded theory-based requirements engineering research in nondistributed projects.

5.1. Metacodes for Geographically Distributed Development. Out of the seven identified GT studies on GDD, we analyzed the codes extracted from six studies. One study did not provide adequate reporting on its codes and subcodes. We collected 31 codes and 95 subcodes resulting in 11 metacodes presented in Table 2.

Table 3 presents a summary of the metacodes we constructed in the geographically distributed development theme. Each metacode represents a large number of codes and sub-codes, samples of which are presented in Table 2. Here we provide an analysis and description for each of the metacodes.

GDD projects are, after all, software development projects, so it was expected to see a number of codes that can be found in a typical software engineering project. Communications in a GDD project play a more prominent role, and it was found in almost every set of codes analyzed. Coordination and adaptation metacodes are closely associated with the GDD nature of the project. That metacode represented codes related to time zone issues, collaboration, level of involvement, and social and cultural issues. All these

Table 2: Metacodes for geographically distributed development.

Number	GDD metacodes	Tags/description
1	Communication	Communication patterns (generating ideas, confirmation, consensus, conflict, humor, and attitude), positive and negative
2	Coordination	Time zone (delay in responses) collaboration, and involvement
3	Adaptation	Social, work, technological, conflict resolution, and lateral thinking
4	Company background	Company size, maturity levels, existing development approaches, and company's culture
5	Stakeholders	Project under study's stakeholders related information, years of experience, and so forth
6	Collaboration technologies	Simple emails, advanced collaboration technologies
7	Requirements challenges due to distance	Inadequate communication, knowledge management, cultural diversity, and time difference
8	Requirements activities	Elicitation, prioritization, negotiation, validation, examining current system, and managing uncertainty specification
9	Involvement of users	Achieving appropriate participation of system users and field personnel
10	Trust	Checking project status, concern about a member doing his task, and trust built progressively
11	Delay	Sources and nature of delay, perceived causes, and delay mitigation actions

Table 3: Metacodes for requirements engineering.

Number	Requirement engineering metacodes	Tags/description
1	Challenges	Distance, communications, knowledge management, customer culture, and awareness of processes for RE
2	Elicitation	These metacodes refer to the standard RE activities
3	Prioritization	
4	Negotiation	
5	Validation	
6	Specification	
7	Examining current system	Existing system attributes
8	Business objectives	Business objectives of the current software project to which RE is being performed
9	Primary business attributes	Nature of users, expected project contribution to business goals, and so forth
10	Primary project attributes	Type of requirements, RE process, and complexity of requirements, etc.
11	RE process attributes	Iterative development, development team and business analysis team, etc.

aspects are related to the geographical nature of the project. Collaboration technologies, requirements challenges due to distance, involvement of users, delay, and trust are metacodes that were found specific to GDD projects.

6. Grounded Theory and Requirements Engineering

Requirements engineering is particularly attractive for grounded theory methodology for a number of reasons. Applications and systems are growing increasingly more complex and involve everincreasing numbers of users and stakeholders. Grounded theory can help discover patterns from a stakeholders' perspective of the system under development that may increase our knowledge of the users' needs and how those stakeholders may perceive aspects related to the new system, like the organization impact of the new system and changes in business tasks and activities.

Requirements management tools now incorporate discussions, communications, and issues related to requirements. This large amount of data can serve as the basis for extensive grounded theory work. Data collection techniques that are typically applicable in social sciences and psychology, from which grounded theory has emerged, are not always as applicable in software engineering. Such requirements management tools provide unbiased data that is otherwise hard, or sometimes impossible, to collect without some level of disturbance of existing business activities.

A prominent work in applying grounded theory to the requirements engineering discipline is the work of Damian and Zowghi [24] where they report on the investigation of requirements engineering challenges introduced by stakeholder's geographical distribution in a multi-site organization. In addition to conducting semistructured interviews, they also analyzed existing documents and observed requirements meetings. In this work, and due to the geographically distributed organization, stakeholders heavily relied on emails and automated requirements engineering tools that provided recorded, as well as detailed, history of discussions and communications.

Qureshi et al. [21] applied grounded theory methodology on a case study of distributed software project management activities. Their study spanned all project phases and used observations and transcripts of electronic communications as their data sources.

A study of the Hewlett-Packard requirements engineering process considered two projects [25]. The first project was small and agile, and was characterized by quick releases, while the second project was large and complex, and was outsourced some of the development. Hewlett-Packard has a large and varying collection of requirements engineering processes. The selection of such processes is influenced by business drivers and constraints as well as characteristics of the project itself. Padula [25] has reported on how Hewlett-Packard selects the requirement engineering process based on project attribute.

Requirements engineering is inherently dynamic due to the nature of continuous change put forward by the various stakeholders. It is argued that information system contexts are soft and ambiguous and are therefore mainly characterized by qualitative data. Such characteristics make grounded theory a suitable methodology for research in the requirements engineering discipline. Galal and Paul [26] presented an analytical technique, based on grounded theory, for developing qualitative scenarios against which statements of requirements can be evaluated.

6.1. Metacodes for Requirements Engineering. For requirements engineering, we collected 26 codes and 54 sub-codes for analysis that resulted in 11 metacodes. We summarize metacodes in Table 3.

Our metacodes include five of the standard requirements engineering activities. It is possible that the referenced studies have used the standard requirement engineering activities as code seeds to initiate their coding process.

7. Other Applications of Grounded Theory

Grounded theory has been applied to a number of other subjects within software engineering that do not fall under our three main themes. Grounded theory has been employed to investigate tool and technology adoption [27], the impact of background knowledge of the performance of software developers [28], the motivation of open source software developers [29], questions developers ask during software maintenance tasks [30], knowledge repositories in software companies [31], barriers to adoption of software reuse [32], cognitive patterns used when explaining or understanding software [33], and new product development management issues and decision-making approaches of development managers [34].

We opted not to include the analysis of codes and sub-codes of this type of GT application for a number of reasons. First, we could not find the sufficient literature of the application of GT to create meaningful new themes. And due to the lack of papers, construction of metacodes using our approach will inevitably result in biased metacodes that reflect more the surveyed studies, rather than the emergence of a pattern observed from a broader coding or subcoding processes.

8. Opportunities and Challenges of GT Application in Software Engineering

Our analysis of the surveyed papers, as well as our experience in applying GT in software engineering, highlights a number of opportunities unique to the software engineering field that makes GT an even more promising research methodology. With opportunities come challenges that software engineering researchers should be aware of while preparing for their research. The analysis we present in this section is extracted from the surveyed literature and does not reflect our own experience with GT.

8.1. Opportunities

(i) The lack of integrated theories in the literature related to a number of areas in software engineering practices suggests the use of an inductive approach that allows theories to emerge based on pragmatic accounts of professionals themselves. For example, the role of communication and trust in distributed development is not formalized in a theory. However, a number of studies reported in this paper have addressed the role of communication and trust in distributed development as reflected by the experience of professionals in GDD projects.

(ii) Grounded theory has well-established guidelines for conducting inductive, theory-generating research.

(iii) Software development is a human-intensive activity and development processes are characterized by heavy reliance on human compliance, emphasizing the human aspects of software engineering [35]. Grounded theory is renowned for its application to the analysis of human behavior.

(iv) Grounded theory is a burgeoning methodology in the information systems arena and has been an established and credible methodology in sociological and health disciplines.

(v) Grounded theory (for the novice researcher or experienced researchers new to interpretive studies) provides a useful template and as such serves as a comfort factor in the stressful and uncertain nature of conducting qualitative research [36].

(vi) Software engineering relies significantly on software tools for managing artifacts, as well as documentation and communications. These tools make available recorded communications, potentially over an extended period of time, which become valuable assets for grounded theory analysis. In comparison to typical social settings, such information is either nonexistent or significantly harder to collect.

8.2. Challenges

(i) Data collection within an organization for research purposes is typically challenging. Business priorities will tend to take precedence over participation in research activities. Ethics committee and managerial approval are required prior to performing such research. The requirement for management approval raises the question of whether the data sampling is actually unbiased and is an honest representation of the organization or activities under study.

(ii) The use of semistructured interviews centers data collection on users' opinions. This can lead to an overemphasis on the participants' perception of what is taking place, which could be at odds with reality. Despite the occasions when there is no supporting evidence, the researcher is obliged to accept what the respondents say during the interviews [37]. However, in certain situations such as decision-making processes, managers base their decisions on their own perceptions, and therefore it is the perception that matters [16]. In addition, semistructured interviews need not be the only data collection activity. As discussed earlier, there are a number of papers reporting the utilization of electronic communications, documentation, and archives as data sources.

9. Adaptation of Grounded Theory

Because grounded theory as a methodology has emerged from the social sciences, one could justifiably adapt the methodology when adopting it in software engineering research. The existence of some variations of grounded theory in social sciences is reported in the literature [38]. In addition, rigid application of grounded theory has been critically questioned [24]. We have noted three major characteristics specific to grounded theory work when applied to software engineering. Those characteristics are related to (a) the literature review prior to the study, (b) selection of participants, and (c) data sources. In this section, we briefly highlight those characteristics.

A prominent characteristic of grounded theory is captured by the advice offered by Glaser and Strauss [1]: "There is a need not to review any of the literature in the substantive area under study. This dictum is brought about by the desire not to contaminate... it is vital to be reading and studying from the outset of the research but in unrelated fields."

Contrary to this advice, a number of grounded theory researchers have explicitly advocated the benefits of the literature and background knowledge of the researcher prior to conducting data gathering activities [8, 36]. It is reported that prior knowledge helps in guiding research and the use of seed categories (such as our metacodes) helps inform analysis. This deviance from the original methodology is justifiable as some background knowledge is needed to help the researcher in the process of interviewing and data collection. The researchers' personal constructs and skills help structure data, and it is the researcher's hermeneutic perspective that maintains the interpretive style rather than the grounded theory method [36].

Selection of participants was particularly challenging for a number of reported research activities in the software engineering arena. While there is normally a criterion for the selection of subjects (based on their role in the study case for example), subject selection was largely affected by management and the participants' availability [37]. In such situations, management could deliberately select participants that would present a favorable picture. Informing management of the objective and purpose of the research and guaranteeing an adequate level of privacy and confidentiality of the data can help mitigate such risks.

Data sources in grounded theory work seem to be overwhelmingly reliant on semistructured interviews. However, in a number of studies, particularly those addressing distributed development, researchers made significant use of documented email communications and chat session histories. Such data sources are typically nonexistent in normal social sciences settings. Researchers also made use of existing manuals and archives.

9.1. Analysis of Metacodes. We present the following remarks about our metacodes presented in Tables 1, 2, and 3.

(1) Communications and trust tend to be central to all GT applications in the three themes under study.

All themes included communications and trust at the first or second level. This may reflect the importance of communication and trust in software development activities. It may also indicate that the existing GT studies focused on studying communications and trust. This may be due to the nature of how GT is developed from interviews, meetings, and so forth. Such data sources may inevitably reflect communications and trust aspects of software development projects.

(2) Metacodes derive their significance from the specific GT application.

In the process of developing our metacodes, we were careful to only select codes from studies that sufficiently addressed the corresponding theme using the GT approach. However, each study had its own unique settings, procedures, objective, and findings. For example, Padula [25] was studying the requirements engineering process with Hewlett-Packard and focused on the study of two particular HP products, while other studies had a different focus. In our analysis, we were careful to select codes and subcodes that were, as much as possible, not related to a specific product or study. At the same time, we also want our metacodes to represent adequate coverage of the existing literature.

(3) Overlapping of themes and codes.

The three themes presented in this paper are not mutually exclusive. For example, the study [24]

addressed GT application for requirements engineering in a GDD environment. This paper was justifiably classified under the two themes GDD and RE. During the process of code analysis, additional care was required to properly select codes that addressed aspects of the project that corresponded to the theme under which the metacodes were listed. For example, in the study by Damian and Zowghi [24], codes that related to GDD were listed under GDD theme and similarly for RE codes.

10. Conclusion

While having its antecedents in sociology in the mid 1960s, grounded theory methodology has been growing in the field of software engineering. Software engineering is a human-intensive activity, and is proving to be an attractive discipline for the application of grounded theory. However, there is little background and guidance on how to apply GT in software engineering domain.

Our work is an initial step towards understanding how to best apply GT methodology in SE. We introduced metacodes that can function as seeds for the construction of codes in GT research in three subdisciplines of software engineering: agile development methodology, requirement engineering, and geographically distributed development. We also presented a critical review of the challenges and opportunities specific to GT research in software engineering as a whole. Our survey revealed characteristics of GT research in software engineering, such as adapting the classic grounded theory rules about prior literature review, data sources, and participant selection.

References

[1] B. G. Glaser and A. L. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*, Aldine de Gruyter, New York, NY, USA, 1977.

[2] W. J. Orlikowski, "CASE tools as organizational change: investigating incremental and radical changes in systems development," *MIS Quarterly*, vol. 17, no. 3, pp. 309–340, 1993.

[3] W. Dilthey and H. Rickman, *Dilthey Selected Writings*, Cambridge University Press, Cambridge, UK, 1979.

[4] H. Blumer, *Symbolic Interactionism: Perspective and Method*, University of California Press, California, Calif, USA, 1986.

[5] L. Calloway and G. Ariav, "Developing and using a qualitative methodology to study relationships among designers and tools," in *Information Systems Research: Contemporary Approaches and Emergent Traditions*, pp. 175–193, 1991.

[6] K. Toraskar, "How managerial users evaluate their decision support: a grounded theory approach," in *Proceedings of the IFIP WG 8. 2 Working Conference*, pp. 195–225, Copenhagen, Denmark, December 1991.

[7] R. Baskerville and J. Pries-Heje, "Grounded action research: a method for understanding IT in practice," *Accounting, Management and Information Technologies*, vol. 9, no. 1, pp. 1–23, 1999.

[8] B. Fitzgerald, "The use of systems development methodologies in practice: a field study," *Information Systems Journal*, vol. 7, no. 3, pp. 201–212, 1997.

[9] G. A. Bowen, "Grounded theory and sensitizing concepts," *International Journal of Qualitative Methods*, vol. 5, no. 3, pp. 1–9, 2006.

[10] O. Markku and K. S. Seija, "Product focused software process improvement," in *Proceedings of the 4th International Conference (profes '02)*, Rovaniemi, Finland, December 2002.

[11] E. Edmonds, "A process for the development of software for non-technical users as an adaptive system," *General Systems*, vol. 19, pp. 215–217, 1974.

[12] K. Beck, M. Beedle, A. van Bennekum et al., "Manifesto for agile software development," Retrieved November, vol 11, pg 2004, 2001.

[13] T. Kahkonen, P. Abrahamsson, N. R. Center, and F. Espoo, "Digging into the fundamentals of extreme programming building the theoretical base for agile methods," in *Proceedings of the 29th Euromicro Conference*, pp. 273–280, IEEE Computer Society, Patras, Greece, 2003.

[14] G. Coleman, "eXtreme Programming (XP) as a "minimum" software process: a grounded theory," in *Proceedings of the 28th Annual International Conference on Computer Software and Applications (COMPSAC '04)*, pp. 30–31, IEEE Computer Society, Honk Kong, September 2004.

[15] G. Coleman and O. Connor R, *Software Process in Practice: A Grounded Theory of the Irish Software Industry*, vol. 4257 of *Lecture Notes in Computer Science*, 2006.

[16] G. Coleman and R. O'Connor, "Investigating software process in practice: a grounded theory perspective," *Journal of Systems and Software*, vol. 81, no. 5, pp. 772–784, 2008.

[17] E. Whitworth and R. Biddle, *Motivation and Cohesion in Agile teams*, vol. 4536 of *Lecture Notes in Computer Science*, 2007.

[18] E. Whitworth and R. Biddle, "The social nature of agile teams," in *Proceedings of the Agile Conference (AGILE '07)*, vol. 3, pp. 26–36, August 2007.

[19] L. Layman, L. Williams, D. Damian, and H. Bures, "Essential communication practices for Extreme Programming in a global software development team," *Information and Software Technology*, vol. 48, no. 9, pp. 781–794, 2006.

[20] B. Ramesh, L. Cao, K. Mohan, and P. Xu, "Can distributed software development be agile?" *Communications of the ACM*, vol. 49, no. 10, pp. 41–46, 2006.

[21] S. Qureshi, M. Liu, and D. Vogel, "A grounded theory analysis of e-collaboration effects for distributed project management," in *Proceedings of 38th Annual Hawaiian International Conference on Systems Sciences*, p. 264, Big Island, Hawaii, USA, January 2005.

[22] M. Last, "Understanding the group development process in global software teams," in *Proceedings of the 33rd Annual Conference on Frontiers in Education (FIE '03)*, vol. 3, November 2003.

[23] A. Martin, R. Biddle, and J. Noble, *When XP Met Outsourcing*, vol. 3092 of *Lecture Notes in Computer Science*, 2004.

[24] D. Damian and D. Zowghi, "Requirements Engineering challenges in multi-site software development organizations," *Requirements Engineering*, vol. 8, no. 3, pp. 149–160, 2003.

[25] A. Padula, "Requirements engineering process selection at Hewlett-Packard," in *Proceedings of the 12th IEEE International Requirements Engineering Conference (RE '04)*, pp. 296–300, IEEE, Palo Alto, Calif, USA, September 2004.

[26] G. Galal and R. Paul, "A qualitative scenario approach to managing evolving requirements," *Requirements Engineering*, vol. 4, no. 2, pp. 92–102, 1999.

[27] D. Oliver, G. Whymark, and C. Romm, "Researching ERP adoption: an internet-based grounded theory approach," *Online Information Review*, vol. 29, no. 6, pp. 585–603, 2005.

[28] J. Carver, "The impact of background and experience on software inspections," *Empirical Software Engineering*, vol. 9, no. 3, pp. 259–262, 2004.

[29] Y. Ye and K. Kishida, "Toward an understanding of the motivation of open source software developers," in *Proceedings of the 25th International Conference on Software Engineering*, pp. 419–429, May 2003.

[30] J. Sillito, G. C. Murphy, and K. De Volder, "Questions programmers ask during software evolution tasks," in *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 23–34, ACM, New York, NY, USA, November 2006.

[31] T. Dingsøyr and E. Røyrvik, "An empirical study of an informal knowledge repository in a medium-sized software consulting company," in *Proceedings of the 25th International Conference on Software Engineering*, pp. 84–92, May 2003.

[32] K. Sherif and A. Vinze, "Barriers to adoption of software reuse: a qualitative study," *Information and Management*, vol. 41, no. 2, pp. 159–175, 2003.

[33] A. R. Murray, *Discourse structure of software explanation: snapshot theory, cognitive patterns and grounded theory methods [Ph.D. thesis]*, University of Ottawa, Ottawa, Canada, 2006.

[34] S. Y. Yahaya and N. Abu-Bakar, "New product development management issues and decision-making approaches," *Management Decision*, vol. 45, no. 7, pp. 1123–1142, 2007.

[35] J. E. Tomayko and J. E. T. O. Hazaan, *Human Aspects of Software Engineering*, Charles River Media, Massachusetts, Mass, USA, 2004.

[36] J. Hughes and S. Jones, "Reflections on the use of grounded theory in interpretive information systems research," *Electronic Journal of Information Systems Evaluation*, vol. 6, no. 1, 2003.

[37] B. H. Hansen and K. Kautz, "Grounded theory applied-studying information systems development methodologies in practice," in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, p. 264, IEEE Computer Society, Big Island, Hawaii, USA, January 2005.

[38] C. Goulding, "Grounded theory: the missing methodology on the interpretivist agenda," *Qualitative Market Research*, vol. 1, no. 1, pp. 50–57, 1998.