Understanding Antecedents of Agile Software Development Methodology Preferences

David Brian Bishop
Dakota State University

Follow this and additional works at: https://scholar.dsu.edu/theses

Recommended Citation
https://scholar.dsu.edu/theses/290

This Dissertation is brought to you for free and open access by Beadle Scholar. It has been accepted for inclusion in Masters Theses & Doctoral Dissertations by an authorized administrator of Beadle Scholar. For more information, please contact repository@dsu.edu.
UNDERSTANDING ANTECEDENTS OF AGILE
SOFTWARE DEVELOPMENT METHODOLOGY
PREFERENCES

A graduate project submitted to Dakota State University in partial fulfillment of the requirements for the degree of

Doctor of Science

in

Information Systems

October, 2014

By
David Brian Bishop

Project Committee:
Dr. Surendra Sarnikar
Dr. Amit Deokar
Dr. Vicki Johnson
Dr. Zixing Shen
We certify that we have read this project and that, in our opinion, it is satisfactory in scope and quality as a dissertation for the degree of Doctor of Science in Information Systems.

Student Name: David Brian Bishop

Dissertation Title: Understanding Antecedents of Agile Software Development Methodology Preferences

Faculty supervisor: Date: ____________

Committee member: Date: ____________

Committee member: Date: ____________

Committee member: Date: ____________
ACKNOWLEDGMENT

I would like to thank Dr. Amit Deokar for his support and help throughout my doctoral work and specifically on my research and dissertation project. His direction was invaluable in making this a reality. In addition I’d like to thank Dr. Surendra Sarnikar for stepping in and helping me complete the dissertation. Also thanks goes to my two additional committee members Dr. Vicki Johnson and Dr. Zixing Shen. I appreciated your support and insights for improving the research and dissertation.

I’d especially like to thank my wife Barb who has stuck with me and supported me in so many ways over my entire graduate efforts. Her sacrifice is immeasurable. Speaking of sacrifice, I’d like to thank my children and their patience when dad was seemingly always working on his doctorate. I know I’ll never get back all those evenings and weekends but thanks for your support and here’s to a more engaged future.

I’d like to dedicate this accomplishment to my mother who although she didn’t achieve significant heights in academia provided me with a great childhood and more encouragement than I deserved. I only wish she was still with us to see this day. And to my father who also completed his education while raising a family, thanks for setting a great example in so many areas.

I would be remiss not to thank Jesus who has made all this possible and provided the strength and ability to make it happen!
ABSTRACT

This study, using a grounded theory methodology, develops a theory of antecedents for the preference of Agile Information Systems Development. The theory emerges from data collected in the form of interviews with a variety of different software developers. Grounded theory practices of coding transcripts into increasing levels of abstraction led to the formation of constructs and their relationships that make up the theory.

Understanding the antecedents of preference for agile software development has important application to management of software development projects. This understanding can lead to more effective formation of teams, better training for developers faced with integrating agile methods into their software development practice on upcoming or current projects. Knowing the factors that software developers find appealing can also assist teams to optimize their software development efforts by focusing on the key components of agile that make it desirable.

Analyzing the data revealed that three broad categories or themes lead to increased preference for agile methods. One of the key factors is self-efficacy. Software developers felt that using agile methods made them effective and efficient. They felt that they produced higher quality software for their customers. And they felt that the agile approach fit better with the reality of changing requirements and customer the customer desire to see and experience the software faster than with the longer cycles of traditional Waterfall methodology. In addition to self-efficacy, affective response emerged as another key factor for preference of agile methods. Developers consistently expressed that they were more engaged, felt a sense of accomplishment, experienced increased confidence and had a better work-life balance when using agile software development methods. These emotional benefits attracted software developers to agile methods.

The other category is intra-inter personal effects. Developers said communication within the team, with stakeholders and particularly customers were improved with agile methods. The short cycle with quicker delivery allowed customers to provide faster and more meaningful feedback. The daily stand-up meetings that were common among participants
also improved team communication. One final concept within the intra-inter personal
dimension is that participants experienced positive social influence. Respected individuals
planted positive thoughts about agile that increased the participants’ preference for agile
methods.

Two concepts were identified that inhibit preference for agile. First, negative social
influence. When either management or customers indicated a desire for Waterfall it
negatively influenced some participants’ preference for agile. Finally, when a developer is
satisfied with the status quo or fears learning a new methodology they have a decreased
preference for agile.
DECLARATION

I hereby certify that this project constitutes my own product, that where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions or writings of another.

I declare that the project describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,

________________________________________
David Brian Bishop
# TABLE OF CONTENTS

**Dissertation Approval Form** ....................................................................................................................... II  
**Acknowledgment** ........................................................................................................................................... III  
**Abstract** ........................................................................................................................................................ IV  
**Declaration** ......................................................................................................................................................... VI  
**Table of Contents** .............................................................................................................................................. VII  
**List of Tables** ....................................................................................................................................................... IX  
**List of Figures** ..................................................................................................................................................... X  
**Introduction** .......................................................................................................................................................... 1  
  - General Background ................................................................................................................................................ 1  
  - Statement of the Problem ....................................................................................................................................... 2  
  - Research Question ................................................................................................................................................ 2  
  - Relevance to Industry ........................................................................................................................................... 2  
**Literature Review** .................................................................................................................................................. 4  
  - Information Systems Development Methodologies ............................................................................................. 4  
  - Acceptance and Adoption Literature ................................................................................................................... 6  
  - Preference versus Acceptance and Adoption ......................................................................................................... 9  
  - Research Gap ...................................................................................................................................................... 10  
**Research Design** ................................................................................................................................................... 12  
  - Research Design Goals ......................................................................................................................................... 12  
  - Unit of Analysis .................................................................................................................................................... 12  
  - Participants .......................................................................................................................................................... 13  
  - Researcher Point of View .................................................................................................................................. 13  
  - Research Tools .................................................................................................................................................... 14  
**Research Methodology** ...................................................................................................................................... 15  
  - Qualitative Grounded Theory ................................................................................................................................. 15  
  - Theoretical Sampling versus Random Sampling .................................................................................................. 15  
  - Data Collection ................................................................................................................................................... 16  
  - Data Analysis ..................................................................................................................................................... 17
LIST OF TABLES

Table 1 - Riemenschneider et al. Acceptance Constructs .............................................. 7
Table 2 - Potential Preference Constructs From Literature Review .............................. 9
Table 3 - Charmaz guidelines for Initial Coding ......................................................... 19
Table 4 - Six Cs Taxonomy ......................................................................................... 20
Table 5 - Interview Summary Information .................................................................. 22
Table 6 - Findings Summary ........................................................................................ 26
LIST OF FIGURES

Figure 1 - Waterfall and agile Comparison ................................................................. 5
Figure 2 - Chan and Thong's Agile Acceptance Conceptual Framework ............... 8
Figure 3 - Glaserian versus Straussian Approaches ................................................. 18
Figure 4 - ATLAS.ti Network Diagram of Codes and Concepts .............................. 24
Figure 5 - Emergence of self-efficacy category ......................................................... 44
Figure 6 - Six Cs self-efficacy context diagram ......................................................... 44
Figure 7 - Emergence of "agile is effective" construct .............................................. 46
Figure 8 - Emergence of "agile is efficient" construct .............................................. 48
Figure 9 - Emergence of "agile increases quality" construct .................................... 50
Figure 10 - Emergence of "agile fits with reality" construct ..................................... 52
Figure 11 - Emergence of affective response category ............................................. 53
Figure 12 - Six Cs affective response context diagram ............................................. 53
Figure 13 - Emergence of "agile increases confidence" construct ......................... 55
Figure 14 - Emergence of "agile is engaging" construct ........................................ 57
Figure 15 - Emergence of "agile produces a sense of accomplishment" construct .... 59
Figure 16 - Emergence of "agile increases life balance" construct ............................ 61
Figure 17 - Emergence of intra-inter personal category ........................................... 62
Figure 18 - Six Cs intra-inter personal category context diagram ........................... 63
Figure 19 - Emergence of "agile improves communication" construct ................... 65
Figure 20 - Emergence of "agile increases timely feedback" construct ................... 67
Figure 21 - Emergence of "agile is a product of professional growth" construct ...... 69
Figure 22 - Emergence of the external preference inhibitors category .................... 70
Figure 23 - Six Cs external preference inhibitors context diagram ........................... 71
Figure 24 - Emergence of "management resistance inhibits agile preference" construct .................................................................................................................. 72
Figure 25 - Emergence of "customer resistance inhibits agile preference" construct.. 73
Figure 26 - Emergence of personal factors inhibitors category ............................. 74
Figure 27 - Six Cs personal preference inhibitors context diagram ........................... 75
Figure 28 - Emergence of "change adversity inhibits agile preference" construct ..... 76
Figure 29 - Emergence of "work style inhibits agile preference" construct 77
Figure 30 - Theoretical Model 78
CHAPTER 1

INTRODUCTION

In this chapter we will discuss the general background of agile methods and the importance that developer preference plays in academia and in the industry. A problem statement will be formulated and a research question developed and described.

General Background

Agile methods are a rapidly growing means of developing software. As of 2011 in the U.S. about 40% of companies were using agile (Glaiel, Moulton, & Madnick, 2013). In a 2009 Forrester Research report, West and Grant (2010) find that about 30% of software developers in the sample of over 1,000 software developers are using some form of agile methods. There is also a growing literature base related to agile methods in software development (Dyba & Dingsoyr, 2008) with ample room for continued study (Balijepally, Mahapatra, & Nerur, 2006; Dingsoyr, Dyba, & Abrahamsson, 2008). Not only is agile significant in business and academia, it holds important considerations for management.

This study is an exploration into factors that influence agile methods preference. As more organizations adopt agile methods on increasing numbers of projects, it will be important for management to understand practitioner preferences and take these into consideration to ensure smooth and effective adoption and diffusion strategies.

As organizations seek to find more efficient means to develop software many are turning to agile methods as a way to reduce costs, speed products to market and deal with rapidly changing customer and business requirements (Boehm & Turner, 2004; Dyba & Dingsoyr, 2008). Dyba and Dingsoyr (2008) note the limited theoretical base utilized in agile research indicating a need for additional empirical based studies that generate and confirm theories related to agile methods.
Statement of the problem

Given the rising adoption of agile methods it is important to understand what drives software developer preference. Because industry and management are interested in using agile approaches it is crucial to identify the factors that cause developers to appreciate the approach. Not every developer likes agile. Agile attitude measurement instruments would be valuable as management assesses individual developer preference for agile. In order to assign developers to appropriate projects and integrate them into agile projects it is imperative to know if training will be effective and if so what factors to emphasize in order to increase agile preference. Unfortunately there is no definitive empirical research or theory that that informs us on the factors that influence preference among software developers.

Research Question

This study aims to answer questions regarding the causes of software developer preference for agile methods. What increases preference? What decreases preference? Are these factors malleable or are they fixed attributes resistant to change?

The main focus of the research is to identify factors that drive increased preference for agile methods. A few factors for decreased preference emerged through the study and will be discussed. What is encouraging is that the factors that emerged from the Grounded Theory research method are they type of factors that appear to be modifiable through education and training. This will have important implications for management; these implications are discussed in later sections.

In this chapter we have noted the rapid rise of agile methods within the industry and increased attention among academics. We have also identified some important impacts on academia and industry regarding developer preference of agile. In addition we have stated the research problem and formulated the research question.

Relevance to Industry

In a 2013 survey, VersionOne noted that out of the 3,501 responses collected that 88% of the respondents had personal experience with agile development practices (VersionOne, 2013). They also noted an 11% increase in the number of people indicating that agile is
helpful in accelerating completion of projects. Given the reality of growing popularity of agile methods it is important that we understand what is driving the appreciation or preference of this method.

Our research question is targeted and understanding the drivers for preference, both for and against agile software development methods. The study uses qualitative interviews and the resulting data to discover factors that influence preference of agile.
CHAPTER 2

LITERATURE REVIEW

This chapter will summarize the literature review that lays the foundation for the research. We will investigate information systems development methodologies in the literature. In addition we will review the information systems acceptance and adoption literature and discuss how it relates to the research at hand. Finally, based on the literature review, we will identify a research gap that will be addressed by the current study.

Information Systems Development Methodologies

Over the last decade two decidedly different approaches to Information Systems Development (ISD) have emerged. The traditional approach is characterized by terms like waterfall, sequential, or even spiral development. These approaches are often called “plan-based” or “plan-driven” in the literature (Boehm & Turner, 2004). They emphasize planning, sequential execution, documentation, specific roles and predictability (Balijepally et al., 2006). Philosophically, traditional approaches have sought to impose order and control on the software development effort (Bonner, 2010).

In contrast to the plan-driven approach are agile methodologies. Rather than control and prediction, agile methods seek to react and adapt (Cockburn & Highsmith, 2001). Agile methods have their roots in the 1990s culminating in a manifesto developed in 2001 which stated the essential concepts at the heart of agile methods. The manifesto lists a set of twelve guiding principles developed by the Agile Alliance (Beck, 2001). Among the emphasis in the twelve principles are the beliefs that working software code is a priority over documentation, early and frequent delivery of working software code is a priority, daily collaboration between users and developers, trust in the front line workers (business and technical), face-to-face communication is better than written documents, progress is measured by working software, consistent pacing rather than periodic heroic efforts, emergent rather than prescriptive
The enduring value and importance of the principles found in the Agile Manifesto is confirmed by a recent study performed by Williams (2012). Balijepally et al. (2006) provide a good summary comparison of agile with traditional waterfall methods in Figure 1.

<table>
<thead>
<tr>
<th>Focus</th>
<th>Traditional</th>
<th>Agile</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Individual creativity</td>
<td>Creative teamwork</td>
</tr>
<tr>
<td></td>
<td>Processes and tools</td>
<td>People and interactions</td>
</tr>
<tr>
<td></td>
<td>Technical</td>
<td>Socio-technical</td>
</tr>
<tr>
<td></td>
<td>Optimization</td>
<td>Adaptation and Learning</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process Characteristics</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Readiness</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Process flow</td>
<td>Predominantly sequential</td>
<td>Parallel and iterative.</td>
</tr>
<tr>
<td>User Involvement</td>
<td>Partial</td>
<td>Complete</td>
</tr>
<tr>
<td>Coding standards</td>
<td>Useful but not critical</td>
<td>Highly important</td>
</tr>
<tr>
<td>Dominant mode of</td>
<td>Documentation</td>
<td>Barrier-free interpersonal</td>
</tr>
<tr>
<td>communication</td>
<td></td>
<td>interaction</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project Management</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Key decision maker</td>
<td>Project manager</td>
<td>Development team members</td>
</tr>
<tr>
<td>Management control</td>
<td>Through plans, processes,</td>
<td>Trust based collaboration</td>
</tr>
<tr>
<td></td>
<td>and verification</td>
<td>and self organization</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>People Issues</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Roles</td>
<td>Defined and relatively</td>
<td>Change, often at developer's</td>
</tr>
<tr>
<td></td>
<td>constant</td>
<td>discretion</td>
</tr>
<tr>
<td>Code ownership</td>
<td>Individual accountability</td>
<td>Team based collective</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ownership</td>
</tr>
<tr>
<td>Rewards</td>
<td>Individual</td>
<td>Team based</td>
</tr>
</tbody>
</table>

Figure 1 - Waterfall and agile Comparison

Recent work has been done in an attempt to solidify an academic definition of agile methods. As is often the case in Information Systems practitioners are moving quickly and lead in defining practices without a rigorous academic and theoretical foundation. Conboy (2009) proposes a definition and taxonomy to address this issue. The resulting definition of agile methods is “the continual readiness of an ISD method to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while contributing to perceived customer value (economy, quality, and simplicity), through its collective components and relationships with its environment” (Conboy, 2009). In addition to the definition he offers a three part taxonomy. First to be agile, an ISD method component must contribute to one or more of the following: (i) creation of change; (ii) proaction in advance of change; (iii) reaction to change; (iv) learning from change. Second, to be agile, an ISD method component must contribute to one or more of the following, and must not detract from any: (i) perceived economy; (ii) perceived quality; (iii) perceived simplicity. And third,
to be agile, an ISD method must be continually ready, i.e., minimal time and cost to prepare the component (Conboy, 2009). This definition and taxonomy will be utilized throughout the proposed research.

Clearly the two approaches have very different orientations, plan-driven is command and control oriented with a focus on documents and artifacts while agile is reactive (or proactive), change embracing and people centric.

**Acceptance and Adoption Literature**

The research employed a grounded theory approach, but some context and a theoretical starting point are provided to bolster the confidence that the research will bear fruit in regard to identifying theoretical constructs for antecedents to agile preference. A variety of research has been done in regard to the adoption of software development methodologies. These have varied in terms of unit of analysis, mandated adoption and empirical versus literature review basis of the research.

Much of the adoption and acceptance literature deals with concepts leading to behavior, either intent to use or usage of a system, object or method. Our interest is in attitude rather than usage. Usage based research hinges on the idea that the individual will be mandated to use something (acceptance) or has the option to use something (adoption) and focuses on the antecedents to behavior. The focus of this research is on the sentiment an individual has toward agile methods. We are not specifically concerned with usage but the attitude developers have toward agile methods and the constructs that influence their attitude. But since this research area is the nearest neighbor to our domain of interest we draw upon it to provide a context for the current research.

An interesting study on developer acceptance of methodologies was performed in the context of evaluating five existing theoretical models (Riemenschneider, Hardgrave, & Davis, 2002). The study considered the determinants of individual software developers’ intentions to use methodologies within an organizational mandate. The theoretical models studied were the Technology Acceptance Model (TAM and TAM2), Perceived Characteristics of Innovating (PCI), Theory of Planned Behavior (TPB), and the Model of Personal Computer Utilization (MPCU). The results of the study indicated that the significant factors were usefulness, voluntariness, compatibility, and subjective norm (Riemenschneider et al., 2002).
Table 1 - Riemenschneider et al. Acceptance Constructs

<table>
<thead>
<tr>
<th>Theory Base</th>
<th>Effective Construct</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAM and TAM2</td>
<td>Usefulness</td>
</tr>
<tr>
<td>TAM2 and PCI</td>
<td>Voluntariness</td>
</tr>
<tr>
<td>TAM2 and TPB</td>
<td>Subjective Norm</td>
</tr>
</tbody>
</table>

TAM (Davis, 1985) and TAM2 (Venkatesh & Davis, 2000) define usefulness as the extent to which an individual considers that a system will enhance their job performance. The second version of the Technology Acceptance Model (Venkatesh & Davis, 2000) and the Perceived Characteristics of Innovating (Moore & Benbasat, 1991) both include the construct called voluntariness which refers to the freedom from mandatory adoption. Finally, TAM2 and the Theory of Planned Behavior (Ajzen, 1985) include a useful construct referred to as subjective norm. A subjective norm is the belief that people of significance around an individual are perceived to value a particular behavior. Riemenschneider et al. (2002) used a survey instrument and quantitative analysis to sift through the various constructs of the five acceptance models to identify these three significant constructs which functioned as determinants for use of a particular Information Systems Development methodology.

This study was conducted based on a specific traditional plan-based methodology but the factors provide an interesting framework for our proposed research and will yield interesting comparative information in terms of agile preference amongst developers particularly in voluntary situations compared to the mandated environment of the Riemenschneider et al. (2002) study. In addition to providing factors for consideration the authors also provide a succinct definition of a methodology as “a comprehensive guide to developing a system.”

Chan and Thong (2009) take a less empirical but more theoretical approach to developing a framework of agile acceptance. Through a comprehensive literature review they derive a potential theoretical model identifying possible factors influencing acceptance of agile methods largely mediated by knowledge management. They do this from a decidedly knowledge management approach rather than the more common tools acceptance model as in Riemenschneider et al. (2002). Their proposal is literature based so it is neither derived
directly from empirical data nor subjected to empirical validation. Their resulting conceptual framework is shown below in Figure 2.

Figure 2 - Chan and Thong's Agile Acceptance Conceptual Framework

The literature review has yielded a working academic definition and taxonomy of agile methods. In addition methodology acceptance and adoption is a demonstrated topic of scholarly interest and has been approached from a variety of perspectives. The research gap identified is the need for an empirically grounded theoretical model of individual software developer preference factors for the preference of agile methods. A summary of potential antecedents to individual develop preference for agile methods are summarized in Error! Reference source not found..
Table 2 - Potential Preference Constructs From Literature Review

<table>
<thead>
<tr>
<th>Potential Preference Constructs From Literature Review</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usefulness</td>
</tr>
<tr>
<td>Voluntariness</td>
</tr>
<tr>
<td>Subjective Norm</td>
</tr>
<tr>
<td>Software Development Methodology Self-Efficacy</td>
</tr>
<tr>
<td>Experience</td>
</tr>
<tr>
<td>Training</td>
</tr>
<tr>
<td>External Support</td>
</tr>
<tr>
<td>Career Consequences</td>
</tr>
<tr>
<td>Top Management Support</td>
</tr>
<tr>
<td>Organizational Culture</td>
</tr>
<tr>
<td>Teamwork</td>
</tr>
<tr>
<td>Communication</td>
</tr>
<tr>
<td>Shared Understanding</td>
</tr>
<tr>
<td>Arduous Relationships</td>
</tr>
<tr>
<td>Perceived Ease of Use</td>
</tr>
<tr>
<td>Perceived Compatibility</td>
</tr>
<tr>
<td>Result Demonstrability</td>
</tr>
<tr>
<td>Perceived Maturity</td>
</tr>
</tbody>
</table>

Preference versus Acceptance and Adoption

In his work on emotions, Scherer (2005) discuses affective phenomena in distinction to emotional response. In this discussion he provides a description of preference as being “relatively stable evaluative judgments in the sense of liking or disliking a stimulus, preferring it or not over other objects or stimuli” (Scherer, 2005). This serves as an excellent definition for this research. Scherer (2005) contrasts preference with emotion and attitude. Noting that emotion is a much shorter term response and that attitude is an enduring belief. Preference fits between emotion and attitude in terms of duration and will be one of the key components of our research.
Previous Information Systems works have focused on acceptance and adoption with little focus on preference. Scherer (2005) mentions that preference generates feelings but from a behavioral aspect only generates affinity or avoidance which is different in kind from acceptance or adoption. Both acceptance and adoption are behaviorally more active and decisive than preference.

Since this research is addressing a gap in the literature there are no previously identified preference constructs that address the specific question at hand. The next best opportunity is to use the results from the literature search from the nearest neighbor domains. The constructs listed in Error! Reference source not found. are from the adjacent domains of acceptance and adoption which are conceptually similar to preference although as previously noted distinct in nature. The research approach described in the next section will take into consideration that these constructs may or may not be fully applicable to preference. They are listed here as a bridge from existing literature to the target research topic as the best available information.

**Research Gap**

The literature is clear that there is a significant need for additional empirically based theories on the subject of agile software development. In addition, there is sparse information on why some software developers like agile and others do not. There has been some effort in the area of software development method acceptance, but as noted previously, we differentiate acceptance from preference. This leaves a research gap that this study addresses. Namely, the need for empirically derived theory regarding the preference of software developers for agile methods. We will seek factors of influence through interviews with a variety of software developers and abstract from these conversations the constructs that influence their preference thus developing a theoretical model of agile preference.

As further discussed in the design and method sections, this study will address the gap in literature by performing Grounded Theory (empirical) research to identify antecedent constructs for agile preference and then through the established techniques of the research methodology we will develop substantive theory to explain the observed phenomena of software developer preference for agile development methods.
We have summarized the literature that serves as the basis for the current study. We canvassed the literature on information systems development methodologies. We also discussed the information systems acceptance and adoption literature and how it relates to the current study. Based on the review, we identified a research gap that will be addressed by the current study.
CHAPTER 3

RESEARCH DESIGN

Here we will elaborate on the study’s design goals, the unit of analysis, the participants and research tools used. The study is designed to discover theoretical constructs that influence preference for agile methods. Consequently a grounded theory research method is adopted. The key technical tool employed is a software program called ATLAS.ti which provides a mechanism for analysis and diagraming findings.

Research Design Goals

The goal of the research is to discover factors that lead to a preference for agile methods. We want to understand why software developers like agile software development methods. Initial attempts focused on personality traits (Bishop & Deokar, 2014). Although the findings indicated a relationship between certain traits and preference for agile the relationship was not significantly explanatory. Therefore the current research was envisioned as an exploration of antecedents to agile preference. Due to the exploratory nature of the research a qualitative approach was selected (Eisenhardt, 1989). Furthermore the grounded theory is utilized since it is designed for the empirical discovery of theory in a social setting, precisely the situation for our questions (Glaser & Strauss, 1967).

Unit of Analysis

The research is focused on the software developer. This unit of analysis allows us to focus on those constructs that make agile desirable from a programmer’s perspective. We specifically excluded management and customer perspectives. Both management and customer perspectives are of interest and may be pursued in subsequent research. But for the study at hand the emphasis is to understand why developers like (or in some cases dislike)
agile methods and to build a theory of antecedents of preference from the software engineers point of view.

**Participants**

In light of this unit of analysis we sought participants that have had significant experience as software developers. These participants are from a wide variety of environments from large multi-national corporations to sole proprietorship consultants. We selected participants from a diversity of industries as well. Our participants range from government employees to employees of large retail companies, some with start-up experience, and others with long standing defense industry backgrounds. These participants also were diverse in their ages with some in the early 20s while and some in their 50s. But one thing all the participants have in common is significant experience as practitioners in software development.

**Researcher Point of View**

As a researcher I do not come to this study as a blank slate. I have professional experience as a software developer and software development manager. I have published on personality traits as possible antecedents to agile preference. Keeping these factors in mind I have taken them into account as I collected and interpreted the data for the current research project and have been careful to avoid undue influence in the collection and interpretation of the data.

In addition I am still coming to grips with my personal preference toward agile methods. My actual project experience has been dominated by traditional Waterfall methods and only in the last five years have I become interested in agile methods. Taking a personal inventory I’d say my personal preference is swinging toward agile. Despite these recognized biases, I have used my curiosity and interest to help drive interview questions, but I have tried to remain objective when it comes to analysis and interpretation of the data.
Research Tools

This research utilized a number of tools. During data collection a digital recorder was used for face-to-face interviews to record the conversation. The majority of interviews were conducted over the phone and TapeACall software was used to record those conversations. Once the conversations were completed the digital recordings were sent to a transcription company called VerbalInk.com. VerbalInk transcribed the digital recordings into Word document format. Next these Word document transcriptions were imported into ATLAS.ti, a software program designed to facilitate grounded theory analysis of data. ATLAS.ti is particularly helpful for coding of the data. In addition the standard Office Suite was employed to take notes, track information and sketch out theoretical models. For the writing of the findings Endnote was also used to track, format and search scholarly references.

We have described the research goals and the unit of analysis. We have also described the participants in the study. In addition we have discussed the tools from digital recorders, transcription services and literature and data analysis software. This chapter has provided a summary of the research design.
CHAPTER 4

RESEARCH METHODOLOGY

In this chapter we will go into detail regarding the selected research methodology. A description of the qualitative grounded theory method is provided. A discussion of random versus theoretical sampling is provided. And finally, data collection and analysis from a grounded theory perspective are explored.

Qualitative Grounded Theory

The research follows the grounded theory methodology. The goal of grounded theory methods is “the discovery of theory from data systematically obtained from social research” (Glaser & Strauss, 1967). Grounded theory involves collecting data and then analyzing this data by breaking it down, sorting it, then synthesizing it (Charmaz, 2006). The aim is to develop theory that emerges from the data itself. In the case of this research the goal is substantive theory rather than formal theory. Glaser and Strauss (1967) distinguish substantive theory as theory limited to a particular area whereas formal theory is far more fundamental and may span multiple areas of study. This research seeks to specifically form theory in the area of preference for Agile Information Systems Development methods.

Theoretical Sampling versus Random Sampling

One of the distinguishing features of the Grounded Theory research method is that in contrast to Quantitative methods, Grounded Theory uses theoretical sampling rather than random sampling (Strauss & Corbin, 1994). Theoretical sampling uses directed selection of data rather than random data collection because the goal is to enrich emerging core categories rather than statistically represent a target group (Jones & Alony, 2011). Random sampling
ensures the external validity in a quantitative study, but external validity is accomplished through the emergent theoretical model in a Grounded Theory study (Yin, 2008).

In this study in the early interviews it became apparent that participants with limited or no practical experience in agile methods were of limited value. Although these early participants provided insight on some of the negative preference influences, their ability to provide rich data on positive preferences for agile were very limited. Subsequent interviews focused on individuals that had more significant experience with agile and provided the depth of information that contributed to forming the concepts and categories ultimately resulting in the emergent theoretical model.

**Data Collection**

Data collection was accomplished through the use of semi-structured interviews. The interview guide is included in Appendix A. These questions formed the basis for the interview but as the discussions progressed a wide range of topics were covered and additional follow-up questions were used to go deeper into areas of interest (Charmaz, 2006). In some cases, it was obvious that certain lines of questioning were of little value and so they were skipped.

Only one interview was conducted face-to-face, all other interviews were conducted over the phone. All interviews except one were recorded and then transcribed into written format. The one exception was due to recording equipment failure. In that particular case the researcher made written notes within an hour of the interview. The written documents form the data repository for the research.

Data collection forms the foundation of the resulting theory. In grounded theory, the theory emerges from the data. The data was collected through sixteen different interviews. The initial set of interviewees was chosen from my social network. At the conclusion of each interview part of the protocol was to ask if the person knew of any additional subjects that might fit the study criteria.

Each interview was recorded. The single face-to-face interview was recorded using a digital recorder while the phone interviews were recorded using an iPhone application called Tape-A-Call. Both recording methods resulted in a digitally recorded file of the conversation. This file was then sent to an online transcription service called VerbalInk. VerbalInk uses
human transcriptionists to create written Microsoft Word documents of the digital files. These transcriptions were reviewed against the recordings and corrections were made to the transcriptions based on the comparisons. In order to maintain confidentiality identifying information was removed and replaced with generic place holders.

The result of the data collection is 397 minutes of recorded conversation which converted to 165 pages of transcribed text. Reviewing the transcriptions and comparing the conversation with the interview protocol it is clear that the interviews were merely semi-structured. The basic outline of questions prescribed in the protocol was adapted as appropriate for each individual participant. As interesting topics surfaced spontaneous questions were posed and additional relevant data was collected. When ambiguity arose follow-up questions were issued to clarify the intended meaning. When participants became repetitive with their answers or it was clear that a subject area was exhausted questions were altered or eliminated for efficiencies sake. Each interview included the core questions or closely worded alternatives. These redacted transcripts in Microsoft Word document format comprised the data that was used for analysis and ultimately revealed the theory proposed by this research.

Data Analysis

In Grounded Theory data collection and data analysis go hand in hand. They are iterative steps involving constant comparison of data (Corbin & Strauss, 1990). The research followed a three-fold approach to coding which is at the core of Grounded Theory data analysis (Charmaz, 2006; Corbin & Strauss, 1990; Glaser & Strauss, 1967).

Two distinct styles of Grounded Theory have emerged. Initially Glaser and Strauss documented Grounded Theory as a research approach in 1967 (Glaser & Strauss, 1967) but later split the approaches. Glaser retained the original form while Strauss developed a more defined data analysis process. The key differences are summarized nicely by Jones and Alony (2011) in Figure 3 below.
This research primarily follows a modified Glaserian approach which includes the use of the three tiered coding strategy proposed by Corbin and Strauss (1990). Two items that both approaches agree on are the need for constant comparison and the writing of memos. Constant comparison is the process of comparing code to code at all levels and the notion that analysis starts early and is done continuously throughout the research. Memos are a means to record analytic thoughts and insights throughout the research process.

<table>
<thead>
<tr>
<th>'GLASERIAN'</th>
<th>'STRAUSSIAN'</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beginning with general wonderment (an empty mind)</td>
<td>Having a general idea of where to begin</td>
</tr>
<tr>
<td>Emerging theory, with neutral questions</td>
<td>Forcing the theory, with structured questions</td>
</tr>
<tr>
<td>Development of a conceptual theory</td>
<td>Conceptual description (description of situations)</td>
</tr>
<tr>
<td>Theoretical sensitivity (the ability to perceive variables and relationships comes from immersion in the data)</td>
<td>Theoretical sensitivity comes from methods and Tools</td>
</tr>
<tr>
<td>The theory is grounded in the data</td>
<td>The theory is interpreted by an observer</td>
</tr>
<tr>
<td>The credibility of the theory, or verification, is derived from its grounding in the data</td>
<td>The credibility of the theory comes from the rigour of the method</td>
</tr>
<tr>
<td>A basic social process should be identified</td>
<td>Basic social processes need not be identified</td>
</tr>
<tr>
<td>The researcher is passive, exhibiting disciplined restraint</td>
<td>The researcher is active</td>
</tr>
<tr>
<td>Data reveals the theory</td>
<td>Data is structured to reveal the theory</td>
</tr>
<tr>
<td>Coding is less rigorous, a constant comparison of incident to incident, with neutral questions and categories and properties evolving. Take care not to ‘over-conceptualise’, identify key points</td>
<td>Coding is more rigorous and defined by technique. The nature of making comparisons varies with the coding technique. Labels are carefully crafted at the time. Codes are derived from ‘micro-analysis which consists of analysis data word-by-word’</td>
</tr>
<tr>
<td>Two coding phases or types, simple (fracture the data then conceptually group it) and substantive (open or selective, to produce categories and properties)</td>
<td>Three types of coding, open (identifying, naming, categorising and describing phenomena), axial (the process of relating codes to each other) and selective (choosing a core category and relating other categories to that)</td>
</tr>
<tr>
<td>Regarded by some as the only ‘true’ GTM</td>
<td>Regarded by some as a form of qualitative data analysis (QDA)</td>
</tr>
</tbody>
</table>
**Open Coding**

Open coding is the first step in analysis of the data. Each relevant piece of information in the written text is analyzed and tagged with a descriptive verbal code. These codes represent the meaning and can often use a particularly vivid word from the dialog itself. These types of codes are referred to as “in vivo” codes (Charmaz, 2006). The tool ATLAS.ti served as a database both for the transcribed content from the interviews and for the coding. This use of a database for research artifacts establishes a chain of evidence which strengthens construct validity (Yin, 2008). The ATLAS.ti tool was used to manage the association of codes with sections of text from the interviews. This tool allowed for systematic organization and the ability to visually represent the relationship of open codes to later abstractions like axial and selective codes and memos using network diagrams.

Charmaz (2006) provides guidelines to be followed by Grounded Theory researchers during the initial coding phase shown in Table 3.

<table>
<thead>
<tr>
<th>Code for Coding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remain open</td>
</tr>
<tr>
<td>Stay close to the data</td>
</tr>
<tr>
<td>Keep your codes simple and precise</td>
</tr>
<tr>
<td>Construct short codes</td>
</tr>
<tr>
<td>Preserve actions</td>
</tr>
<tr>
<td>Compare data with data</td>
</tr>
<tr>
<td>Move quickly through the data</td>
</tr>
</tbody>
</table>

These guidelines provided direction for the research performed in this study.
**Axial Coding**

Once the granular data has been tagged with succinct and descriptive codes (open/initial coding) it is ready to be sorted and grouped into related conceptual clusters. This is variously described as axial coding or focused coding (Charmaz, 2006).

The goal with focused coding is to begin to form conceptual groupings. In the open coding activities the interview data was fractured into its most granular units. During focused coding the research compares code against code and looks for emergent relationships that form conceptual groupings. These concepts form the constructs of the developing theory.

**Selective/Theoretical Coding**

The next phase of coding is theoretical or selective coding. In this phase the researcher abstracts from the concepts to broader categories that form the essence of the emergent theory. Corbin and Strauss (1990) suggest that a core unifying category should emerge during selective coding. In the original writing by Glaser and Strauss (1967) they did not have as granular separation into three types of coding. They outlined two phases and did not suggest a single unifying core category. They allowed for the emergence of themes that as they developed formed theory. Theoretical codes are developed to explain and describe the relationships between the categories developed at the axial or focused coding level (Charmaz, 2006).

**The Six Cs Approach**

Corbin and Strauss (1990) introduce a structured approach to provide context and relationships to the emerging categories. Their approach provides guidance to the grounded theory research and a mechanism to communicate the emergent nature of theoretical constructs and their relationship to the emergent categories. The six Cs are listed and described in Table 4.

Table 4 - Six Cs Taxonomy

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context</td>
<td>The setting where the category is at play</td>
</tr>
<tr>
<td>Condition</td>
<td>A factor that is prerequisite for the category to emerge</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>Cause</td>
<td>A reason for the category to occur</td>
</tr>
<tr>
<td>Consequence</td>
<td>Outcomes or effects as a result of the occurrence of the category</td>
</tr>
<tr>
<td>Contingency</td>
<td>A moderating factor between categories and consequences</td>
</tr>
<tr>
<td>Covariance</td>
<td>Categories or parts thereof can co-vary with each other, meaning that a change in one category inflicts a change in the other</td>
</tr>
</tbody>
</table>

Our approach will utilize 5 of the 6 Cs in Table 4. Covariance is excluded because our data is limited and does not provide enough information to perform comparative analysis. The diagram that will be used in a later section to depict the modified six Cs is adapted from an article by Van Waardenburg and Van Vliet (2013).

In this chapter we have described the qualitative grounded theory method of research. We have also contrasted theoretical sampling with random sampling. The chapter then concludes with coverage of grounded theory data collection and data analysis describing the process for discovering emergent theory from the data.
CHAPTER 5

FINDINGS AND THEORETICAL MODEL

FINDINGS

This chapter will provide information about the interviewed participants. It will elaborate on the detailed codes and their relationships that emerge from the data and provide a diagram of the same. A table overview of the findings is presented followed by a detailed discussion of each construct. The chapter concludes with a discussion of the theoretical model and the relationships between the constructs.

Interview Information

The research involved 16 participants ranging from junior professional level and age 22 to senior professional level with an age of 54. These 16 individuals work in 10 different companies ranging from small individual proprietorships to large multi-national corporations. The interviews resulted in 165 pages of transcribed text based on 397 minutes of conversations. This information is summarized in the table below.

Table 5 - Interview Summary Information

<table>
<thead>
<tr>
<th>Demographic Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of participants</td>
</tr>
<tr>
<td>Number of sites</td>
</tr>
<tr>
<td>Age range</td>
</tr>
<tr>
<td>Company size</td>
</tr>
<tr>
<td>Experience level</td>
</tr>
</tbody>
</table>
Analysis of the 165 pages of transcripts resulted in 178 individual codes (see Appendix B: Initial Codes). Not all of these codes ended up being significant. These codes are the result of the open coding phase of analysis. These initial codes emerged from the interview transcriptions to capture the meaning and ethos of the thoughts of the participants. These codes were grouped into twenty seven initial concepts (see...
Appendix C: Focused Codes). These concepts constitute the focused coding. Some concepts were thinly represented in the data and deemed not to be significant from a theoretical perspective. Ultimately fifteen concepts formed the basis for the three main categories or themes that emerged from the data. With ATLAS.ti the relationship between the open codes and concepts are visualized in the network diagram as shown in Figure 4 below.

Figure 4 - ATLAS.ti Network Diagram of Codes and Concepts
Table 6 shows the progression from open coding to concepts. Open codes are editorial summaries of participant narrative from the interviews. A special case of open codes are called in-vivo codes which are codes stated in the same words that a participant used. In Table 6 there is no distinction made between researcher created codes and in-vivo codes. The researcher analyzes, often line by line, the interview text and coins words or phrases to interpret and capture the meaning from the participant’s conversation. Concepts are also known as axial codes in the Corbin and Strauss taxonomy (1990). Concepts are an increasing level of abstraction eliciting the meaning from related codes into a word or phrase and typically become the constructs of the emerging theory. Finally, categories are the overarching labels that capture the theoretical significance of a set of related concepts. Corbin and Strauss (1990) refer to categories as selective codes in their taxonomy.
The open, focused and theoretical coding results are summarized in Table 6.

Table 6 - Findings Summary

<table>
<thead>
<tr>
<th>Categories – abstractions based on the concepts</th>
<th>Concepts – abstractions from the open codes</th>
<th>Open Codes – created by the researcher based on interview transcription narratives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agile increases self-efficacy in software developers which leads to preference for agile methods</td>
<td>Effective</td>
<td>Waterfall never worked, good technique, more effective, productivity, best way, better results, how software should be done</td>
</tr>
<tr>
<td></td>
<td>Efficient</td>
<td>Avoids useless work, quick up-to-speed, great learning, focus on what is important, faster productivity, early id of blocking issues, learn product and project, timely delivery</td>
</tr>
<tr>
<td></td>
<td>Quality</td>
<td>Improved quality, more reliable estimates, design for change, innovative</td>
</tr>
<tr>
<td></td>
<td>Negative perceived impact on code</td>
<td>Speed breeds poor quality, short term impact on productivity due to learning, inconsistent with low-level coding, retroactive unit test writing, feature attention deficit disorder, daily disruption of stand-ups</td>
</tr>
<tr>
<td></td>
<td>Good fit/harmony</td>
<td>Fit with industry demands, best practices, reality, natural</td>
</tr>
<tr>
<td></td>
<td>Negative of partial adoption</td>
<td>Waterfall in agile clothing, written requirements</td>
</tr>
<tr>
<td>Agile produces positive affective responses leading to preference for agile methods</td>
<td>Engagement</td>
<td>Happy because coding, love to create software, better management involvement, visualize rather than imagine, happier, small task size, creative, fun</td>
</tr>
<tr>
<td></td>
<td>Accomplishment</td>
<td>Sense of accomplishment, well defined work, accountability, clearly defined product, clearly defined task, immediate gratification, fun, coding is what I do, programmers don’t like writing docs, feels right, (-) Kanban feels relentless, self-improvement, busy</td>
</tr>
<tr>
<td></td>
<td>Confidence</td>
<td>Unit testing, test driven development, change without worry, safety net, confidence, no fear, reduced risk, adaptable</td>
</tr>
<tr>
<td></td>
<td>Balance</td>
<td>Work-life balance, doesn’t rely on heroes, no death march</td>
</tr>
<tr>
<td>Intra-Inter Personal factors drive preference for agile methods</td>
<td>Communication</td>
<td>(-) large scale, collaborative, collective, efficient problem solving, (-) distributed team difficulties, stand-up, customer involvement, constant feedback, proximity</td>
</tr>
<tr>
<td></td>
<td>Feedback</td>
<td>Quick feedback, short cycle, knowledge/learning</td>
</tr>
<tr>
<td></td>
<td>Positive Social Influence</td>
<td>Planted a seed, opens my mind, picked up from peers, everyone is moving to it, social pressure, modern, they were successful, world of respect</td>
</tr>
<tr>
<td></td>
<td>Negative Social Influence</td>
<td>Neutral is negative, management resistance, Customer requires waterfall</td>
</tr>
<tr>
<td></td>
<td>Negative personal characteristics</td>
<td>Fear of learning, no interest in personal growth, opposed to change, isolationist/independent, skeptical of anything new</td>
</tr>
</tbody>
</table>
The process of open or initial coding resulted in over 170 initial or open codes. Throughout the initial coding phase the codes were constantly compared and continually sorted to identify relationships and common concepts in the interview data. Through this process of comparison and sorting common themes emerged from the data. These themes form a higher level of abstraction above the codes to provide a unifying conceptual basis for the related codes. These concepts form the middle column of Table 6 above. As the concepts became firm another level of abstraction was embarked upon. A similar comparison and sorting effort was applied to the concepts in order to identify high-level categories that account for the related concepts that emerged from the data. These categories form the left most column in Table 6 and provide structure to the concepts. As we will see later in this chapter the concepts form the basis for the propositions in the emergent theoretical model.

Through the Atlas.ti tool there is a clear chain of evidence from the concept, back to the open code and ultimately to the interview narrative. This line of evidence solidifies the emergent nature of the theory to specific narrative from the participants producing “grounded theory”. Later in this chapter we illustrate the emergence from interview quote through open code all the way to theoretical construct to provide confidence in the relationship between the theoretical constructs and the actual narrative from the participant interviews.

The findings can be summarized into three broad categories or themes. The first category is self-efficacy. Participants conveyed that agile methods affected their behavior in a positive manner in regard to creating software and delivering value to customers. They contrasted this with the behavioral aspects of the traditional or waterfall methods which had the effect of impeding the delivery of value to customers. We will explore this theme in more detail as we amplify it with concepts and detailed codes that emerged from the data.

The second category is labeled “affective response” because there were certain emotional benefits that participants described as reasons that they preferred agile methods. These feelings ranged from a sense of satisfaction and accomplishment to being described as just plain fun. The participants said the most satisfying part of their job as a software developer was writing software and because that is the focus of agile methods they said they derived pleasure, confidence and a sense of life balance when they engaged in agile methods.

The third category is labeled “intra-inter personal.” Many of the statements from participants had to do with either personal or inter-personal effects. This category included
factors that increased preference and also decreased preference. Participants conveyed that inter-personal communication and collaboration was enhanced through agile methods. This category also contains a few factors that led to some disfavor towards agile methods. When software development teams grew in size or the number of teams exceeded a threshold communication became more difficult detracting from the perceived effectiveness of agile methods and consequently decreasing preference.

Self-efficacy

Self-efficacy can be defined as participants feeling more empowered and effective when they used agile methods. They described being productive and efficient, producing higher quality software, and that agile fit the way software is really developed. One participant who worked on a project that only partially adopted a few agile practices did not experience this same self-efficacy.

Effective

Effectiveness is one of the strong concepts that emerged from the data. Developers that appreciated agile felt that the traditional software development methods were less effective and believed that agile was more effective. A typical comment in regard to the effectiveness of agile is “I just found that things like agile development seemed to produce better results” or “traditional waterfall … never worked.” Many developers noted that agile implemented industry best practices and that it was the best way to develop software.

One of the appealing factors of agile is that projects are more effective as reflected in their higher success rate. A participant mentioned “I have seen comparatively with Waterfall and agile, the success rate is really high in agile.” Other participants mentioned that agile is more productive due to agile’s focus on the actual software over documents and the related notion of enhanced inter-personal communication from face-to-face conversations rather than communication mediated through documentation. Another self-efficacy contributor was the perception that agile allowed the project to be broken into small manageable chunks of work. This allowed the developer to focus on a specific task contributing to their effectiveness. One participant said that after experimenting with a number of approaches he felt that agile made him the most effective.
One participant said “I think agile is how software should be developed.” This is an extremely strong statement, practically a moral imperative. This developer felt so strongly about the effectiveness of agile that he believed it should be the normative approach for all software development. Clearly there is a sense among developers that have experience with both Waterfall and agile that agile is the more effective method of software development. This sense of self-efficacy raised the preference of these developers for agile methods emphatically.

Efficiency

The idea of efficiency resonated with a number of participants. They pointed out that in traditional/waterfall approaches it was easy for developers to become disengaged and waste time especially during documentation phases where there was a specific time allotted and then customers had to review which lead to developers being unproductive. In contrast to that their experience in agile was that the granular task sizes, daily discussions about progress and focus on producing working code made much better use of their time and allowed them to produce more features per time period than did the waterfall approach. As one person said, “Agile is fast.”

A common detractor from waterfall and consequently a positive factor for preferring agile is the disconnectedness of documentation suggesting that documentation as done in Waterfall is wasteful. A number of participants noted that although they felt like some form of documentation was beneficial in their experience documentation in the waterfall process was too heavy, detailed and quickly became out of synch with the real code in the project. Then since the documentation was out of sync with the actual code for the project it lost its utility and value. Thus documentation as practiced in traditional waterfall was inefficient. As participant #13 says:

We either have to spend time updating the spec now like over and over and over again or else what would be common to me is that the specs then just doesn’t [sic] get updated and now it’s not accurate, so it’s not a useful document anymore. Both of those things seem to me to take up a lot of time.

This same participant also mentioned “I usually feel like our documentation is a lot bigger, wordier than it needs to be. I think some of the times when it goes into real specific
detail [it] is usually not helping.” Capturing a similar sentiment about the inefficiency of waterfall but not specifically related to documentation, participant #16 says:

When I used to be developing as part of waterfall cycle … there were like two days a week where I would just work fifteen hours and write the whole code and then forget about it, three days I would be just testing it, taking it easy, you know.

Another element of efficiency that surfaced was an increased ability to get up to speed on a project and understand both the big picture and the granular details. This is a result of two components of agile method practices. Daily stand-up meetings are a common feature of agile practices. Each day members of the team get together and discuss what they accomplished since the previous meeting, what they intend to accomplish before the next meeting and any impediments they are facing (Schwaber, 2004). New members to a project found these daily stand-up meetings an excellent venue to get the big picture and understand the range of activities occurring on the project. The sense was that in Waterfall you would have to read the entire documentation to grasp the scope of the project and read through myriads of status reports to determine the current state of affairs. But through agile’s stand-up meetings new team members quickly got up to speed on the project by listening to each team member during the daily meetings. A common practice in eXtreme Programming (XP), a popular agile method, is pair programming. This is where two team members work at one computer and program together. When a new member joins a project and is paired with a more experienced team member it provides a natural context to learn about the details of the current application area. A participant noted that through pair programming it was easy to ask questions and since the two enjoyed a common context there was no need to provide background information, the questions and answers fit into their present experience making the learning process both efficient and socially straightforward.

An important efficiency of agile is the involvement of the customer. It bypasses the need to constantly search through requirements documents to get clarification. Instead you can go directly to the customer and get the issue resolved. As one participant said:

The ability as you get deep into creating the product, and hit a roadblock you know you don’t have to continue revisiting the requirements or revisiting the deliverable that you’re working on and saying, “Hey, look what do you think?” And to be one to want to invest three days of work and effort to work around these small requirements here or can we just go back and check with the customer and see if it’s really worth it? In
the past I guess you spent a lot more time with the Waterfall with a very fixed signed-off requirement saying it shall do this.

Agile in general encourages periodic team retrospection (Beck, 2001). This allows teams to fine tune their processes and improve efficiency. This was communicated by a developer in the following statement, “we’ve analyzed it and these things don’t really help us and we don’t have to do them; so that will give us more time to work on the actual functionality.” This self-optimization process increases efficiency over time and accelerates the delivery of software products.

One final efficiency related consideration is that agile suggests working on the highest value features first. A participant mentioned that in his experience with Waterfall they may spend too much time on small details or unimportant features, thus wasting valuable resources and time. He says:

It was a long time before delivering any kind of value to the customer. It allowed the development teams to spend … large amounts of time developing portions of software that weren’t necessarily as critical to the customer. Especially sometimes the characteristics of some of the best elements they wanted to kind of make them polished them [sic] or pretty, but many times a large portion of that product used aren’t needed, they spent their effort on a product that really wasn’t required.

Quality

Improvements to the end product were a frequent comment from participants. They attributed much of the improvement to quality to the use of unit testing. Unit testing helped them focus on the task at hand, which as mentioned previously was usually small in scope due to the short iteration cycle. They also designed and wrote their code in such a way as to be adaptable and flexible, resilient to change. In addition to the code itself were the automated unit tests which provided a safety net that gave them confidence to make changes without negatively affecting the intended functionality. The automated unit test framework then fostered a willingness to adapt to changing requirements and improve the structure of the code. The net result is higher quality code that better reflects the user’s needs as illustrated in this quote:

The second thing would be on the issue of quality. [I] mentioned unit testing, test and development, I think that those things have huge dividends for me, especially at being able to move very rapidly and adapt to changes very quickly, and that’s the whole idea
behind unit testing, obviously, is that you have a safety net and your design has been
designed with change in mind. But actually being forced to go through that exercise I
think helps get you there and it really does pay for itself when you get later on in the
project and changes don’t seem as intimidating as they would have otherwise seemed.
So quality is benefited as a result for sure.

One negative factor was mentioned in relation to quality and agile methods. A
participant thought that the emphasis on short cycles and quicker delivery of features
sometimes resulted in reduced quality since developers may duplicate code rather than
research to find where it may have been implemented already or find a similar piece of code
and generalize it to meet both needs. Only one of the sixteen participants mentioned this as
an issue but it is something to consider:

Well what you’ve just done is you’ve added to the code base, you’ve probably
implemented it in a way that’s different than anyone else who’s implemented it, and
you really didn’t consult anybody because you’re trying to get done very quickly. The
way I see it playing out on the actual code base is it just mushrooms the amount of
code that’s sitting out there and creates inconsistency in the product.

This isn’t an inherent problem of the agile method since the manifesto explicitly
focuses on “continuous attention to technical excellence and good design enhances agility”
(Beck, 2001) but could be a perceived issue with agile’s emphasis on early delivery of value
to customers.

**Fit**

Fit with reality is a strong concept that emerges from discussions with participants. In
their experience it turns out that the Waterfall method just doesn’t fit with the way things
work in the real world of software development. More often than not change happens and it
happens even when the specification has been signed off by the users as complete and
accurate. When developers work on features as specified they frequently encounter change.
The specification was incorrect or the user has altered their desire, regardless, the feature
needs to be modified. With Waterfall, not only does the code and tests that have been created
need to be updated so does the associated documentation. Because this is such a common
occurrence it becomes discouraging to those who have followed the Waterfall method. Here’s
how one participant summarized this experience:
My experience was with failures at Waterfall and I found that in my early projects there, we’d be working with so-called signed-off specs but, you know, the spec had to end up being rewritten at least once if not a number of times, both after it was supposedly signed off. So just – it never worked, and getting away from the idea of hey, this is done code against it and just sort of realizing that it’s gonna be an ongoing, evolving process I think made it a bit less frustrating.

Or as another participant put it:

Something that I kind of think about a lot I would say where we spend a lot of time – it’s like usually what happens is the PM [Project Manager] comes to look in and now I have [the] spec it’s ready, let’s start working on the feature now and then as soon as you start making the feature then we potentially kind of go through each little piece of functionality. We kind of go and say, “Oh now that I’ve charged work on it I think maybe we should do this other approach, maybe that would work better.” We discuss it and we say, “Okay, yeah, we’re actually going to do it different from the spec,” and then we either have to spend time updating the spec now like over and over and over again or else what would be common to me is that the specs then just doesn’t get updated and now it’s not accurate, so it’s not a useful document anymore. Both of those things seem to me to take up a lot of time.

A number of participants expressed that agile just felt right. Many said they had been doing it without knowing that is what it is called. It fit their perception of the best way of building software, the best way to deal with the reality of quick deliver, quick feedback, changing priorities, and changing requirements. As one participant described his feelings when practicing agile, “I feel right in my own self.” Another participant says:

So I’ve been using agile without knowing that it was actually agile at a personal level and it just kind of you know, carried in towards my educational and later on, … into my professional life and it [sic] just kind of like … the way I was doing things.

Participants mentioned the complexity of software, the difficulty in nailing requirements down, the fact that sometimes you don’t know how to solve the problem until you start working on it which naturally leads to a highly iterative approach consistent with agile methods. They felt that agile is more consistent with the reality of how software is built. Some felt like agile was how they dealt with life, not just software, so it was a natural extension of how they deal with complexity and ambiguity. When discussing the agile approach to software development one participant said, “To me, on a personal basis, that’s how I look at life.”
Partial Adoption

One participant had a negative experience in terms of self-efficacy and it is attributable to the way that his organization adopted agile. It was only for a single project, and agile was only partially and shallowly implemented. The group did regular stand-up meetings and used shorter cycles but their requirements were gathered by someone else with no developer-customer interaction. In fact, the requirements were entered into a system and then retrieved by the developer. The developer implemented the requirements and then passed the code on to testers and only dealt with bug reports. It became clear during the interview that the approach was a form of Scrum but the actual practice was still Waterfall in most respects. Interestingly enough this resulted in a negative perception toward agile for this participant. It seems like a surface level or partial adoption impinged upon preference for agile. Only one participant had this type of partial adoption experience so this will not form a significant component of the emerging theory but may be a fruitful avenue for further exploration and research.

Self-Efficacy Summary

Participants clearly felt better able to develop software following an agile methodology over a traditional Waterfall approach. They described being more productive, focusing on work that mattered to customers. They were more efficient, performing less rework due to changing requirements and priorities; they focused on smaller tasks and better utilized their time. The quality of their work improved due to quick feedback and improved communication with customers and quicker assimilation into a new project. Developers also felt that agile just fit better with the reality of software development and for that matter the way life is lived.

Affective Response

Participants mentioned a number of emotional benefits that they experience while practicing agile software development. They talked about being more engaged, having a heightened sense of accomplishment and a sense of satisfaction. They were more confident and felt they had a better life balance because of agile methods.
Engagement

As we have seen in some of the previous quotes participants frequently felt frustrated with Waterfall when change occurs and rework is required, especially when the rework involved updated documentation. For example, here is what one developer said, “My eyes glazed over now because this is spec time and it’s boring and later on the fun stuff happens.” In contrast to this participants found agile highly engaging, even fun. One reason for this heightened engagement with agile is, as more than one participant said, it focuses on coding and coding is fun. It is why people become software developers, to create software. As one participant said, “I think the happiness of the teams are higher, because in a classic waterfall thing the only time the teams are really happy is in the middle when they’re developing code.” He continues “Because software engineers love to create real things. Like all the morale events in the world, all the bonuses, all that kind of stuff, like let them create real things and that’s going to do the most to improve their morale.”

Not only are the developers more engaged with agile it turns out that management is also more engaged. Note what one developer described about management on an agile project:

They'll be … more involved because they are to meet – they are to schedule a meeting on a day-to-day basis to find out what’s going on and all that stuff. So yeah, overall, yeah they have to be involved to get good facts about what’s going on at the early stages, I think, unlike waterfall where things get lost for months. So yeah I think they’ve been involved.

Others described the PM as being much more involved and collaborative. On one team prior to moving to agile the developer said the PM had a tendency to hand off the specification and then disengage only checking in to get progress reports. This was frustrating because sometimes there were problems with the spec but it was totally up to the programmer to figure out inconsistencies or to fill in missing information. This all changed when they moved to agile. Now there is much more collaboration and therefore the PM is involved with helping solve the problems resulting in higher engagement, they are solving problems together and the PM takes an interest in each issue.

Another area of increased engagement is with the customer. In a Waterfall approach customers are typically engaged in the early requirements gathering stage and then again in
the User Acceptance Testing phase. In between there may be months if not years where the
customer is distanced while the developers do their thing. Numerous difficulties arise. For
one thing it is difficult for the customer to visualize the actual software during the
requirements phase. As one participant said, “If you're not a good visualizer – as business
owner, if you don’t have good visualization, then the project is in trouble in waterfall.” For
another, they are out of the loop during the design, development and system testing phases
increasing their anxiety over the outcome of the project. As one participant said:

I personally prefer agile because of the visualization perspective in the sense like the
customers can see what they're going to get in a shorter period of time rather than just
waiting and waiting until they get to – until they get to Year 3, they cannot see the
screens or the product that they wanted to see for so long.

So there are a number of dimensions that engagement is increased through the agile
method from individual programmer to project manager to customer, each will be more
engaged on an agile project. This increased engagement is a driver for preference of agile
methods by developers.

**Accomplishment**

Developers described a sense of accomplishment when employing agile methods for
software development. Many participants mentioned that they like the fact that in agile their
work is broken down into small tasks and the tasks can be implemented fairly quickly making
progress is visible. This provides the sense of accomplishment. Note how one participant
described this:

I think part of the fun factor is just, everybody likes a sense of accomplishment and we
are an immediate gratification type of society if you’re not I’d question whether or not
you’re human. I think you get that more of an immediate gratification and, I’ve coded
something, tested it, I’ve delivered it and I’ve gotten some feedback on it. So I’ve
gotten some feedback and I’ve got the feedback loop established and then I go and I
do the next iteration and it’s that same repeated pattern, it’s that same repeated instant
gratification pattern. And that’s one thing I hear from … a pseudo exit interview if you
will and a lot of it is just a sense of accomplishment. They feel as if they’re walking
out of this door, they’ve written some software that’s actually gonna be used by
somebody.

Another participant shared similar sentiments:
What I like about it is what you’re doing either with Kanban or with the Scrum part is you’re having a succinct block of work, like a small block of work that is well defined and it has that end goal.

The accomplishment is in two dimensions, first it derives from the small task size that provides a well-defined unit of work that can be completed and progress noted. The other is that the short cycles deliver working software to customers. The regular delivery of software to customers provides a sense of accomplishment.

**Confidence**

Confidence emerged as a strong benefit of employing agile methods. Many different participants highlighted their feeling of confidence resulting from agile. This is due to two different factors. The first is the confidence to make changes. Change may come in two forms, technical improvement or customer driven. The basis for the confidence to make changes is based on automated unit testing. Without tests participants spoke of fear and made statements like:

> When you had to fix a bug or make changes you absolutely did not do anything but fix only the exact little spot, small as possible and make that change. So even if you notice that oh, there’s several lines of code here that looks like it’s repeated over there, maybe I should put that into a method to clean it up. Nope, can’t do that. Because of the fear that you’re gonna break something and that you won’t know you broke it.

Contrast that sense of fear with the confidence to change that grows from knowing that you have a comprehensive set of automated unit tests as expressed by this participant:

> Unit testing, test and development, I think that those things have huge dividends for me, especially at being able to move very rapidly and adapt to changes very quickly, and that’s the whole idea behind unit testing, obviously, is that you have a safety net and your design has been designed with change in mind. But actually being forced to go through that exercise I think helps get you there and it really does pay for itself when you get later on in the project and changes don’t seem as intimidating as they would have otherwise seemed.

The terminology used by participants in this area is illuminating. They talk of “fear” and “safety nets” and “confidence.” There is a true sense of the emotional nature of software development and how there is emotional satisfaction when using agile methods.
In addition to the confidence that comes from the safety net of unit tests there is also a sense of confidence that comes from the reduced risk of short cycles, early delivery and quick user feedback. As one participant said:

Because in IT especially you know, I mean look at the problems that people have faced with waterfalls. Like you know, you don’t find until the whole thing has really gone too far. Like you’re hitting – it you lose, you’re going to lose big time. So the risk is very high. In agile, that’s not there. You’re just like you know, doing repeated evaluations of where you are. And to me you know, on a personal basis, that’s how I look at life right?

Another participant related his confidence because using agile engaged stakeholders early and allowed them to visualize the system, to make the intangible tangible and allow them to provide feedback and see progress thus reducing risk.

So agile methods increase confidence of software developers and embolden them to make the necessary technical and feature changes all the while knowing that overall project risk is being reduced by early and frequent delivery giving customers the opportunity to provide quick feedback which facilitates value in each cycle.

**Balance**

“Waterfall, in my experience, towards the end of the release, it tends to be hell.” So says one participant who goes on to say “People’s work-life balance suffers immensely so I think that’s another sort of discrepancy. People don’t talk about that a lot but it’s an important distinction between Waterfall and Agile.” This sentiment is well known among those that have experienced a difficult project using Waterfall. The final phase is often known as the “death march.” This leads to dissatisfaction and disruption in the personal lives of software developers. Agile recommends an indefinitely sustainable pace (Beck, 2001). Participants noted the difference between work-life balance when using agile methods compared to traditional approaches finding that agile provides a better balance.

Another balance issue is the hero mentality. Often on Waterfall projects schedules slip and toward the end of a milestone there is a need for pent up work to be finished quickly. This develops the need for a coding hero that can get all these things done quickly in order to salvage the schedule. As one participant put it a project “relies on the individual … on heroes that you know [are] very smart people. There are a few smart people that a lot of people
depend on and then there’s a lot of like helper people.” This leads to pressure and stress on those “heroes” (and to some degree a sense of uselessness on the “helper people”).

**Affective Response Summary**

The way developers talk about software development sheds light on the fact that it is an emotional endeavor not just a cognitive effort. Agile methods bring many emotional or affective benefits to software developers. They enjoy confidence, balance, accomplishment and engagement. This is a refreshing change from the “hell” of a death march and the “fear” of change or the hero mentality that participants associated with Waterfall based methods.

**Inter-Intra Personal**

The final category that emerged from the data relates to inter and intra personal factors. Communication, feedback and social influences arise in relation to interpersonal factors. Intrapersonal dimensions relate to professional development and individual characteristics.

**Communication**

Improved communication was a widely mentioned factor of preference for agile methods. Participants felt that collaboration between team members, management and customers increased due to the use of agile methods. For example, in a Waterfall method there may be long periods where there is little communication and accountability. As one participant says, “my observation is that in a lot of cases a developer won’t think twice about wasting two, three days struggling through an issue that could have been resolved in ten minutes of conversation.” But with a daily stand-up meeting developers are accountable for daily progress and are more likely to resolve issues rather than waste time. The same participant goes on to say:

> Whether it’s pride or laziness or just stubbornness, they [developers] don’t get out there and ask the people the questions. And having that standup and being called out every day forces that person to communicate when they otherwise wouldn’t volunteer that information. I think that’s very helpful in overcoming that dependency.
This is a very similar line of thinking that another developer mentioned, “when we do our daily standup we tell – we discuss our issues. It gets you going rather than you get distracted on some other path you know?”

One participant mentioned the increased collaboration that their team is experiencing because of co-location. They have a team-room where the project team sits together and this provides an incubator effect for innovative solutions. In his own words:

We actually now have physical surroundings of support and collaboration in the form of [a] team room, where everybody is involved both engineering and PMs and management are all in a combined team room. We get a lot of cross-pollination there, which is good.

Feedback

Another dimension of the interpersonal category is feedback. Participants mentioned that they really liked the short cycles and quick feedback that agile affords. We heard from one participant earlier that the shorter delivery cycles made the project more tangible for customers and this allows them to see the software so not only is the feedback more frequent, more timely it is also more concrete and relevant. The feedback may not always be positive but it is helpful. Here’s how one participant put it, “Even if the customer says that they don’t like what we’re doing, I’d much rather hear that early rather than later.” The short cycles providing quick feedback contributes to the instant gratification phenomena that were mentioned previously. “I think you get that more of an immediate gratification and, I’ve coded something, tested it, I’ve delivered it and I’ve gotten some feedback on it.”

As another developer says:

I think the big downside people too with Waterfall nowadays is having these huge long release cycles where you’re doing this huge feature and you don’t really review it or have a customer review until it’s already had a ton of hours put into it. Where with Agile people hope to get it reviewed faster even if it’s just a prototype that’s being reviewed and then they can adapt and change direction a lot more easily early on than if they wait until later.

One personal benefit was mentioned related to shorter feedback cycles. A participant pointed out that as a developer experiencing a shorter time period between creating a defect, identifying it and fixing accelerates the learning process. As he says:
I see that the teams are happier and I think the teams have an opportunity to get better faster. I’ll explain it like this by contrasting. In the past you know you would a bunch of development and then you would have this stabilization phase. And during the stabilization phase we’re just improving the quality of the product by fixing bugs, right? … The real key thing is that when you’re looking at the bug and you’re fixing it, the point of the discovery and the fixing point is so far away from the point when it got injected into the product in the first place, it could be months of difference. So the opportunity for you to look at this and say, “How can I learn so that I do not enter these kinds of bugs again,” goes way down. So then it comes around to the next development cycle, it’s long forgotten all the bugs that they injected, but in an iterative cycle where within two weeks you’re introducing product and you’re fixing all the bugs in that little bit of product. Like you have a very short cycle where you are tied together the injection point and the discovery point of the bug are very close together and you can actually learn and then you do it all over again the next two weeks.

**Positive Social Influence**

Another intra-inter personal factor that positively influenced people’s preference for agile was positive social impacts. These came in the form of colleagues, managers, blogs, books, and training. When these forms of social influence favored agile it had a positive impact on the preference of the participants. As one developer said, “the social pressure of well this is what everyone is moving too and it’s kind of becoming a new standard, so we’re going to feel like we’re behind if we’re doing an old method, everyone else is in the name.” Another developer, referring to meetings on agile says “I’d bet some of my opinions have been picked up from my peers in those meetings.” And yet another, speaking of a respected colleague’s positive influence on his attitude toward agile said, “I have a world of respect for Shaun on a lot of levels, and the fact that he was that strong a proponent of it, yeah, that does go into my good formula, absolutely.”

**Negative Social Influences and Personal Influences**

Two negative factors emerged in the inter-intra personal area. We mentioned social influence as a positive influence, but it can also be a negative factor. This came in two forms, customers and managers. If customers express a preference or requirement that a project use Waterfall methods then some developers felt compelled to drop agile and use Waterfall methods. The same developer would also override his personal preference for agile if management expressed a preference for Waterfall. This was an isolated case but interesting none the less.
The other negative factor that emerged had to do with a perceived personal characteristic. Only a few participants explicitly expressed a form of this characteristic, although others mentioned they observed or suspected it in others. One explicit expression concerning an aversion toward agile was mentioned like this, “Obviously I was more skeptical of it because it was unknown and new.” This exemplified the thoughts of other participants who witnessed some of their colleagues preferring the traditional Waterfall over agile because it was new, different, required them to learn something.

Another developer said “I’m probably a creature of habit so I’d probably start with Waterfall ‘cause it’s what I know.” Other participants said they felt that some developers just preferred the status quo because they had a fear of learning or perhaps appearing inadequate or just wanted to finish off their career doing what they had always done. One participant put it this way:

My experience was that, with a few exceptions, a lot of the people who had been in that job or in that career field for a while just didn’t share that same desire to keep growing their skills; they had kind of adopted a certain level of comfort with, okay, this is how we do it.

**Inter-intra Personal Summary**

So although the inter-intra personal theme primarily led to increased preference for agile methods there were a few exceptions. On the positive influence side we saw that developers believed that communication, short feedback cycles and social influence were enhanced by agile methods increasing developer preference for agile. On the negative side social influences as well as personal resistance to change and learning can negatively affect one’s preference for agile methods.

**EMERGENT THEORETICAL MODEL**

As mentioned previously the goal of the research is to generate theory grounded in the data. The analysis of the data resulted in a number of concepts which were grouped into themes or categories. The three main positive preference categories are self-efficacy, affective response and inter-intra personal factors. In addition two negative influence
categories are represented in the model as external and internal preference inhibitors. Each of these categories is made up of propositions. Each proposition and will be discussed and then the emergent theoretical diagram with associated constructs and relationships is shown in Figure 30 below.

Overview

The model represents many features. The primary concepts that lead to an increased preference in agile methods are listed as Self-efficacy (P1), Affective Response (P2), Inter-Intra Personal Factors (P3) on the left side of the diagram. Within each primary concept are the supporting elements labeled alphabetically within each conceptual category. Consequently the proposition “Agile is effective” P1(a) can be understood to state that the belief that agile is effective increases a developer’s preference for agile methods. And by extension we have propositions P1(a) through P3(c) that positively influence a software developer’s preference for agile methods.

Note that there are two categories of preference inhibitors, External (P4) and Personal (P5) on the right side of the diagram. These are concepts that decrease a software developer’s preference for agile methods. “Management resistance,” P4(a), states that if a software developer believes that management is resistant to agile methods then this will inhibit the developer’s preference for agile methods. Similarly “customer resistance” P4(b), “change adversity” P5(a), and “work style” P5(b) are theorized to inhibit a developer’s preference for agile methods.

Self-Efficacy Positively Influences Preference for Agile: P1

We will now discuss each proposition in the model and relate it to the data demonstrating their emergent nature. Figure 5 shows how the underlying concepts developed in the Findings section above contribute to the theoretical category emphasizing the emergence of the self-efficacy category from the underlying concepts.
Figure 5 - Emergence of self-efficacy category

Figure 6 follows the modified six Cs approach to provide the context and further illustrate the emergence of the self-efficacy category and its relationship to the supporting constructs. In the following subsections each “cause” in the diagram will take the form of a construct in the emerging theoretical model. Each construct will be further illuminated.

Figure 6 - Six Cs self-efficacy context diagram
**Agile is effective: P1(a)**

First is the self-efficacy group of propositions. Agile is effective P1(a), was mentioned by multiple participants and was often reiterated throughout the conversations as a key reason why the developer preferred agile over other methodologies. Developers expressed that agile helped them get the project done and provided a focus on steps that demonstrated value to the customer throughout the project lifecycle starting very early compared to other methods. Numerous participants mentioned that “waterfall never worked.”

The interview context related to questions from the interview protocol about what and why the developer liked agile methods compared to traditional methods. This typically led to a discussion of factors of preference for agile methods and often the participant contrasted the preference factors for agile with the contrasting parts of the plan-driven/waterfall approach. In the case of the “agile is effective” construct participants pointed to a number of causes for agile’s effectiveness. Comments indicated that agile makes the developer more productive, it focuses on the right activities, and task sizes were small and manageable. The result of agile being more effective is that developers preferred agile over traditional waterfall methods.

Figure 7 shows an example of how the theoretical construct emerged from an example participant quote through the open, axial, and selective coding phases resulting in the theoretical construct P1(a).
Agile is efficient: P1(b)

Participants also pointed out that other methods have steps that did not bring value to either the customer or the technical team which made the other methods inefficient compared to agile. These steps often included detailed documentation that became out of synch with the actual code as the project proceeded and thus became misleading. The lightweight nature of the agile method focused on “just enough” to satisfy the customer and this efficient approach, Agile is efficient P1(b), was a clear attraction to the participating developers. In addition the
daily stand-up meetings and pair-programming made learning efficient. By listening and watching new members of the team could get up to speed quickly and without slowing down other team members.

The interview context is the same as for the previous construct, it is related to questions from the interview protocol about what and why the developer liked agile methods. This typically led to a discussion of factors of preference for agile methods and often the participant contrasted the preference factors for agile with the contrasting parts of the plan-driven/waterfall approach. In the case of the “agile is efficient” construct participants’ identified significant causes for agile’s efficiency. Participants noted that agile’s streamlined approach, eliminating unnecessary documentation, focusing on high priority features first and frequent delivery allowed them to work much more efficiently. The consequence of agile being more efficient is that developers preferred agile over traditional waterfall methods.

Figure 8 shows an illustration of how the theoretical construct emerged from an example participant quote through the open, axial, and selective coding phases resulting in the theoretical construct P1(b).
Agile increases quality: P1(c)

Another aspect in the self-efficacy group was the sense that quality was improved by using the agile software development method, Agile increases quality P1(c). The most common comment was that unit tests, which most participants felt was an integral part of agile, provided high quality and increased confidence in the software. Unit tests reduced risk that changes to the software might have negative side effects. Participants also mentioned that their estimates were more accurate because of the short cycle time of each delivery.
Participants also mentioned that with agile they designed for change which resulted in higher quality even as requirements shifted because the code was designed to be modified. One other contribution to quality was the innovation that came from working in a “team room” where all members of the team, both technical and non-technical, were co-located in an open area resulting in increased communication and many innovations.

The interview context is a discussion about why developers like agile. This typically led to a discussion of factors of preference for agile methods and often the participant contrasted the preference factors for agile with the contrasting parts of the plan-driven/waterfall approach. In the case of the “agile increases quality” construct participants’ highlighted numerous reasons for agile’s increased quality. Some key factors were an emphasis on unit testing, team room dynamics, daily stand-up meetings and designing for change. The significance of agile increasing quality is that developers preferred agile over traditional waterfall methods.

Figure 9 shows an illustration of how the theoretical construct emerged from an example participant quote through the open, axial, and selective coding phases resulting in the theoretical construct P1(c).
Agile fits with reality: P1(d)

One last item in the self-efficacy group is agile’s fit with reality. Developers noted that the Waterfall method had many steps where developers just went through the motions and the steps did not produce any lasting value. They recognized that agile preserves the core
steps that do bring value and eliminates those that don’t. Agile fit better with the way
developers actually to develop software, Agile fits with reality P1(d). A few participants said
that they had personally started to develop software in an agile way even before they had
heard of what agile was. They noted it just fit with the way they liked to work and were
surprised to find out that there was a name for their approach. They said agile just fit with the
way they did their work.

The interview context is a discussion about why developers like agile. This typically
led to a discussion of factors of preference for agile methods and often the participant
contrasted the preference factors for agile with the contrasting parts of the plan-
driven/waterfall approach. In the case of the “agile fits with reality” construct participants’
the causes for their belief were that many of them were practicing agile without even knowing
there was such a thing. Others mentioned that agile seemed more natural and practical where
waterfall seemed more academic and bureaucratic. The outcome of agile having a better fit
with reality is that developers preferred agile over traditional waterfall methods.

Figure 10 shows an illustration of how the theoretical construct emerged from an
example participant quote through the open, axial, and selective coding phases resulting in the
theoretical construct P1(d).
The second grouping of positive preference factors is labeled “Affective Response” because these propositions have an emotional flavor. Figure 11 shows how the underlying concepts developed in the Findings section above contribute to the theoretical category emphasizing the emergence of the affective response category from the underlying concepts.
Figure 11 - Emergence of affective response category

Figure 12 follows the modified six Cs approach to provide the context and further illustrate the emergence of the affective response category and its relationship to the supporting constructs. In the following subsections each “cause” in the diagram will take the form of a construct in the emerging theoretical model. Each construct will be further illuminated.
Agile increases confidence: P2(a)

The first proposition in this group, Agile increases confidence P2(a), summarizes the comments made by participants about increased confidence. This increase arises primarily from the ubiquitous implementation of automated unit tests. The ability to make changes and then validate the changes through the set of unit tests gives the developer the confidence to make requested changes to the code. These changes may come from the customer or they may come from refactoring to make the code easier to understand or more efficient.

The interview context is a discussion about why developers like agile. The discussion turned to how participants feel about using agile methods. In the case of the “agile increases confidence” construct participants’ listed numerous causes for increased confidence. The primary reason for confidence was that in their experience of agile there was an emphasis on automated unit tests. These automated unit tests emboldened developers to make the necessary changes to the code without fear of breaking existing working code. The outcome of agile increasing developer confidence is that developers preferred agile over traditional waterfall methods.

Figure 13 shows an illustration of how the theoretical construct emerged from an example participant quote through the open, axial, and selective coding phases resulting in the theoretical construct P2(a).
**Agile is engaging: P2(b)**

Agile is engaging P2(b) captures the comments from participants about the appealing nature of agile methods. Developers are involved and enjoy their work from the start of the project because they are developing code right away rather than going through long stages of documenting requirements and producing detailed designs for the entire project before
actually writing code like they do in other methodologies. The participants stated that they found delivering code in short cycles much more engaging letting them do what they like to do as professionals. As one participant said, “my eyes glazed over” when doing the requirements and design documentation knowing that coding was months away. And as another participant spoke about developers using agile methods, “they are happy because they are coding.” The engagement doesn’t stop with the development team. Participants noted that all stakeholders are more engaged when they see working software and can visualize the feature because they see it on their screens.

The interview context is a discussion about why developers like agile. The discussion turned to how participants feel about using agile methods. In the case of the “agile is engaging” construct participants’ noted some reasons why agile is engaging. Overwhelmingly the reason agile is engaging is because of its emphasis on coding over documentation. Another example of why agile is engaging is that the project manager role is diminished and the manager becomes more involved with solutions rather than just assigning and monitoring tasks. The outcome of agile’s engaging nature is that developers preferred agile over traditional waterfall methods.

Figure 14 shows an illustration of how the theoretical construct emerged from an example participant quote through the open, axial, and selective coding phases resulting in the theoretical construct P2(b).
Another affective outcome of agile is that it produces a sense of accomplishment which is stated in the model as Agile produces a sense of accomplishment P2(c). The quick delivery cycle and more immediate feedback are satisfying to developers and give them the
feeling that they are quickly and regularly meeting the needs of the customer. One developer mentioned that he liked the short cycle and the resulting well defined units of work. In previous methods that were used the requirements were less focused and he became bogged down trying to understand what was desired. Agile’s short delivery time frame and fewer focused features alleviated this problem.

The interview context is a discussion about why developers like agile. The discussion turned to how participants feel about using agile methods. In the case of the “agile produces a sense of accomplishment” construct participants’ listed a variety of causes. The reasons for a feeling of accomplishment ranged from increased accountability at stand-up meetings, to agile’s emphasis on well-defined units of work making progress more tangible. The outcome the sense of accomplishment is that developers preferred agile over traditional waterfall methods.

Figure 15 shows an illustration of how the theoretical construct emerged from an example participant quote through the open, axial, and selective coding phases resulting in the theoretical construct P2(c).
Figure 15 - Emergence of "agile produces a sense of accomplishment" construct

**Agile increases life balance: P2(d)**

The last proposition in the Affective Response group is related to balance, Agile increases life balance P2(d). As one participant said, “Waterfall, in my experience, towards the end of the release, it tends to be hell.” Contrast this with the comments from participants who said that agile is more accommodating to their non-work lives. The short cycles allow
for planning around life events and according to the Agile Manifesto the desire is to maintain an indefinitely sustainable pace. One developer said that with agile “you sort of get into being able to think about work life balance. I mean that’s important too because you wanna keep employees happy so they stay in your group.” So the avoidance of the end of project “death march” and increased flexibility around non-work activities is a positive factor for developer preference of agile methods.

The interview context is a discussion about why developers like agile. The discussion turned to how participants feel about using agile methods. In the case of the “agile increases life balance” construct participants’ listed some explanations for this feeling. The primary reason was the contrast to the typical “death march” experience in waterfall compared to the consistent rhythm of agile. One participant also noted the dependence on “heroes” in waterfall and contrasted that with the whole team approach of agile. The outcome of the feeling of better life balance is that developers preferred agile over traditional waterfall methods.

Figure 16 shows an illustration of how the theoretical construct emerged from an example participant quote through the open, axial, and selective coding phases resulting in the theoretical construct P2(d).
Intra-Inter Personal Effects Positively Influences Preference for Agile: P3

The last set of positive preference factors are the intra-inter personal propositions. Figure 17 shows how the underlying concepts developed in the Findings section above
contribute to the theoretical category emphasizing the emergence of the affective response category from the underlying concepts.

<table>
<thead>
<tr>
<th>Underlying Concepts</th>
<th>Emergent Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication</td>
<td>Intra-inter personal factors drive preference for agile methods</td>
</tr>
<tr>
<td>Feedback</td>
<td></td>
</tr>
<tr>
<td>Positive social feedback</td>
<td></td>
</tr>
</tbody>
</table>

Figure 17 - Emergence of intra-inter personal category

Figure 18 follows the modified six Cs approach to provide the context and further illustrate the emergence of the intra-inter personal category and its relationship to the supporting constructs. In the following subsections each “cause” in the diagram will take the form of a construct in the emerging theoretical model. Each construct will be further illuminated.
Agile improves communication: P3(a)

Agile improves communication P3(a), is the first proposition in this group. Many participants mentioned the improved communication that comes with using agile methods. Comments ranged from the previously noted “team room” concept where all team members are in an open room working together and freely talking. Regular team “stand-up” meetings also provided great communication opportunities while limiting the amount of time spent in the meeting itself. Another developer mentioned how agile engaged the program manager as part of the problem solving team and required him to discuss options and solutions rather than delegate it to the developer to figure out the problem on their own. Another valuable outcome is the increased communication with the customer. Many teams had customers on the team.
that were involved regularly if not daily with defining features and behavior, solving problems and testing software.

The interview context is a discussion about why developers like agile. This part of the conversation focused on personal and inter-personal factors. In the case of the “agile improves communication” construct participants’ consistently mentioned the stand-up meeting as improving communication both between team members and with management. Other enhancements to communication came from co-location in a team room and regular customer interaction throughout the development process. The outcome the sense of improved communication was an increased preference for agile over traditional waterfall methods.

Figure 19 shows an illustration of how the theoretical construct emerged from an example participant quote through the open, axial, and selective coding phases resulting in the theoretical construct P3(a).
Figure 19 - Emergence of "agile improves communication" construct

**Agile increases timely feedback: P3(b)**

Agile increases timely feedback P3(b) captures the idea that agile makes timely feedback from customers and stakeholders available much earlier than other methods. The short delivery cycle of working features allowed the customers and others to provide quick
comments and adjustments could be made in the next iteration. Many participants appreciated this outcome of using agile methods.

The interview context is a discussion about why developers like agile. This part of the conversation focused on personal and inter-personal factors. In the case of the “agile increases timely feedback” construct participants’ recognized that agile produces more frequent and earlier feedback than other methods. As the illustration quote in Figure 20 shows, even if the feedback is negative the fact that agile teams get this feedback early is beneficial to the project. The outcome of experiencing increased and timely feedback using agile methods is an increased preference for agile over traditional waterfall methods.

Figure 20 shows an illustration of how the theoretical construct emerged from an example participant quote through the open, axial, and selective coding phases resulting in the theoretical construct P3(b).
Positive social influence increases agile preference: P3(c)

In an inter-personal sense the comments made by participants indicated that peers, mentors, media and other respected people had a positive influence on their preference for agile. Through conversations, training, blogs, books and articles these social influences...
shaped the participants perspective on agile and generally increased their preference because these influences presented agile in a positive light.

The interview context is a discussion about why developers like agile. This part of the conversation focused on personal and inter-personal factors. In the case of the “positive social influence increases agile preference” construct participants’ identified things like blogs, books, and respected colleagues influenced their appreciation of agile. The result of these positive social influences was an increase in their preference for agile.

Figure 21 shows an illustration of how the theoretical construct emerged from an example participant quote through the open, axial, and selective coding phases resulting in the theoretical construct P3(c).
Figure 21 - Emergence of "agile is a product of professional growth" construct

By abstracting the comments from participants we find three groups of positive antecedents for preference of agile methods, self-efficacy, affective response and intra-inter personal factors. Now we will discuss the two groups of negative influences on preference for agile.
External Factors Inhibits Preference for Agile: P4

The first group of negative influences that inhibit preference for agile is summarized as external factors, P4, in the theoretical model. Figure 22 shows how the underlying concepts developed in the Findings section above contribute to the theoretical category emphasizing the emergence of the external preference inhibitors category from the underlying concepts.

<table>
<thead>
<tr>
<th>Underlying Concepts</th>
<th>Emergent Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management resistance</td>
<td>External factors inhibit preference for agile</td>
</tr>
<tr>
<td>Customer resistance</td>
<td></td>
</tr>
</tbody>
</table>

Figure 22 - Emergence of the external preference inhibitors category

Figure 23 follows the modified six Cs approach to provide the context and further illustrate the emergence of the external preference inhibitors category and its relationship to the supporting constructs. In the following subsections each “cause” in the diagram will take the form of a construct in the emerging theoretical model. Each construct will be further illuminated.
Management resistance inhibits agile preference: P4(a)

Management resistance P4(a) is the first external factor that exhibits a negative influence on developer preference for agile. One participant initially indicated that he preferred agile. But when asked what he would recommend for a project he said it depended on what management preferred. If managed preferred Waterfall then the participant would recommend Waterfall over agile. He also noted that on many projects his management required agile. From this we arrive at the principal of management resistance inhibits developer preference for agile methods. See Figure 24 below for an illustration of how this proposition emerges from a sample participant quotation.
Figure 24 - Emergence of "management resistance inhibits agile preference" construct

**Customer resistance inhibits agile preference: P4(b)**

P4(b) is labeled customer resistance in the diagram. Similar to management resistance participants noted that if a customer demonstrates a desire for the Waterfall method this would have a significant impact on the preference of the developer. One developer mentioned
that he would use the method the customer wanted even if it was Waterfall. The emergence of this construct is illustrated below in Figure 25.

**Figure 25 - Emergence of "customer resistance inhibits agile preference" construct**

**Personal Factors Inhibit Preference for Agile: P5**

The second group of preference inhibiting factors is the personal group. Figure 26 shows how the underlying concepts developed in the Findings section above contribute to the
theoretical category emphasizing the emergence of the personal factors inhibitors category from the underlying concepts.

**Underlying Concepts**

- Change adversity
- Work style

**Emergent Category**

- Personal factors inhibit preference for agile

Figure 26 - Emergence of personal factors inhibitors category

Figure 23 follows the modified six Cs approach to provide the context and further illustrate the emergence of the personal preference inhibitors category and its relationship to the supporting constructs. In the following subsections each “cause” in the diagram will take the form of a construct in the emerging theoretical model. Each construct will be further illuminated.
Change adversity inhibits agile preference: P5(a)

Change adversity, P5(a), is an inhibitor that emerged from the data. Here is a quote from one participant, “of course I was skeptical because it (agile) is new and different.” This particular participant did not prefer agile even though he had worked on a project using agile methods. His comment assumes that a person would be skeptical of anything new. His tone indicated that he was change adverse and that he preferred to stick with what he had experience with which was Waterfall. Many participants discussed that they observed co-workers who were change adverse. They did not want to put forth the effort to learn a new methodology because they were more comfortable with their existing methods. The emergence of this construct is illustrated below in Figure 28.
Work style inhibits agile preference: P5(b)

Work style, P5(b), also surfaced as a personal inhibitor to agile preference. One participant, who does not prefer agile, mentioned that he does not like the increased communication, co-location and daily meetings associated with agile. He preferred to work alone with written requirements and he would go from there. His preferred work style just didn’t fit with agile and thus he did not like to use agile methods. An example of how this construct emerged from the data is shown below in Figure 29.
The model contains agile preference inhibitors and communicates these constructs as external and personal. Management and customer resistance can negatively influence a developer’s preference for agile. In addition adversity to change and personal working style can also negatively influence a developer’s preference for agile.
The Theoretical Model

The model communicates that those software developers that prefer agile prefer it because they believe it makes them more effective, using agile methods has positive emotional benefits and they believe agile methods improve both their interpersonal relationships and their own self.

We have organized and represented these concepts and themes in a diagram representing the theoretical model as shown in Figure 30.

Figure 30 - Theoretical Model

In this chapter we have explained information about the study participants, followed by a discussion of the detailed codes and their relationships that emerged from the interview data. The summarized data was shown in an ATLAS.ti network diagram. An overview of the findings was provided in table format and is followed by a detailed discussion of each construct along with demonstration of how the constructs emerged from the interview data. The chapter concluded with a discussion of the emergent theoretical model and the relationships between the constructs culminating in the theoretical model diagram.
CHAPTER 6

CONCLUSIONS

The findings from the previous chapter are discussed in relation to their application to future research and implications for industry. Of particular interest are the implications of the findings for management and how the theoretical model may aid in developing training programs to increase developer preference for agile methods.

Malleable Factors

One of the exciting findings is that the discovered constructs of preference are based on factors that are malleable, that is, they can be modified. This is encouraging since if a particular person currently has a low preference for agile the constructs for preference are such that through education and experience their preference can be increased. Prior studies investigated immutable characteristics like personality traits (Bishop & Deokar, 2014). Fortunately those did not turn out to have significant influence on preference for agile. If they did then an individual’s preference could only be changed if their personality could be modified. Since the positive constructs that emerged from the data are all modifiable there will be opportunities to develop programs to increase preference for agile. Following are some suggestions on how this research can be used in practice.

Implications for Management

Although the unit of analysis is at the individual software developer level there are a number of implications that apply to the management of software developers and software projects. We will look at three areas: team formation, team optimization and team retention.

Agile team formation can be positively influenced by careful selection of those individuals who prefer to work in an agile fashion. Future research will develop empirical testing measures for the dimensions of preference, but in the meantime managers can query
potential team members along the lines of the model’s constructs to determine if they are a
good fit for working on an agile team. If the potential team member buys into the three main
concepts and the supporting constructs, P1(a) – P3(d), then it is likely that the candidate will
engage in agile software development. If the candidate does not adhere to these concepts and
constructs then they may resist using agile methods.

Perhaps a team is being transitioned to agile methods. Understanding what drives
preference can be used to develop training tools to educate team members about the benefits
that other software developers derive from agile. By raising team member awareness of the
key factors that other developers appreciate about agile there is a better chance that they will
develop a preference for agile methods.

Another application of the theory to practice is ensuring that an active agile team is
emphasizing the elements that have been identified as generating the most preference among
developers. Ensuring that the positive constructs of the model are practiced by a team will
likely produce a higher satisfaction among the team members. In addition since one of the
main conceptual categories is self-efficacy those constructs should make the team more
productive in addition to increased satisfaction and retention.

Future Research

One important next step in the research will be a quantitative study to test the
hypothesis outlined in the theoretical model. The research is envisioned to be survey based to
determine if there is a correlation between the suggested constructs and the preference
outcome. This will require two key features; a preference instrument (see below) and an
instrument aligned with the theoretical factors of the emergent model described in this
research. Once the appropriate sample data has been acquired then statistical analysis can be
performed to affirm the accuracy and validity of the modeled preference drivers and then
determine their relationship to agile preference to verify the theoretical model.

As mentioned above, in order to carry out the proposed validation it will be necessary
to have a validated instrument to ascertain an individual software developer’s level of agile
preference in order to then assess the constructs of the theoretical model. Prior work has been
done (Bishop, 2013; Bishop & Deokar, 2014) in regard to developing an instrument, but there
is room for improvement to the structure of the instrument and the resulting validity. Improving those instruments will be a useful and productive research path to follow.

Another direction for future research is to examine preference from a customer perspective. How do customers perceive agile, is it better and preferred over other approaches? A similar grounded theory approach could be used with customers as participants to understand what aspects they like or dislike about agile compared to alternative software development methodologies.

A final suggested research direction that would be an excellent complementary study to this research is to study the factors that drive management preference for agile. A similar Grounded Theory approach should be utilized but with a different audience of participants. This proposed study would examine the drivers for management preference of agile software development methods. This study may include an adoption as well as preference flavor. Because unlike individual developers who may not be able to influence the selection of a project’s development methodology, managers are in a position to make choices that are implemented by the team. Consequently the study could encompass preference and extend to adoption and usage.

Summary

In conclusion then this research has set out to answer questions regarding what influences individual software developer preference for agile methods. The Grounded Theory approach of research was employed and 16 participants shared over 390 minutes of dialog leading to a robust theoretical model that conceptualizes three main factor categories that drive developer preference: self-efficacy, affective response and intra-inter personal factors. These categories are made up of fifteen different factors which form the basic hypothesis suggested by the model. In addition the research identified two categories, external and personal, that may decrease developer preference for agile methods.


APPENDICES

APPENDIX A: INTERVIEW PROTOCOL QUESTIONS

- Get demographic info: age, gender, location, industry, programming language, years of experience, company size, team size
- Can you tell me about your software development experiences to date?
- How do you build software now? What processes do you follow?
- How long have you done it this way? Has that changed over time?
- Do you associate the way you build software with any particular method? Why or why not?
- Have you heard of agile software methods? How would you describe them?
- How do agile methods compare with your preferred method of developing software?
- Do you like agile methods? How do you know you like or dislike them? What indicators of your level of preference are there?
- What drives your level of preference for agile software methods?
- When did you develop your perspective on agile software methods?
- Are you an active advocate/antagonist with regard to agile methods? How have you expressed this? Have you ever defended agile methods or made arguments against them? What did you do?
- Did anything or anyone have a particular influence on your level of preference?
- Were there any specific experiences that made you lean one way or the other in regard to agile methods?
- Are there any external influences on your perspective on agile methods? Magazines, books, internet discussions or blogs, conferences, colleagues, friends, management, training?
- Have you ever had training on one or more software development methodologies? Which ones? Has training or lack of training influenced your level of preference for agile?
• Have you ever been required to use agile methods? Did that have any influence on your level of preference?
• Have you ever been prohibited from using agile methods? Did that have any influence on your level of preference?
• Given your choice what SDM would you choose?
• Given a list of methods like waterfall, spiral, RAD, or agile how would you order them according to your preference? How did you decide on that order?
• What factors have contributed to your level of preference for agile? If you have a low/high preference for agile, what are the reasons you don’t/do like it?
• What do you find attractive about your preferred SDM? If not agile, what do you find unattractive about it?
• How do you see agile being received in the industry? Is that a good thing or not? Why?
• Are you interested in management’s perspective on agile SDM?
• Now that we have covered many different aspects of your agile SDM preference, is there anything that strikes you as having influence on your level of preference for agile SDM?
• Do you know of any other developers that might be willing to participate in an interview like this? They just need to be software developers, preferably with some exposure to agile methods.
APPENDIX B: INITIAL CODES

- advocate of agile to peers
- age: middle aged
- age: young
- agile experience reason: someone else (project leader) wanted to try it
- agile experience: high
- agile experience: intermediate
- agile experience: minimal
- agile experience: none
- agile preference factor: better communication from standup meetings
- agile preference factor: better fit with reality
- agile preference factor: confidence to change because of unit testing
- agile preference factor: engagement
- agile preference factor: improved quality
- agile preference factor: more realistic/less academic
- agile preference factor: programmer is happier than with waterfall
- agile preference factor: satisfying due to clearly defined product and tasks
- agile preference factor: satisfying due to sense of accomplishment
- agile preference factor: standup meeting forces communication which avoids prolonged problem solving
- agile preference factor: well defined goal
- agile preference factor: well defined unit of work
- agile preference factor: adaptability to changing requirements
- agile preference factor: agile advocates good programming techniques
- agile preference factor: agile has good development practices
- agile preference factor: avoids writing documents that become out of sync anyway
- agile preference factor: better fit with best coding practices
- agile preference factor: better fit with industry demands of frequent releases
- agile preference factor: better involvement of program manager/product owner
• agile preference factor: better results
• agile preference factor: better than the alternatives
• agile preference factor: customer involvement
• agile preference factor: early delivery of product
• agile preference factor: early identification of blocking issues
• agile preference factor: efficiency
• agile preference factor: efficient learning of product and project
• agile preference factor: engagement of team
• agile preference factor: engaging, creative, fun
• agile preference factor: flexible implementation of features
• agile preference factor: happier because they are coding
• agile preference factor: improved quality due to adaptability due to unit tests
• agile preference factor: improved quality due to design for change
• agile preference factor: keeps a person busy
• agile preference factor: less frustrating due to better fit w/reality
• agile preference factor: love to create software for customers
• agile preference factor: management support would increase preference
• agile preference factor: more effective approach to development
• agile preference factor: more reliable estimates to customers because of short time frame
• agile preference factor: pairing provided great learning experience
• agile preference factor: produce faster
• agile preference factor: productivity
• agile preference factor: quicker feedback on finished work
• agile preference factor: reading positively influenced preference for agile
• agile preference factor: reduced risk due to frequent evaluation
• agile preference factor: self-improvement by learning to code better
• agile preference factor: sense of accomplishment
• agile preference factor: sense of collaboration
• agile preference factor: short feedback cycle
- agile preference factor: shorter/faster code-test-fix cycles
- agile preference factor: small task size
- agile preference factor: social influence
- agile preference factor: standup meetings
- agile preference factor: team member retention due to work life balance
- agile preference factor: team room breeds innovation
- agile preference factor: team room facilitates communication
- agile preference factor: timely delivery
- agile preference factor: training is important
- agile preference factor: unit testing/test driven design
- agile preference factor: visualize requirements rather than imagine them
- agile preference factor: volatile requirements
- agile preference factor: work life balance
- agile preference negative factor: chaotic at enterprise level with many teams
- agile preference negative factor: disruption of stand-up meetings
- agile preference negative factor: distributed teams
- agile preference negative factor: emphasis on speed enables expanded code base and inconsistencies
- agile preference negative factor: fear of learning
- agile preference negative factor: fear of period of lower productivity due to learning
- agile preference negative factor: feature attention deficit disorder
- agile preference negative factor: framework or low-level code is harder to adapt to agile
- agile preference negative factor: Kanban can feel relentless
- agile preference negative factor: lack of information continuity
- agile preference negative factor: lack of reading
- agile preference negative factor: lack of training
- agile preference negative factor: no management support
- agile preference negative factor: no social influence
- agile preference negative factor: people don't like change
• agile preference negative factor: prefers to work in isolation
• agile preference negative factor: proj mgmt must change their involvement level
• agile preference negative factor: project management opposition
• agile preference negative factor: retroactively creating unit tests
• agile preference negative factor: scaling inter-team communications
• agile preference negative factor: scaling to large numbers of teams
• agile preference negative factor: some people oppose personal growth
• agile preference negative factor: stand-ups for distributed teams
• agile preference negative factor: stick with status quo
• attitude toward agile: skeptical because it was new
• avoiding documentation and jus..
• believes customers like defined milestones
• C Sharp, Java, Python, JavaScript, TypeScript, OCaml, and Haskell
• Certainly putting working soft..
• changes without worrying that ..
• collectively working through t..
• company size: medium
• company size: large
• company size: small
• constant positive feedback asp..
• customers prefer waterfall
• dev experience: intermediate
• dev experience: senior
• even with volatile requirements he chose spiral equal with agile
• everybody likes a sense of acc..
• experience driven by customer requirements
• experience driven by industry requirements
• experience driven by management
• experience with ad-hoc methods
• experienced in waterfall
• fear that you’re gonna break s..
• I feel right in my own self
• I just found that things like ..
• I only know that I would not e..
• I see value in the standup mee..
• I think agile is how software ..
• I think it’s just fun
• I’ve been using Agile without ..
• incomplete method definition: agile
• Increased accountability
• industry: defense - structured, government
• industry: manufacturing
• industry: online retail
• industry: software
• industry: software - image processing and embedded systems
• industry: telecommunications
• It’s a nice way of being able ..
• it’s coding it’s doing what so..
• Lack of available unit tests causes fear
• location: california
• location: kansas
• location: missouri
• location: north dakota
• location: redmond, washington
• location: south dakota
• location: utah
• management influence on preference
• management likes defined milestones
• must be open and interested in professional growth
• my eyes glazed over now becaus..
• no software developer really e..
• not willing to commit to agile as best solution
• partial use of agile - mostly agile
• partial use of agile - mostly waterfall
• programming languages: C and C++
• programming languages: C Sharp, SQL and JavaScript
• programming languages: C, C++, Java, Pearl, Python, .NET scripting
• programming languages: C++, Java, C Sharp
• programming languages: Java, Objective-C, C Sharp, C++
• programming languages: Oracle, SQL, PLSQL
• programming languages: vb.net and JQuery
• rely on heroes
• senior devs less open-minded
• seniority breeds self-conciousness
• stated preference: agile
• stated preference: waterfall
• team size: large
• team size: small
• the best way to develop softwa..
• the team started to kinda move..
• to me you know, on a personal ..
• traditional waterfall that nev..
• waterfall experience: minimal
• waterfall in agile clothing
• waterfall negative: allowed for too much work on unimportant features
• waterfall negative: too much risk to to late delivery
• waterfall never worked
• waterfall ruins work life balance at end of project
• Waterfall, in my experience, t..
• we are an immediate gratificat..
• when I learn about Agile, I kn..
• without even knowing what Agil..
• you have a safety net
• you have the confidence
APPENDIX C: FOCUSED CODES

- Accomplishment
- Adaptability
- Communication
- Confidence
- Effective
- Efficient
- Emotional
- Engagement
- Feedback
- Good Fit
- Incomplete implementation has a negative influence
- Knowledge
- Negative agile factor - Enterprise Scalability
- Negative customer influence
- Negative impact on concentration
- Negative Personal Characteristics
- Negative Perceived Impact on Code
- Negative Social Influences
- Positive Experience
- Preference driven by desire for short code-test-fix cycle
- Preference driven by management expectations for waterfall which negatively influence weak developer preference of agile
- Process
- Quality
- Satisfying
- Social
- Social Influence
- Work-Life Balance