7-3-2016

On Understanding Preference for Agile Methods Among Software Developers

David Brian Bishop
Dakota State University

Amit V. Deokar
University of Massachusetts at Lowell

Surendra Sarnikar
California State University, East Bay

Information Resources Management Journal
Volume 29 • Issue 3 • July-September 2016

On Understanding Preference for Agile Methods among Software Developers

David Bishop, Dakota State University, Madison, SD, USA
Amit V. Deokar, The Roberts J. Manning School of Business, University of Massachusetts Lowell, Lowell, MA, USA
Surendra Sarnikar, California State University East Bay, CA, USA

ABSTRACT

Agile methods are gaining widespread use in industry. Although management is keen on adopting agile, not all developers exhibit preference for agile methods. The literature is sparse in regard to why developers may show preference for agile. Understanding the factors informing the preference for agile can lead to more effective formation of teams, better training approaches, and optimizing software development efforts by focusing on key desirable components of agile. This study, using a grounded theory methodology, finds a variety of categories of factors that influence software developer preference for agile methods including self-efficacy, affective response, interpersonal response, external contingencies, and personality contingencies. Each of these categories contains multiple dimensions. Preference rationalization for agile methods is the core category that emerges from the data. It informs that while the very essence of agile methods overwhelmingly and positively resonates with software developers, the preference is contingent on external and personality factors as well.

KEYWORDS

Agile, Grounded Theory, Preference, Software Developer, Software Development Methodology

1. INTRODUCTION

Today there are two decidedly different approaches to Information Systems Development (ISD). 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, Mahapatra, & Nerur, 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 et al., 2001). Among the emphases in the twelve principles are that working software code, early and frequent delivery of working software code, daily collaboration between users and developers, trust in front line workers (business and technical), and face-to-face communication is better than written documentation. In addition, progress is measured by working software, consistent pacing rather than periodic heroic efforts, emergent rather than prescriptive design/architecture, and reflective team adjustments. 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. Table 1 illustrates these ideas.

DOI: 10.4018/IRMJ.2016070102

Table 1. Waterfall and agile methodology comparison

	Methodology	
	Traditional	**Agile**
Planning	Predictive	Adaptive
Progress Measure	Milestones	Delivered Software
Values	Process	People
Change	Minimized	Embraced
Communication	Documents	Conversations
Roles	Specialized	Generalist

Agile methods are a rapidly growing means of developing software. As of 2011, in the U.S. about 40% of companies use agile (Glaiel, Moulton, & Madnick, 2013). In a 2009 Forrester Research report cited by West, Grant, Gerush and D'Silva (2010) found that about 30% of software developers in a sample of over 1,000 are using some form of agile methods.

Since industry and management are interested in using agile approaches, it seems appropriate to identify factors that influence developer preference. Not every developer likes agile. Agile attitude measurement instruments would be valuable as management assesses individual developer preference for agile. To effectively assign developers to appropriate projects and integrate them into agile projects it is important to know if training will be effective and if so what factors increase agile preference. Since there are no definitive empirical research or theory that informs us on the factors that influence preference for agile methods among software developers, we embarked on our study to address this gap.

In this article, we present a qualitative study of factors influencing preference for agile software development methods from a developers' perspective. We begin with a review of recent research on the adoption and developer preferences for systems development methodologies and identify the need in-depth research on agile methods adoption from a developer perspective. We then present an overview of Grounded Theory research approach used to address the research goal. Next we present a detailed overview of our findings and a conceptual model that describes developer preferences for agile methods. Discussion of the findings with implications for research and practice are then discussed before concluding remarks.

2. LITERATURE REVIEW

There have been continuing theoretical development to extend agile development principles to a number of different contexts such as large and dynamic software development projects (Batra, VanderMeer, & Dutta, 2011) distributed software development projects (Bergadano, Bosio, & Spagnolo, 2014) data warehousing and business intelligence projects (Rahman, Rutz, & Akhter, 2011), and game design and development projects (Cano, González, Collazos, Muñoz-Arteaga, & Zapata, 2015). However, understanding developer attitudes towards development methodologies such as agile methods is an important research question to study as organizations explore approaches to drive improvements in software quality (Hendersen, Sheetz, & Bélanger, 2012).

Previous studies indicate that perceptions of methodology output, and perceptions about the obstacles encountered while applying a methodology, influence perceptions about usefulness and ease of use about the methodology (Kacmar, McManus, Duggan, Hale, & Hale, 2009). Changes in software

development methodologies may also require changes in developer mindset (Armstrong, Nelson, Nelson, & Narayanan, 2008). Systems development methodologies can also impact organizations from a human relations (HR) perspective such as in terms of staff turnover and developer satisfaction (Koch & Turk, 2011).

The above studies provide a good overview of the impact of system development methodologies in general or in comparison between agile and waterfall methodologies, however there is a need for in-depth research on factors that influence agile method adoption from a developer perspective. While there are some studies and theoretical models for agile adoption such as the technological frames model for agile adoption (Abdelnour-Nocera & Sharp, 2012), there is limited empirically grounded literature with focus on developer preferences and attitudes towards adoption of agile methodologies.

Studying agile software development project methodologies can also yield significant benefits from an organizational productivity and agility perspective. Given the significant investments and resources that organizations allocate for IT project management, several recent studies have explored issues methodology and IT project related issues such as continuation intention for IT project management (Korzaan & Brooks, 2015), and the impact of methodology fit on project performance (Xu & Yao, 2014). An emerging concept in this area is that of information systems agility. In order for organizations to adapt to ever changing business environment, the underlying information systems that form the core of an organization information resource also need to be agile (Chaudhary, Hyde, & Rodger, 2015). Recent research in this area explores mechanism for improving IS and organizational agility through the development of IT governance (Teoh & Cai, 2015; Teoh & Chen, 2013) mechanisms and new mechanism for executing change (Chaudhary et al., 2015). Understanding the factors that influence the adoption of agile methods, may also provide some insight into the factors that influence IS agility specifically and overall organizational agility in general.

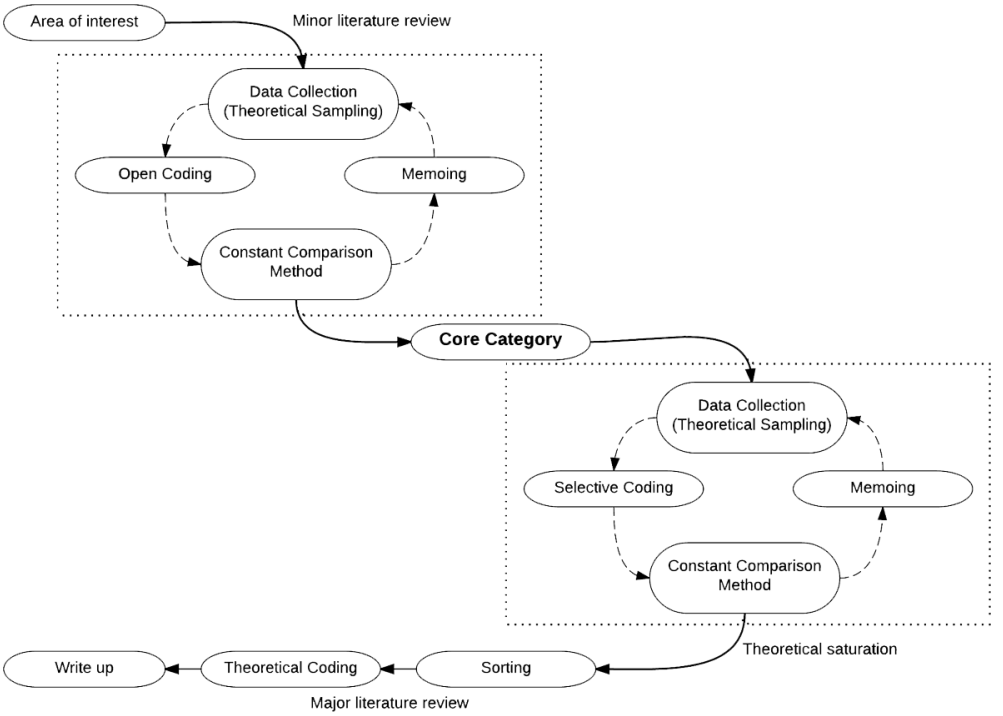
From an industry and management perspective, agile development methods are attractive because of the belief that they will deliver software faster. Agile methods are aimed at reducing bureaucracy and focusing on the essentials of delivering value to stakeholders (Boehm & Turner, 2004). Given theoretical and industry interest in agile methods, our study explores factors that influence software developers' preference for agile methods. It is our belief that, as more organizations adopt agile methods on increasing numbers of projects, it is important for management to understand practitioner preferences and take these into consideration to ensure smooth and effective adoption and diffusion strategies.

3. RESEARCH METHODOLOGY

The Grounded Theory (GT) method aims to generate theory from data about a substantive area through a rigorous and consistent process (Glaser, 1992). Grounded Theory was chosen for several reasons. First, software development in organizations is inherently a team endeavor. GT, as a qualitative research method, allows study of social interactions and behavior of software developers engaged in practicing various software development strategies (Parry, 1998). Second, GT supports theory generation instead of testing or extending theories (Glaser & Strauss, 1967). Third, GT is particularly suited for researching subject areas, which are relatively new in terms of research maturity levels such as software developers' preferences for software development methods. Finally, GT has received considerable attention and popularity in the fields of information systems in general, and software engineering in particular, over recent years, which illuminates the potential of GT as a research method for tackling research problems in the broader discipline (see Adolph, Hall, and Kruchten (2011; 2012), Birks, Fernandez, Levina, and Nasirin (2013), Hoda, Noble, and Marshall (2013), and van Waardenburg and van Vliet (2013)).

Figure 1 shows the key steps in the Grounded Theory approach employed in this study. We briefly describe each of the steps below.

Figure 1. Overview of Grounded Theory process steps adopted



We began by broadly defining the area of interest as studying software professionals and their perspectives on various software development methodologies. Consistent with Glaser (1978), we refrained from formulating a research problem or question in a precise manner at the outset. This step was purposeful to avoid corrupting the study with preconceived research problems and extant literature in the area. Another goal was to uncover perspectives and problems of the participants in an emergent manner rather than superimposing the problem definition. Both strategies used GT to to generate theory inductively.

We did not conduct an extensive literature review in the early stages of the study. The rationale was similar to that of avoiding pinning down a research question and clouding our minds with preconceived ideas and notions, as discussed above. Another reason for not conducting a formal literature review at the beginning of the study was our background as researchers. One of us had professional experience as a software developer and software development manager for over a decade. In those roles, most projects dealt with using traditional waterfall methods with some exposure and interest in agile methods in recent years. The other researchers did not have extensive background as software professionals yet our education and training prepared us with general information about various software development methodologies including the terminology and key steps in various methodologies. Based on shared experiences and prior knowledge, we were generally cognizant that a particularly good theory about how software developers perceive software development methodologies did not exist. Given this background, we made a conscious effort not to engage in further literature review related to software development methods and their use, until much later stages in the study. Most of the reading during the study was focused on applications of GT in areas different than the substantive area of research we were focusing on (O'Reilly, Paper, & Marx, 2012). Despite these recognized biases, we, as researchers, used our curiosity and interest to help drive interview questions, but tried to remain objective when it came to analysis and interpretation of the data.

During data collection, we sought participants with significant experience as software developers. Participants were from a wide variety of environments including large multi-national corporations to sole proprietorship consultants. All participants were working in the United States. They worked in California, Washington, Utah, South Dakota, Kansas and Missouri. We selected participants from a diversity of industries. Participants range from government employees to employees of large retail companies, some with start-up experience, and others with long standing defense industry backgrounds. Participants were also diverse in age from early 20s to 50s. But all participants have significant experience as practitioners in software development. Table 2 shows participant summary information. Also, given that the study aim was to uncover perspectives of software engineers' toward using software development methodologies, we specifically excluded management and customer participant roles. Both management and customer perspectives are of interest and may be pursued in subsequent research, but do not relate to the current substantive area of research.

As part of the semi-structured interview questions we probed what the developer considered as a definition of agile. The answers ranged from a hodgepodge of industry buzzwords to very informed views. Through the interview the semi-structured questions provided clarity and direction to the developers to ensure that the data gathered focused on current themes of agile definitions from the literature.

In terms of data gathering medium, 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 transcribed documents formed the data repository for the research. Overall, data was collected through sixteen different interviews. The initial set of interviewees was chosen from one of the author's 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, which led to additional participants. Each interview was recorded digitally and then transcribed via a transcription service. Transcriptions were reviewed against the recordings and corrections were made based on the comparisons. To maintain confidentiality, identifying information was removed and replaced with generic placeholders. The result of data collection was 397 minutes of recorded conversations, which translated to 165 pages of transcribed text.

Data collection was accomplished through use of semi-structured interviews. While interviews were largely conversation-based, a few typical questions asked were:

- 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?

Table 2. Participant summary information

Demographic Information	
Number of participants	16
Number of sites	10
Age range	22 – 54
Company size	Small – Large Multi-national
Experience level	Entry level to senior
Data Collection Information	
Minutes of recording	397
Pages of transcript	165

- Do you associate the way you build software with any particular method? Why or why not?
- Did anything or anyone have a particular influence on your level of preference?

These questions formed the basis for the interviews, but as discussions progressed a wide range of topics were covered and additional follow-up questions were used to delve deeper into areas of interest. We employed the four stages of constant comparison (Glaser and Strauss, 1967), namely: (a) comparing incidents relevant to each category, (b) integrating categories and their characteristics or properties, (c) delimiting the theory, and (d) writing the theory to guide data analysis efforts. Through simultaneously gathering, coding and analyzing data, we aimed to uncover new conceptual problems or ideas that were either related or unrelated to the study's initial conceptual notion. After the first few interviews, and subsequent category refinement, it became evident that there was a consistent preference for agile methods among developers. At the onset of data collection, based on our prior experience and background, it was generally expected that there would be distinct camps of software engineers who would prefer one methodology to another. However, emergence of the preference of agile methods theme informed our initial conceptualizations and shifted our investigation from why developers prefer certain software methodologies to why developer preference toward agile methods of software development is increasing. In subsequent interviews, questions such as the following were also typically included:

- Have you heard of agile software methods? How would you describe them?
- How do agile methods compare with your preferred method of developing software?
- Were there any specific experiences that made you lean one way or the other in regard to agile methods?
- What drives your level of preference for agile software methods?

As noted earlier, data analysis was conducted in tandem with data gathering through constant comparison and theoretical sampling. Data analysis resulted in two types of code, namely *substantive codes* and *theoretical codes*. Substantive codes are the categories of theory that surface from the data (Glaser, 2005). Theoretical codes organize, abstract and relate substantive codes (Glaser, 2005). Substantive codes result from open coding and selective coding, whereas theoretical codes result from theoretical coding.

Initially we engaged in open coding, evaluating relevant pieces of information in the written text, analyzing and tagging the text with descriptive codes. These codes represent the meaning and can often use a particularly vivid word from the dialog itself. These types of codes are often referred to as *in vivo* codes. We used ATLAS.ti to serve as a database for the transcribed content with codes and diagrams establishing a chain of evidence. 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 selective and theoretical codes, and memos using network diagrams.

During the open coding phase, we remained open to emergent concepts staying close to the data while constructing short codes with an emphasis on preserving actions constantly comparing and moving quickly through the data. This process provided direction for the research performed in this study. For example, after each interview and infusion of new data, analysis was performed through each coding level. Our coding levels included open and selective coding resulting in substantive codes and categories. It also included theoretical coding resulting in concepts and generating new theoretical insights.

The end of open coding was marked by the emergence of the *core category* (Glaser, 1992). As Glaser (1978) states, the core category explains the majority of variation within the data. Several criteria were used in selecting the core category, which included: (a) it is central and related to several other categories and their characteristics, (b) recurrent theme in the data, (c) theoretically saturates

after other categories, and (d) relates conceptually and meaningfully with other categories. In this study, as described in the findings, we found "*preference rationalization for agile methods*" as the core category.

Upon establishing the core category, data collection and analysis continued with selective coding as the focus. As stated by Glaser and Holton (2004), the objective is to focus on codes that are relevant to the core category resulting in a parsimonious theory. The core category helped guide further data collection, analysis and theoretical sampling. The goal was to sufficiently elaborate the core category, its properties and theoretical connections to other categories. This process continued until we could find no additional data that would further enhance the properties of the category (theoretical saturation).

Throughout the process of data analysis, memoing was an integral component. It allowed us to elaborate on ideas about the codes and their relationships as they emerged from the data. Memo writing focused on free-flowing description and theoretical analysis of the codes. Once theoretical saturation was reached, we began sorting memos and writing content. During the process, we used theoretical coding as a mechanism to abstract the general notion captured in the core variable and its relationship to the other concepts. In this case, we found that "*rationalization*" was the key theoretical code, implying a weighing and balancing act that software developers engage while shaping their preference for agile methods.

4. FINDINGS

This section describes the key results of our study and the emergent model for preference rationalization in choosing between agile methods and traditional methods of software development. The findings resulted from an iterative process of collecting, coding and analyzing participant data to uncover theoretical underpinnings of the phenomenon. Figure 2 presents a visual summary of the findings. As can be seen, *preference rationalization for agile methods* emerged as the core category with *self-efficacy*, *affective response*, *interpersonal response*, *personality contingencies*, and *external contingencies* as underlying categories, each relying on several concepts emergent from the raw data.

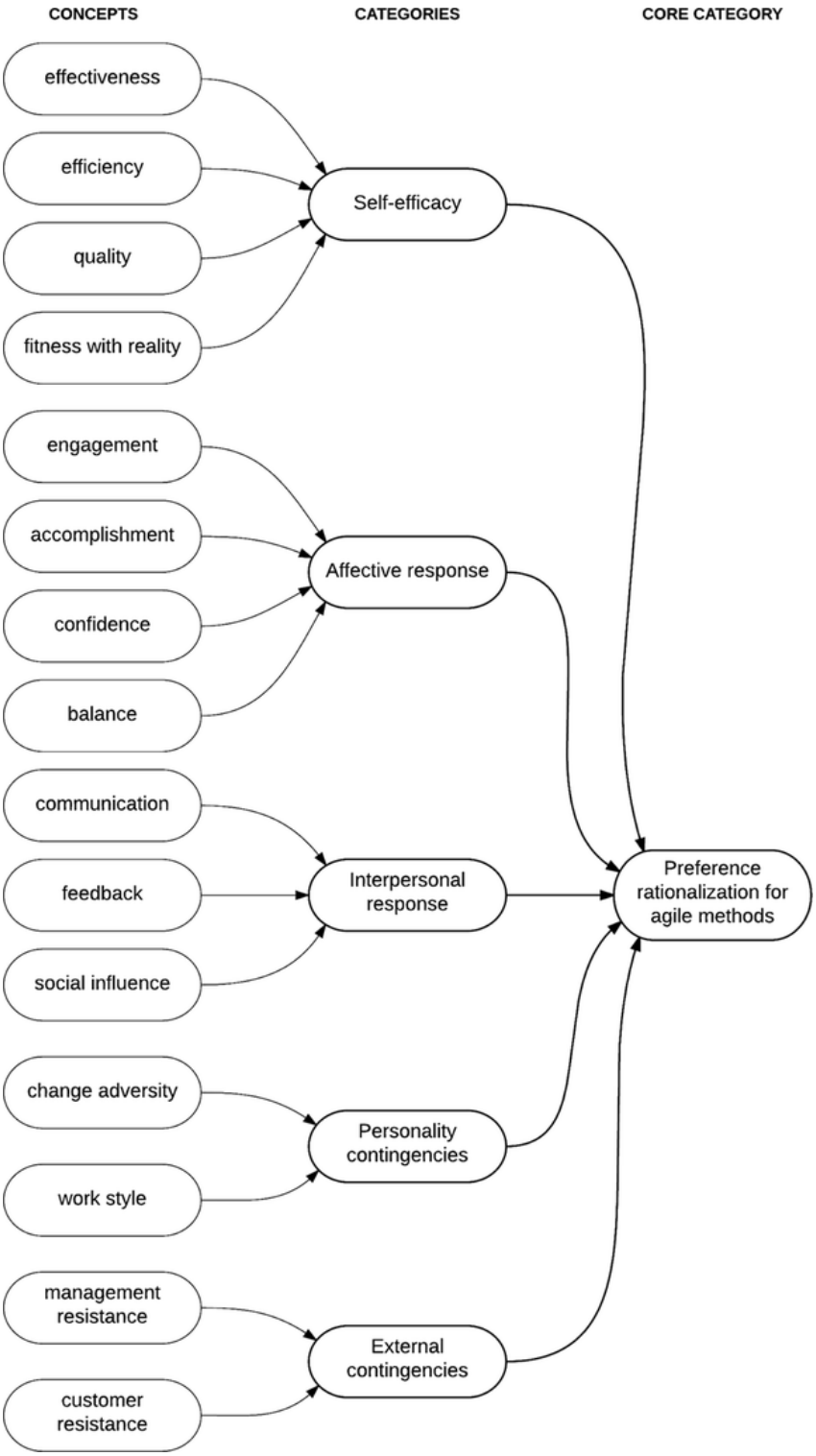
4.1. Self-Efficacy

Self-efficacy as a category emerged through constant comparison and theoretical sampling. Upon reaching theoretical saturation, self-efficacy as a category captured the idea that participants feel 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. The data suggests that software developers broadly perceive that agile methods enhance their self-efficacy. Various dimensions of self-efficacy in relation to agile methods were evident from the analysis, and were consistent across participants. These include – (a) effectiveness, (b) efficiency, (c) quality, and (d) fitness with reality. Each of these dimensions is elaborated below.

4.1.1. Effectiveness

Effectiveness emerged as a consistently strongly grounded concept. Software developers perceive agile methods to be more effective in comparison to traditional methods. This notion became clear after few early interviews, which was then explored further iteratively through additional data collection, coding, and analysis. Numerous participants mentioned that "waterfall never worked." 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. Developers that appreciated agile felt that the traditional software development methods were less effective and believed that agile was more effective. A typical comment related the effectiveness of agile methods was "I just found that things like agile development seemed to produce better results" (P4) or "traditional waterfall … never worked (P4). Many developers noted that agile implemented

Figure 2. Emergence of preference rationalization core category



industry best practices and that it was the best way to develop software, as can be seen in one of the comments:

I tried experiment[ing] with different things and said, you know, what works, what doesn't, what makes me more effective, I just found that things like agile development seemed to produce better results. (P3)

One of the appealing factors of agile is that projects are perceived to be more effective as reflected by higher success rates. A participant mentioned "I have seen comparatively with Waterfall and agile, the success rate is really high in agile" (P9) Other participants mentioned that agile is more productive due to agile methods' 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" (P8). 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.

4.1.2. Efficiency

The concept of efficiency resonated with a number of participants. Participants 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, their experience in agile methods 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 succinctly, "Agile is fast" (P16).

A common detractor from waterfall methods and, in turn, a positive factor influencing preference for agile methods is the disconnectedness caused by excessive disengaged documentation suggesting that documentation as done in waterfall tends to be 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 sync with the real code in the project. 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 perceived as inefficient. This comment is telling:

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. (P13)

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" (P13). Capturing a similar sentiment about the inefficiency of waterfall methods but not specifically related to documentation, another participant 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. (P16)

Another aspect 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. Evidently, 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 (see 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 approaches 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 methods' stand-up meetings new team members quickly got up to speed on the project by listening to each team member during 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, questions and answers fit into their present experience making the learning process both efficient and socially straightforward.

An important aspect of efficiency that works in favor of agile methods is the involvement of the customer. It bypasses the need to constantly search through requirements documents to get clarification. Instead one 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. (P12)

Agile methods in general encourage periodic team retrospection (Beck et al., 2001). This allows teams to fine tune their processes and improve efficiency. This was communicated by a developer in her 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" (P13). This self-optimization process increases efficiency over time and accelerates the delivery of software products.

Another efficiency related consideration that became evident is that agile methods suggest working on the highest value features first. A participant mentioned that in his experience with waterfall approaches they might 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. (P10)

4.1.3. Quality

Improvements to the end product were a frequent comment from developers. They attributed much of the improvement to quality to the use of unit testing in agile methods. 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 were able to design and write 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 was 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. (P4)

One negative aspect also emerged from the data in relation to quality and agile methods, although it was not consistently supported in subsequent data collection and analysis. 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. The following participant comment is informative:

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. (P5)

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 et al., 2001) but could be a perceived issue with agile methods' emphasis on early delivery of value to customers.

4.1.4. Fitness with Reality

Fitness with reality is a strong concept that emerges from discussions with participants. In their experience, the waterfall approach 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 the waterfall or traditional approaches, 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. (P4)

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. (P13)

A number of participants expressed that agile methods just felt right. Many said they had been doing it without knowing that is what it is called. It naturally 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" (P7). Another participant said:

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. (P16)

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" (P16).

4.2. Affective Response

Affective response emerged as an abstraction of the recurrent notion that participants experienced a number of emotional benefits while practicing agile software development in comparison to traditional waterfall methods. 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. Each of these dimensions is elaborated below.

4.2.1. 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 updating 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" (P4). In contrast, participants found agile highly engaging, even fun. One reason for this heightened engagement with agile is that, as more than one participant said, it focuses on coding and coding is fun. That is, 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" (P8). 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" (P8).

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. (P14)

Others described the project manager (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 instance, 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" (P9). The customers are out of the loop during the design, development and system testing phases increasing their anxiety over the outcome of the project. He 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. (P9)

There are a number of dimensions where engagement is increased with agile; from individual programmer to project manager to customer, each is more engaged. Increased engagement is a driver for preference of agile methods by developers.

4.2.2. Accomplishment

Developers described a sense of accomplishment when employing agile methods for software development. Many participants mentioned that they like that with agile their work is broken down into small tasks and the tasks can be implemented fairly quickly making progress visible. This provides a 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. (P15)

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. (P5)

Accomplishment is represented by two dimensions. First, it derives from a small task size that provides a well-defined unit of work that can be completed and progress noted. Second, short cycles regularly deliver working software to customers. The regular delivery of software to customers provides a sense of accomplishment.

4.2.3. Confidence

Confidence emerged as a strong benefit of employing agile methods. Many different participants highlighted their feeling of confidence resulting from agile. This was due to two different factors. The first is the confidence to make changes. Change may come in two forms, technical improvement and customer-driven. 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. (P7)

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. (P4)

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 – if 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? (P16)

Another participant related his confidence to using agile-engaged stakeholders early and allowing them to visualize the system, to make the intangible tangible, and 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 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.

4.2.4. Balance

"Waterfall, in my experience, towards the end of the release, it tends to be hell" (P10). 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" (P10). This sentiment is well known among those that have experienced a difficult project using the Waterfall method. 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 et al., 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" (P7). This leads to pressure and stress on those "heroes" (and to some degree a sense of uselessness on the "helper people").

4.3. Interpersonal Response using Agile Methods

The final category that emerged relates to interpersonal response factors. Communication, feedback and social influences arise in relation to interpersonal factors.

4.3.1. 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 instance, in a Waterfall method there may be long periods where there is little communication and accountability. As one participant said, "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" (P4). 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. (P4)

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" (P14)?

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. (P12)

4.3.2. 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 feedback more frequent and more timely, it is also more concrete and relevant. 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" (P12). 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" (P15).

As another developer said:

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. (P13)

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 said:

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. (P8)

4.3.3. Social Influence

Another interpersonal factor that emerged as positively influencing 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. (P13)

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" (P11). 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. (P1)

4.4. External Contingencies

The data also revealed two contingent categories, external contingencies being one of them. The concepts of management resistance and customer resistance together shaped to form this category.

4.4.1. Management Resistance

A number of participants' discussed the influence that management has on their desire to use agile methods. One particular participant was aware of management resistance. He perceived that his management preferred Waterfall because it allowed them to some degree to distance themselves from the implementation details allowing them to focus on things like schedules and specifications. He said:

I think the second source of opposition was kind of on the program manager's side, you know, the non-coder side. I think they were just so used to kind of being the ones who – you know, we sit down, we figure out a spec, we all go through months of talking about it and we sign off and then you guys just go work on it, and it really kinda changed the way that they interacted … They used to really distance themselves from the design and implementation problems. (P4)

Another participant said that management liked Waterfall because of the ability to track project progress against milestones. When the participant was asked if he preferred agile he initially said yes. But when the question was modified to say that management had a slight preference for Waterfall he immediately changed his answer and said he would defer to whatever approach management preferred. This indicates the impact that management resistance has on some developers' preference.

Yet another participant was also heavily influenced by management. He had never participated in an agile project because management chose to use Waterfall. He had not done any reading on agile or discussed it with any colleagues because he had chosen to follow management's lead away from agile. Consequently, management has inhibited his interest and desire to pursue agile software development methods.

The data tells us that management resistance is real and has the capability of inhibiting developer preference for agile methods.

4.4.2. Customer Resistance

Some developers were particularly sensitive to customer-stated preference in terms of development approach. Two of the participants worked in the defense industry and felt that there was a bias from their customers toward the waterfall software development approach. This customer bias significantly influenced these developers' preference by inhibiting their desire to use agile. Their perception of the customer's preference influenced their own preference even when they felt that agile might be the better development methodology.

In some cases developers said that customers mandated that Waterfall be used which superseded the developers' personal preference of agile:

Not by my choice, certainly not. It was by mandate. It was because that's what the customer … well, my impression at the time is that's what the customer expected because the state and local governments, they were comfortable with that [Waterfall]. (P1)

Another participant said that the magnitude of the project in terms of features and cost drive some customers to prefer Waterfall and that impacts his own decisions about agile versus Waterfall preference:

It all depends on the size of the – yeah, key things that were deciding agile versus waterfall is the way I see it is the size of the project from the financial perspective and also the duration or time to implement the project and also the availability of the business owners … all these are constraints for a project manager or for any organization to decide which one [agile or Waterfall] is the better way to go. (P9)

4.5. Personality Contingences

The other type of contingent category that was discovered from the data is that personality contingencies. Concepts of change adversity and work style together shaped to form this category.

4.5.1. Change Adversity

Change adversity is an inhibitory agent that emerged from discussions with developers. A quote from one participant reads like this, "of course I was skeptical because it [agile] is new and different" (P3). 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. His tone in the conversation indicated that he was change adverse and that he preferred to stick with what he had experience with, which was the Waterfall methodology. Other participants mentioned that they observed co-workers who were change adverse and therefore did not appreciate agile methods. The developers' perception was that change adverse people did not want to put forth the effort to learn a new methodology because they were comfortable with their existing methods.

One participant put it like this:

It's the idea that in a lot of cases you're talking about people who have kind of become entrenched in their career and they're not necessarily interested in changing or taking on new things. They're – not to badmouth them, but we're not talking here about people who are really interested in continuing to grow their skills, it's more like trying to just sort of get by with what you already know. (P4)

In addition, he also said:

I think that frankly on the side of just opposition to the agile approach and kind of sticking with the status quo, I think the main opposition just came from reluctance to change 'cause, you know, we don't really like change. (P4)

Another developer said "I'm probably a creature of habit so I'd probably start with Waterfall 'cause it's what I know" (P6). 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.

Clearly, there is a segment of software developers who are content with traditional methods like Waterfall and because of their adversity to change are not as interested in adopting something new like agile. Thus, change adversity has a moderating effect in shaping software developers' preference for agile.

4.5.2. Work Style

Another dimension to personality factors that may impede preference for agile methods is work style. One participant 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. His preferred work style just didn't fit with agile's emphasis on face-to-face communication.

I'm one of those [people] where tell me what to do and then let me get at it and I know what I'm doing so let me do it. (P6)

This participant also didn't like test driven development, which he considered part of an agile process. He did not like starting off with failure and then trying to make the code work. This approach was counter to his preferred work style. He states, "I definitely didn't like that one, where you gotta design it to fail. I'm like no, I don't design things to fail, that is just wrong to me" (P6).

A developer's preferred work style may conflict with agile principles, which then leads to diminished preference for agile.

4.6. Preference Rationalization for Agile Methods

Analysis of the data revealed a recurrent theme emerging from the data that suggests that developers rationalize their preference for agile by weighing the various categories discussed to arrive at their preference value for agile software development methods.

Analysis of self-efficacy in relation to other categories made it clear that self-efficacy was a key factor in shaping preference of developers toward agile methods. 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.

One participant noted 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. 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-based 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. This observation led to further investigating the conditions under which the participants perceive self-efficacy using agile methods. It became clearly evident from the data that the systematic use of agile methods in a holistic manner as opposed to a partial-use or a splintered view in implementing

agile principles is key in developers perceiving increased self-efficacy. Thus the issue of holistic adoption of agile methods is to be recognized as a concept that conditions perceived self-efficacy. Also, this type of partial adoption experience is interesting in itself as a fruitful avenue for further exploration and research in a separate study.

Affective response was also noted as a clear driving force in shaping developers' preferences in favor of agile methods. The way developers talked about software development shows that it is an emotional endeavor not just a cognitive effort. Based on the participant comments, it can be said that 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.

Interpersonal response category, when viewed alongside other categories, was also noted to be a driver in increasing developers' preference toward agile methods. In brief, developers believed that communication, short feedback cycles and social influence were enhanced by agile methods increasing developer preference for agile.

In contrast to the previously mentioned drivers of preference, we noted that developers' preferences are also shaped by contingent factors, although negatively, thus leading to a tension in preferring one method to another. The data showed that at least two categories of personality contingencies to agile preference emerged – one external and the other personality based. On the external side, both management and customers were found to exert influence on developers' preference We found that developers, in most cases, showed deference to management direction and if the management steered away from agile so did the developers. Similarly, we found that some developers chose to defer to either perceived or explicit customer preference in terms of software development method preference. Essentially, if the management or customers make it known that they want Waterfall over agile the developers are apt to listen and exhibit diminished preference for agile.

We also found that developer personality also may moderate their preference for agile. Agile emphasizes a social environment to guide development, which may be contrary to some developers' desires. There are also specific work habits of developers that may not harmonize with certain agile practices. When this is the case, these personality characteristics may temper a developer's preference for agile. In comparison to the preference drivers noted earlier, these external and personality contingencies were found to moderate preference for agile methods and thus may be viewed as a preference inhibitor.

Essentially, we found that drivers for preference come in the form of self-efficacy, affective response and interpersonal response while contingencies are expressed through external factors and personality. As one participant put it:

We went through different models. Then when we created different models, that's when I came to an understanding of how this agile works, how it has better helped these projects [be] successful rather than waterfall and what [it] is that we are gaining. (P9)

Another developer illustrates the preference rationalization process when he said:

The Waterfall approach just involves ultimately writing and lots of documentation and while that appeared to be productive work at the time, but usually the degree of detail that you ended up going into wasn't necessarily needed by the customer … you really kind of miss the bigger picture sometimes. The avoiding documentation and just delivering functionality earlier has always been much more successful. (P12)

Developers weigh their experiences (and those they have heard of) using an internal calculus based on previously described categories to arrive at a value judgment with regard to agile methods. Individuals use varying inputs both internal (personal experience) and external (management and customers). These significant voices speak uniquely to each developer but the mix influences their preference and results in either a positive disposition or in a few cases a negative stance toward agile methods. It is surprising that for many developers it only takes a few vivid experiences to guide them into a particular feeling about agile.

Not all developers arrive at the same conclusion as we have seen due to external factors imposed by management and customers or by work style or personality issues. The uniqueness of experience is mirrored by the uniqueness of responses to common experiences. One of the semi-structured interview questions was, "how would you respond if your manager said they are no longer going to use agile?" Those heavily influenced by external factors capitulated and said they would follow their management's direction (in this case away from agile). Those developers giving more weight to internal factors, when faced with a prohibition by management went so far to say they would find a different company to work for. This illustrates the diversity of the individual calculus applied by developers. But those of whom positive experiences with self-efficacy, affective response, and interpersonal response outweigh the inhibitors, there surfaces an inclination toward agile software methods.

5. DISCUSSION AND IMPLICATIONS

We now discuss the findings from the study in relation to findings from other studies, their application to future research as well as the implications for management in practice.

Williams (2012, p. 76) reports results from two survey studies conducted to gauge software developer communities' views on principles of agile methods laid out in the Agile Manifesto. The overall finding was that the principles succinctly captured the "essence of the agile trend that has transformed the software industry over more than a dozen years." We find that many of these principles resonated among the factors (e.g., efficiency, effectiveness, quality) contributing toward developers' preference towards choosing agile methods. Another study by Yu and Petter (2014) found that agile practices are instrumental in developing shared understanding (i.e., mental models) among developers and customers in software development teams. This finding also aligns with several factors found in our study related to shaping preference for agile methods among developers. Yet another study by Hoda, Noble and Marshall (2012) reports that self-organizing agile teams engage in a number of balancing acts including (a) cross-functionality and specialization, (b) freedom and responsibility, and (c) continuous learning and iteration pressure. Finally, Dybå and Dingsøyr (2008) conducted a review of extant studies till 2005 and report four studies that found positive perceptions of developers toward using agile methods and their willingness to use in future projects.

One of the exciting findings in our study 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 software developer currently has a low preference for agile, the constructs for preference are such that through education and experience their preference can be increased. Since the positive constructs that emerged from the data are all modifiable there may be opportunities to develop programs to increase preference for agile.

Another implication of this research is for management professionals. Although the unit of analysis is at the individual software developer level there are a number of implications that apply to the management of teams of software developers and software projects. In the future, we intend to 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. Managers can engage in learning about the team members' preferences. Team members that do not have prior experience with agile methods may be given opportunities to shadow experienced professionals as well as participate on pilot projects. Software

developers' responses in terms of self-efficacy, affective response and interpersonal factors can inform about the candidate's affinity for preferring agile methods to traditional methods. Building on current findings, further research is needed in developing and validating an instrument to measure software developer preferences that can also guide the aforementioned process. If the candidate does not adhere to these concepts and constructs, then they may resist using agile methods.

Team optimization opportunities may exist in organizations or teams transitioning to agile methods. Understanding what drives software developers' preference can be useful in developing 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 likelihood that members of transitioning teams will develop a preference for agile methods.

An active agile team can leverage the findings of the study from a team retention perspective by emphasizing the elements that have been identified as key influencers of software development method preference among developers. Ensuring that the discovered principles are practiced by a team will likely lead to higher satisfaction and engagement among the team members. Also, given that one of the key conceptual categories discovered is self-efficacy, these factors may potentially lead to higher productivity in addition to increased satisfaction and retention among software development teams employing agile methods. Validating these propositions is an avenue for future research.

Another direction for future research being considered by the researchers is to examine preference from a customer perspective. How do customers perceive agile, is it better and preferred over other approaches? With customers as participants, a study, perhaps also employing grounded research approach, can inform the aspects that matter to customers (e.g., like/dislike, care/do not care) about agile methods compared to alternative software development methodologies.

6. CONCLUSION

In this study, we adopted a grounded theory approach to shed light on factors that influence software developers' preference for using agile methods. The discovery of these factors is the key contribution of this study. Through a method of constant comparison and an iterative process of data collection (theoretical sampling) and analysis, we discovered *self-efficacy*, *affective response*, *interpersonal response*, *external contingencies*, and *personality contingencies* as categories of factors influencing developers' preference for agile methods. A recurrent theme of *preference rationalization for agile methods* emerged as the core category from the data. We observed that the categories that positively influence preference for agile methods are malleable and can thus inform management of software development teams adopting agile methods in regards to team formation, optimization, and retention. Our work adds to the continuing stream of research on adoption of development methodologies (Kacmar et al., 2009), developer attitudes and preferences (Hendersen et al., 2012) and complements research that explores issues such as methodology fit (Xu & Yao, 2014) and project commitment (Korzaan & Brooks, 2015) on organizational productivity.

REFERENCES

Abdelnour-Nocera, J., & Sharp, H. (2012). Understanding conflicts in agile adoption through technological frames. *International Journal of Sociotechnology and Knowledge Development*, *4*(2), 29–45. doi:10.4018/jskd.2012040104

Adolph, S., Hall, W., & Kruchten, P. (2011). Using grounded theory to study the experience of software development. *Empirical Software Engineering*, *16*(4), 487–513. doi:10.1007/s10664-010-9152-6

Adolph, S., Kruchten, P., & Hall, W. (2012). Reconciling perspectives: A grounded theory of how people manage the process of software development. *Journal of Systems and Software*, *85*(6), 1269–1286. doi:10.1016/j.jss.2012.01.059

Armstrong, D. J., Nelson, H. J., Nelson, K. M., & Narayanan, V. K. (2008). Building the IT workforce of the future: The demand for more complex, abstract, and strategic knowledge. *Information Resources Management Journal*, *21*(2), 63–79. doi:10.4018/irmj.2008040104

Balijepally, V. G., Mahapatra, R. K., & Nerur, S. P. (2006). Assessing personality profiles of software developers in agile development teams. *Communications of the Association for Information Systems*, *18*(1), 4.

Batra, D., VanderMeer, D., & Dutta, K. (2011). Extending agile principles to larger, dynamic software projects: A theoretical assessment. *Journal of Database Management*, *22*(4), 73–92. doi:10.4018/jdm.2011100104

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., & Grenning, M.F.J. … Thomas, D. (2001). Manifesto for Agile Software Development. Retrieved from http://www.agilealliance.org/the-alliance/the-agile-manifesto/

Bergadano, F., Bosio, G., & Spagnolo, S. (2014). Supporting collaboration between customers and developers: A framework for distributed, agile software development. *International Journal of Distributed Systems and Technologies*, *5*(2), 1–16. doi:10.4018/ijdst.2014040101

Birks, D. F., Fernandez, W., Levina, N., & Nasirin, S. (2013). Grounded theory method in information systems research: Its nature, diversity and opportunities. *European Journal of Information Systems*, *22*(1), 1–8. doi:10.1057/ejis.2012.48

Boehm, B., & Turner, R. (2004). Balancing agility and discipline: Evaluating and integrating agile and plan-driven methods.*Proceedings of the 26th International Conference on Software Engineering* (pp. 718–719). IEEE. doi:10.1109/ICSE.2004.1317503

Bonner, N. A. (2010). Predicting leadership success in agile environments: An inquiring systems approach. Academy of Information and Management Sciences Journal, 13(2).

Cano, S. P., González, C. S., Collazos, C. A., Muñoz-Arteaga, J., & Zapata, S. (2015). Agile software development process applied to the serious games development for children from 7 to 10 years old. *International Journal of Information Technologies and Systems Approach*, *8*(2), 64–79. doi:10.4018/IJITSA.2015070105

Chaudhary, P., Hyde, M., & Rodger, J. A. (2015). Attributes for executing change in an agile information system. *International Journal of Technology Diffusion*, *6*(2), 30–58. doi:10.4018/IJTD.2015040103

Cockburn, A., & Highsmith, J. (2001). Agile software development: The people factor. *IEEE Computer*, November(11), 131–133.

Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, *50*(9-10), 833–859. doi:10.1016/j.infsof.2008.01.006

Glaiel, F., Moulton, A., & Madnick, S. (2013). *Agile project dynamics: A system dynamics investigation of agile software development methods* (No. CISL# 2013-05). Cambridge, MA. Retrieved from http://web.mit.edu/smadnick/www/wp/2013-05.pdf

Glaser, B. (2005). *The grounded theory perspective III: Theoretical coding*. Mill Valley, CA: Sociology Press.

Glaser, B., & Strauss, A. (1967). *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Chicago: Aldine Publishing Company.

Glaser, B. G. (1978). *Theoretical Sensitivity: Advances in the Methodology of Grounded Theory*. Mill Valley, CA: The Sociology Press.

Glaser, B. G. (1992). *Emergence vs. Forcing: Basics of Grounded Theory Analysis*. Mill Valley, CA: Sociology Press.

Glaser, B. G., & Holton, J. (2004). Remodeling Grounded Theory. *Forum: Qualitative Social Research, 5*(2). doi:10.1016/j.clae.2007.06.001

Hendersen, D., Sheetz, S. D., & Bélanger, F. (2012). Explaining developer attitude toward using formalized commercial methodologies: Decomposing perceived usefulness. *Information Resources Management Journal*, *25*(1), 1–20. doi:10.4018/irmj.2012010101

Hoda, R., Noble, J., & Marshall, S. (2012). Developing a grounded theory to explain the practices of self-organizing Agile teams. *Empirical Software Engineering*, *17*(6), 609–639. doi:10.1007/s10664-011-9161-0

Hoda, R., Noble, J., & Marshall, S. (2013). Self-organizing roles on agile software development teams. *IEEE Transactions on Software Engineering*, *39*(3), 422–444. doi:10.1109/TSE.2012.30

Kacmar, C. J., McManus, D. J., Duggan, E. W., Hale, J. E., & Hale, D. P. (2009). Software development methodologies in organizations: Field investigation of use, acceptance, and application. *Information Resources Management Journal*, *22*(3), 16–39. doi:10.4018/irmj.2009070102

Koch, S., & Turk, G. (2011). Human resource related problems in agile and traditional software project process models. *International Journal of Information Technology Project Management*, *2*(2), 1–13. doi:10.4018/jitpm.2011040101

Korzaan, M. L., & Brooks, N. G. (2015). The binding and blinding influence of project commitment. *Information Resources Management Journal*, *28*(1), 57–74. doi:10.4018/irmj.2015010104

O'Reilly, K., Paper, D., & Marx, S. (2012). Demystifying grounded theory for business research. *Organizational Research Methods*, *15*(2), 247–262. doi:10.1177/1094428111434559

Parry, K. W. (1998). Grounded theory and social process: A new direction for leadership research. *The Leadership Quarterly*, *9*(1), 85–105. doi:10.1016/S1048-9843(98)90043-1

Rahman, N., Rutz, D., & Akhter, S. (2011). Agile development in data warehousing. *International Journal of Business Intelligence Research*, *2*(3), 64–77. doi:10.4018/jbir.2011070105

Schwaber, K. R. (2004). *Agile Project Management with Scrum* (Vol. 7). Redmond, WA: Microsoft Press.

Teoh, S. Y., & Cai, S. (2015). The process of strategic, agile, innovation development: A healthcare systems implementation case study. *Journal of Global Information Management*, *23*(3), 1–22. doi:10.4018/JGIM.2015070101

Teoh, S. Y., & Chen, X. (2013). Towards a strategic process model of governance for agile it implementation: A healthcare information technology study in China. *Journal of Global Information Management*, *21*(4), 17–37. doi:10.4018/jgim.2013100102

Van Waardenburg, G., & van Vliet, H. (2013). When agile meets the enterprise. *Information and Software Technology*, *55*(12), 2154–2171. doi:10.1016/j.infsof.2013.07.012

West, D., Grant, T., Gerush, M., & D'Silva, D. (2010). *Agile development: Mainstream adoption has changed agility*.

Williams, L. (2012). What agile teams think of agile principles. *Communications of the ACM*, *55*(4), 71–76. doi:10.1145/2133806.2133823

Xu, P., & Yao, Y. (2014). Methodology fit in offshoring software development projects. *Information Resources Management Journal*, *27*(4), 66–81. doi:10.4018/irmj.2014100104

Yu, X., & Petter, S. (2014). Understanding agile software development practices using shared mental models theory. *Information and Software Technology*, *56*(8), 911–921. doi:10.1016/j.infsof.2014.02.010

David B. Bishop is an Associate Professor in Information Systems at the College of Business and Information Systems, Dakota State University, Madison, South Dakota. His research interests are in agile software development and software engineering. In addition, Dr. Bishop has over 20 years of experience in software development and holds a BS in Mathematics/Computer Science from Western Washington University, a MS in Information Systems from Dakota State University and a DSc in Information Systems from Dakota State University.

Amit V. Deokar is an Assistant Professor of Management Information Systems in the Robert J. Manning School of Business at the University of Massachusetts Lowell. Dr. Deokar received his PhD in Management Information Systems from the University of Arizona. He also earned a MS in Industrial Engineering from the University of Arizona and a BE in Mechanical Engineering from VJTI, University of Mumbai. His research interests include predictive analytics, business intelligence, process management, and collaboration processes and technologies. His work has been published in journals such as Journal of Management Information Systems, Decision Support Systems (DSS), The DATA BASE for Advances in Information Systems, Information Systems Frontiers, Business Process Management Journal (BPMJ) and IEEE Transactions. He is currently a member of the editorial board of DSS and BPMJ journals. He has been serving as the Business Analytics, Big Data and Knowledge Management Track Chair at the international AMCIS 2014-16 conferences, and Program Chair of the AIS Special Interest Group on Decision Support and Analytics (SIGDSA). He was recognized with the 2014 IBM Faculty Award for his research and teaching in the areas of analytics and big data.

Surendra Sarnikar is an Associate Professor in Information Systems at the College of Business and Economics, California State University East Bay. He holds a PhD in Management Information Systems from the University of Arizona. He has taught healthcare informatics, design research and knowledge management at the Dakota State University. He has published several conference and Journal publications in the area of healthcare information systems, knowledge management systems, and information retrieval. He has won best paper awards for his work in healthcare information systems at the Hawaii International Conference on system Sciences and the International Conference in Information Systems.