How Agile Practices Influence the Performance of Software Development Teams: The Role of Shared Mental Models and Backup

Completed Research Paper

Christoph T. Schmidt
University of Mannheim
Mannheim, Germany
christoph.schmidt@uni-mannheim.de

Thomas Kude
University of Mannheim
Mannheim, Germany
kude@uni-mannheim.de

Armin Heinzl
University of Mannheim
Mannheim, Germany
heinzl@uni-mannheim.de

Sunil Mithas
University of Maryland
College Park, Maryland, U.S.A.
smithas@rhsmith.umd.edu

Abstract

This study draws on team adaptation theory to examine how agile behavior within Information Systems development (ISD) teams influences team performance. We conceptualize agile behavior as the degree to which ISD teams use agile practices and test a theoretical model that links agile practice use to two key components of team adaptation—shared mental models and backup behavior. Moreover, in line with team adaption theory, shared mental models among team members are hypothesized to increase backup behavior, which in turn is suggested to lead to higher levels of ISD team performance in complex environments. To test our hypotheses, we collected data from Scrum masters, project leaders and more than 490 professional software engineers of a global enterprise software development company. Our findings broadly confirm our theoretical model linking agility, adaptation, and ISD team performance, leading to several theoretical and practical contributions.

Introduction

Many software companies have adopted an agile development approach during the last years in order to achieve process flexibility and cope with volatile user requirements (Abrahamsson et al. 2009; Baskerville et al. 2006; Dybå and Dingsøyr 2008; MacCormack et al. 2001). While initially introduced as a counter-movement from the traditional, plan-driven approach, agile development is mainstream today (VersionOne 2012; West et al. 2010). The traditional approach divided the development process into sequential phases of planning, implementation, validation and a single software release at the end of the project (Boehm 2006). Agile development, by contrast, follows an iterative approach with frequent software releases of incremental functionality (Highsmith and Cockburn 2001; Lindstrom and Jeffries 2004). This requires developers to assure software quality throughout the entire development process rather than relying on dedicated testers that validate the software after it is fully implemented.

As a consequence, numerous quality-oriented development practices have become popular during the last years, such as pair programming, code review, or automated testing (Beck 2000), labeled as agile practices. Agile developers write test cases to automatically check their software for bugs (automated testing), ask their colleagues for feedback on work results (code reviews), or even solve the development task together working on a single computer (pair programming). In addition, the popularity of Scrum (Schwaber and Beedle 2002), an agile project management framework, emphasizes the importance of teamwork in most software development organizations.

A number of studies within the IS field have examined software development processes and particularly software development teams (Faraj and Sproull 2000; Harter and Slaughter 2003; Henderson and Lee 1992; Hoegl and Gemuenden 2001). While previous studies have focused on agile software development (Lee and Xia 2010; Maruping et al. 2009; McAvoy and Butler 2009; Ramesh et al. 2012; Sarker and Sarker 2009; Strode et al. 2012; Vidgen and Wang 2009), a comprehensive understanding and evaluation of the agile approach in software development teams is still in its infancy (Dingsøyr et al. 2012). There is a clear need for more theory-based, industrial studies on agile software development teams (Dingsøyr et al. 2012; Dybå and Dingsøyr 2008).

This study addresses this research gap by investigating the impact of using agile development practices on the performance of software development teams. Our key assertion is that the use of agile practices not only directly influences the performance of software development teams, but that it also affects various teamwork mechanisms. Hence, the study draws on team effectiveness research from organization sciences and social psychology (Mathieu et al. 2008; Salas et al. 2005) to answer the central research question: how do agile software development practices influence the performance of software development teams. Team adaptation theory (Burke et al. 2006) holds that shared mental models and backup behavior are central makers of adaptive teams. This study posits that the use of agile practices is positively related to both markers providing a so far unrecognized explanation for performance effects of agile software development on team performance building on the idea that agile development increases team adaptability.

We first develop a research model focusing on shared mental models and backup behavior as determinants of team performance in agile software development teams. We test our hypotheses with survey data from more than 490 professional software developers working in 81 development teams in a large, international enterprise software company. Finally, we discuss the study's findings.

Theoretical and Conceptual Background

Agile Development Practices

Organizations increasingly follow an agile software development approach with software engineers using agile development practices such as pair programming, code review, or automated testing (Beck 2000; Highsmith and Cockburn 2001). These development practices are behavioral norms introduced, e.g. through training programs, to standardize developers' work processes (Mintzberg 1980) and to direct developers towards company-wide goals, e.g. by encouraging software engineers to focus on software quality right from writing the first line of code.

In many agile software companies, however, software development teams are not obliged to apply agile practices. Developers mostly have a great level of freedom to decide how they organize their personal work and how they collaborate within their teams. One the one hand, agile software engineers are not formally monitored or controlled how they fulfill their tasks (Cockburn 2001; Highsmith and Cockburn 2001; Schwaber and Sutherland 2011). Instead, they have a high level of autonomy for decision-making and task fulfillment. On the other hand, software developers have the responsibility for software quality assurance knowing that there is no team-external validation step before the software gets shipped to the customer. Hence, team autonomy coincides with responsibility to deliver software quality that meets customer expectations. To fulfill these responsibilities and to frame this high degree of autonomy, agile teams therefore develop and follow organizational work norms. This study takes a more general perspective and examines the behavioral norms framed by the agile development approach. We assume that agile software practices are a specific way to instantiate the agile work norms.

There are numerous agile software development practices that software development teams use. In this study, we focus on pair programming, code reviews, and automated testing. These practices fundamentally influence software developers' implementation work based on the core agile values and principles. Moreover, our research context comprises thousands of software developers familiar with these three practices as the studied company had decided to formally introduce them through a training program. Finally, these practices are among the most widely adopted agile methods according to a world-wide survey among software engineers (VersionOne 2012). Hence, we conceptualize our independent variable, called 'agile practices use', as the degree to which software development teams apply pair programming, code reviews, and automated testing as a means for standardizing work norms. Even though these particular development practices are specific to software development, we view them as a context-specific instantiation of an agile work mode. This work style may be relevant for new product development teams working in dynamic and uncertain environments.

Team Adaptation Theory

Previous research suggests that most software development teams work in highly dynamic project contexts (MacCormack and Verganti 2003; Schmidt et al. 2001). Lee and Xia (2005) found that business- and technology-related volatility is the major sources of change in software development projects. Regarding business-related change, the team setup may shift, there might be a change in the functional or non-functional system requirements, or due dates may be modified. Second, technology-related change might occur. For instance, the used technology, the system architecture, or other technical components the software relies upon could change unpredictably (Kude et al. 2014a).

We propose the ability of a software development team to adapt to novel situations as an essential antecedent of high performance software development teams. Therefore, we draw on team adaptation theory (Burke et al. 2006) and develop a research model that links the use of agile development practices to team performance explained through two central teamwork factors proposed by the theory (see Figure 1). Team adaptability is defined as a team's "ability to change team performance processes in response to cues from the environment in a manner that results in functional team outcomes" (Burke et al. 2006). Team adaptation theory holds that adaptive teams successfully manage to (1) assess situations appropriately and build a coherent understanding of a new situation, (2) adjust their plans accordingly, (3) coordinate their work to fit the new situation, and (4) learn by evaluating their effectiveness. While iterating this four-step adaptation process, the team develops emergent states and behavioral markers (Marks et al. 2001; Rosen et al. 2011). This adaptation model is very complex and therefore difficult to

apply. Hence, researchers suggested studying behavioral markers which are expected to be more pronounced in adaptive teams. We follow Salas et al. (2005) who proposed that team adaptation is positively related with a high sharedness of team members' mental models (DeChurch and Mesmer-Magnus 2010) and strong team backup behavior (Marks et al. 2001).

In the following sections, we argue that a team's shared mental models as well as team backup behavior are key antecedents for the performance of software development teams. Both are proposed to be either directly or indirectly affected by the use of agile software development practices.

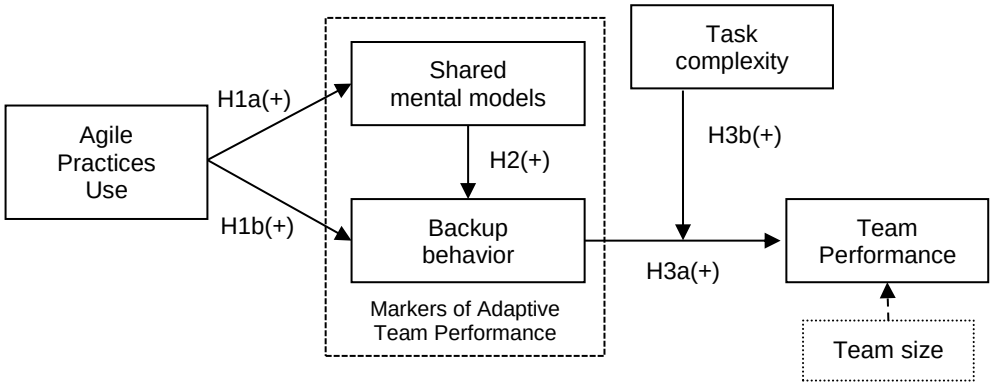

Figure 1: Research Model

Shared Mental Models

Mental models are organized knowledge structures consisting of the content as well as the structure of the concepts in the mind of individuals (Byrne 2001). Mental model theory was first proposed by Johnson-Laird (1980) as a cognitive concept that can represent knowledge structures and relationships. People use mental models to describe, explain, and predict a system or people they interact with. For example, software engineers may possess a mental model of the software architecture or mental models about the expertise of their coworkers. This helps them coordinate collaboration within the team or make good software design decision if the model properly mirrors the actual functioning of the software.

Shared mental models are team members' shared, organized understandings and mental representation of knowledge about key elements of the team's work environment (Klimoski and Mohammed 1994). They are defined as the degree of similarity among the mental models of members (Burke et al. 2006; Byrne 2001; Klimoski and Mohammed 1994; Mohammed and Dumville 2001; Salas et al. 2005). Shared mental model theory holds that effective teams need to maintain a shared understanding within the team that allows team members to interpret information in a similar way, leading to better communication and coordination (Cannon-Bowers et al. 1993). Byrne (2001) suggested that task-related, i.e. shared knowledge about the task, and team-related mental models, i.e. shared knowledge about roles and responsibilities of team members, exist in parallel. Both have been found to have a positive impact on team adaptation in novel situations (Burke et al. 2006). Previous IS studies have applied shared mental models theory to show how agile development methods might influence collaboration within the team (Yu and Petter 2014) and how team members' personality influences the emergence of shared mental models (Yang et al. 2008).

Previous research suggested social interaction (Levesque et al. 2001) such as frequent communication, information sharing, participation, or negotiation (Klimoski and Mohammed 1994) to be the primary mechanism through which shared mental models develop. Second, spending time together to learn the dominant logic prevailing within a team strongly influences the development of a shared understanding. Furthermore, role differentiation plays a major role in the development of shared mental models. For teams with diverse roles, cross-training and carrying out other team members' duties can have a positive influence on the development of shared mental models (Marks et al. 2002). Finally, performance monitoring and self-correction was found to be positively related to higher mental model congruency (Rasker et al. 2000).

The usage of agile software development practices may increase the sharedness of the mental models of team members. Developers working in a pair with a programming partner socially interact over a longer period of time while discussing how to solve the assigned development task. Both developers frequently change their roles, i.e. one developer has the role of the observer who monitors the work results of the so-called navigator. During these programming sessions, the two developers learn from each other while spending time to solve an implementation task together. Pair programing can be considered as an extreme form of code review as pair programmers not only review the developed software after it has been finished, but constantly monitor the development process. As such, code review and pair programming may lead to similar outcomes.

When using automated testing, developers write test cases that they automatically execute to check if their software behaves as expected, i.e. if the software produces an expected output for a set of input conditions. Taking a cognition perspective, software development professionals form mental models that represent functional or non-functional requirements of a software system. These mental models are made explicit by writing test cases and then shared in the team's 'test suite'. Test suites contain all test cases which were developed over time by all software developers working on that software code line. Thus, developing and regularly executing test suites to check software for functional correctness will likely increase a team's shared mental model. Hence, we posit the following hypothesis:

H1a: The more intensively a team uses agile practices, the more shared the mental models of the team members.

Team Backup Behavior

Team backup behavior can take many forms such as helping, carrying out a task, or providing feedback to team members. The intention of backup behavior is to better achieve the team goals when potential failure is apparent (Porter et al. 2003). Marks et al. (2001) identify three means of providing backup as (a) providing feedback and coaching to improve performance, (b) assisting teammates in performing a task, and
(c) completing a task when overload is detected. Prior literature suggests mutual monitoring and team orientation among team members as key conditions for backup behavior to emerge within teams (Salas et al. 2005). Both mutual monitoring and team orientation may be particularly present in teams intensively using agile development practices.

Software developers primarily use pair programming and code reviews to discuss newly developed or modified code, i.e. provide feedback to each other. Thus, developers assist each other on how to implement a task before writing software code through developing a solution approach together or right after the code has been written. Thereby, developers can immediately discuss occurring mistakes to avoid them in the future. Thus, applying personal feedback practices implies that team members monitor each other in the process of developing software. Moreover, while constantly engaging in task-related interaction with other team members, the use of pair programming and code review will likely lead to increased levels of team orientation.

Automated test can be seen as a means to automate the feedback process. Previous expectations about the software's behavior are made explicit as an automated test to be used in the future or by other developers on the team. Therefore, automated tests can be considered as a form of feedback provision that would otherwise be given, at best, verbally when developers test the software manually. Many teams work with continuous integration servers that continuously and automatically run these test cases. When a single test fails, the entire team gets noticed and can judge whether the error could be solved by the developer who had caused it or if actual assistance is required. As such, running automated test cases provides a convenient way to implement mutual monitoring and to receive instant feedback on the current state of the software. Also, automated testing implies that own code is constantly integrated with the team's shared code-base, thus increasing the team orientation of individual team members. Therefore, we propose the following link for our research model:

H1b: The more intensively a team uses agile practices, the more backup its team members provide to each other.

Backup behavior is particularly important for software development teams because individual developers may not be aware of their own performance deficiencies (Salas et al. 2005). Software engineers develop a product that can be used in very different contexts and that has a high level of complexity encapsulated in different software layers. Therefore, it can be very complex to find one's own mistakes (bug fixing) and to help others fix bugs or finish up a task when needed. Therefore, team members have to recognize the need for help and effectively judge the trade-off between providing help or accomplishing their own task to effectively engage in backup behavior. Doing so requires a common understanding of a colleague's task, of the specific architecture relevant to this task as well as his or her engineering capabilities to make that decision. Only if team members have that understanding, they can provide backup (Dickinson and McIntyre 1997).

Backup behavior is a response to a genuine request for assistance and means that a developer only provides assistance to a colleague if help is actually required (Porter et al. 2003). This suggests that shared mental models are necessary antecedents for backup behavior because they build the foundation for decisions when help is actually needed (Salas et al. 2005). In line with previous assertions in literature, we posit the following hypothesis:

H2: The more shared the mental models of a team's members, the more backup its team members provide to each other.

Team Performance

In complex software systems, small bugs can have detrimental effects. Undetected software errors may lead to fatal consequences as, for instance, unpredictable system behavior that may require excessive efforts to be solved. Hence, the ability of a team to quickly detect and fix software defects is important for its overall performance. If major software defects are not detected, the quality of the software may gradually deteriorate over time. Consequently, the team would have to spend more time finding and fixing problems rather than developing new features. This process of finding and fixing is mostly a time-consuming and unpredictable task that can affect the team's ability to forecast its development schedule. As a consequence, stakeholders might perceive the team as unreliable when the delivered software quality does not meet its expectations or when the forecasted software features are not delivered on time. By providing feedback and assistance, developers can call their peers' attention to flawed software and help each other to fix problems when necessary. Teams that live a high level of backup as part of their work style, however, may shift their priorities and balance their workload between team members when facing such situations. Therefore, behavior may help the team to mitigate such problems and improve its team performance.

Previous research has shown that backup behavior is particularly important for complex tasks and in volatile work environments (Porter et al. 2003). In such situations, individual team members are most likely to face workloads that surpass their capacity. Hence, other team members need to take over in order to avoid a negative impact on team performance. To compensate, team members shift workloads within the team to adapt to novel situations. However, only when underutilized team members assist their overloaded colleagues, the team can adjust and perform at a level that could otherwise not be achieved. For non-complex tasks, however, backup can even be detrimental for the performance as misplaced backup behavior might lead to redundant rather than complementary teamwork. Following Salas et al. (2005), we propose that team backup behavior can be used to improve the performance of a group of individual developers.

H3a,b: In case of high task complexity, the more backup a team's members provide to each other, the better its team performance.

Method

Sample and Data Collection

To test our hypotheses, we conducted a field study at a large global enterprise software company between December 2013 and February 2014. The company has more than 400 Scrum teams in its headquarters, 300 of which had been trained on agile software development at the time of data collection (Schmidt et al. 2014). The teams had participated in a one week training program which was complemented by a three week coaching phase. Furthermore, most teams could draw from at least six months of experience with the agile development approach. Despite the company's investment into the training program, the teams were not obliged to use agile methods in their daily work. They could rather voluntarily decide if they wanted to apply the agile approach.

For our study, we drew a sample of 81 co-located development teams. In total, more than 490 software engineers, 79 Scrum Masters, and 65 project leaders agreed to participate in our study. Data was gathered on site using a standardized, paper-based questionnaire. Respondents' participation in the study was strictly voluntary and data collection followed a structured format. After a short introduction of the study to the team, the paper-based questionnaires were handed over to the team members to be completed individually. The researcher was still in the room to respond to any clarifying questions. Each data collection session took about 30 to 60 minutes. Afterwards, the answers were transcribed twice to ensure high quality of the dataset.

Table 1 provides an overview of the study sample. The average team size in the sample was 9.0. For 65 teams, our dataset is complete with answers from the Scrum master, at least four developers and at least one project leader. All teams were *stable*; more than 85% of all team members had worked for at least one year in their team. The developers are very *experienced*; the average developer had more than 10 years of work experience, one third of developers had more than 15 years of experience.

Team setup			**Technology†**		
Team Size	Less than 7 team members	15%	Programming Language	ABAP	78%
	7 – 9 team members	44%		Java	29%
	10 –12 team members	34%		Java Script	43%
	More than 12 team members	6%		C / C++ / C#	5%
Project Size	Less than 4 teams in project	58%	Software Type	Software Platform	23%
	4 – 6 teams in project	27%		Software Application	76%
	7 – 9 teams in project	10%		Mobile Apps	19%
	More than 9 teams in project	5%		Others	9%

†Some teams use multiple technologies.

Table 1: Sample Characteristics

We used role specific questionnaires to collect insights from the different perspectives. The Scrum master and the developers provided their team-internal perspective, the project leader provided his or her perspective on the performance of the teams. We asked 120 randomly selected teams to participate, 81 responded, i.e. we had a response rate of 68%. The average response per team was 7.1 (including developers and Scrum master), i.e. a response rate of 79%, with a minimum of five responses and a maximum 12 response.

Measures

We used existing measurement instruments from previous quantitative studies in IS or social psychology wherever possible. However, prior literature and our initial empirical insights suggested that no adequate instruments exist to capture some of the constructs of interest. In these cases, we developed new scales based on our knowledge and interactions with professionals at our research site. To avoid measurement biases, we collected data from different roles of the team including developers, Scrum masters, and project leaders. The list of items used in our study is shown in the Appendix.

Team Performance

We measured team performance using four items which we developed based on field interviews with project leaders and the organization's consultants for software development processes before conducting our study. The items are designed to cover different perspectives on teams' performance including projects leaders, managers, and peer teams' perspectives. We asked the project leaders to provide their assessment of the respective teams.

Use of Agile Practices

We asked developers to rate their personal adoption level of the studied agile development practices. In line with previous studies (Maruping et al. 2009),we asked the developers, for instance, to indicate how much of their coding time they programed with a programming partner (pair programming), how much of their code was reviewed by at least one other team member, or for how much of their newly developed code they wrote automated test cases. Face validity of the measurement items was pre-tested during interviews with senior software developers at the company and piloted in a quantitative pre-study to ensure construct validity. The specific questions are listed in the appendix of this paper. The developers rated their personal behavior on a 9-point scale that ranges from "0-10%" (coded 0.0) to "90-100%" (coded 0.9). A factor analysis with varimax rotation revealed a single factor that explains 75% of the variance of these items. Against this backdrop, we averaged the six questions to calculate an index with equal weights of agile practices for each developer in the study. The individual-level scores were averaged to the team level score.

Task Complexity

We used three items to measure the teams' task complexity that were adapted to our research context from Nidumolu (1995). The three items ask if preexisting knowledge and work procedures could be applied to solve the team task during the last six months. The Scrum master of the team indicated her or his assessment for the team on a 7-point agreement Likert scale. We added the answers for the three questions to a single index of task complexity.

Shared Mental Models

Measuring shared mental models has been discussed as a very challenging endeavor (DeChurch and Mesmer-Magnus 2010). Existing literature offers no consistent methodology, mostly because of the context-dependent nature of shared mental models (Mohammed et al. 2010). Previous approaches addressed accuracy and sharedness of team members' mental models. We are particularly interested in the sharedness of the mental models as it is at the core of the construct referring to the degree to which members' mental model are consistent or converge with one another (Mohammed et al. 2010). In line with previous literature, we interpret "sharing" in the sense of "having in common" rather than "dividing up" (Cannon-Bowers et al. 1993).

Different elicitation techniques have been used to capture shared mental models, ranging from paired comparison ratings, concept maps, card sorts, to qualitative techniques (Mohammed et al. 2010). These measurement approaches require a profound understanding of the mental models specific to the team context and are very time-consuming to deploy. Given that we study software professionals that work on very diverse software products and contexts, we had to find a context-insensitive measurement approach.

We were interested in the degree of sharedness of relevant knowledge areas for software development teams. We draw on previous literature (He et al. 2007) that found four specific areas of knowledge relevant to software development: knowledge on the application domain of the software, on the underlying technology, the development procedure, and on the overall project vision. Based on He et al.'s work, we derived six statements on the sharedness of knowledge between two developers. For these statements, the developers indicated their level of agreement regarding their work relationship to other team members on a 7-point Likert scale, e.g. "we have a similar understanding of our software architecture". All items were pretested and refined based on feedback from senior software developers to ensure face validity.

In our team-based study, we asked every participating team member to indicate her or his agreement with these items considering three members A, B, and C of his or her team (see Figure 2 for a screenshot of the used questionnaire). We followed a specific procedure to ensure the best possible coverage of bidirectional relations for each studied team. For every possible team size, we prepared a set of sign plates similar to the one depicted in Figure 2. All developers randomly drew such a sign and positioned it in front of them on the desk. As such, a number was assigned to every team member and every team member knew for which of his or colleagues A, B, and C he or she was asked to fill out the questionnaire. We designed the sign plates in such a way that the highest coverage of bidirectional relations in the team was collected. Due to this data collection approach, our dataset includes a team-specific graph of '3 x number of participating team members' bidirectional relations with the answers as weights on the edges for every item in the questionnaire. To aggregate to the team level, we averaged the weights of all edges in the graph for every item.

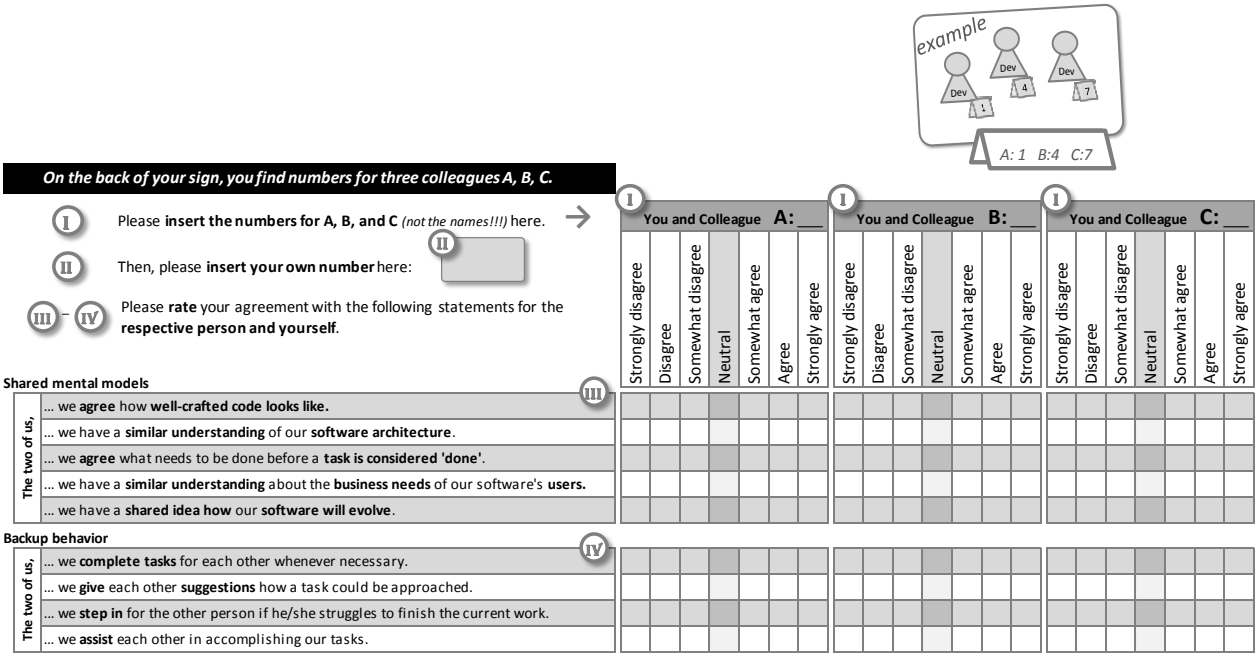

Figure 2: Questionnaire Extract for Measuring Shared Mental Models

Backup Behavior

We measured the backup behavior of the team following the same procedure as described for the shared mental models. The items were newly developed and based on previous work on backup behavior (Burke et al. 2006; Den Hartog et al. 2007; Salas et al. 2005). All answers were averaged to the team level score.

Controls

As most other team-based studies, we controlled for the size of the team. The Scrum master provided that number.

Construct Validity and Reliability

The measurement items of agile development practices (AGILE), shared mental models (SHARED), backup behavior (BACKUP), and team performance (PERFORMANCE) are designed as reflective measures. Our exploratory factor analysis of these variables (principal component analysis with varimax rotation) helped us to find four distinct measurements. The loadings on the respective items are on average .78 and between .58 and .95. Table 2 displays the descriptive statistics and the correlation matrix. The Cronbach alphas for all variables were higher than .86 showing adequate construct reliability. Task complexity was conceptualized as a formative construct as it captures three independent sources of complexity that a team may face; thus it was measured as an additive variable.

Table 2: Descriptive Statistics and Correlations of the Model Variables

Variable	Alpha	Mean	S.D.	Min	Max	1	2	3	4	5
1. AGILE	0.87	0.38	0.17	0.05	0.80					
2. SHARED	0.86	5.42	0.42	4.2	6.39	0.23*				
3. BACKUP	0.94	5.64	0.53	3.89	6.56	0.26**	0.40***			
4. PERFORMANCE	0.95	5.07	1.28	2.25	7	0.03	0.12	-0.07		
5. COMPLEX	-	9.47	3.98	3	21	-0.02	-0.20*	0.05	-0.17	

*** p<0.01, ** p<0.05, * p<0.1

Model Specification and Estimation Procedure

To test our hypotheses, we specified the following equations. We mean-centered backup behavior and task complexity for reasons of a better interpretability of the interaction effect (Aiken and West 1992). Parameters were estimated with ordinary least squares (OLS) regression.

$$\text{SHARED} = \alpha_0 + \alpha_1 \text{TeamSize} + \alpha_2 \text{AGILE} + \varepsilon_1 \quad (1)$$

$$\text{BACKUP} = \beta_0 + \beta_1 \text{TeamSize} + \beta_2 \text{AGILE} + \beta_3 \text{SHARED} + \varepsilon_2 \quad (2)$$

$$\text{PERFORMANCE} = \gamma_0 + \gamma_1 \text{TeamSize} + \gamma_2 \text{BACKUP} + \gamma_3 (\text{BACKUP x COMPLEX}) + \varepsilon_3 \quad (3)$$

The performance of the studied teams was evaluated by the team's project leader. As indicated in Table 1, most of the teams work in multi-team projects. Consequently, there are various teams from the same development project in our dataset. For these teams, the performance was simultaneously assessed by their common project leader. Therefore, we expect the error terms of his or her answers not be independent from each other. To take this characteristic into consideration, we clustered those datasets which contain answers from the same project leader when we estimate the second model (Cameron et al. 2011). This procedure is not required for estimating the parameter of the first model as all variables are indicated by team-specific respondents. We checked for multicollinearity. The variance inflation factors for the independent variables in all four models (Table 3 & 4) are smaller than .98; hence multicollinearity is not a concern in our analyses.

Results

The results for the parameters for the first two equations of our model (i.e. backup behavior and shared mental models as dependent variables) are displayed in Table 3. The results show support for our hypothesis H1a, H1b, and H2. The use of agile development practices of a software development team positively influences the sharedness of team members' mental models as well as how much backup behavior team members provide to each other (see Table 3). Column 3 of Table 3 shows that higher levels of team backup behavior are associated with a higher sharedness of team members' mental models (beta = 0.45, $p < 0.01$). In total, 21% of the variance of team backup behavior can be explained with our model.

Table 3: Parameter Estimates for Shared Mental Models and Backup Behavior

	(1) Shared Mental Models	(2) Backup Behavior	(3) Backup Behavior
Agile Practices Use	0.58**	0.75**	0.50
	(0.28)	(0.34)	(0.33)
Shared Mental Models			0.45***
			(0.13)
Team Size	-0.00	-0.04	-0.04
	(0.02)	(0.02)	(0.02)
Constant	5.25***	5.69***	3.36***
	(0.22)	(0.27)	(0.74)
Observations	81	81	81
R-squared	0.06	0.09	0.21
F test	2.31*	3.99**	6.79***

Standard errors in parentheses; *** p<0.01, ** p<0.05, * p<0.1

Next, we estimated the relationship between team backup behavior and team performance under varying levels of task complexity. Our analysis does not show any direct effect of backup behavior on team performance at the mean value of task complexity (see Table 4). However, there is a strong positive impact in case of high task complexity. Specifically, if task complexity is high the impact of backup behavior on team performance is positive and significant (beta = 0.19, $p < 0.01$).

Figure 3 displays this interaction effect at the mean and plus/minus one standard deviation level of task complexity with mean centered values of task complexity and backup behavior (Aiken and West 1992). As can be inferred from Figure 3, the prediction line is only positive at high levels of task complexity, whereas a negative effect can be observed in case of average or low task complexity.

Table 4: Results of the Team Performance Model

	(4) Team Performance
Backup Behavior	-0.16 (0.268)
Task Complexity	-0.06* (0.03)
Backup Behavior x Task Complexity	0.19*** (0.06)
Team Size	0.02 (0.07)
Constant	4.95*** (0.65)
Observations	62
R-squared	0.10
F test	3.57**

Standard errors in parentheses; *** p<0.01, ** p<0.05, * p<0.1
Backup behavior and Task complexity values are mean-centered in this model
The number of observations drops from 81 to 62 due to missing data.

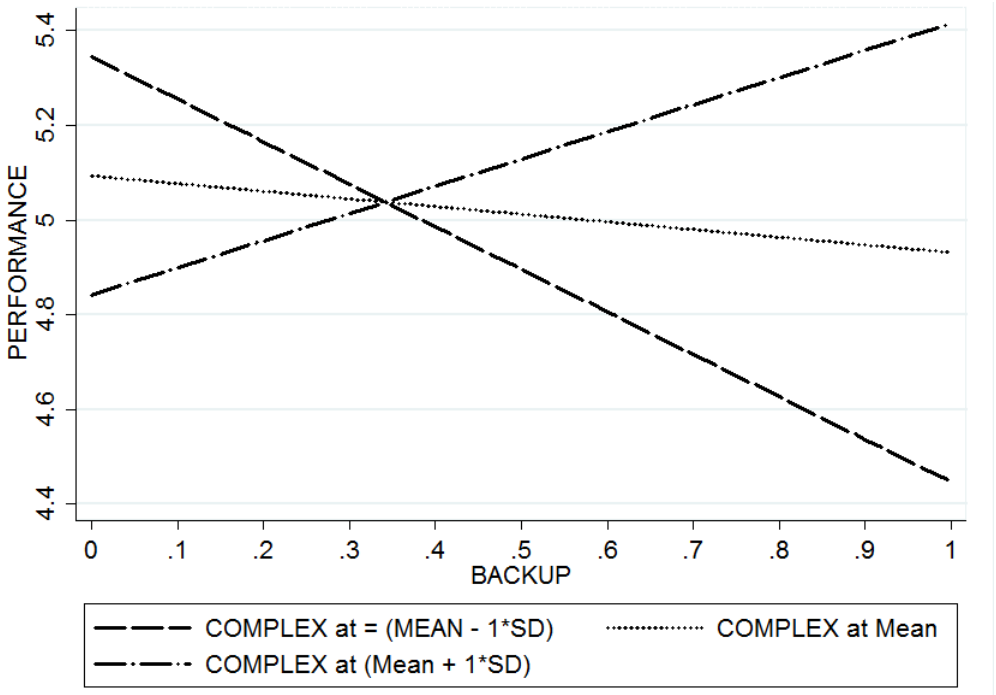

Figure 3: Interaction Effect of Task Complexity on the Predicted Level of Team Performance Dependent on Team Backup (Mean Centered Values)

Discussion

The goal of this study was to better understand the theoretical relationship between applying agile software practices and team performance. In particular, we aimed at moving beyond direct impacts on code quality and individual performance, but instead theorize and empirically examine the effect of applying agile practices on software development team performance. Our empirical study of 81 software development teams within a large enterprise software company shows that agile behavior of IS developers has an influence on software development team performance. More specifically, our results suggest that teams which apply agile quality assurance practices such as pair programming, code reviews, and automated testing show a higher degree of shared mental models within the team and are more likely to give feedback to each other, assist each other, or shift workloads (i.e., show more backup behavior). Backup behavior was in turn found to lead to increases in software development team performance when the team faces high task complexity.

Before discussing contributions and implications, we point to some limitations that should be considered when interpreting the results of our study. Although our study was conducted at a single company which serves to control for any influence of corporate culture, we call for studies in other high-technology or R&D environments for further generalizability. To the extent teams in our setting work on very different software products across different business units, we would expect the findings to hold in other knowledge work situations that are characterized by high clock-speed and innovation. Second, our goal in this study was to understand the effect of using agile practices on overall team performance, we call for further research to focus on specific aspects of performance such as innovation or cycle-time that may be more salient in some settings and which may require considering a different set of contextual or mediating factors for a more complete understanding (Kude et al. 2014b).

This study makes several contributions to existing literature on software development and team adaptability theory. Our study extends prior work on the performance effects of agile software development practices (Dingsøyr et al. 2012; Dybå and Dingsøyr 2008) beyond individual effects of applying the practices in laboratory settings or in the context of student projects. Also, much of the existing knowledge on agile software development is derived from anecdotal evidence or practitioner reports. Our study responds to the calls for theory-based empirical studies in the context of agile software development teams (Dingsøyr et al. 2012; Dybå and Dingsøyr 2008; Maruping et al. 2009). By drawing on theories from social psychology and by rigorously studying agile software development practices within teams, we create new insights to contribute to a better understanding of the effects of applying agile practices within teams on the way how teams work together and on team cognition (Davern et al. 2012).

Our results provide empirical support for the theorized relationships between agile software development practices within teams and team performance. Whereas previous literature mostly focused on direct effects on software quality or individual performance (Harter and Slaughter 2003; Subramanyam et al. 2012), our theoretical model that draws on team adaptability theory provides a strong explanation for effects of agile software development on a team level. In particular, we theoretically link agile behavior within teams to shared mental models and backup behavior as key emergent states in team adaptability theory. By doing so, our study helps reconcile the so far disconnected views on agility either as a behavior (in terms of applying agile methods and practices) or as a capability (in terms of being able to react to changes). Our results suggest that agile behavior and agile capabilities are two distinct but related concepts, and that agile behavior leads to team adaptability as an agile capability within teams.

The results of our study also contribute to prior literature on team adaptability theory by linking shared mental models and backup behavior—two key components of team adaptability—to the idiosyncrasies of software development and software development teams. First, our study shows that in the context of software development teams, the use of agile practices is an important antecedent of both shared mental models and backup behavior. This is important because it shows how specific software development practices increase team adaptability. Such agile practices may be applicable beyond the context of software development. Thus, our findings may have wider implications for team behaviors that increase team adaptability in other knowledge-intensive contexts.

Second, our study adds to team adaptability literature by linking it to task complexity as an IS-specific factor. In particular, our results generally confirm the importance of backup behavior as a determinant of team performance. It is important to note, however, that in the context of software development teams, the effect of backup behavior on team performance can either be positive or negative. For simple tasks, backup behavior has a negative impact, for complex tasks, it is positive (see Figure 3). This finding suggests that while it may be fruitful to borrow from team effectiveness research, we need to further analyze and understand its applicability in the IS domain. Future research should develop IS-specific theoretical models by focusing on other context factors such as the influence of team task diversity (component teams vs. feature teams), team member role diversity (cross-functional teams vs. teams with specific roles as architects, testers, user-interface specialists, etc.), project-specific contingency factors (intensity of customer interaction, inter-team task dependency in multi-team projects, etc.), or technical contingency factors (e.g. role of software architecture, use of software development specific collaboration tools, etc.).

Third, although our research context studies agile practices in the context of software development, our findings provide an opportunity for further theorizing and developing notions of ambidexterity that can be studied at multiple levels. For example, one could view agile practices as an instantiation of a more general construct of disciplined autonomy in a software development context, to the extent it reflects a voluntary application of standardized work process templates (Mintzberg 1993). In other words, although teams have discretion and autonomy to apply specific practices, they are disciplined by implicit behavioral norms of agile templates. In that sense, disciplined autonomy may help to strike a balance between flexibility and quality at the same time. More generally, while work process autonomy may foster flexibility to react to novel situation or to adapt to new technologies; compliance to standardized work processes may help to meet company-wide quality goals. Further theorizing along these lines can be particularly valuable for extending the nascent literature on ambidextrous strategies in other contexts (Birkinshaw and Gibson 2004; Duncan 1976; Miles et al. 1978; O'Reilly III and Tushman 2004; Raisch and Birkinshaw 2008). For example, IS researchers have pointed to the need for managing trade-offs related to knowledge management and firm performance by developing ambidextrous capabilities (Im and Rai 2008; Tafti et al. 2007). However, the extent to which such ambidexterity can be enabled by IT investments or by changing IT development practices remains to be studied. Understanding these systems level or team level enablers of ambidexterity can be a valuable contribution to prior literature which has so far suggested structural (Duncan 1976), temporal (Adler et al. 1999; Cao et al. 2013) and contextual notions of ambidexterity (Gibson and Birkinshaw 2004).

The results of our study also provide valuable insights for practitioners. In contrast to other fields of engineering, software engineering experiences more frequent changes and is characterized by higher industry dynamism and turbulence (Boehm 2006). Therefore, new development methods, new technologies, or new management approaches frequently emerge and this trend is likely to continue in the coming years as the global economy becomes more information-intensive. Not every new direction is likely to become mainstream and many of the new "fashions" might just recycle existing concepts or ideas. This poses a challenge for managers to understand if and when they should deploy new software development methods. Frequently, many organizations adopt a trial-and-error approach or simply follow the herd.

Our study suggests that by understanding the underlying concepts and mechanisms that drive software development methodologies, decision makers will be in a better position to predict the impact and effectiveness of the methodologies in new contexts. By pointing to the impact of agile practices on shared mental models and backup behavior, our study provides decision makers with important factors to consider when evaluating the merits of new software development practices. However, our results also show that backup behavior is not a silver bullet for software development teams, despite the proposed positive effect found in team effectiveness research. Instead, the positive implications of backup behavior on team performance may depend on various context factors, task complexity as one such important context factor. This should help practitioners to decide when to apply agile practices, as using these practices may only be valuable when task complexity is high, but not in the context of simpler tasks.

Appendix: Measurement Items

Variable	Items	Scale	Respondents	Source
Agile Practices Use AGILE	How much of your code do you develop with a programming partner? How much of your coding time do you work with a programming partner? How much of your new code is reviewed by at least one colleague? How much of your modified code is reviewed by at least one colleague? For how much of your new code do you write automated tests at all? How much of your new functionality is regularly tested with automated integration tests?	0-10% - 90-100%	Developers	Self-developed based on Maruping et al. (2009)
Shared Mental Models SHARED	The two of us ... ... we agree how well-crafted code looks like. ... we have a similar understanding of our software architecture. ... we agree what needs to be done before a task is considered 'done'. ... we have a similar understanding about the business needs of our software's users. ... we have a shared idea how our software will evolve.	7-point Likert	Developers	Self-developed based on He et al. (2007)
Backup Behavior BACKUP	The two of us ... ... we complete tasks for each other whenever necessary. ... we give each other suggestions how a task could be approached. ... we step in for the other person if he/she struggles to finish the current work. ... we assist each other in accomplishing our tasks.	7-point Likert	Developers	Self-developed based on Burke et al. (2006); McIntyre and Salas (1995)
Team Performance PERFORMANCE	When asked for a high performance team, other teams would reference this team. I consider this team a high performance team. Reports on the performance of this team are always favorable. Peer teams consider this team a great success.	7-point Likert	Project Leader	Self-developed based on interviews with 15 project leaders
Task Complexity COMPLEX	Concerning the last six months, the team faced task ... ... for which there was a clearly known way how to solve them. ... for which the team's preexisting knowledge was of great help to solve them. ... for which the team's preexisting work procedures and practices could be relied upon to solve them.	7-point Likert	Scrum Master	Nidumolu (1995)

References

Abrahamsson, P., Conboy, K., and Wang, X. 2009. "Lots Done, More to Do: The Current State of Agile Systems Development Research," *European Journal Of Information Systems* (18:4), pp. 281-284.

Adler, P.S., Goldoftas, B., and Levine, D.I. 1999. "Flexibility Versus Efficiency? A Case Study of Model Changeovers in the Toyota Production Systems," *Organization Science* (10:1), pp. 43-68.

Aiken, L.S., and West, S.G. 1992. *Multiple Regression: Testing and Interpreting Interactions*. Newbury Park, CA, USA: Sage Publications, Inc.

Baskerville, R., Ramesh, B., Levine, L., and Pries-Heje, J. 2006. "High-Speed Software Development Practices: What Works, What Doesn't," *IT Professional* (8:4), pp. 29-36.

Beck, K. 2000. *Extreme Programming Explained: Embrace Change*. Reading, MA, USA: Addison-Wesley.

Birkinshaw, J., and Gibson, C. 2004. "Building Ambidexterity Into an Organization," *MIT Sloan Management Review* (45:Summer), pp. 47-55.

Boehm, B. 2006. "A View of 20th and 21st Century Software Engineering," *International Conference On Software Engineering*, Shanghai, China.

Burke, C.S., Stagl, K.C., Salas, E., Pierce, L., and Kendall, D. 2006. "Understanding Team Adaptation: A Conceptual Analysis and Model," *Journal Of Applied Psychology* (91:6), pp. 1189-1207.

Byrne, B.M. 2001. *Structural Equation Modeling with AMOS : Basic Concepts, Applications, and Programming*. Mahwah, N.J.: Lawrence Erlbaum Associates.

Cameron, A.C., Gelbach, J.B., and Miller, D.L. 2011. "Robust Inference with Multiway Clustering," *Journal Of Business & Economic Statistics* (29:2), pp. 238-249.

Cannon-Bowers, J.A., Salas, E., and Converse, S. 1993. "Shared Mental Models in Expert Team Decision Making," in *Individual and Group Decision Making: Current Issues,* J.N. Castellan (ed.). Castellan, NJ, USA: Lawrence Erlbaum Associates, pp. 221-246.

Cao, L., Mohan, K., Ramesh, B., and Sarkar, S. 2013. "Evolution of Governance: Achieving Ambidexterity in IT Outsourcing," *Journal of Management Information Systems* (30:3), pp. 115-140.

Cockburn, A. 2001. *Agile Software Development: Software through People*. Amsterdam , Netherlands: Addison-Wesley Longman.

Davern, M., Shaft, T., and Te'eni, D. 2012. "Cognition Matters: Enduring Questions in Cognitive IS Research," *Journal Of The Association For Information Systems* (13:4), pp. 273-314.

DeChurch, L.A., and Mesmer-Magnus, J.R. 2010. "Measuring Shared Team Mental Models: A Meta-Analysis," *Group Dynamics: Theory, Research, And Practice* (14:1), pp. 1-14.

Den Hartog, D.N., De Hoogh, A.H., and Keegan, A.E. 2007. "The Interactive Effects of Belongingness and Charisma on Helping and Compliance," *Journal Of Applied Psychology* (92:4), pp. 1131-1139.

Dickinson, T.L., and McIntyre, R.M. 1997. "A Conceptual Framework for Teamwork Measurement," in *Team Performance Assessment and Measurement,* M.T. Barnnick, E. Salas and C. Prince (eds.). Mahwah, New Jersey: Lawrence Erlbaum Associates, Inc., pp. 19-43.

Dingsøyr, T., Nerur, S., Balijepally, V., and Moe, N.B. 2012. "A Decade of Agile Methodologies: Towards Explaining Agile Software Development," *Journal Of Systems And Software* (85:6), pp. 1213–1221.

Duncan, R.B. 1976. "The Ambidextrous Organization: Designing Dual Structures for Innovation," in *The Management of Organizational Design,* R. Kilman and L. Pondy (eds.). New York: North Holland, pp. 167-188.

Dybå, T., and Dingsøyr, T. 2008. "Empirical Studies of Agile Software Development: A Systematic Review," *Information And Software Technology* (50:9), pp. 833 - 859.

Faraj, S., and Sproull, L. 2000. "Coordinating Expertise in Software Development Teams," *Management Science* (46:12), pp. 1554-1568.

Gibson, C., and Birkinshaw, J. 2004. "The Antecedents, Consequences, and Mediating Role of Organizational Ambidexterity," *Academy of Management Journal* (47:2), pp. 209-226.

Harter, D.E., and Slaughter, S.A. 2003. "Quality Improvement and Infrastructure Activity Costs in Software Development: A Longitudinal Analysis," *Management Science* (49:6), pp. 784-800.

He, J., Butler, B.S., and King, W.R. 2007. "Team Cognition: Development and Evolution in Software Project Teams," *Journal Of Management Information Systems* (24:2), pp. 261-292.

Henderson, J.C., and Lee, S. 1992. "Managing I/S Design Teams: A Control Theories Perspective," *Management Science* (38:6), pp. 757-777.

Highsmith, J., and Cockburn, A. 2001. "Agile Software Development: The Business of Innovation," *Computer* (34:9), pp. 120-127.

Hoegl, M., and Gemuenden, H.G. 2001. "Teamwork Quality and the Success of Innovative Projects: A Theoretical Concept and Empirical Evidence," *Organization Science* (12:4), pp. 435-449.

Im, G., and Rai, A. 2008. "Knowledge Sharing Ambidexterity in Long-Term Interorganizational Relationships," *Management Science* (54:7), pp. 1281-1296.

Johnson-Laird, P.N. 1980. "Mental Models in Cognitive Science," *Cognitive Science* (4:1), pp. 71-115.

Klimoski, R., and Mohammed, S. 1994. "Team Mental Model: Construct or Metaphor?," *Journal Of Management* (20:2), pp. 403-437.

Kude, T., Bick, S., Schmidt, C.T., and Heinzl, A. 2014a. "Adaptation Pattern in Agile Information Systems Development Teams," *European Conference On Information Systems*, Tel Aviv, Israel.

Kude, T., Schmidt, C.T., Mithas, S., and Heinzl, A. 2014b. "Disciplined Autonomy and Innovation Outcomes: The Role of Team Efficacy and Task Volatility." University of Mannheim, Working Paper.

Lee, G., and Xia, W. 2005. "The Ability of Information Systems Development Project Teams to Respond to Business and Technology Changes: A Study of Flexibility Measures," *European Journal Of Information Systems* (14:1), pp. 75-92.

Lee, G., and Xia, W. 2010. "Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility," *Management Information Systems Quarterly* (34:1), pp. 87-114.

Levesque, L.L., Wilson, J.M., and Wholey, D.R. 2001. "Cognitive Divergence and Shared Mental Models in Software Development Project Teams," *Journal Of Organizational Behavior* (22:2), pp. 135-144.

Lindstrom, L., and Jeffries, R. 2004. "Extreme Programming and Agile Software Development Methodologies," *Information Systems Management* (21:3), pp. 41-52.

MacCormack, A., and Verganti, R. 2003. "Managing the Sources of Uncertainty: Matching Process and Context in Software Development," *Journal Of Product Innovation Management* (20:3), pp. 217-232.

MacCormack, A., Verganti, R., and Iansiti, M. 2001. "Developing Products on 'Internet Time': The Anatomy of a Flexible Development Process," *Management Science* (47:1), pp. 133-150.

Marks, M.A., Mathieu, J.E., and Zaccaro, S.J. 2001. "A Temporally Based Framework and Taxonomy of Team Processes," *Academy Of Management Review* (26:3), pp. 356-376.

Marks, M.A., Sabella, M.J., Burke, C.S., and Zaccaro, S.J. 2002. "The Impact of Cross-Training on Team Effectiveness," *Journal Of Applied Psychology* (87:1), pp. 3-13.

Maruping, L.M., Venkatesh, V., and Agarwal, R. 2009. "A Control Theory Perspective on Agile Methodology Use and Changing User Requirements," *Information Systems Research* (20:3), pp. 377-399.

Mathieu, J.E., Maynard, M.T., Rapp, T., and Gilson, L. 2008. "Team Effectiveness 1997-2007: A Review of Recent Advancements and a Glimpse into the Future," *Journal Of Management* (34:3), pp. 410-476.

McAvoy, J., and Butler, T. 2009. "The Role of Project Management in Ineffective Decision Making within Agile Software Development Projects," *European Journal Of Information Systems* (18:4), pp. 372-383.

McIntyre, R.M., and Salas, E. 1995. "Measuring and Managing for Team Performance: Emerging Principles from Complex Environments," in *Team Effectiveness and Decision Making in Organizations,* R.A. Guzzo and E. Salas (eds.). Jossey-Bass, pp. 9-45.

Miles, R., Snow, C.C., Meyer, A.D., and Coleman, H.J. 1978. "Organizational Strategy, Structure, and Process," *Academy of Management Review* (3:3), pp. 546-562.

Mintzberg, H. 1980. "Structure in 5's: A Synthesis of the Research on Organization Design," *Management Science* (26:3), pp. 322-341.

Mintzberg, H. 1993. *Structure in Fives: Designing Effective Organizations*. San Francisco, CA: USA Prentice-Hall, Inc.

Mohammed, S., and Dumville, B.C. 2001. "Team Mental Models in a Team Knowledge Framework: Expanding Theory and Measurement across Disciplinary Boundaries," *Journal Of Organizational Behavior* (22:2), pp. 89-106.

Mohammed, S., Ferzandi, L., and Hamilton, K. 2010. "Metaphor No More: A 15-Year Review of the Team Mental Model Construct," *Journal Of Management* (36:4), pp. 876-910.
Nidumolu, S. 1995. "The Effect of Coordination and Uncertainty on Software Project Performance: Residual Performance Risk as an Intervening Variable," *Information Systems Research* (6:3), pp. 191-219.
O'Reilly III, C.A., and Tushman, M.L. 2004. "The Ambidextrous Organization," *Harvard Business Review* (82:4), pp. 74-81.
Porter, C.O., Hollenbeck, J.R., Ilgen, D.R., Ellis, A.P.J., West, B.J., and Moon, H. 2003. "Backing up Behaviors in Teams: The Role of Personality and Legitimacy of Need," *Journal of Applied Psychology* (88:3), pp. 391-403.
Raisch, S., and Birkinshaw, J. 2008. "Organizational Ambidexterity: Antecedents, Outcomes and Moderators," *Journal of Management* (34:3), pp. 375-409.
Ramesh, B., Mohan, K., and Cao, L. 2012. "Ambidexterity in Agile Distributed Development: An Empirical Investigation," *Information Systems Research* (23:2), pp. 323-339.
Rasker, P.C., Post, W., and Schraagen, J. 2000. "Effects of Two Types of Intra-Team Feedback on Developing a Shared Mental Model in Command & Control Teams," *Ergonomics* (43:8), pp. 1167-1189.
Rosen, M.A., Bedwell, W.L., Wildman, J.L., Fritzsche, B.A., Salas, E., and Burke, C.S. 2011. "Managing Adaptive Performance in Teams: Guiding Principles and Behavioral Markers for Measurement," *Human Resource Management Review* (21:2), pp. 107-122.
Salas, E., Sims, D.E., and Burke, C.S. 2005. "Is There a 'Big Five' in Teamwork?," *Small Group Research* (36:5), pp. 555-599.
Sarker, S., and Sarker, S. 2009. "Exploring Agility in Distributed Information Systems Development Teams: An Interpretive Study in an Offshoring Context," *Information Systems Research* (20:3), pp. 440-461.
Schmidt, C.T., Srinivasa, G., and Heymann, J. 2014. "Empirical Insights into the Perceived Benefits of Agile Software Engineering Practices: A Case Study from SAP," *International Conference On Software Engineering*, Hyderabad, India.
Schmidt, R., Lyytinen, K., Keil, M., and Cule, P. 2001. "Identifying Software Project Risks: An International Delphi Study," *Journal Of Management Information Systems* (17:4), pp. 5-36.
Schwaber, K., and Beedle, M. 2002. *Agile Software Development with Scrum*. Upper Saddle River, NJ, USA: Prentice Hall.
Schwaber, K., and Sutherland, J. 2011. "The Scrum Guide."
Strode, D.E., Huff, S.L., Hope, B., and Link, S. 2012. "Coordination in Co-Located Agile Software Development Projects," *Journal Of Systems And Software* (85:6), pp. 1222-1238.
Subramanyam, R., Ramasubbu, N., and Krishnan, M. 2012. "In Search of Efficient Flexibility: Effects of Software Component Granularity on Development Effort, Defects, and Customization Effort," *Information Systems Research* (23:3), pp. 787-803.
Tafti, A., Mithas, S., and Krishnan, M.S. 2007. "Information Technology and the Autonomy-Control Duality: Toward a Theory," *Information Technology and Management* (8:2), pp. 147-166.
VersionOne. 2012. "7th Annual State of Agile Development Survey" VersionOne Inc., http://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf.
Vidgen, R., and Wang, X. 2009. "Coevolving Systems and the Organization of Agile Software Development," *Information Systems Research* (20:3), pp. 355-376.
West, D., Grant, T., Gerush, M., and D'silva, D. 2010. "Agile Development: Mainstream Adoption Has Changed Agility", http://www.forrester.com/Agile+Development+Mainstream+Adoption+Has+Changed+Agility/fulltext/-/E-RES56100?objectid=RES56100.
Yang, H.-D., Kang, H.-R., and Mason, R.M. 2008. "An Exploratory Study on Meta Skills in Software Development Teams: Antecedent Cooperation Skills and Personality for Shared Mental Models," *European Journal Of Information Systems* (17:1), pp. 47-61.
Yu, X., and Petter, S. 2014. "Understanding Agile Software Development Practices Using Shared Mental Models Theory," *Information And Software Technology* (56:8), pp. 911-921.