How agile software development methods reduce work exhaustion: Insights on role perceptions and organizational skills

Viswanath Venkatesh1 | James Y. L. Thong2 | Frank K. Y. Chan3 | Hartmut Hoehle4 | Kai Spohrer5

1Department of Information Systems, University of Arkansas, Fayetteville, Arkansas
2Department of Information Systems, Business Statistics and Operations Management, The Hong Kong University of Science and Technology, Kowloon, Hong Kong
3Department of Information Systems, Decision Sciences and Statistics, ESSEC Business School, Cergy-Pontoise, France
4Department of Enterprise Systems, University of Mannheim, Mannheim, Germany
5Department of General Management and Information Systems, University of Mannheim, Mannheim, Germany

Abstract
Agile methods are widely used in the software industry as a way to more rapidly develop and deliver new software. They define iterative work processes, advocate self-organization and openness for change, and prescribe how software developers interact with each other and external stakeholders. Despite their popularity, it is unclear how agile methods influence work exhaustion in software developers and how developer skills play into this effect. On the one hand, agile methods may reduce software developers’ work exhaustion by levelling out their workload across the entire duration of a project. On the other hand, agile methods exert a high level of pressure on software developers to continuously deliver working software, create many intensive social interactions, and to frequently adapt to changes. In light of these effects, prior research could not explain why some software developers become less exhausted from using agile methods, whereas others perceive the exact opposite. Based on the job demand-control model, we develop a theoretical model connecting agile method use to individual developer skills and to two established determinants of employee exhaustion: role conflict and role ambiguity. We tested our research model in a field study among 1894 software developers in 217 project teams that used agile methods. The random coefficient modelling results show that agile method use facilitates the achievement of clear and unambiguous role perceptions.
and thereby reduces work exhaustion in developers, particularly if developers possess the organizational skills to effectively interact with others in their organization. We highlight implications for theory on the individual-level effects of software development methods and provide practical insights for software companies.

**KEYWORDS**
agile systems development, developer skills, organizational skills, role ambiguity, role conflict, work exhaustion

1 | INTRODUCTION

With a global spending volume of nearly US$400 billion in 2018 (Costello & Omale, 2019), the enterprise software industry constitutes a significant segment of the global economy that has become increasingly fast paced in recent years (Fitzgerald & Stol, 2017; Gallagher, Kaiser, Simon, Beath, & Goles, 2010; Sarker & Sarker, 2009). For example, rapid technological developments, fast changing market dynamics, a growing variety of data sources, and increasing needs for enterprise software to span multiple domains of business force software companies to adapt more quickly to altering conditions, to continuously redefine development processes and roles, and to facilitate the ongoing acquisition of new skills in their workforce (Akter, Wamba, Gunasekaran, Dubey, & Childe, 2016; Davis, Niederman, Greiner, Wynn, & York, 2006; Fosso Wamba et al., 2017; Ke & Zhang, 2010; Weerakkody, Irani, Kapoor, Sivarajah, & Dwivedi, 2017). These rapid changes place great demands on software companies and especially on their software developers who consequently suffer from increased stress and work exhaustion (Thong & Yap, 2000; Venkatesh, Rai, & Maruping, 2018; Watson, Boudreau, York, Greiner, & Wynn, 2008; Windeler, Maruping, & Venkatesh, 2017). Increased work exhaustion in software developers is highly problematic because it accounts for more than 30% of all technical development errors (Furuyama, Arai, & Iio, 1994; Furuyama, Arai, & Iio, 1997), causes substantive maintenance costs, impacts development teams’ performance, and can derail entire projects (Venkatesh et al., 2018; Windeler et al., 2017). In addition to negative effects on their performance, high work exhaustion negatively impacts software developers’ health and makes them more likely to quit their jobs (Moore, 2000; Ply, Moore, Williams, & Thatcher, 2012). In order to achieve high productivity and long-term retention of their developers, it is therefore critical for software companies to understand which work conditions and skills help their developers to alleviate work exhaustion during software development.

The use of software development methods is one important factor that can reduce or increase software developers’ stress and work exhaustion (Drury, Conboy, & Power, 2012; Furuyama et al., 1997). As such, software development methods affect developers’ daily work by defining roles and interactions, partitioning projects into smaller steps, and laying the foundations for team work and the division of labour (Fitzgerald, 1998). Particularly, agile development methods, such as extreme programming (XP) and Scrum, have become widespread in industry during recent years because they facilitate adaptation to changing environments and customer needs (Lee & Xia, 2010; Maruping et al., 2009a, 2009b; Yu & Petter, 2014). These agile methods entail a people-centric, iterative development process that is based on self-organizing teams of software developers who continuously solicit and evaluate feedback on their progress in developing a desirable software product (Hoda & Murugesan, 2016). Although experience in agile methods has become an important occupational qualification (Cram & Newell, 2016; Dingsøyr, Nerur, Balijepally, & Moe, 2012), it is largely unclear how the use of agile methods actually affects individual developers in their jobs.
Industry reports are particularly inconsistent regarding the effects of agile methods on the work exhaustion of software developers. On the one hand, some developers report that the use of agile methods helps them to achieve a sustainable, less exhausting working pace (Laanti, 2013). On the other hand, there are developers who report increased exhaustion from the use of agile methods due to the "constant strive for improvement, the relentless drive for feedback, the subsequent changes in direction and the incessant social interactions" (Balbes, 2017). Software companies, therefore, have contradictory guidance regarding the questions how the use of agile methods affects their core resource, namely, software developers and why some developers become more exhausted than others when using agile methods.

Prior research has recently started to examine the effects of agile software development on individual developers, eg, regarding job satisfaction, job autonomy, and work exhaustion (Hoda, Salleh, Grundy, & Tee, 2017; Tripp et al., 2016; Tuomivaara, Lindholm, & Käänilä, 2017). With regard to software developers’ work exhaustion, extant research focused primarily on whether agile methods for project management indeed help to balance the workload of development projects across the entire project duration, thereby preventing exhausting phases with extreme workload prior to milestones or toward the end of a project (Tuomivaara et al., 2017). Although valuable, such a project management perspective does not account for differences between individual developers and does not explain why some developers may become more exhausted than others when using agile methods. A possible reason for such differences could be that software developers can be more or less skilled in coping with the demands of specific software development methods. For example, agile methods demand frequent and intensive social interactions and tremendous informal communication (Hummel, Rosenkranz, & Holten, 2013) that may be more exhausting for those developers who lack the skills to effectively engage in such activities. Yet, research currently provides little insight into how agile methods influence work exhaustion in software developers or what role individual developer skills play in this regard. This is a theoretically and practically relevant issue because psychological theories suggest that work exhaustion often results from a mismatch between employees' capabilities and the demands of their work environment (Häusser, Mojzisch, Niesel, & Schulz-Hardt, 2010; Jung, Schneider, & Valacich, 2010; Karasek, 1979; Karasek et al., 1998). Failure to understand the role of individual developer skills in the relationship of agile methods and software developers' work exhaustion consequently limits our ability to achieve a more complete understanding of how to manage and staff contemporary software development projects. Against this backdrop, our research aims to answer two overarching research questions:

1. What is the impact of agile method use on work exhaustion in individual developers?
2. How do individual developer skills influence the relationship of agile method use and developers' work exhaustion?

To address these questions, we rely on and extend an established theory. First, we draw on the job demand-control model (JDCM; Karasek, 1979) to understand the influence of agile methods on software developers' work exhaustion. Second, we draw on the taxonomy of information systems (IS) skills by Nelson (1991) to understand the role of software developer skills in this relationship. More specifically, we focus on (a) role conflict and role ambiguity to theorize why agile method use influences work exhaustion in developers (Häusser et al., 2010; Karasek et al., 1998; Van der Doef & Maes, 1999) and (b) organizational skills of developers to theorize how the skills of individual developers alter the effect of agile method use on work exhaustion (Gallagher et al., 2010; Lee, Trauth, & Farwell, 1995; Nelson, 1991; Wade & Parent, 2001). To date, systematic research that aids in understanding the effects of agile methods on individual developers is scarce (Tripp et al., 2016; Tuomivaara et al., 2017). Scholars have therefore called for more theory-driven research to examine the consequences of agile method use in quantitative studies (Chan & Thong, 2009; Chuang, Luor, & Lu, 2014; Dingsøyr et al., 2012; Mangalaraj, Nerur, Mahapatra, & Price, 2014). We respond to this call with a field study on the influence of agile method use on software developers’ work exhaustion. To this end, we test our model in a field study among 1894 developers in 217 project teams.
2 | THEORETICAL BACKGROUND

2.1 | Agile software development methods and work exhaustion

The rise of agile software development has resulted in a variety of agile methods, such as XP (Beck, 2000), Scrum (Schwaber & Beedle, 2002), and Lean Programming (Poppendieck & Poppendieck, 2003). Although the various agile methods differ in the practices they comprise, all adhere to the same set of underlying values that they instantiate in different ways (Fowler & Highsmith, 2001). They advocate an iterative, people-centric approach to software development that facilitates quick adaptation to changing business requirements, markets, and technologies and directs development efforts toward perceived customer value (Conboy, 2009; Ramesh, Cao, Kim, Mohan, & James, 2017). Agile methods, therefore, generally follow an incremental process that builds on self-organizing, empowered teams of developers who frequently interact with customers to solicit feedback on their development outcomes and adapt their targets and development activities accordingly (Hoda & Murugesan, 2016). There is persuasive evidence that agile methods benefit team performance and project outcomes (Hoda et al., 2017). As such, agile method use increases software quality and project performance by making development teams more responsive to uncertainty and requirements volatility (Lee & Xia, 2010; Maruping et al., 2009a, 2009b) and by letting them come to a more accurate and shared understanding of customer needs and development obstacles (Ghobadi & Mathiassen, 2016, 2017; Yu & Petter, 2014).

Self-organization and continuous adaptation, which are germane to agile methods, necessitate that developers communicate frequently with their colleagues and customers, force them to confront potentially controversial opinions, and require them to coordinate their actions informally based on mutual adjustment rather than on explicit role descriptions or detailed process models (Conboy, 2009; Ghobadi & Mathiassen, 2017; Hoda & Murugesan, 2016; Matook & Maruping, 2014; Ramesh, Mohan, & Cao, 2012; Yu & Petter, 2014). In addition, agile methods mandate that developers continuously deliver working software and increase the value of their software product in each step (Drury et al., 2012). They enforce that developers identify issues with extant solutions, communicate with colleagues to draw on their team’s combined expertise, and show extensive backup behaviour (Kudaravalli, Faraj, & Johnson, 2017). Although all this makes agile methods effective for improving outcomes of development projects (Lee & Xia, 2010; Maruping et al., 2009a, 2009b), it is not yet fully understood how these specific demands of agile methods affect the individual developers who need to fulfill them (Tripp et al., 2016). On the one hand, developers may benefit from increased job control through self-organization of their teams. On the other hand, the specific job demands entailed in agile methods may exhaust developers if they lack the skills to cope with the demands. In line with these contradictory views, industry reports show that some developers feel more exhausted from using agile methods whereas others see agile methods as a safeguard against work exhaustion (Balbes, 2017; Laanti, 2013).

Prior research on software developers’ work exhaustion and agile methods has primarily taken a project management perspective. It examined whether agile methods indeed help balance the workload of development projects across the entire project duration, a long-standing promise of agile methodology (Beck, 2000; Fowler & Highsmith, 2001). This line of work found support for a reduction of workload peaks and a subsequent reduction in developers’ work exhaustion (Tuomivaara et al., 2017). However, research has remained silent about the effects of agile methods on developers’ work exhaustion that go beyond balancing workload. As such, there are also developers who do not cope well with the demands of agile development, who feel that agile methods increase work stress and feel forced into exhausting activities by them (Laanti, 2013). Some developers have actually been found to struggle with their jobs when using agile methods and to become emotionally drained (Balbes, 2017; Ghobadi & Mathiassen, 2017). These ambivalent observations suggest that there are differences at an individual level that influence how exhausting agile methods are for software developers. For example, agile methods emphasize self-organization and adaptation, and thus require frequent and intensive informal communication as well as social interaction with team members, customers, and other stakeholders (Hummel et al., 2013; Sarker &
Sarker, 2009). However, not all developers are equally skilled in communicating and interacting informally (Wade & Parent, 2001). The effects of agile methods may consequently be related to individual software developers’ skills and how well they fit with the demands of agile methods.

To date, rigorous research on the effects of agile method use has not yet addressed these issues. It has, in fact, largely overlooked differences between individual developers in coping with their job demands (Tripp et al., 2016). Yet, this topic is important because frequent failure to cope with the challenges of one’s job, here created by agile methods, can have long-term effects such as burnout (Häusser et al., 2010; Moore, 2000). We turn to an established theory, ie, the JDCM, to help understand the effects of job demands and potential controls on employees' work exhaustion.

### 2.2 A job demand-control perspective on agile methods

To better understand the impact of agile methods on software developers’ work exhaustion, we base our research on one of the most impactful theories on organizational psychology, namely, the JDCM (Häusser et al., 2010; Karasek, 1979; Van der Doef & Maes, 1999). The JDCM holds that there are two central factors influencing employees’ appraisal of their job and consequent psychological and physiological reactions: job demands—ie, activities and outcomes necessary to complete one’s organizational tasks—and job control—ie, ability to direct or manage one’s own work activities. The JDCM posits that employees working in a high-strain context, ie, when job demands are high and job control is low, are prone to suffering from negative impacts such as psychological stress, feelings of helplessness, and exhaustion (Van der Doef & Maes, 1999). In contrast, such adverse effects are less likely in low-strain contexts, ie, when job demands are low and job control is high (Häusser et al., 2010).

In line with JDCM, software engineers have been found to experience high work exhaustion and low satisfaction when facing very high job demands (Schreurs & Taris, 1998). In addition to causing work exhaustion in software developers, overly high job demands impair team processes and harm work results if not managed adequately (Maruping, Venkatesh, Thatcher, & Patel, 2015; Windeler et al., 2017). In laymen's terms, “work exhaustion” is often used interchangeably with the term “job burnout” (Moore, 2000). Although there are more detailed conceptualizations of exhaustion and other physiological and psychological facets of burnout in prior research, work exhaustion is widely seen as an equivalent of burnout (Häusser et al., 2010). Consistent with prior IS literature (Moore, 2000), we therefore define work exhaustion as the depletion of mental resources in the workplace.

Software projects can put particularly high strain on software engineers by affecting their role perceptions, specifically role conflict and role ambiguity (Windeler et al., 2017). Role ambiguity refers to the degree to which a software developer’s role expectations are unclear (Rizzo, House, & Lirtzman, 1970). Developers who work in a group of interdependent members but have no guidance regarding their individual responsibilities may not know which tasks to complete in which order, how to communicate with external stakeholders, and how to evaluate their work (Häusser et al., 2010; Swanson & Power, 2001; Windeler et al., 2017). Although self-organization inherent to agile methods may support development teams in clarifying role expectations, prior work suggests that individual developers sometimes suffer from ambiguous role perceptions even when using agile methods. For example, Moe, Dingsøyr, and Dybå (2010) state that Scrum leaves many important development activities unspecified and puts high demands on development team members to interactively clarify what needs to be done and by whom. Hoda and Murugesan (2016) emphasize that some developers perceive it as particularly challenging to clarify, select, and self-assign their development tasks. Yet, elaborations of the individual-level factors that drive these challenges and corresponding empirical tests have been left to future research (Hoda & Murugesan, 2016). Although developers can work together effectively with either overlapping or nonoverlapping areas of expertise (Kudaravalli et al., 2017), it is important for each individual to know whether a task, be it self-assigned or assigned top-down, falls into one’s area of expertise and responsibility or not (Faraj & Sproull, 2000). Otherwise, uncertainty about individual responsibilities translates to role ambiguity.
Role conflict refers to the degree to which behaviours expected from an individual are inconsistent (Rizzo et al., 1970). For example, managers often need to balance software quality, functional scope, and resource consumption in development projects. They face role conflict if they are equally constrained by customer demands for quality and scope as by internal demands for minimized resource investments. Conflicting and ambiguous role perceptions are theoretically and empirically established causes of work exhaustion. They have frequently been examined in the JDCM literature (Häusser et al., 2010; Swanson & Power, 2001; Wong, DeSanctis, & Staudenmayer, 2007) and are linked to dissatisfied, work overload, and turnover intentions (Joseph, Ng, Koh, & Ang, 2007; Moore, 2000; Ply et al., 2012). Yet, it is unclear how they relate to the use of software development methods or to the skills of individual developers.

To understand individual differences in how exhausting the use of agile methods is for developers, we further examine individual developer skills because the evaluation of job demands and job control is generally influenced by skills (Karasek, 1979). In this regard, organizational skills refer to an individual’s knowledge and capability to effectively manage interpersonal communication, interpersonal behaviour, and group dynamics as a member of a work group (Nelson, 1991). In the past, scholars found that the work of software developers depended much more on technical skills (eg, skills in programming and the use of software packages) than on organizational skills (Gallagher et al., 2010; Lee et al., 1995; Nelson, 1991; Wade & Parent, 2001). Given the emphasis on people, interactions, and self-organization in agile methods (Hoda & Murugesan, 2016), this assumption may no longer hold in contemporary, agile software development.

3 | THEORY DEVELOPMENT

In light of prior work and the gaps discussed, we theorize that a larger extent of agile method use helps developers to better understand their roles and responsibilities, thereby reducing role ambiguity and role conflict and that organizational skills amplify this effect. More specifically, we theorize that organizational skills facilitate communication and mutual adjustment among developers that are particularly important during the use of agile methods because these methods require developers to engage in frequent personal interactions, confront controversial opinions of colleagues and customers, and informally align perspectives throughout a team. Lower role ambiguity and role conflict are then expected to be associated with lower work exhaustion. Figure 1 depicts these relationships that we develop further in this section.

3.1 | Use of agile methods

Although agile methods have been widely adopted, the software industry rarely relies exclusively on either agile or plan-driven methods (Ramesh et al., 2012). Once adopted, agile methods become instead part of an organization’s portfolio of diverse software development methods that can be drawn on selectively, in sequence or in combination (Ramasubbu, Bharadwaj, & Tayi, 2015; Ramesh et al., 2012). Over time, software developers may therefore apply agile methods to varying degrees, alternate between plan-driven and agile methods, and even apply plan-driven and agile methods at the same time (Bick, Spohrer, Hoda, Scheerer, & Heinzl, 2018; Ramasubbu et al., 2015). We therefore conceptualize the extent of agile method use as the frequency and intensity with which developers apply agile methods in a particular project.

In order to complete their tasks successfully, software developers need to understand the role they play in their team, be aware of their own responsibilities compared with those of their team members, and coordinate their activities with their colleagues (Espinosa, Slaughter, Kraut, & Herbsleb, 2007; Faraj & Sproull, 2000; Zhang, Venkatesh, & Brown, 2011). Although this holds true for software development teams following plan-driven methods with specialized roles and for software development teams following agile methods with more flexible roles (Kudaravalli et al., 2017), we expect that the use of agile methods helps software developers to more effectively achieve clear and
congruent role perceptions. As such, agile methods stimulate self-organization and continuous adaptation, entail short and comprehensible development cycles, foster the creation of shared mental models, and provide developers with a set of guiding core values (Conboy, 2009; Ramesh et al., 2017). These elements of agile method use can be expected to reduce both role ambiguity and role conflict for individual developers.

More specifically, agile methods empower teams to self-organize and make relevant decisions about the design and implementation of the software product they are building. Such autonomy fosters feelings of self-determination (Deci & Ryan, 1985; Hoda & Murugesan, 2016; Wang, Schneider, & Valacich, 2015) that intrinsically motivate software developers and help them to be more proactive about information gathering to define their own role (Ke & Zhang, 2011; Windeler et al., 2017). Using agile methods, developers can also more easily resolve situations of role conflict. Developers in self-organized teams are empowered to make relevant development decisions and can thus decide to follow the expectations that are more congruent with their team goals. Self-organization inherent to agile methods consequently provides software developers with a sense of control that reduces role conflict. Similarly, the core values underlying agile methods reduce the potential for conflicting expectations toward individual developers. For example, agile methods consistently emphasize software quality over following a process or creating detailed documentation. These core values guide decisions and thus reduce the chance for developers to get caught between conflicting expectations, for example, between the expectations of project managers pushing for clean documentation and process adherence on the one hand and customers pushing for the rapid and complete development of the software product on the other hand.

Lastly, agile methods help reduce role conflict through short, structured development iterations that entail collaborative interactions with team members and customers. Collaborative agile practices such as pair programming, daily meetings, and continuous integration stimulate communication and exchange between developers about solution strategies, development obstacles, and intermediary work results. The use of agile methods thereby facilitates the mutual adjustment of developers and helps them establish shared perspectives that reduce the risk of conflicting role expectations toward individual developers. Short development cycles entail frequent exchange of software developers with customers and other stakeholders about product increments, customer needs, and development goals. These regular interactions help developers reduce the salience of the strong boundary between customers and developers. Plan-driven development methods typically span this boundary with specific roles and personnel (e.g., business analysts) who often face role conflict because expectations toward them differ on the two sides of the boundary (Baroudi, 1985; Joseph et al., 2007; Speier & Venkatesh, 2002). Agile methods, by contrast, have customers and developers interact frequently and force them to regularly discuss small product increments. These interactions and discussions help both groups to align their expectations and reduce the chance of role conflict in developers. Thus, we hypothesize:

**H1:** The extent of agile method use will be negatively associated with role conflict.
Using agile methods, software developers engage in communication and knowledge sharing with team members and other stakeholders to self-organize their development activities. Developers can use these discussions to solicit more information about the expectations of team members, managers, and customers about them and about their work output. The frequent, informal discussions help to achieve a common understanding of requirements, challenges, and areas of developer expertise (Kudaravalli et al., 2017; Yu & Petter, 2014). They thereby reduce ambiguity in who should do what in a software development team and help each developer understand their personal role in it. In self-organized agile teams, members do not have firmly specialized roles (Hoda & Murugesan, 2016). Consequently, informally exchanged knowledge about team members’ expertise and mutual expectations allow developers in such teams to shape their own roles and get a clearer understanding of their responsibilities.

Agile methods moreover make developers iteratively collect feedback from customers and team members that reduces uncertainty about their expectations. It helps establish clearly understood and agreed upon target visions for the next steps of a project. In particular, iterative evaluations of their work results allow developers to refine their understanding of the problem domain and even force them to explicitly adapt their goals (Ramesh et al., 2012). This aids in making developers’ own expectations of themselves clearer and less ambiguous. Lastly, close collaboration enforced by agile practices, such as shared code ownership, pair programming, and shared coding standards, stimulates the transfer of tacit knowledge within a team through socialization (Dingsøyr et al., 2012). Imitating coding styles, design patterns, and problem-solving strategies of colleagues, for example, lets team members implicitly come to joint perspectives on challenges and supports the emergence of common norms. The close collaboration enforced by agile methods thereby results in clearer developer expectations of each other. Thus, we hypothesize:

H2: The extent of agile method use will be negatively associated with role ambiguity.

3.2 | Organizational skills

Given that self-organization, continuous adaptation, and intensive collaboration are intrinsic parts of agile method use, software developers need organizational skills to effectively benefit from agile methods. As such, the beneficial effects of agile method use on role conflict and role ambiguity may be particularly pronounced for developers with extensive organizational skills because organizational skills facilitate interpersonal interactions and direct communication that are crucial elements of effective agile method use. Whereas plan-driven methods build on specialized role definitions and authoritative task assignments by managers, generalist members of self-organized agile teams need to engage in discussions with their team to understand and clarify their implicit roles (Hummel et al., 2013). For example, developers may realize that their colleagues expect them to work on tasks that do not fit their own perceptions of their roles and responsibilities. Especially agile development teams are known to critically review self-assignment practices of their members and discuss individual task self-assignments in order to prevent undesired specializations of team members and knowledge silos (Hoda & Murugesan, 2016). Developers with better organizational skills can more easily understand the specific arguments of their counterparts in such discussions and can clarify the situations based on sound argumentation and persuasive reasoning. Better organizational skills thereby allow developers to communicate more easily with their team members to reduce role conflict during agile method use. In contrast, lack of organizational skills may be particularly harmful for developers’ role perceptions when using agile methods. As such, lack of organizational skills may result in a failure to resolve conflicting expectations in agile development contexts because agile methods repudiate formal, specialized roles and hierarchies. Instead, they urge developers to distribute work and responsibilities in their teams based on informal communication and team processes (Dingsøyr et al., 2012). The distribution of roles and responsibilities in agile teams thereby becomes subject to group dynamics, and developers cannot point to formal role specifications that would help them make their point. Instead, they need to cope with group discussions and possibly more persuasive counterparts to resolve conflicting
viewpoints on their roles, even though they may lack the skills to do so effectively. Developers in agile teams who lack organizational skills will therefore more frequently run into role conflicts with team members, customers, and managers that they are unable to resolve. Thus, we hypothesize:

**H3:** Organizational skills will moderate the negative relationship between the extent of agile method use and role conflict, such that the relationship will be stronger at higher levels of organizational skills.

Agile method use includes frequent, periodic meetings and discussions with customers and other stakeholders that aim at sharing knowledge and opinions to probe new ideas and adjust misguided developments (Ghobadi & Mathiassen, 2016). Individuals possessing better organizational skills may use these meetings to seek more effective clarification and understand existing expectations and requirements more easily. Moreover, collaborative agile practices, such as pair programming, build on the synchronized and joint cognitive efforts of developers that involves a significant amount of verbalizing ideas, putting forth own perspectives, and taking others’ perspectives (Balijepally, Mahapatra, Nerur, & Price, 2009; Mangalaraj et al., 2014). Developers with better organizational skills may more easily conduct these activities and thereby reflect more effectively on their activities, roles, skills, and responsibilities. They may therefore face less role ambiguity when using agile methods. In contrast, developers who lack organizational skills may experience high role ambiguity and high job strain when using agile methods. As such, the lack of formal developer role descriptions in agile methods forces developers to understand the specifics of their roles primarily based on informal interpersonal communication (Hummel et al., 2013). For developers who lack organizational skills, it is, however, harder to discern the meaning behind others’ suggestions and comments during such informal encounters. They may thus face particularly strong challenges in clearly understanding all facets of their roles and the expectations that others have. Failure to communicate effectively with team members may moreover be particularly stressful for developers who are frequently forced to engage in intense collaborative agile practices, like pair programming, with their colleagues. They may more frequently fail to comprehend the meaning of team members’ problem analyses and proposed solutions, leaving them with unclear expectations about the task at hand, the roles of others, and their own role. Thus, we hypothesize:

**H4:** Organizational skills will moderate the negative relationship between the extent of agile method use and role ambiguity, such that the relationship will be stronger at higher levels of organizational skills.

### 3.3 Work exhaustion through role ambiguity and role conflict

In line with prior research, we expect both role ambiguity and role conflict to increase work exhaustion of developers. Prior work has consistently suggested that role conflict and role ambiguity are linked to negative outcomes such as job dissatisfaction, anxiety, exhaustion, reduced job engagement, and turnover intentions (Joseph et al., 2007; Tubre & Collins, 2000). Conflicting and ambiguous role perceptions are problematic because they obfuscate the clear and single flow of authority within organizations that allows employees to experience less uncertainty, increase accountability, and discern meaning about their positions regarding how they will be evaluated (Rizzo et al., 1970). Employees will face lower cognitive demands for decisions related to their responsibilities if expectations are clear and congruent (Feldman & Rafaeli, 2002). When multiple expectations from different sources are introduced, employees need to cognitively and emotionally cope with more demands that cannot necessarily be reconciled. Consequently, employees who face role conflict or role ambiguity spend more cognitive resources on weighing different expectations, become more hesitant in decision making, and have to resort to trial-and-error in meeting expectations (Pettijohn, Pettijohn, Taylor, & Keillor, 2001; Wang et al., 2015).

In software development, incongruent expectations of developers’ behaviour and unclear responsibilities both put high strain on software engineers and cause them psychological stress (Windeler et al., 2017), a well-known
antecedent of work exhaustion (Häusser et al., 2010; Van der Doef & Maes, 1999). This is particularly prevalent for individuals who engage in boundary spanning and are continuously exposed to multiple sources of influence such as vendors, managers, and users (Baroudi, 1985; Speier & Venkatesh, 2002; Windeler et al., 2017). Collecting and evaluating role-related information from these various sources creates an additional burden on developers (Joseph et al., 2007; Moore, 2000). Lastly, developers within the interdependent structures of a team need to work with and rely on the output of their team members in order to benefit from team work rather than working alone (Espinosa et al., 2007; Faraj & Sproull, 2000; Mangalaraj et al., 2014). In this context, role ambiguity and role conflict reduce individuals' sense of control over their own work and the possibility to do justice to their responsibilities, resulting in anxiety and stress (Wang et al., 2015; Windeler et al., 2017). Thus, we hypothesize:

**H5:** Role ambiguity will be positively associated with work exhaustion.

**H6:** Role conflict will be positively associated with work exhaustion.

In sum, our research model suggests that agile method use facilitates the achievement of clear and unambiguous role perceptions and thereby reduces work exhaustion for developers, particularly if they possess the organizational skills to effectively interact with others in their organization.

### 4 | Method

#### 4.1 | Study setting and participants

To test our model, we conducted a field study at a leading Indian business software vendor with software developers who were actively involved in projects using agile methods. The developers worked mostly in projects to custom develop and provide software solutions for clients. The software vendor used agile methods in their portfolio of development methodologies. Although there were no fixed rules about the application of agile software development methods, the company encouraged its developers to make use of the agile development practices of XP. Development teams could draw from a broad set of software development methods, and developers would engage to different degrees in agile practices. The developers in our sample therefore all had a basic understanding of various agile methods. Once companies have progressed beyond the initial adoption of agile methods, such situations of process diversity with developers applying multiple different methods to varying degrees over time are quite common in the software industry (Ramasubbu et al., 2015).

We focused on XP as the agile method of our investigation given its use by the software vendor. XP consists of 12 specific practices, namely, the planning game, small releases, use of metaphor, simple design, coding standards, collective ownership, sustainable pace, testing, refactoring, pair programming, continuous integration, and involvement of an on-site customer (Beck, 2000). Together, these practices instantiate the core values of the agile manifesto. Prior research has examined their use in different settings (Ramesh et al., 2012; Ramesh et al., 2017; Sarker & Sarker, 2009; Sharp & Robinson, 2008), and to some degree, their outcomes (Balijepally et al., 2009; Mangalaraj et al., 2014; Maruping et al., 2009a, 2009b). We focus on XP to understand the effects of the application of agile methods on software developers’ work exhaustion for several reasons: (a) XP consists of an established collection of agile best practices, adheres to the values outlined in the agile manifesto, and closely resembles agile principles (Beck, 2000); (b) XP is used widely in the software industry and one of the most popular agile methods (Dingsøyr et al., 2012); and (c) there is a substantive body of scholarly research on this particular method and its practices (Chan & Thong, 2009; Dingsøyr et al., 2012; Dybå & Dingsøyr, 2008; Fitzgerald, Hartnett, & Conboy, 2006; Hoda et al., 2017; Hong, Thong, Chasalow, & Dhillon, 2011; Maruping et al., 2009a).
Our sampling frame consisted of 3989 software developers working in development projects at our target software vendor in 2016. We approached the developers via the company’s management team who invited them to participate in our two-stage survey. We discarded responses from developers who did not complete both surveys. Moreover, we restricted our sample to projects that had no overlap in terms of developers in order to prevent confounding effects of multiple project/team membership. These exclusions resulted in a final sample of individual-level data from 1894 developers who worked in 217 software development projects.

The average age in our sample was 29.4 years ($SD = 4.75$) and average tenure at the company was 2.98 years ($SD = 1.60$), with 58% of the respondents being men. On average, developers had 3.1 years of experience in software development ($SD = 1.8$), and 1.9 years of experience in agile methods ($SD = 1.5$). To minimize potential nonresponse bias, we obtained demographic data of everyone in our sampling frame and compared them with our sample. Comparing respondents and nonrespondents, there were no significant differences in terms of gender ($z = 1.17; P = .24$), age ($z = -1.08; P = .28$), tenure ($z = -1.40; P = .16$), software development experience ($z = -1.75; P = .08$), or XP experience ($z = -1.48; P = .14$).

### 4.2 Measurement

We relied on established and validated scales to measure all constructs and modified them only where necessary to fit the specific research context. The measurement scales can be found in the Appendix A. Where not indicated otherwise, constructs were measured with multiple items on a 7-point Likert scale ranging from 1 (strongly disagree) to 7 (strongly agree). To create variable scores, we averaged the responses to the respective items.

#### 4.2.1 Extent of agile method use

Consistent with prior research (Maruping et al., 2009a), we measured the extent of agile method use by capturing the frequency with which developers used the six key XP practices, namely, pair programming, continuous integration, refactoring, unit testing, collective ownership, and coding standards. These practices are identified as instrumental in enabling software development teams to respond to requirement changes (Maruping et al., 2009a) and thus expected to influence developers’ understanding of their constantly evolving roles in the development process. In total, the measure consisted of 18 items (three for each XP practice) that we averaged to create a score for the extent of agile method use. The reliability of this measure was $\alpha = .79$. Although it could be reasonably argued that these practices should each be modelled as separate constructs, given our conceptualization and the fact that in our dataset, the correlations ranged from .74 to .86 across these six practices, we modelled them as a single construct—an approach that is consistent with Windeler et al. (2017) when first-order specifications are highly correlated.

#### 4.2.2 Role conflict and role ambiguity

We measured role conflict and role ambiguity using the established scales of Rutner, Hardgrave, and McKnight (2008). Although recent research has proposed refined scales for role conflict and role ambiguity (Bowling et al., 2017), we used the scale of Rutner et al. (2008) for the following reasons. First, the scales by Bowling et al. (2017) are relatively new and were not available during our instrument design phase. Second, the scales of Rutner et al. (2008) are well established and have been used to measure role conflict and ambiguity in the context of software development (Windeler et al., 2017). Using these scales allows for easier comparisons to prior work. The reliability of these scales was $\alpha_{\text{conflict}} = .75$ and $\alpha_{\text{ambiguity}} = .77$. 
4.2.3 | Organizational skills

The measure for organizational skills was adapted from Wade and Parent (2001), based on the definition of Nelson (1991), and encompasses an individual’s skills in interpersonal communication, interpersonal behaviour, and project group dynamics. The reliability of this scale was $\alpha = .70$.

4.2.4 | Work exhaustion

We measured developers’ work exhaustion using the scale of Rutner et al. (2008). This scale focuses on emotional exhaustion as an integral element and measurable symptom of job burnout and as an established predictor of turnover intentions (Joseph et al., 2007; Moore, 2000). The reliability of this scale was $\alpha = .75$.

4.2.5 | Control variables

In order to isolate the effects of agile method use as well as role conflict and role ambiguity, we controlled for individual-level and project-level characteristics. We controlled for age, gender, and team tenure as developers may become accustomed to formal and informal tasks and responsibilities over time, become more aware of their role, and gain legitimate authority in the eyes of colleagues, thereby making them less strained by managing unclear social relationships (Faraj & Sproull, 2000; Lewis, Belliveau, Herndon, & Keller, 2007). We controlled for experience in software development (in general and in using XP in particular) as employees face challenges of uncertainty and increased effort requirements when first adopting new development methodologies and need time to understand the specific roles and responsibilities they entail (Cram & Newell, 2016). We thereby ensured that any significant effects of agile method use were not caused by a lack of job experience or experience with agile methods. As prior research suggested that they may influence coping behaviours and exhaustion of employees, we further controlled for perceived autonomy and perceived fairness of rewards (Furuyama et al., 1997; Moore, 2000; Rutner et al., 2008). Moreover, workload is known as an important factor associated with work exhaustion (Moore, 2000; Ply et al., 2012), and Tuomivaara et al. (2017) proposed that agile method use reduces developers’ work exhaustion primarily through reduced workload. We therefore controlled for perceived workload to examine whether the effects of role perceptions are relevant for software developers’ work exhaustion beyond the effects of workload. At a project level, we controlled for team size and requirements uncertainty because they can make it harder to establish clear and unambiguous role perceptions (Windeler et al., 2017). In addition, we controlled for the presence of a client representative in the project team because it can be an important aid for agile teams relying on XP (Beck, 2000) and may influence how easily developers can access customer feedback to refine their role perceptions and how well development teams can engage in self-organization (Hoda, Noble, & Marshall, 2011).

4.3 | Procedure and analysis

The software vendor used agile methods in 325 projects in 2016, and these were our target for data collection. After exclusions, as described earlier (eg, overlapping team members), we collected data in two waves during the projects—ie, at the beginning and at the end of the project. The projects lasted between 80 and 140 days, consistent with the general philosophy underlying agile projects. In the first wave, we measured job skills, capabilities, and prior training, ie, organizational skills, experience in software development, and experience in XP. In the second wave, we measured individuals’ work exhaustion, extent of agile method use, and job-related perceptions, including role
ambiguity, role conflict, perceived workload, autonomy, and fairness of rewards. After the projects, we obtained the scores for team size, requirements uncertainty, and the presence of a client representative from archival data.

Table 1 shows the descriptive statistics, reliabilities, and correlations. Convergent and discriminant validity were assessed using factor analysis with oblimin rotation allowing for correlated factors. All items loaded significantly on their specified constructs, with factor loadings greater than .70 and cross-loadings below .35. Cronbach alpha exceeded .70 for all constructs. A power analysis suggested that our sample size of 1894 developers was large enough to detect even small effects with the conventionally assumed power of .80 (Cohen, 1988).

We used random coefficient modelling (RCM) to test our model. Specifically, we employed the R studio multi-level package for data analysis. We chose RCM for its capability to adequately account for the nested nature of our data. In our sample, 1894 developers were nested in 217 projects. This leads to nonindependence of observations and calls for testing a “1-1-1” two-level model that corrects relationships between variables at a lower (ie, individual) level for error induced by commonalities of observations at a higher (ie, project) level (Bauer, Preacher, & Gil, 2006). RCM tools remedy many of the threats of type I and type II errors due to nested data by explicitly modelling non-independence (Bliese & Hanges, 2004). RCM has been widely used and accepted in IS for analysing multilevel effects on team and individual levels (eg, Maruping & Magni, 2015, Venkatesh et al., 2018, Windeler et al., 2017). To report the explained variance that is proportionally reduced for level 1 and level 2 errors, we provide Snijders and Bosker (1999)’s overall pseudo $R^2$ ($\tilde{R}^2$). To establish that it is reasonable to use a multilevel RCM approach, we first ran a two-level null model with no predictors included. Results of a $\chi^2$ test indicate sufficient variability at both levels of analysis. In the null model, 62% of variance in developers’ work exhaustion was attributable to individual-level differences, whereas 38% ($\chi^2 = 1112.66, P < .001$) of variance was attributable to differences between projects. This reinforced the need to control for project-level effects.

Following Venkatesh et al. (2018), we conducted multiple tests to exclude common method bias. We ran Harman one factor test (Podsakoff, MacKenzie, Lee, & Podsakoff, 2003) with an unrotated factor analysis. The first factor extracted only about 10% of the variance, thus reducing the concern of common method bias in this study. In addition, we conducted a marker variable test (Lindell & Whitney, 2001). The resulting attenuation was below 0.04 in the correlations between the variables, and the significance levels remained stable. This test further reduces the concern for common method bias.

## RESULTS

The results of testing the two-level model for H1 to H4 are shown in Table 2A. Models 1a and 1b represent the baseline models, models 2a and 2b represent the main effects only models, and models 3a and 3b represent the moderated models. All models account for the nested nature of our data. In H1 and H2, we predicted that a larger extent of agile method use would be negatively associated with developers’ perceptions of role ambiguity and role conflict. The results show that a larger extent of agile method use is negatively associated with both role ambiguity ($\gamma = -0.16, P < .01$) and role conflict ($\gamma = -0.18, P < .01$), thus supporting H1 and H2. Moreover, the results show that there are significant interaction effects of agile method use and organizational skills on role ambiguity ($\gamma = -0.20, P < .01$) and role conflict ($\gamma = -0.28, P < .001$). To understand the nature of the interactions, we plotted the significant interactions, per Aiken and West (1991). Figure 2A,B illustrates the simple slopes of agile method use on role ambiguity and role conflict, respectively, at high and low values of organizational skills (ie, 1 SD above and below the mean). As shown in Figure 2A,B, a larger extent of agile method use is more negatively related to role ambiguity and role conflict when developers have high organizational skills than when they have low organizational skills. This suggests that organizational skills amplify the relationship between agile method use and clearer and less ambiguous role perceptions. Taken together, these results provide support for our H3 and H4 on the moderating relationship of organizational skills. All models, for main effects (models 2a and 2b) and for interaction effects (models 3a and 3b),...
<table>
<thead>
<tr>
<th>Variables</th>
<th>Avg</th>
<th>SD</th>
<th>CA</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Age</td>
<td>29.4</td>
<td>4.75</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>2. Gender</td>
<td>0.42</td>
<td>.50</td>
<td>NA</td>
<td>.05</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>3. Tenure</td>
<td>2.98</td>
<td>1.60</td>
<td>NA</td>
<td>.21</td>
<td>.05</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>4. Experience in software development</td>
<td>3.1</td>
<td>1.80</td>
<td>NA</td>
<td>.19</td>
<td>.19</td>
<td>.17</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>5. Experience in XP</td>
<td>1.9</td>
<td>1.51</td>
<td>NA</td>
<td>.15</td>
<td>.14</td>
<td>.16</td>
<td>.22</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>6. Perceived workload</td>
<td>5.46</td>
<td>1.90</td>
<td>NA</td>
<td>.16</td>
<td>.05</td>
<td>.08</td>
<td>.09</td>
<td>.13</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>8. Fairness of rewards</td>
<td>4.81</td>
<td>1.81</td>
<td>NA</td>
<td>.24</td>
<td>.19</td>
<td>.19</td>
<td>.16</td>
<td>.13</td>
<td>.05</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>9. Team size</td>
<td>8.73</td>
<td>1.93</td>
<td>NA</td>
<td>.79</td>
<td>.10</td>
<td>.09</td>
<td>.05</td>
<td>.12</td>
<td>.14</td>
<td>.12</td>
<td>.04</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>10. Client representative</td>
<td>0.55</td>
<td>0.50</td>
<td>NA</td>
<td>.07</td>
<td>.05</td>
<td>.08</td>
<td>.09</td>
<td>.13</td>
<td>.13</td>
<td>.13</td>
<td>.08</td>
<td>.15</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>11. Requirements uncertainty</td>
<td>24.87</td>
<td>11.39</td>
<td>NA</td>
<td>.10</td>
<td>.08</td>
<td>.06</td>
<td>.04</td>
<td>.05</td>
<td>.10</td>
<td>.08</td>
<td>.06</td>
<td>.17</td>
<td>.18</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>12. Extent of agile method use</td>
<td>4.66</td>
<td>1.53</td>
<td>NA</td>
<td>.79</td>
<td>.17</td>
<td>.17</td>
<td>.21</td>
<td>.21</td>
<td>.24</td>
<td>.17</td>
<td>.08</td>
<td>.17</td>
<td>.13</td>
<td>.15</td>
<td>.16</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>13. Organizational skills</td>
<td>4.88</td>
<td>2.17</td>
<td>NA</td>
<td>.27</td>
<td>.23</td>
<td>.23</td>
<td>.23</td>
<td>.08</td>
<td>.19</td>
<td>.19</td>
<td>.13</td>
<td>.16</td>
<td>.07</td>
<td>.10</td>
<td>.14</td>
<td>.20</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>14. Role ambiguity</td>
<td>4.98</td>
<td>1.64</td>
<td>NA</td>
<td>.77</td>
<td>.17</td>
<td>.08</td>
<td>.10</td>
<td>.23</td>
<td>.13</td>
<td>.14</td>
<td>.19</td>
<td>.20</td>
<td>.13</td>
<td>.17</td>
<td>.37</td>
<td>.20</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

Note. Gender is coded as male = 0 and female = 1. Variables team size, client representative, and requirements uncertainty on level 2 (project, n = 217), all other variables on level 1 (individual developer, n = 1894).
Abbreviations: CA, Cronbach alpha; XP, extreme programming.
*P < .05; **P < .01; ***P < .001.
show significant improvements in deviance and explained variance that lends further support for H1 to H4. In total, the moderated models explain 25% and 30% of variance in role ambiguity and role conflict, respectively.

Table 2B shows the results of the RCM analysis predicting role ambiguity and role conflict. As predicted by H5 and H6, both role ambiguity ($\gamma = 0.19$, $P < .001$) and role conflict ($\gamma = 0.21$, $P < .001$) have significant positive effects on work exhaustion beyond the effects of the control variables ($\Delta R^2 = 0.08$, $P < .01$). Two individual-level control variables—age and workload—and two project-level control variables—requirements uncertainty and the presence of a client representative—have significant effects on work exhaustion, but their path coefficients are lower for the model including role ambiguity and role conflict (model 2). This lends support to the reasoning that the effects of agile method use through role perceptions have a significant impact on developers’ work exhaustion beyond project-level effects and beyond workload effects that have been suggested by prior work (Tuomivaara et al., 2017). Overall, the model explains 32% of the variance in work exhaustion.

We next examined the direct and indirect effects of the extent of agile method use on work exhaustion in more detail. To do so, we conducted a moderated mediation analysis following Edwards and Lambert (2007) in which bootstrapping was used to create bias-corrected confidence intervals and simple effects. Table 3 shows the results of

### Table 2A

<table>
<thead>
<tr>
<th>Variables</th>
<th>Role Ambiguity</th>
<th>Role Conflict</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Model 1a</td>
<td>Model 2a</td>
</tr>
<tr>
<td>L1 control variables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Age</td>
<td>.13* (.007)</td>
<td>.12* (.010)</td>
</tr>
<tr>
<td>Gender</td>
<td>.04 (.012)</td>
<td>.03 (.022)</td>
</tr>
<tr>
<td>Tenure</td>
<td>.05 (.023)</td>
<td>.03 (.027)</td>
</tr>
<tr>
<td>Experience in software development</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Experience in XP</td>
<td>.08 (.031)</td>
<td>.04 (.031)</td>
</tr>
<tr>
<td>L2 control variables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Team size</td>
<td>.08 (.040)</td>
<td>.05 (.044)</td>
</tr>
<tr>
<td>Client representative</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements uncertainty</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Experience in XP</td>
<td>.13* (.010)</td>
<td>.11* (.012)</td>
</tr>
<tr>
<td>Main effects</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extent of agile method use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Organizational skills</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interaction effect</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extent of agile method use × organizational skills</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note. Results for two-level model: variables team size, client representative, and requirements uncertainty on level 2 (project, n = 217), all other variables on level 1 (individual developer, n = 1894). Standard errors are shown in parentheses. Abbreviations: RCM, random coefficient modelling; XP, extreme programming.

* $P < .05$; ** $P < .01$; *** $P < .001$. 

We next examined the direct and indirect effects of the extent of agile method use on work exhaustion in more detail. To do so, we conducted a moderated mediation analysis following Edwards and Lambert (2007) in which bootstrapping was used to create bias-corrected confidence intervals and simple effects. Table 3 shows the results of
this analysis. It outlines the simple effects of agile method use on developers’ work exhaustion as mediated by role conflict and role ambiguity and moderated by organizational skills. The results indicate that a larger extent of agile method use has significant negative indirect, direct, and total effects on developers’ work exhaustion that are all significantly stronger for higher levels of organizational skills. The results thus provide strong support for the hypothesized moderating effect of organizational skills. The indirect and direct effects of agile method use on work exhaustion are negative and significant when mediated by role ambiguity and role conflict. This suggests that the mediated effects of agile method use and its interaction with organizational skills through role perceptions entail significant consequences for individual developers’ work exhaustion. More specifically, the results show that role ambiguity and role conflict partially mediate the effects of agile method use on software developers’ work exhaustion conditional on organizational skills. Although this partial mediation lends support to our reasoning in H1 through H6, it also suggests that role perceptions are not the only mechanism through which agile method use influences work exhaustion in software developers.

We further took procedural precautions to demonstrate the robustness of our results. Specifically, endogeneity could constitute a problem for our results if the extent of agile method use or organizational skills were endogenous to role perceptions that would result in omitted variable bias or selection bias (Wooldridge, 2012). Thus, we took several steps to rule out potential threats of endogeneity. First, as part of our research design, the multi-wave data collection reduced endogeneity concerns because effects that occurred and were measured later in the projects (e.g., role perceptions and work exhaustion) could unlikely influence what occurred and was measured earlier in the projects (e.g., organizational skills). Second, consistent with prior work (e.g., Hsieh, Rai, & Xu, 2011, Mani, Barua, & Whinston, 2012, Venkatesh, Shaw, Sykes, Wamba, & Macharia, 2017), we conducted a two-stage Heckman procedure to address endogeneity concerns regarding the extent of agile method use. Originally intended to correct estimates for selection bias, this approach is often used to rule out broader endogeneity threats (Venkatesh et al., 2017). Appendix B reports the results of this procedure. The results do not hint toward any endogeneity threats and indicate that selection bias does not constitute a threat to our analyses (see Appendix B). Third, we followed the approach suggested by Frank (2000) to assess the danger of possible omitted variable bias. We calculated the impact threshold for a confounding variable (ITCV) at which an omitted variable would render the effect of an independent variable (e.g., extent of agile method use) on a dependent variable (e.g., role ambiguity or role conflict) nonsignificant. The ITCV determines the minimum correlations of a potential

FIGURE 2  A, Interaction effect of agile method use and organizational skills on role ambiguity; B, Interaction effect of agile method use and organizational skills on role conflict
### Table 2B: Results of RCM analysis predicting work exhaustion

<table>
<thead>
<tr>
<th>Variables</th>
<th>Work Exhaustion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Model 1</td>
</tr>
<tr>
<td><strong>L1 control variables</strong></td>
<td></td>
</tr>
<tr>
<td>Age</td>
<td>.13* (.009)</td>
</tr>
<tr>
<td>Gender</td>
<td>.07 (.024)</td>
</tr>
<tr>
<td>Tenure</td>
<td>.05 (.026)</td>
</tr>
<tr>
<td>Experience in software development</td>
<td>-.15* (.011)</td>
</tr>
<tr>
<td>Experience in XP</td>
<td>-.13* (.011)</td>
</tr>
<tr>
<td>Perceived workload</td>
<td>.13* (.012)</td>
</tr>
<tr>
<td>Autonomy</td>
<td>.14** (.006)</td>
</tr>
<tr>
<td>Fairness of rewards</td>
<td>.06 (.027)</td>
</tr>
<tr>
<td><strong>L2 control variables</strong></td>
<td></td>
</tr>
<tr>
<td>Team size</td>
<td>.14* (.010)</td>
</tr>
<tr>
<td>Client representative</td>
<td>-.17** (.012)</td>
</tr>
<tr>
<td>Requirements uncertainty</td>
<td>.19** (.011)</td>
</tr>
<tr>
<td><strong>Main effects</strong></td>
<td></td>
</tr>
<tr>
<td>Role ambiguity</td>
<td>.19*** (.010)</td>
</tr>
<tr>
<td>Role conflict</td>
<td>.21*** (.011)</td>
</tr>
<tr>
<td>Deviance</td>
<td>2155.12</td>
</tr>
<tr>
<td>$\chi^2$</td>
<td>371.68***</td>
</tr>
<tr>
<td>$R^2$</td>
<td>.24</td>
</tr>
<tr>
<td>$\Delta R^2$</td>
<td>.08</td>
</tr>
</tbody>
</table>

*Note. Results for two-level model: Variables team size, client representative, and requirements uncertainty on level 2 (project, $n = 217$), all other variables on level 1 (individual developer, $n = 1894$). Standard errors are shown in parentheses. Abbreviations: RCM, random coefficient modelling; XP, extreme programming. *$P < .05$; **$P < .01$; ***$P < .001$.  

### Table 3: Simple effects analysis of agile method use on work exhaustion

<table>
<thead>
<tr>
<th>Moderator variable</th>
<th>Mediation through Role Ambiguity</th>
<th>Mediation through Role Conflict</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Direct</td>
<td>Indirect</td>
</tr>
<tr>
<td>Low (mean − 1 SD)</td>
<td>-.06*</td>
<td>-.07**</td>
</tr>
<tr>
<td>High (mean + 1 SD)</td>
<td>-.19***</td>
<td>-.11**</td>
</tr>
<tr>
<td>Difference</td>
<td>.13**</td>
<td>.04*</td>
</tr>
</tbody>
</table>

*Note. Results for direct, indirect, and total effects following Edwards and Lambert (2007). Bootstrap estimates based on 1000 resamples. *$P < .05$; **$P < .01$; ***$P < .001$
omitted variable with both the independent variable and the dependent variable that are necessary to render the
effect of the independent variable on the dependent variable nonsignificant after controlling for all covariates. We
calculated ITCV scores for the hypothesized relationships H1 to H4. Of all these relationships, the relationship of
the extent of agile method use with role ambiguity showed to be the least robust against omitted variable bias.
Still, a potential omitted variable would have to be correlated with both the extent of agile method use and role
ambiguity at more than 0.488 after controlling for covariates to invalidate our inferences. Given that this is much
higher than the correlations observed in our sample (see Table 1), we deem it unlikely that an omitted variable
could fulfil these conditions. As all other effects were even more robust, there are no indications that omitted vari-
able bias would constitute an issue for our results. In sum, these precautions heavily reduce endogeneity concerns
regarding H1 to H4.

### 6.1 Theoretical contributions

First, we elaborate on the effects of using agile development methods on work exhaustion in software developers
and show that the use of agile methods reduces work exhaustion by lowering both role ambiguity and role conflict in
software developers. This finding is important because work exhaustion in software developers leads to program-
ing errors and subsequently high maintenance costs, reduces team performance, and increases employee turnover
(Furuyama et al., 1997; Venkatesh et al., 2018; Windeler et al., 2017). A better understanding of how the use of pop-
ular agile methods influences software developers' work exhaustion consequently helps to better understand suc-
cess and failure in contemporary software development projects. Prior work in this area has primarily taken a project
management perspective and found that agile methods reduce work exhaustion by distributing the workload in soft-
ware development projects more evenly over time (Tuomivaara et al., 2017; Vidgen & Wang, 2009). Our research

<table>
<thead>
<tr>
<th>TABLE 4 Key contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Research Stream</strong></td>
</tr>
<tr>
<td>Effects of agile development methods on work exhaustion</td>
</tr>
<tr>
<td>Skills in software development</td>
</tr>
<tr>
<td>Effectiveness of software development methods</td>
</tr>
</tbody>
</table>

---

6 | DISCUSSION

We set out to explain the effects of agile development methods on software developers’ work exhaustion. This was
accomplished by developing and testing a model suggesting that agile method use facilitates the achievement of
clear and unambiguous role perceptions and thereby reduces work exhaustion for developers, particularly if they
possess the organizational skills to effectively interact with others in their organization. We found strong support for
our model in a field study on 1894 software developers in 217 project teams that used agile methods. Our findings
have several theoretical and practical implications. Table 4 outlines the key contributions of our study.
theorized and empirically found support for a different effect of agile methods on developers: Beyond the effects on their workload, agile methods let developers come to clearer and more congruent role perceptions that in turn lower developers’ work exhaustion. Our work thus provides a new, behavioural explanation of individual-level effects of agile method use on developers’ work exhaustion. Whereas workload effects hinge primarily on the project-wide use of agile methods for project management (Tuomivaara et al., 2017; Vidgen & Wang, 2009), the frequency and intensity in which developers engage individually in agile practices vary within projects and thus influences developers at an individual level rather than at a project level. Future research on the effectiveness of software development methods needs to take into account that methods such as XP may not only change project management activities but also have distinctive effects on individual developers’ role perceptions and work exhaustion. Although a moderated mediation analysis lent strong support to our research model and the indirect effects of agile method use on work exhaustion through role perceptions, it also showed significant direct effects of agile method use on work exhaustion that were not mediated by role perceptions. Future research should investigate the mechanisms that underlie these unexplained direct effects to further extend the emerging stream of research on the effects of agile methods on individual software developers (Balijepally et al., 2009; Tripp et al., 2016; Tuomivaara et al., 2017).

Second, our work shows that software developers’ organizational skills facilitate the benefits of agile methods in terms of improved role perceptions and reduced work exhaustion. Based on the JDCM, we found empirical support for our theorizing that the use of agile methods places specific demands on software developers that they can best address if they possess organizational skills to effectively manage interpersonal communication and group dynamics. This finding is important because prior work on the effects of agile methods has not fully accounted for individual-level differences between software developers. Specifically, prior work could not explain why some software developers become less exhausted from using agile methods than others who perceive the exact opposite (Balbes, 2017; Laanti, 2013). Tapping into this gap, our work suggests that organizational skills facilitate software developers’ interactions with team members and external stakeholders that become necessary when following agile methods. Organizational skills thereby allow developers to reap the benefits of agile methods. In line with this reasoning, a moderated mediation analysis showed that the use of agile methods without appropriate organizational skills resulted in significantly higher levels of role ambiguity, role conflict, and work exhaustion in software developers. The more general literature on the effects of software development methodologies learns from our research that distinct developer skills can decide about the favourable or unfavourable effects of software development methodologies. Prior work in this direction stressed the importance of experience with software development in general and with specific development methodologies in particular for coping with software development work (Ang, Thong, & Yap, 1997; McManus, 2003; Sultan & Chan, 2000). We add to this stream of research by pinpointing the importance of specific skills, rather than experience, for coping with job demands and preventing work exhaustion during software development.

Third, our findings contribute to research on the importance of technical and organizational skills in software development (eg, Ang et al., 1997, Wade & Parent, 2001). Prior work often argued that software developers primarily need technical skills and that deficiencies in organizational skills should not be as dramatic for them as for other occupational groups (Gallagher et al., 2010; Nelson, 1991; Wade & Parent, 2001). Our findings show that this assumption is questionable with regard to contemporary software development. Given the widespread use of agile methods, developers may increasingly need organizational skills to effectively apply these methods. In fact, agile software development today appears to follow predictions by Lee et al. (1995) who envisioned organizational skills to become continuously more important.

Lastly, prior research has shown that agile methods are particularly beneficial for outcomes in projects with high complexity and requirements volatility (Maruping et al., 2009a). Extant theory holds that this advantage results from improved project-level control structures (Maruping et al., 2009a; Venkatesh et al., 2018). Our findings add to this view by suggesting that agile methods not only improve the project-level control but also allow developers to increase control at the individual level as they gain less conflicting and less ambiguous role perceptions. Examining possible interactions between these project-level and individual-level effects of agile methods may therefore be a
fruitful avenue for future research. In fact, recent work suggests that especially multi-level views on developer role perceptions can strongly improve our understanding of project performance (Windeler et al., 2017).

6.2 Practical implications

IT organizations have previously been found to hire applicants primarily based on their technical skills, largely ignoring their organizational skills (Wade & Parent, 2001). Our findings call this practice into question, at least for software developers working with agile methods. Our results suggest that software companies that are interested in preventing work exhaustion and burnout of their developers should provide training in organizational skills and agile methods, or they need to make sure that their software developers already have these skills when they are hired.

Role conflict and role ambiguity are especially prevalent in software projects with high technical risk and requirements volatility (Windeler et al., 2017). Our results show that agile methods can be used to reduce role conflict and role ambiguity. From an individual-level perspective, agile methods may therefore fit particularly well with high risk projects and volatile requirements. Given that technical risk factors and volatile requirements continue to be two of the most common and influential obstacles to successful software development work (Hoda & Murugesan, 2016; Maruping, Venkatesh, Thong, & Zhang, 2019; Venkatesh et al., 2018), our findings have a substantial impact for the software industry and can help reduce the notoriously high failure rates of software development projects.

Lastly, software vendors can learn in which way the use of agile development methods relates to one of their core resources, namely, software developers. The increased application of agile methods is associated with reduced software developers’ work exhaustion. To preserve the long-term performance and employability of their developers, even conservative organizations may consequently want to utilize agile methods, at least to a certain degree in some of their projects.

6.3 Limitations and future research

Our work focused on software developers’ work exhaustion through role conflict and role ambiguity. This focus was reasonable because contemporary theory in organizational psychology has established role conflict and role ambiguity as crucial determinants of work exhaustion (Häusser et al., 2010; Karasek et al., 1998; Van der Doef & Maes, 1999) that are particularly relevant in the context of empowered teams (Windeler et al., 2017). Future research may want to replicate our study using the refined conceptualizations and measurements of role conflict and role ambiguity that were proposed in recent research (Bowling et al., 2017). Although we controlled for a number of potentially confounding variables, there are also other sources of job strain possibly related to agile methods that this study did not account for. For example, time pressure, project complexity, and the temporal distribution of workload affect how developers feel and how they interact with their team members (Maruping et al., 2015; Tuomivaara et al., 2017; Venkatesh et al., 2018; Venkatesh, Maruping, & Brown, 2006). Likewise, process maturity and how well a software development method is implemented in a project might influence developers’ role perceptions (Ramasubbu et al., 2015). Given that such factors would influence all members of a project team in the same way and that our RCM analysis controlled for project-level nonindependence of observations, our results are to some degree robust against the confounding effects of these factors. Moreover, our robustness checks for potential omitted variable bias did not raise any concerns. Nonetheless, we suggest that future research should examine these potential sources of job strain and their effects more explicitly. Another avenue for research is to evaluate individual level conditions, such as Internet addiction, that might influence work exhaustion in software developers, especially if they work in contexts with high autonomy such as agile teams (Venkatesh, Sykes, Chan, Thong, & Hu, 2019). Future research may also investigate psychological and performance-related outcomes that can have more complex relationships with job strain (Onyemah, 2008).
Our study examined the use of XP, a particularly popular agile method for which there is a body of scholarly research. Yet, there are other agile methods. Scrum, for example, is partly more concerned with project management than with actual collaborative development work that differentiates it from XP (Dingsøyr et al., 2012). The amount of collaborative development work may, however, influence how often developers can engage with team members, users, customers, and external stakeholders to seek mutual understanding and clarify their roles. Future research should therefore examine how other agile development methods influence role perceptions. For example, qualitative investigations into multiple other development methods may be used to extend and complement our quantitative results. In addition, our study was set in a context where XP had already been introduced as the company's choice of agile method. This meant that XP was one method in the company's overall portfolio of development methods. XP was therefore used selectively and to varying degrees of intensity. Such selective use of development methods is typical for most organizations in the software industry that have progressed beyond the initial adoption of agile methods (Ramasubbu et al., 2015) that makes our findings relevant for a variety of organizations. Yet, we did not study the introduction of agile methods or a comparison of projects that exclusively used either agile or plan-driven methods. Consequently, our findings do not speak to the question of whether the introduction of agile methods alleviates developers' work exhaustion or possibly even causes more role conflict and role ambiguity. Future research that examines such effects can draw on an extensive body of literature on the individual-level effects of introducing new technology in teams and organizations (Bala & Venkatesh, 2015; Dennis, Venkatesh, & Ramesh, 2008; Venkatesh & Windeler, 2012). Likewise, it is a limitation of our research that we did not examine patterns of simultaneous use of multiple development methods that can affect software quality and project performance (Bick et al., 2018; Ramasubbu et al., 2015). For example, if teams combine XP with elements of other methods, such as Scrum, the combined effects of the individual methods could create synergies or cancel each other out. These issues and how they relate to individual developer's work exhaustion are interesting directions for future research.

Our study paid special attention to developers' organizational skills because we expected and found that organizational skills are important for developers who use agile methods. In doing so, we did not aim at drawing a full picture of all the relevant job skills of developers and did not collect data on many other skills that successful developers may need (Gallagher et al., 2010; Nelson, 1991; Wade & Parent, 2001). This limitation of our work is an opportunity for future work that may draw a more comprehensive picture of the relevant job skills for developers. In light of our findings, this line of work should evaluate the relative importance of different skills in agile and plan-driven development environments.

Finally, our empirical focus on a single organization allowed us to naturally control for inter-organizational differences. Such a focus on a single vendor firm to study IT projects is consistent with prior research (eg, Kudaravalli et al., 2017; Rai, Maruping, & Venkatesh, 2009, Ramasubbu et al., 2015, Venkatesh et al., 2018, Windeler et al., 2017) as it helps to control for organizational-level effects. But it also calls for replicating our work in other organizations to increase generalizability. There is substantial evidence that organizational and national cultures influence the adoption and adaptation of software development methods (Persson, Mathiassen, & Aaen, 2012; Rai et al., 2009; Ramesh et al., 2017; Sarker & Sarker, 2009). Future research should therefore aim to understand the cultural aspects of how agile methods affect developer role perceptions and should try to outline how agile methods can be used to reduce developers' work exhaustion in globally distributed and culturally heterogeneous settings.

### CONCLUSION

We presented a research model with the goal to explain the effects of agile software development methods on individual developers' work exhaustion. We theorized that agile method use helps software developers to achieve clear and unambiguous role perceptions that reduce work exhaustion if the developers have organizational skills to effectively interact with customers and stakeholders within their organization. We found strong support for our model in a field study of 1894 software developers applying the popular agile development method XP in 217 project teams.
Our findings have substantial implications for theory on the individual-level effects of agile development methods and show that the effectiveness of development methodologies can hinge on specific skills of individual developers. We encourage organizations to use agile software development methodologies but caution that developers need to have organizational skills to be able to reap the benefits of these methods.

ACKNOWLEDGEMENTS
This project was funded by a grant from the Research Grants Council of Hong Kong (GRF693313).

ORCID
Viswanath Venkatesh https://orcid.org/0000-0001-8473-376X
James Y. L. Thong https://orcid.org/0000-0002-1640-0581
Frank K. Y. Chan https://orcid.org/0000-0001-9301-7634
Hartmut Hoehle https://orcid.org/0000-0001-8117-0105
Kai Spohrer https://orcid.org/0000-0001-8659-7554

REFERENCES
Bliwise, P. D., & Hanges, P. J. (2004). Being both too liberal and too conservative: The perils of treating grouped data as though they were independent. Organizational Research Methods, 7, 400–417.


How to cite this article: Venkatesh V, Thong JYL, Chan FKY, Hoehle H, Spohrer K. How agile software development methods reduce work exhaustion: Insights on role perceptions and organizational skills. Info Systems J. 2020;1–29. https://doi.org/10.1111/isj.12282

APPENDIX

A. MEASUREMENT SCALES

Extent of agile method use (Maruping, Venkatesh, & Agarwal, 2009a)

Pair programming
1. How often is pair programming used on this team? (1 = never; 7 = all the time)
2. On this team, we do our software development using pairs of developers.
3. To what extent is programming carried out by pairs of developers on this team? (1 = never; 7 = all the time)

Collective ownership
4. Anyone on this team can change existing code at any time.
5. If anyone wants to change a piece of code, they need the permission of the individual(s) that coded it.
6. Members of this team feel comfortable changing any part of the existing code at any time.

Coding standards
7. We have a set of agreed upon coding standards in this team.
8. Members of this team have a shared understanding of how code is to be written.
9. Everyone on this team uses their own standards for coding.

Continuous integration
10. Members of this team integrate newly coded units of software with existing code.
11. We combine new code with existing code on a continual basis.
12. Our team does not take time to combine various units of code as they are developed.

Refactoring
13. Where necessary, members of this team try to simplify existing code without changing its functionality.
14. We periodically identify and eliminate redundancies in the software code.
15. We periodically simplify existing code.

Unit testing
16. We run unit tests on newly coded modules until they run flawlessly.
17. Members of this team actively engage in unit testing.
18. To what extent are unit tests run by this team? (1 = never; 7 = all the time)
Role ambiguity (Rutner et al., 2008)
1. I know exactly what is expected of me.
2. I have clear, planned goals and objectives for my task assignment.
3. I have a defined role in my team.

Role conflict (Rutner et al., 2008)
1. I sometimes have to buck a rule or policy in order to carry out an assignment.
2. I often perform work for two or more parties who operate quite differently.
3. In my work, I have to try to balance two or more conflicting preferences.

Organizational skills (Nelson, 1991; Wade & Parent, 2001)
1. I can communicate effectively with others.
2. I can recognize and manage personality problems which interfere with job completion.
3. I can work effectively in groups.
4. I can manage projects.

Work exhaustion (Rutner et al., 2008)
1. I feel emotionally drained from my work.
2. I feel used up at the end of the workday.
3. I feel burned out from my work.

Perceived workload (Rutner et al., 2008)
1. I feel that the number of requests, problems, or complaints I deal with is more than expected.
2. I feel that the amount of work I do interferes with how well it is done.
3. I always feel busy.

Autonomy (Rutner et al., 2008)
1. In my work, I usually do not have to refer matters to my direct supervisor for a final decision.
2. Usually, my direct supervisor does not have to approve my decisions before I can take action.
3. I can usually do what I want on this job without consulting my direct supervisor.

Fairness of rewards (Rutner et al., 2008)
1. I think my level of pay is fair.
2. Overall, the rewards I receive here are quite fair.

Client representative (Rai et al., 2009)
Following Rai et al. (2009), we used a dummy variable that indicated whether a project team had a client representative present or not.

Requirements uncertainty (Rai et al., 2009)
The number of formal, written changes to the project contract between our focal software vendor and the respective customer was used to measure requirements uncertainty.

B. TWO-STAGE HECKMAN ANALYSIS

We conducted a two-stage Heckman analysis with an instrumental variable to address endogeneity concerns regarding the extent of agile method use. This analysis aims to correct path estimates for sample selection bias and
provides insight into whether selection bias constitutes a threat to the original analysis of a sample (Wooldridge, 2012, p. 619). For example, one would expect the second-stage results of the Heckman analysis (i.e., path estimates corrected using the Inverse Mills Ratio) to differ strongly from the original estimations if there were mentionable sample selection bias.

Because we had controlled for reasonable explanatory variables, we had no options remaining for additional suitable instrumental variables. We therefore dropped experience in XP that had served as a control variable from the original models and used it as an instrumental variable for the extent of agile method use after re-estimating the models. Following standard procedures (Wooldridge, 2012), we examined the relevance and exogeneity of experience in XP for the relationships of the extent of agile method use with role ambiguity and role conflict. Experience in XP lent itself as an instrument as it can reasonably be expected to influence the use of XP and is highly correlated with it in our sample whereas it cannot reasonably be caused by role perceptions and does not show high correlations with them in our sample (see Table 1).

Models M2 and M5 in Table B1 display the re-estimated models of the original analysis after dropping experience in XP. These models serve as a baseline for comparing corrected and uncorrected estimates. Models M3 and M6 in Table B1 depict the corrected results based on the two-stage Heckman procedure using experience in XP as an instrumental variable for the extent of agile method use. The results of the analysis show that model M3 does not differ strongly from model M2, and model M6 does not differ strongly from model M5. Instead, the estimates remain qualitatively the same after correcting for the Inverse Mills Ratio. This suggests that sample selection bias does not constitute a relevant threat to our original analyses and thus reduces endogeneity concerns.

### TABLE B1  Two-stage Heckman procedure for extent of agile method use and role perceptions

<table>
<thead>
<tr>
<th>Models</th>
<th>Role Ambiguity</th>
<th>Role Conflict</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M1</td>
<td>M2</td>
</tr>
<tr>
<td></td>
<td>Original (Model 2a Table 2A)</td>
<td>Dropping instrumental var.</td>
</tr>
<tr>
<td>L1 control variables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Age</td>
<td>.12* (.010)</td>
<td>.13* (.011)</td>
</tr>
<tr>
<td>Gender</td>
<td>.03 (.022)</td>
<td>.05 (.020)</td>
</tr>
<tr>
<td>Tenure</td>
<td>.03 (.027)</td>
<td>.07 (.029)</td>
</tr>
<tr>
<td>Experience in software development</td>
<td>-.10 (.012)</td>
<td>-.11* (.009)</td>
</tr>
<tr>
<td>Experience in XP</td>
<td>.04 (.031)</td>
<td></td>
</tr>
<tr>
<td>L2 control variables</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Team size</td>
<td>.05 (.044)</td>
<td>.02 (.040)</td>
</tr>
<tr>
<td>Client representative</td>
<td>-.10 (.022)</td>
<td>-.10 (.018)</td>
</tr>
<tr>
<td>Req. uncertainty</td>
<td>.11* (.012)</td>
<td>.13* (.009)</td>
</tr>
<tr>
<td>Main effects</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Extent of agile method use</td>
<td>-.23*** (.008)</td>
<td>-.24*** (.007)</td>
</tr>
<tr>
<td>Organizational skills</td>
<td>-.13* (.010)</td>
<td>-.13* (.009)</td>
</tr>
</tbody>
</table>

(Continues)
<table>
<thead>
<tr>
<th>Models</th>
<th>Role Ambiguity</th>
<th>Role Conflict</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>M1</td>
<td>M2</td>
</tr>
<tr>
<td>Model details</td>
<td>Original (Model 2a Table 2A)</td>
<td>Dropping instrumental var.</td>
</tr>
<tr>
<td>Heckman correction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inverse mills ratio (for extent of agile method use)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deviance</td>
<td>951.48</td>
<td>996.50</td>
</tr>
<tr>
<td>$\chi^2$</td>
<td>329.17***</td>
<td>341.60***</td>
</tr>
<tr>
<td>$R^2$</td>
<td>.17</td>
<td>.17</td>
</tr>
</tbody>
</table>

Note. Results for two-level model: variables team size, client representative, and requirements uncertainty on level 2 (project, n = 217), all other variables on level 1 (individual developer, n = 1894). Standard errors are shown in parentheses. Abbreviation: XP, extreme programming.

* $P < .05$; ** $P < .01$; *** $P < .001$. 