Acrobats and Safety Nets: Problematizing Large-Scale Agile Software Development

KNUT H. ROLLAND, University of Oslo, Norway
BRIAN FITZGERALD, University of Limerick and Lero, Ireland
TORGEIR DINGSØYR, Norwegian University of Science and Technology and SimulaMet, Norway
KLAAS-JAN STOL, University College Cork and Lero, Ireland, and SINTEF, Norway

Agile development methods have become a standard in the software industry, including in large-scale projects. These methods share a set of underlying assumptions that distinguish them from more traditional plan-driven approaches. In this article, we adopt Alvesson and Sandberg's problematization approach to challenge three key assumptions that are prevalent in the large-scale agile literature: (1) agile and plan-driven methods are mutually exclusive; (2) self-managing and hierarchically organized teams are mutually exclusive; and (3) agile methods can scale through simple linear composition. Using a longitudinal case study of large-scale agile development, we describe a series of trigger events and episodes whereby the agile approach was tailored to address the needs of the large-scale development context, which was very much at odds with these fundamental assumptions. We develop a set of new underlying assumptions which suggest that agile and plan-driven practices are mutually enabling and necessary for coordination and scaling in large-scale agile projects. We develop nine propositions for large-scale agile projects based on these new alternative underlying assumptions. Finally, we summarize our theoretical contribution in a generic process model of continuously adjusting agile and plan-driven practices in order to accommodate process challenges in large-scale agile projects.

CCS Concepts: • **Software and its engineering** → **Agile software development**; **Software development process management**; *Collaboration in software development*;

Additional Key Words and Phrases: Large-scale agile, problematization, assumptions, literature review, case study, software architecture, requirements engineering, multiteam project management

This publication was financially supported by Science Foundation Ireland under grant nos. 13/RC/2094_P2 and 15/SIRG/3293, and by the competence-building project Agile 2.0, supported by the Research Council of Norway through grant no. 236759 and by the companies DNV GL, Equinor, Kantega, Kongsberg Defence & Aerospace, Sopra Steria, and Sticos. For the purpose of Open Access, the authors have applied a CC-BY public copyright license to any Author Accepted Manuscript version arising from this submission.

Authors' addresses: K. H. Rolland, Department of Informatics, University of Oslo, Ole-Johan Dahls hus Gaustadalléen 23B, 0373 Oslo; B. Fitzgerald, Lero—the Science Foundation Ireland Research Centre for Software, Tierney Building, Department of Computer Science and Information Systems, University of Limerick, Limerick, V94 NYD Ireland; e-mail: brian.fitzgerald@lero.ie; T. Dingsøyr, Department of Computer Science, Norwegian University of Science and Technology (NTNU), Center for Effective Digitalization of the Public Sector, SimulaMet, IT Building, Sem Sælandsvei 9, Gløshaugen, 7491 Trondheim, Norway; e-mail: torgeir.dingsoyr@ntnu.no; K.-J. Stol, School of Computer Science and Information Technology, Western Gateway Building, University College Cork, Western Road, Cork, Lero—the Science Foundation Ireland Research Centre for Software, Cork, T12 XF62 Ireland, and SINTEF, Strindvegen 4, Department of IT Management, P.O. Box 134, 1325 Lysaker, Norway; e-mail: k.stol@ucc.ie.

1049-331X/2023/12-ART33
https://doi.org/10.1145/3617169

ACM Reference format:
Knut H. Rolland, Brian Fitzgerald, Torgeir Dingsøyr, and Klaas-Jan Stol. 2023. Acrobats and Safety Nets: Problematizing Large-Scale Agile Software Development. *ACM Trans. Softw. Eng. Methodol.* 33, 2, Article 33 (December 2023), 45 pages.
https://doi.org/10.1145/3617169

1 INTRODUCTION

The Acrobat's Pole and Safety Net
Picture a circus acrobat on a tightrope; two elements are obvious. One, a long pole which twitches as the acrobat wobbles in maintaining balance on the tightrope — this local flexibility is vital to maintain overall stability. The second, a safety net, is also crucial, but for a completely different reason. The acrobat could practice close to the ground but in order to scale to greater heights (and survive the process), the safety net is required — just as the brakes in a car permit faster driving, the safety net permits the acrobat to scale to greater heights.
Inspired by Bateson [1972] and Farjoun [2010]

Agile methods continue to increase in popularity and are by far the dominant mode of software development today [State of Agile 2022]. Agile methods were initially suggested to be best suited to (a) small projects with (b) co-located teams, and (c) non-critical projects [Abrahamsson et al. 2009; Williams and Cockburn 2003]. These constraints have been challenged and there are now many successful exemplars of the use of agile methods in distributed environments (e.g., Moe et al. [2014]), as well as the use of agile in safety-critical and regulated domains, including aerospace [Hanssen et al. 2017], automotive [Hilt et al. 2016], life-sciences [Fitzgerald et al. 2013], medical devices [McCaffery et al. 2016], and railway [Stålhane et al. 2012]. However, the successful use of agile methods in large projects remains an outstanding challenge [Booch 2015; Dingsøyr et al. 2019; Edison et al. 2021].

To assist organizations in 'scaling up' agile principles to large-scale projects, a multitude of solutions have been proposed. These methods include Water-Scrum-Fall, Large-Scale Scrum (LeSS), the Scaled Agile Framework (SAFe), Disciplined Agile Delivery (DAD), Nexus, and hybrid processes [Edison et al. 2021; Conboy and Carroll 2019; Dingsøyr et al. 2019; Šmite et al. 2019]. However, as we illustrate in this article, these solutions apply the fundamental assumptions underpinning agile methods in large-scale development settings, without critically questioning or challenging them. This, we argue, contributes to an over-reliance on principles and practices as encoded in agile methods in contexts where those no longer apply, leading to confusion and misunderstanding, including a misplaced insistence that plan-driven practices are somehow 'undesirable' and that to be 'agile,' all practices must be compliant with the Agile Manifesto. While some literature acknowledges the potential co-existence of agile and plan-driven methods [Boehm 2002], and more recently hybrid approaches [Kuhrmann et al. 2018; Kuhrmann 2021], we argue that prior work falls short on providing any theoretical foundation to help understand and explain the tensions between these different approaches. To allow for a deeper understanding and to help resolve the perceived tensions between plan-driven and agile methods, we (a) propose to reposition this relationship as a *duality* rather than a *dualism*,[1] and (b) develop new theory to guide the field further.

[1]Summarizing briefly, dualism views paired concepts as separable, whereas duality views paired concepts as interdependent which cannot exist independently from each other [Giddens 1979; Jackson 1999].

Given that large-scale agile appears to be the ‘final frontier’ for agile methods [Booch 2015; Dingsøyr et al. 2019] and that the complexity and scale of such development projects have been perceived as requiring a plan-driven approach, we are particularly interested in challenging the fundamental assumptions underpinning large-scale agile development. Agile methods were originally positioned as lightweight methods in response to plan-driven approaches that were the industry norm up to the 1990s and which were characterized as ‘documentation-heavy’ and ‘rigid’ (phase-based) approaches [Boehm 2002]. Over the years, there has been an increased acknowledgment that even for large-scale projects, traditional plan-driven approaches no longer suffice (e.g., Ambler and Lines [2012]; West et al. [2011]). The need for agility is driven by changing market trends; organizations that cannot respond quickly to market demands will suffer consequences. Even behemoths such as Microsoft, which traditionally followed a document-heavy process [Cusumano and Selby 1997] have adopted agile methods, and organizations around the world seek to tailor agile methods to large-scale projects [Berntzen et al. 2023; Kuhrmann et al. 2021; Russo 2021].

Although reporting a successful case of scaling up agile methods to large projects in itself is interesting, our goal is to go beyond this. Through a systematic analysis of the literature on large-scale agile, we observe a set of recurring patterns, namely, a set of assumptions in relation to the use of agile methods in large-scale projects. We adopt Alvesson and Sandberg’s [2011] problematization methodology to identify fundamental assumptions in prior research on this topic. We then challenge these assumptions and, drawing on an in-depth and longitudinal case study of a successful large-scale agile project, we propose a set of revised assumptions that provide a better theoretical foundation to capture the duality between agile and plan-driven practices. Using this new set of assumptions, we theorize on the specific balanced combinations of such practices to provide better insights regarding the scaling up of agile methods. To the best of our knowledge, this is the first attempt to theorize large-scale agile methods.

In this article, we take an alternative approach to the more common practice of gap spotting and adopt a problematization approach. This approach is not widely used within software engineering research. Hence, this article is also structured in an unconventional way (see Figure 1), consisting of two major phases. In Section 2, we outline Alvesson and Sandberg’s problematization method and its relevance for software engineering and agile software development. In particular, we draw on the framework of Farjoun [2010], which was originally designed to distinguish between dualism and duality, to help understand how agile and plan-driven methods are traditionally positioned as opposites. The first phase of our research approach is a systematic review of the large-scale literature through which we identify three major assumptions—Section 3 presents the details of this review. We then shift to the second phase of our research, which is a longitudinal case study of a successful large-scale agile software development project. Section 4 presents the design of the case study. Section 5 presents the results of our analysis, which comprises a series of ‘episodes’ that illustrate in detail how the three assumptions were challenged, and how agile and plan-driven methods were successfully combined. We then revisit the framework of Farjoun [2010] in Section 6 to position and reflect on our observations, and propose a process model [Newman and Robey 1992] that explains the dynamics in large-scale agile software projects. We develop a set of alternative assumptions and further develop theory by deriving a series of propositions to guide future research. We conclude in Section 7 with an outlook on how our propositions might inform future work.

2 PROBLEMATIZING FUNDAMENTAL ASSUMPTIONS IN LARGE-SCALE AGILE SOFTWARE DEVELOPMENT

Alvesson and Sandberg [2011, 2013] propose the problematization of both conventional assumptions and the consensus in prevailing theoretical perspectives. They point out that the typical mode

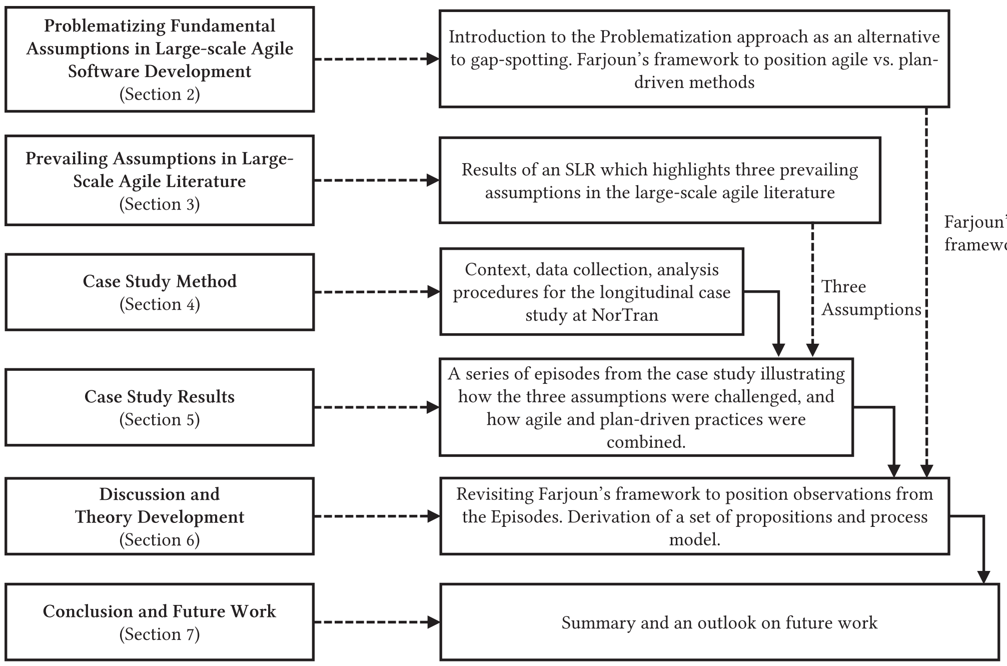
Problematizing Fundamental Assumptions in Large-scale Agile Software Development (Section 2)
Introduction to the Problematization approach as an alternative to gap-spotting. Farjoun's framework to position agile vs. plan-driven methods
Prevailing Assumptions in Large-Scale Agile Literature (Section 3)
Results of an SLR which highlights three prevailing assumptions in the large-scale agile literature
Farjoun's framework
Case Study Method (Section 4)
Context, data collection, analysis procedures for the longitudinal case study at NorTran
Three Assumptions
Case Study Results (Section 5)
A series of episodes from the case study illustrating how the three assumptions were challenged, and how agile and plan-driven practices were combined.
Discussion and Theory Development (Section 6)
Revisiting Farjoun's framework to position observations from the Episodes. Derivation of a set of propositions and process model.
Conclusion and Future Work (Section 7)
Summary and an outlook on future work

Fig. 1. Structure of this article.

of framing research is to identify research questions through spotting gaps (or even 'constructing' gaps) in current theories or empirical studies—what they have termed the *gap-spotting approach*. Research in software engineering and in many other scientific fields is expected to draw from, and build upon, established bodies of research. Researchers often construct their research questions through finding 'gaps' in an existing body of literature. In the context of large-scale agile software development, Paasivaara et al. [2012, p. 236] identified a 'gap' in the research literature: *"[We] found only three research articles briefly reporting experiences from projects with more than ten Scrum teams."* By pointing out the scarcity of research on Scrum projects that involve more than ten teams, the authors identified a gap upon which they position and legitimate their research on Scrum-of-Scrums.

The literature on agile software development offers several examples of consensus building around important research challenges [Gregory et al. 2016], complemented by special issues on agile systems development (e.g., Abrahamsson et al. [2009]). Most of these initiatives can be characterized as 'gap identifying,' with gaps being uncovered in prior research or gaps between the literature and industry needs.

While gap-spotting is clearly a useful approach that has led to significant contributions in most research fields, a fundamental problem with the gap-spotting approach is that it can result in *reinforcing* existing theories rather than *challenging* them. By adopting the assumptions from prior studies without critical reflection, these assumptions are amplified. Researchers tend to then build entirely from the premises of prior literature without challenging the consensus.

As a primarily practitioner-driven phenomenon, agile methods have been viewed as having relatively weak theoretical grounding [Abrahamsson et al. 2009; Conboy 2009]. Rather than 'constructing' a gap in the current large-scale agile literature, we adopt a problematization approach that challenges the prevalent assumptions in that literature to form an alternative perspective that

Table 1. Typology of Assumptions [Alvesson and Sandberg 2011]

Assumption Type	Description	Examples Related to Agile Literature
In-house assumption	Assumptions that are shared by subgroups of researchers within an area of research and which affect how they conceptualize or measure a particular subject matter. Each agile method can be seen as a 'school of thought,' including Scrum and XP.	The role of Product Owner within Scrum as a surrogate for the customer, and who is assumed to know the business requirements.
Root metaphor assumptions	Assumptions that signify a deeper aspect of the subject matter by using conceptual images to understand the topic of study.	"Waterfall" as a metaphor for the software development process, with strictly unidirectional phases separated by 'stage gates.'
Paradigmatic assumptions	Assumptions concerned with the underlying epistemological and ontological views of the research in the dominant literature. These views may relate to the research paradigm (e.g., positivist vs. constructivist research, or qualitative vs. quantitative), and also the theoretical paradigm.	Overemphasis on success stories and case studies which are typically analyzed from a positivistic viewpoint [Paasivaara et al. 2012].
Ideological assumptions	Ideological assumptions include those that relate to political, moral, and gender-related issues.	'Agile Manifesto' as an ideological statement: the term 'manifesto' is commonly used as a declaration or statement of intent in politics (e.g., "Communist Manifesto").
Field assumptions	Field assumptions can be identified within several different branches of a field of study.	The assumption that agile methods are best suited to small projects, with co-located teams, and for non-critical development contexts. These assumptions have long been held for all agile methods, but these have been challenged (e.g., Hanssen et al. [2017]).

can help us understand the dilemma between agility and rigidity seemingly inherent in large-scale development.

Alvesson and Sandberg [2011] provide methodological guidance in the form of a typology of assumptions and a series of 'principles' for identifying and challenging assumptions. The typology defines five types of assumptions (see Table 1). This typology provides useful prompts for researchers to identify candidate assumptions that can subsequently be further studied. In our study, we used this taxonomy to increase our sensitivity while reviewing prior agile literature (see Section 3).

The second part of the methodological guidance is a set of principles for identifying and problematizing assumptions (Alvesson and Sandberg 2011, p. 260]:

(1) Identify a domain of literature: What main bodies of literature and key texts make up the domain?
(2) Identify and articulate assumptions: What major assumptions underlie the literature within the identified domain?
(3) Evaluate articulated assumptions: Are the identified assumptions worthy of being challenged?
(4) Develop alternative assumptions: What alternative assumptions can be developed?

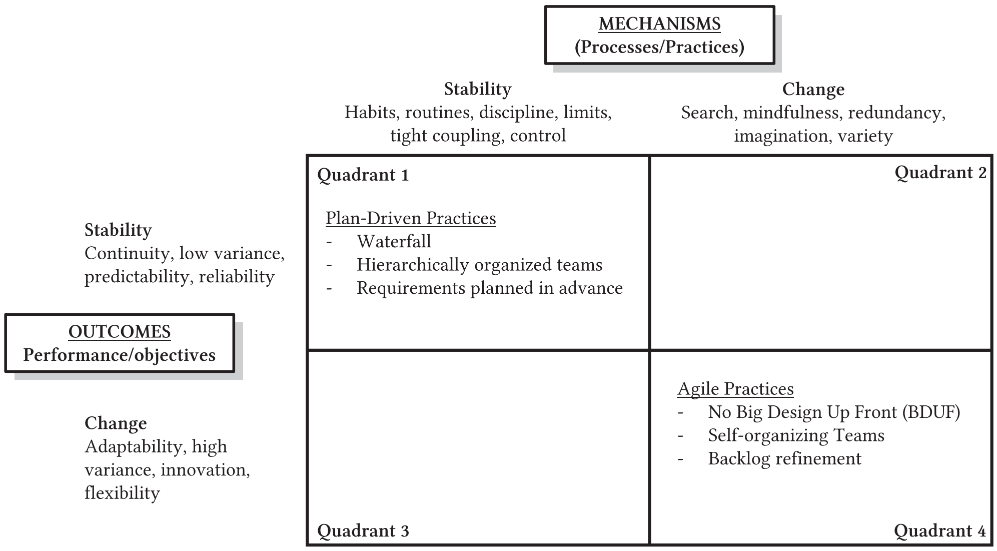
MECHANISMS
(Processes/Practices)
Stability
Habits, routines, discipline, limits, tight coupling, control
Change
Search, mindfulness, redundancy, imagination, variety
Quadrant 1
Plan-Driven Practices
- Waterfall
- Hierarchically organized teams
- Requirements planned in advance
Quadrant 2
Stability
Continuity, low variance, predictability, reliability
OUTCOMES
Performance/objectives
Change
Adaptability, high variance, innovation, flexibility
Agile Practices
- No Big Design Up Front (BDUF)
- Self-organizing Teams
- Backlog refinement
Quadrant 3
Quadrant 4

Fig. 2. Stability and change as mechanisms and outcomes to characterize plan-driven versus agile software development (based on Farjoun [2010]).

(5) Relate assumptions to audience: What major audiences hold the challenged assumptions?
(6) Evaluate alternative assumptions: Are the alternative assumptions likely to generate a theory that will be regarded as interesting by the audiences targeted?

In this article, we focus on the literature on large-scale agile software development specifically, rather than the agile literature more broadly (Step 1 above). We identify and articulate major assumptions in this body of literature (Step 2), and we argue in the same section that these are worthy of being challenged (Step 3), and demonstrate that these are prevalent in many papers on large-scale agile (Step 5). Through a longitudinal case study (Sections 4 and 5), we illustrate these assumptions and develop a set of alternative assumptions (Step 4) in Section 6. We seek to develop theory by deriving a series of propositions (Step 6) and discuss the implications of our alternative assumptions (Section 6).

To start our investigation, we draw on Farjoun [2010] to position plan-driven software development and agile methods (Figure 2). Farjoun proposes two key dimensions — namely, *mechanisms* and *outcomes* — both of which may feature stability or change. Mechanisms are the processes and practices that people employ, whereas outcomes are the objectives that people enacting those mechanisms seek to achieve. This conceptualization is especially relevant to, and resonates well with, the traditional positioning of plan-driven and agile approaches to software development.

The waterfall approach emerged as a reaction to the “software crisis” identified in the early years of the software field [Naur and Randell 1968] whereby most software development projects ended in failure to meet user needs. However, plan-driven, waterfall approaches led to lengthy, multi-year development times [Taylor and Standish 1982] without resulting in software that met user needs. This issue led to the emergence of the Agile Manifesto, which sought to address these issues.

Plan-driven approaches to software development were first and foremost seen as a way of securing a development process that is stable, predictable, and produces low variance [Boehm 2002; Cho 2009; Humphrey 1989; Vinekar et al. 2006]. For example, Figure 2 suggests that the waterfall approach, as a series of consecutive steps in the software development lifecycle that are separated by stage gates, is such a plan-driven mechanism. Another plan-driven mechanism (or practice) is to organize teams hierarchically, each responsible for a specific part (component) of the overall software system. Each of these software components is carefully planned and analyzed, resulting in a detailed set of fixed requirements before development starts. These are ‘stable’ mechanisms, rather than ‘change’ mechanisms, and traditionally have been thought to lead to stability, and associated attributes, such as predictability [Humphrey 1989; Vinekar et al. 2006].

In contrast, agile practices and processes are often seen as a way of achieving flexibility, innovative solutions, and adaptability [Conboy 2009]. Figure 2 suggests that agile practices such as no Big Design Up Front (BDUF), self-organizing teams, and backlog refinement are change mechanisms that lead to change outcomes, and associated attributes such as adaptability. From the start, agile and plan-driven approaches have been positioned as opposites, whereby plan-driven methods suggest the use of stable mechanisms leading to stable outcomes, and agile methods offer flexible mechanisms leading to changeable outcomes [Nerur et al. 2005; Vinekar et al. 2006].

As the figure suggests, this dichotomous positioning suggests that both plan-driven and agile methods are ‘extreme’ opposites. A number of empirical studies show that large-scale agile methods rely on a variety of underlying mechanisms, ways of organizing, and roles [Dingsøyr et al. 2018; Paasivaara and Lassenius 2014]. However, these studies tend to draw on the same underlying assumptions as small-scale agile projects [Rolland et al. 2016], as we discuss in the next section.

3 PREVAILING ASSUMPTIONS IN LARGE-SCALE AGILE LITERATURE

In the first step of our research, we conducted a systematic review of the relevant literature on large-scale agile development to identify the assumptions underpinning prior research. The goal of the literature review was to investigate common assumptions in this body of work which were specifically related to the nature of *large-scale* software development using agile methods. Appendix A presents the protocol for this literature review and describes details of our analysis. Our review comprised 67 relevant papers. This number is consistent with a recent systematic review of large-scale agile [Edison et al. 2021]; our number of papers is slightly lower as we did not include experience reports, but only empirical studies. The reading of the papers was divided among the author team. During the review process, we met at regular intervals, both in person and at online meetings. We also conducted several in-person full-day workshops that allowed for in-depth discussions. As each paper was read and discussed by the author team, we continuously asked: ‘*what assumption does the paper make in relation to the use of agile methods in large-scale settings?*’ For each paper, we kept short notes in temporary memos, which were subsequently discussed among the team. During the iterative analysis process, we identified three central and recurring assumptions that are key to large-scale software development: (1) agile and plan-driven methods are perceived to be mutually exclusive; (2) self-managing and hierarchically organized teams are perceived to be mutually exclusive; and (3) scaling of agile methods is seen as a simple linear composition.

Table 2 presents a fragment of the full table (Table A.1) in the appendix that captures relevant quotes in relation to these three assumptions that emerged during our analysis. For example, Bick et al. [2016] suggest: *“Top-down planning refers to a mechanistic, centralized approach. Bottom-up adjustment, on the on the other hand, is largely organic and decentralized.”* This clearly dichotomizes coordination approaches as seen from agile and plan-driven perspectives, thus emphasizing the assumption that self-organization and hierarchical coordination are mutually exclusive. Several other studies make similar observations that suggest the same assumption. Likewise, two of the references in Table 2 provide evidence for Assumption 3, namely, that agile methods can be scaled through “simple” linear composition. For example, Cho et al. [2006] suggest that the Scrum ceremony can be scaled up linearly by conducting a “Scrum of Scrums” (SoS), a meeting among Scrum Masters from different Scrum teams.

Table 2. Quotes from Selected Sources that Exhibit Existence of Assumptions

Reference	Assumption 1: Agile and plan-driven methods are perceived to be mutually exclusive.	Assumption 2: Self-organization and hierarchical coordination are perceived to be mutually exclusive.	Assumption 3: Scaling of agile methods is seen as a linear composition.
Bick et al. [2016]		Top-down planning refers to a mechanistic, centralized approach. Bottom-up adjustment, on the other hand, is largely organic and decentralized.	Scaling via Iterative Proxy Collaboration — CPO, SoS, central architecture team.
Cao et al. [2004]	Agile methods lack of up-front design and documentation.	We didn't have layers and layers of management. We got rid of those. Decentralizing development-oriented decision-making is critical for a successful agile — push decision-making down, empower the people who are actually doing the work.	
Cho et al. [2006]			The daily Scrum of Scrums is a daily meeting for SMs from multiple Scrum teams.
Costa et al. [2014]		In opposition to the previous methodologies, agile development processes are based on self-organized teams resolving their problems.	

Table 3. Frequency of Observed Assumptions in the Sample of Reviewed Papers (see Appendix Table A.1 for References to Papers)

Assumption	Number of papers
Assumption 1: Agile and Plan-driven Methods as Mutually Exclusive	40
Assumption 2: Self-Organization and Hierarchical Coordination as Mutually Exclusive	33
Assumption 3: Scaling through Simple Linear Composition	37

As mentioned, Table A.1 in the appendix presents extracted text from all 67 papers included in our review. Table 3 summarizes the frequency of observations of these three assumptions in the reviewed papers. Taken together, these quotes provide evidence that these three assumptions are widespread throughout the large-scale agile literature. We now discuss these core assumptions in more detail to explain what they mean and how they manifest in the literature.

3.1 Assumption 1: Agile and Plan-Driven Methods are Mutually Exclusive

From the outset, agile methods were positioned to be in contrast to so-called plan-driven methods which followed a waterfall life-cycle (Beck and Boehm 2003; Boehm and Turner 2003]. Table A.1 (see appendix, column 2) lists several such claims that contrast agile and plan-driven methods; for example, Cao et al. [2004] wrote: *"Agile methods lack up-front design and documentation."*

Agile advocates argued that all too often development processes were ‘plan-oriented’ rather than ‘reality oriented’ in that the software produced by traditional processes failed to meet actual business needs, despite significant and formal up-front planning. This view is captured in the acronyms BDUF (Big Design Up Front) and YAGNI (You Ain’t Gonna Need It) [Abrahamsson et al. 2010]. Agile is characterized as avoiding this ineffective planning, with the assumption in Extreme Programming (XP) that a system metaphor can be sufficient to guide development. The agile approach is thus positioned as a reaction against traditional ‘relay-style’ development with exhaustive up-front planning and design [Hannay and Benestad 2010]. Being adaptable to change may be seen to be in contradiction to planning [Gunyho and Gutiérrez Plaza 2011]. The fundamental assumption behind traditional methods is that systems are fully specifiable and built through meticulous and extensive planning. Agile methods, on the other hand, assume that systems can be built from scratch through continuous design, improvement, and testing based on rapid feedback while responding to changing requirements [Nerur et al. 2005; Dingsøyr et al. 2018].

Several researchers have suggested ‘decision rules’ to determine whether an agile method would be more appropriate than a plan-driven approach [cf. Boehm 2002; Boehm and Turner 2003; Hobbs and Petit 2017]. However, when transitioning to agile, Weiss and Brune [2017] suggested staying *“as close as possible to pre-defined agile practices,”* citing Boehm and Turner’s [2003] rationale that it is more effective to *“build up processes rather than tailoring them down.”* Note that studies of “small scale” development suggest that methods are “adopted and adapted” to suit the context of development [Dittrich et al. 2020] and there are often deviations between formal descriptions of methods and “methods-in-action” [Fitzgerald 1996].

The Agile Manifesto suggests that the nature of planning in agile projects is emergent (e.g., Adikari et al. [2009]; Bjarnason et al. [2011]; Ernst and Murphy [2012]; Ramesh et al. [2012]). The ceremony of ‘backlog refinement’ further illustrates the assumption of emergent planning. The backlog reflects the requirements as they are known and prioritized, and the assumption is that, as additional learning occurs during development, the backlog planning estimates can be updated or refined in preparation for development. The fact that requirements inevitably change (or become obsolete), allied to the cost of design carry-through from earlier stages, mitigates against upfront planning [Elshamy and Elssamadisy 2006]. At best, *“a little design up front approach”* might be used, for example, an ‘architecture spike’ might be considered to establish an initial architectural platform, but overall, these spikes are acknowledged as exceptions which are counter to agile principles [Alsaqaf et al. 2017].

3.2 Assumption 2: Self-managing Teams and Hierarchically-organized Teams are Mutually Exclusive

One of the key ideas in agile software development lies in the Agile Manifesto principle that the *“best architectures, requirements, and designs emerge from self-organizing teams.”* The notion of self-managing teams is certainly a fundamental assumption for early influential advocates of agile methods [Fowler and Highsmith 2001], and is a key aspect in Scrum [Deemer et al. 2010; Hoda et al. 2013]. Table A.1 (see appendix, column 3) lists the claims in several papers that contrast self-management and traditional top-down organization. For example, Bass and Haxby [2019] wrote: *“Self-organizing teams relinquish some autonomy toward an architecture board or design authority that determines common policies and approaches.”* This assumption is also present in large-scale frameworks such as Large-Scale Scrum (LeSS) and the Scaled Agile Framework (SAFe) which state that a team should be a *“self-organizing, self-managing, and cross-functional group of five to nine people”* [Leffingwell 2016]. Eckstein [2016] states that while some believe that a large team cannot be self-organized, establishing a hierarchy for control *“is often understood as a sign of mistrust, which has negative effects on morale.”* A particular challenge when migrating to agile methods is the change from a hierarchically-oriented organization with heavy specialization to self-managing teams. The roots of self-organization in software development have been traced back to research on complex adaptive systems and complexity theory [Nerur at al. 2010; Vidgen and Wang 2009]. With self-management, the focus is *"more on adding value to the team's primary objectives rather than on their job functions alone"* [Nerur et al. 2010].

Maruping et al. [2009] suggest that autonomy can be both beneficial and detrimental in an agile development context. However, we did not identify any studies on large-scale agile development that directly challenge the role of autonomy. On the contrary, current research tends to take self-managed teams for granted, even as the scale increases [Gustavsson 2018]. For example, Paasivaara and Lassenius [2019] argue that *"the whole development organization [can form] a single empowered community,"* suggesting that empowerment can be driven by self-organization. Schwaber [2013] cautions that traditional coordination practices can suffocate team autonomy and thus restrict self-management.

Hoda and Murugesan [2016] identify several challenges that arise from self-management. They suggest that self-management will be problematic in trying to establish cross-functional teams, as there will be a natural tendency in self-managing teams to choose specializations that suit the skills and preferences of team members. Šablis and Šmite [2016] contrast the bottom-up 'empower and reflect' style of team management with the top-down command and control style, and associate the former with the agile concept of 'team autonomy.' However, some studies identify the need for boundaries on autonomy and flexibility. For example, Hannay and Benestad [2010] and Dingsøyr et al. [2018] report restrictions on autonomy in order to provide control in large programs. Likewise, Bass and Haxby [2019] identify the need for restrictions on team self-governance to ensure compliance with standards and guidelines. For example, the agile concept of 'emergent architecture' may lead to noncompliance with architectural standards and rules set within an organization.

It has been suggested that scaling of self-managing teams should be organic [Tessem and Maurer 2007] with the addition of specialist roles (e.g., a champion to elicit top management support for team self-management [Hoda and Murugesan 2016]). A significant facilitator of self-organization is the notion of direct face-to-face communication, also a core theme of agile methods. Direct communication facilitates timely feedback and reflection on activities that have taken place. This, in turn, facilitates continuous improvement [Batra et al. 2011], which is the goal of the Sprint Retrospective ceremony in Scrum, for example. In large projects with distributed teams, distortion of communication can occur across organizational layers [Dingsøyr et al. 2014].

3.3 Assumption 3: Linear Composition versus Multi-faceted Scaling

There are several examples which illustrate the assumption that agile activities can be scaled unproblematically in a straightforward linear fashion. The Scrum of Scrums concept [Arseni 2016] is a clear example of this, with all Scrum Masters meeting for a higher-level Scrum meeting, 'metascrum' [Dingsøyr et al. 2017], or 'forum of forums' [Šāblis and Šmite 2016], or even 'Scrum-of-Scrum-of-Scrums (SoSoS)' [Paasivaara et al. 2012]. Gupta et al. [2017] propose a linear temporal scale of daily, weekly, and bi-weekly Scrum of Scrums. Even in terms of organizing the transformation from plan-driven to agile, Laanti [2017] suggests that work should be arranged into *"backlogs of smaller batches"* as opposed to the traditional model, again based on the assumption that reductionism is possible in transforming from a large plan to a scaled-down version.

Linear composition is also evident in how roles are amalgamated across teams. For example, in relation to the Product Owner role in Scrum, both Gustavsson [2017] and Putta et al. [2018] suggest the role of Area Product Owner who would report to the Chief Product Owner, a role also identified by Bick et al. [2016]. Similarly, Gupta et al. [2017] identify Chief Product Owner and

Chief Scrum Master roles, who lead teams of Product Owners and Scrum Masters, respectively. Arseni [2016] suggest a "product owner team" to manage the product backlog.

This linear scaling also appears in the assumption that complex requirements and features, which may be expressed in epics, can be broken down into smaller tasks or user stories that can be accomplished by small teams over a shorter time period [Sekitoleko et al. 2014]. However, breaking down development work in this manner serves to mask interdependencies which may spill across teams [Crowston et al. 2016].

3.4 Summary

In summary, the fundamental assumptions underpinning agile tend to be treated as sacrosanct in the literature, to the extent that some approaches suggest restructuring the organization and breaking it down into smaller units rather than altering the basic tenets of the agile methodology [Crowston et al. 2016]. These fundamental assumptions are important because they guide how developers and other stakeholders respond to the typical process challenges in large-scale agile. These mutually exclusive assumptions also tend to be seen as a *dualism*, in that it is one or the other. However, as we will argue subsequently, these assumptions are better conceived as a *duality* in that they mutually reinforce each other.

4 CASE STUDY METHOD

In the second step of this research, we carried out a longitudinal case study [Runeson and Höst 2009] between September 2014 and December 2019 of a successful large-scale agile software development project in a governmental organization (hereafter referred to as 'NorTran') in the transport sector with 7,000 employees across 70 locations. The project was business critical and involved a core legacy system which had been running for many years. The focus of the case study was to understand changes in the development process during the project. This was one of three cases studied in the research project Agile 2.0 [Dingsøyr et al. 2018, Dingsøyr et al. 2023] where one key topic was "large-scale agile development."

NorTran signed a contract for this project with a major consulting company, ConsultCorp (a pseudonym). ConsultCorp had experience with large-scale agile at another government agency. The project was a large agile project, involving more than 120 participants over a 4-year period. Hence, the selection of this case gave us a unique opportunity to study the adaptation and scaling of agile methods in a large project in a complex organizational setting. The project is further described in Section 5.1. In the following, we present the information on data sources and data analysis procedures.

4.1 Data Collection

The case study draws on multiple data sources (see Table 4). The main source of empirical data is 27 in-depth semi-structured interviews with participants from both the customer and ConsultCorp, conducted by the first and third authors. Participants were carefully selected based on their level of participation and knowledge of the project. We interviewed informants across all roles in the project, including project managers, designers, architects, developers, testers, and Scrum Masters. In addition, we interviewed participants on the customer-side of the project — including the project manager, test manager, project architects, user representatives, and domain experts. The interviews ranged from 60 to 90 minutes and were transcribed. The interviews were done towards the end and after project completion.

The semi-structured nature of the interviews encouraged participants to speak freely, to share their insights and interpretations, and to steer the topics of discussion as they saw fit. However, a checklist of topics was used to guide the interviews and included the agile and non-agile

Table 4. Summary of Data Sources for the Longitudinal Case Study

Interviews	Workshops and Meetings	Supplementary Sources
27 interviews with informants from both NorTran and ConsultCorp, conducted over a period of 24 months, including: • Project managers • Designers • Architects • Developers • Testers • Scrum Masters • Test manager • Project architects • User representatives • Domain experts	Eleven meetings with ConsultCorp to discuss historical and contextual insights on the project, organizational aspects, agile practices preferred by consultants, commercial and competence aspects. Two workshops focusing on specific issues identified by participants as important for project outcome.	Unrestricted access to project documentation, including issue tracker, internal wiki containing material on user stories, contract documents, and other project management documentation.

practices being used, team coordination, scaling of practices and organizing, integration challenges, and collaboration with customers and stakeholders. As the interviews were conducted over a prolonged period, the checklist was refined over time in order to cover emerging themes. A second source of data was a series of 11 meetings with ConsultCorp. We used the information from these meetings to inform further in-depth interviews. These meetings also provided important historical and contextual insights on the project, the initial assumptions behind its organization, and the agile practices preferred by the consultants. A third important source of data was two workshops, which focused on specific issues the participants identified as especially important for the outcomes of the project at different phases in the process. Finally, a fourth data source was the project documentation to which we had unrestricted access, including the issue tracker and an internal wiki containing material comprising all user stories, contract documents, and other documentation used by project management. This triangulation of data sources, as well as the triangulation among researchers, were two tactics that helped us to better establish the credibility of this study [Lincoln and Guba 1985].

4.2 Data Analysis

First, drawing on the guidelines by Miles and Huberman [1994], we conducted an extensive process of descriptive coding over three iterations of the transcribed interviews emphasizing characteristics of practices, processes, and major events such as deliverables, changes in organizing, and roles in the project. The dominant data source was the interview material. Examples of such descriptive codes included *method adaptation, integration team, refactoring, ready-to-sprint process, champion roles, architect,* and *developer.* A total of 58 descriptive codes were used. As recommended by Miles and Huberman [1994], some of these were defined upfront as 'seed codes.' Some initial codes were changed in later iterations, and a substantial number of codes were added during the first iteration of descriptive coding. We used the software package HyperResearch to support the qualitative analysis. Second, we linked the relevant descriptive categories to the assumptions that we identified in the literature review. For example, the information coded as 'champion roles' was linked to the issue of 'self-managing teams versus hierarchically organized teams' since the champion role is an example of how teams became *less* self-managing as they needed to relate to how other teams worked and solved similar problems. Consequently, it does not fit the underlying assumption in the agile literature. Similarly, the 'ready-to-sprint process' code was linked to the theme of 'agile versus plan-driven' as it can be characterized as a plan-driven activity that was added with the aim of making each individual team more agile, thus illustrating that it is not a question of either agile or plan-driven practices, but rather how they are *combined*. Finally, we also linked the relevant descriptive codes to the identified theme of 'simple linear composition versus complex multi-faceted scaling.' One example of this is the 'code architect.' This was used to tag information regarding how the architect role was practiced and how it changed over time in the project to facilitate scaling of the project in a non-linear fashion.

In order to focus the analysis on tension points that arise in a large-scale agile development setting and how these tensions unfold over time, we drew on the social process model of Newman and Robey [1992]. We used the temporal bracketing strategy [Langley 1999; Langley et al. 2013] in order to focus on specific, critical phases in the project. This strategy helps to decompose critical events in a *"stream of longitudinal data"* and *"are constructed as progressions of events and activities separated by identifiable discontinuities in the temporal flow"* [Langley et al. 2013]. Temporal bracketing is frequently used to identify phases or stages; in this study, we also identified critical events that caused "discontinuities" in how the project was run. Our focus is not on demonstrating a strict sequence of events or identifying phases or stages, but instead on the shifts that took place that separate the "before" from the "after." For this reason, unlike many other studies using this strategy, e.g., Sabherwal et al. [2001], we do not present a timeline of events to identify phases or stages.

We drew primarily on interview data and on information from the two workshops at which the participants identified the major events in the project. Workshops were used to check preliminary analysis results with informants as a form of member checking [Lincoln and Guba 1985]. We identified six episodes that were triggered by the tensions represented in the three assumptions in the literature. This 'process approach' complements a 'factor approach' that has been more commonly used in other studies, which have focused on 'success factors' for large-scale agile transformations [Dikert et al. 2016; Russo 2021]. The latter focuses on identifying conditions or predictors that give rise to a certain outcome, but the process by which this outcome is achieved remains a 'black box.' The process approach that we followed, on the other hand, opens up that black box by identifying the antecedents and the sequence of events that led to a particular outcome. The process approach seeks to identify 'triggering events' that challenge existing ways of organizing development and development practices, along with episodes in which the project responds to those challenges.

5 CASE STUDY RESULTS

We first provide a brief history of the case project. Then, using a process analysis approach, we present the findings according to the three broad assumptions identified in the literature (see Section 3) and illustrate through six episodes how these assumptions were challenged. This is summarized in Table 5 and discussed in detail below. Apart from these six episodes, several others were identified, which are summarized in Appendix B.

5.1 The Project

The project used a delivery model based on the Scrum development framework with four different deliverables involving sets of new features to be put into production at the same time. The incorporation of agile methods was chosen by NorTran in order to *"maximize flexibility, and to avoid specifying all details up front,"* as the NorTran project manager pointed out. In our case analysis, we refer to the first two deliverables as the first phase (Episodes 1 and 2) and the final two as the second phase (Episodes 3 to 6) because of the differences in the development process that was used in these two phases. From NorTran's perspective, the first phase was a failure overall, as they only developed around 75% of the expected scope. During this phase, it became evident to NorTran that they lacked the competence and experience to implement large-scale agile software development projects. This had unfortunate consequences for the capability of the project

Table 5. Dominant Assumptions from Literature on Large-Scale Agile Software Development Identified in Episodes of the Case Study

Dominant assumption	**Episode description**	**Challenging the assumption**
Agile and Plan-Driven Methods Are Perceived to be Mutually Exclusive	*Episode 1: Establishing "Tornado meetings"* In response to initial challenges with the agile approach to tackle architectural issues, a plan-driven practice called the "Tornado meeting" was introduced as an arena for solving problems related to architecture and developing better user stories. This practice also increased common understanding across teams and roles.	Plan-driven practice: Tornado meetings as stage-gate to up-front planning, which also served as a means to coordinate a common solution across teams. This stage-gate helped to resolve issues, allowing the agile development process to proceed more smoothly.
	Episode 2: Establishing "Ready-to-sprint process" Triggered by increasing complexity and interdependencies across teams, a new plan-driven practice referred to as "ready-to-sprint process" was established. This improved the capability to recognize important interdependencies. In turn, this reinforced agile practices within teams.	Plan-driven practice: Ready-to-sprint as a stage-gate prior to sprint planning. Allowed teams to be more flexible and reduced need for coordination. Plan-driven practice: Definition of Prepared to ensure greater standardization and agreement across development teams.
Self-managing Teams and Hierarchically Organized Teams Are Perceived to be Mutually Exclusive	*Episode 3: Reorganizing the project* Triggered by problems due to distribution of integration tasks across all teams, management decided to reorganize the project. The project was reorganized from generic feature teams into more specialized teams. One "integration team" was established as a means for reducing interdependencies across teams. This both reduced and increased the teams' level of self-management.	Plan-driven practice: Central integration team as 'control structure' to take ownership of integration; this additional top-down mechanism took away some level of management but also helped teams to self-organize; thus, self-management and hierarchical organization are not mutually exclusive.
	Episode 4: Introducing cross-team roles and task forces A bottom-up initiative to establish "champion roles" for specific technology areas for supporting all teams. Also introducing temporary "task forces" to solve specific problems that were show-stoppers for work within the teams. This both reduced and increased the teams' level of self-management.	Agile practice: Cross-team champions who set standards and provided advice to others. Task forces as multi-skilled and self-organizing teams to address emergent technical problems, which were dissolved after a problem was resolved.
Scaling through Simple Linear Composition	*Episode 5: Scaling across complex interdependencies and multiple actors* Triggered by the complexity of multiple actors and systems to be integrated in the project, architect roles and meetings were scaled. Architect roles and architecture meetings were added in order to deal with continuously reoccurring architectural challenges. Plan-driven practices were also needed for coordinating activities with external actors and projects.	Plan-driven practice: Additional meetings among architects to facilitate more fine-grained coordination.

(Continued)

Table 5. Continued

Dominant assumption	Episode description	Challenging the assumption
		Agile practice: Architect role shifted from setting the architecture to more informal roles of 'facilitator' and 'knowledge broker'; this contradicts the concept of 'linear composition' where architects would become 'supervisors.'
	Episode 6: Downscaling Toward the end of the project, it became essential to reduce the number of teams and developers without reducing quality and productivity. Some developers were given a "freelance role" outside teams.	Agile practice: New 'freelance role' to take on work outside teams to work on issues that required more personnel. Scaling down was just not the reverse of "scaling up," not merely taking people off the project, but rather through a more flexible process to ensure quality.

to scale. In turn, this was one of the main causes for the failure of the first phase. According to the contractual agreement, ConsultCorp established a project with four teams consisting of more than 50 participants working full-time. However, NorTran, who hired ConsultCorp, had only four dedicated participants. This imbalance in staff numbers became evident when the customer side of the project struggled to develop user stories, with dependencies identified, in time for the next sprint. A NorTran architect described the situation as follows:

> "It is hard to follow the supplier [ConsultCorp] and answer their requests. They want to work in an iterative and agile manner, and expect to have two or three domain experts in each team. We were never able to deliver those resources — we were not organized for that." (Software Architect, NorTran)

The typical Scrum team in the project consisted of around 12 people — 7 developers, one User-Experience (UX) designer, one technical architect, one functional architect, one domain expert who served as a customer proxy representative, and one Scrum Master who also served in the more traditional role of team lead.

The project was organized as a matrix organization, similar to other large-scale agile projects (e.g., [Dingsøyr et al. 2018]), implying that a person would be a member of a specific team as well as a member of an organizational unit for Business, Architecture, Development, Maintenance, Test, or Infrastructure. The organizing structure was partly replicated for both the customer (NorTran) and the consultancy company (see Figure 3).

The Scrum-based delivery model involved splitting this large project into a set of deliverables. For each deliverable, the process involved first defining user stories on a low level, then architectural design, overall user experience (UX) design, and refinement of user stories — but with minimal effort in order to refrain from too much up-front planning.

5.2 Combining Agile and Plan-driven Practices

Rather than viewing agile and plan-driven approaches as mutually exclusive (Assumption 1), the project drew upon both agile and plan-driven practices in *combination.* In fact, these different practices were interdependent and strengthened each other: plan-driven practices *increased* the agility

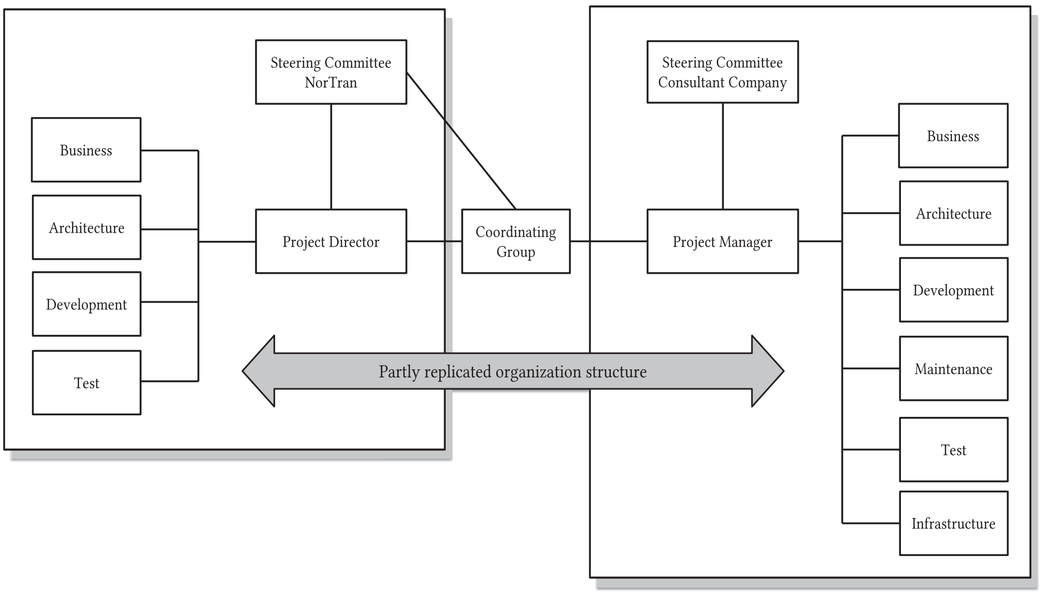

Fig. 3. Project organization.

of the teams; in turn, the agile mindset of bottom-up process innovation helped establish new plan-driven practices. The combination of these practices was triggered by challenging situations and events in the project.

5.2.1 Episode 1: Establishing "Tornado meetings". During the first phase, this episode was triggered by the initial challenge of handling complex architectural issues while using an agile approach with iterative short sprints. After the first phase, the project was almost a year behind schedule. The project ran into considerable difficulties related to the initial software architecture, technical issues concerning integration with other systems, and scaling the project with additional feature teams and customer representatives. One of the most pressing issues was that the initial plan for delivering the architecture with standard off-the-shelf components did not meet the requirements of the project. Hence, there was also a considerable scope creep in this phase.

In order to ensure discovery of such major obstacles at an earlier stage, a new ceremony referred to as the "Tornado meeting" was established prior to starting work on any future deliverable. This term was used because six groups would 'throw around' user stories without a set agenda and the discussions could take unexpected directions. This practice added a stage in up-front planning. Representatives from NorTran and ConsultCorp would meet, including functional architects, technical architects, testers, and the graphical user interface designer. One of the pressing problems solved in the Tornado meeting was an architectural issue with off-the-shelf components:

> "This problem with off-the-shelf architectural products emerged particularly during the Tornado meetings. We were supposed to use off-the-shelf products, but these turned out not to meet user needs. We realized that it was too complicated to solve with standardized products. It was less costly to develop everything ourselves. This made integration easier." (Technical Architect, ConsultCorp)

Hence, in this episode, the failure to deliver according to plan that resulted from the problems of integrating different off-the-shelf products triggered the need for better planning and problem-solving across teams. The new practice that emerged can be characterized as primarily plan driven because it introduced a new type of meeting that sought to establish a plan for further development but also because issues were identified, analyzed, and solved during these Tornado meetings rather than letting these issues emerge during development and dealing with them as needed. This, in turn, allowed the agile development process to proceed more smoothly. Recognizing that the introduction of new plan-driven practices was "in conflict" with the agile approach that NorTran sought to follow, a functional architect described Tornado meetings as *"a pestilence and plague, but one of the smartest things we introduced."* Despite being seen as "non-agile," it was deemed highly necessary and successful by participants, illustrating that plan-driven practices and agile methods are not mutually exclusive. The new Tornado meetings facilitated the coordination of a common solution across teams. Because of this new practice, the upfront architectural design process was completely redesigned, and was described as *"extreme service-oriented architecture."* The 'extreme' prefix was chosen to reflect the agility of responding rapidly to the architecture not meeting the actual user needs. In addition to agile practices such as refactoring, short iterations, and prototyping, more plan-driven activities were introduced. These plan-driven practices were not a substitute for agile practices; rather, they were complementary practices.

5.2.2 Episode 2: Establishing "Ready-to-sprint process". As the project progressed, it increasingly faced challenges arising from interdependencies between different teams. This triggered a need for more comprehensive coordination *prior* to commencing the regular cadence of development sprints. An additional up-front planning step called the 'ready-to-sprint process' was established. This was a formal meeting which was practiced frequently during sprints, not to be confused with the 'Sprint 0' concept that takes place *once,* prior to commencing the sprint schedule. The ready-to-sprint process was introduced in order to optimize completion of user stories. Thus, similar to, but more extensive than the continuous agile practice of 'backlog refinement,' this practice involved more people and was done in a three-day seminar, making it a stage-gate prior to sprint planning.

Before commencing work on user stories, the solution description was discussed by technical and functional architects from both customer and provider to that ensure the main design decisions were fully discussed and that work would not be delayed when assigned to an actual development Scrum team. This practice was described as ensuring agreement on what was called the 'Definition of Prepared' (Functional Architect, ConsultCorp), mirroring the agile practice of 'Definition of Done.' This practice ensured greater standardization and agreement across development teams.

Again, this plan-driven practice of 'ready-to-sprint' was not in conflict (or mutually exclusive) with agile practices already in use, but rather complementary, in that it *increased* the flexibility of the teams as less coordination was necessary during subsequent sprints. A team leader described this process as *"key to ensuring good quality of user stories and flow of tasks"* (Development Manager, ConsultCorp). Such plan-driven practices also meant that the teams had a better overview of what to expect in the upcoming sprint. This, in turn, allowed developers and UX designers to select more appropriate agile practices, such as prototyping:

> "We had a team of customer representatives and developers. However, identifying how a solution should work is not necessarily trivial. It is hard to know if we really had a common understanding of new functionality. Although the ready-to-sprint process helped in that respect, there were misunderstandings. Thus, I started to develop prototypes." (Developer, ConsultCorp.)

In this way, the agile practice of introducing prototypes helped to improve communication and identify functional requirements. In contrast to the dominant assumption of agile and plan-driven practices being mutually exclusive, the introduction of plan-driven practices seemed to *enable* more agile practices such as prototyping, thus creating a positive outcome for the project.

5.3 Combining Hierarchical Organizing and Self-organizing

Self-management is a fundamental principle in agile development, in contrast to more hierarchical forms of management. Self-management is associated with benefits such as flexibility in solving tasks and increased employee satisfaction. In the project, there was a mixture of hierarchical ways of organizing and a degree of self-management within teams.

5.3.1 Episode 3: Reorganizing the Project. As described in Episode 1, the project experienced a major setback in the second deliverable, which led to a project reorganization. This reorganization was triggered by the complexity of integration tasks. All four teams were initially doing development work involving integration across external projects and systems. In particular, the project was supposed to follow the 'extreme service-oriented-architecture' practice as explained above. Consequently, all teams strove to implement integration through the architecture, but this led to misunderstandings and increased workload in coordination across teams. One developer explained how this was addressed through the forming of an integration team:

> "In the beginning we had purely generic teams where each team had their own modules. All teams were supposed to have the competence to do everything. But we soon realized that the scope was too large for any single team, so we established an integration team. They were responsible for connecting the different systems and making things function as a whole. Those who were interested in working on integration were then moved to the integration team." (Developer, ConsultCorp)

Project management reorganized the project based on competence and interest in doing back-end integration work, thereby *reducing* the self-management of the teams and increasing an element of hierarchical organization by management. However, later in the project, when the integration team had existed for some months, the distinction between the integration team and the other teams became more 'fluid':

> "During sprints there were several teams that worked on the same domain. We distributed the task among us. We got the integration part, but also helped some of the other teams." (Developer, ConsultCorp)

The self-organization now took place *within* the structures that were imposed from the top (such as the integration team mentioned above). These findings suggest that self-organization and hierarchical management are not mutually exclusive, and positioning these as opposing approaches is too simplistic. Rather, self-organization can still happen even within hierarchical management.

5.3.2 Episode 4: Introducing Cross-Team Roles and Task Forces. Triggered by the lack of specific competence or problem-solving capabilities within each team, new roles that cut across teams were introduced in Episode 4. The project introduced more than a dozen 'Champion' roles. These champions were individuals with specialized skills in a specific technology such as databases, service integration, GUI-related skills, and information security. The champions provided advice and reviewed solutions. At technical kick-offs for sprints, champions presented standards and 'recipes.' One informant explained:

> "The JavaScript Champion set the standard. He was the most skilled developer and taught others how to work with JavaScript across teams. He had a mandate to provide advice and tips on the internal chat-channel, to do code reviews." (Development Manager, ConsultCorp).

Many challenges identified throughout the project required solutions involving multiple project roles across different teams. One example related to the technical issue of combining thousands of source code components into a working software product. The project established a ‘task force’ that included people from the Operations department, from the customer, and from the ConsultCorp teams. Such a task force would work for “a week or two” (Developer, ConsultCorp) to solve technical problems in the automatic build process. Another task force was established to solve technical problems with the service bus, a component which should offer increased flexibility when integrating the new system with legacy systems.

The ‘task force’ practice was an agile reaction that involved the rapid and ad hoc creation of a multi-skilled and self-organizing team to address an emergent technical problem, which would then provide future stability for the overall development process to proceed as planned. After a task force had solved a problem, it was dissolved. While this practice provided a flexible solution, it did involve a form of role assignment, thus reducing teams’ level of self-management that is uncharacteristic of cross-functional, agile teams. In this case, the level of expertise necessary was not always present in teams, requiring a more cross-team solution.

5.4 Scaling as a Complex Sociotechnical and Ongoing Process

The literature on agile software development suggests that scaling is achieved in a linear fashion — for example, through establishing Scrum-of-Scrums as a coordinating mechanism, or by more sophisticated Communities-of-Practice [Paasivaara and Lassenius 2014]. In the project, scaling involved much more than setting up Scrum-of-Scrums. Importantly, scaling also involved changing practices and applying a new configuration of both agile and plan-driven practices in tandem. Our study revealed two specific episodes of scaling.

5.4.1 Episode 5: Scaling Across Complex Interdependencies and Actors. As noted in Episode 1 above, the Tornado meetings improved the development process by adding a crucial coordination mechanism for planning across teams. In these meetings, problems concerning the upfront architecture were resolved. However, the problems associated with scaling in terms of negotiating the more fine-grained details could not be solved indefinitely, but presented an ongoing challenge throughout the project. One architect stated that:

> “A significant challenge was how to manage transactions in the system, this was a discussion between architects and the chief architect. ... all teams worked on the same development and test servers. And if there were things causing problems — sometimes the servers would go down when introducing new code ... this was handled by coordinating amongst architects across the development teams.” (Developer, ConsultCorp)

In order to scale the architecture, a new meeting among architects was established at the end of Deliverable 1 to enable coordination and communication of details of the software architecture across different teams. Over time, the frequency and importance of these architecture meetings increased, as the different feature teams needed to use many of the same services when developing new features. Although each feature team was responsible for a specific domain, the interdependencies between domains increased, which made coordination and communication of these interdependencies increasingly important. Large-scale systems cannot simply be decomposed into smaller-scale components and features that are delivered by feature teams. Instead, the scaling requires considerable fine-grained coordination to identify and resolve key interdependencies.

The architect role also shifted in focus. While initially focused on the overall software architecture to ensure a maintainable, efficient, and evolvable architecture, the architect role evolved into a *translator role* between the feature teams and the rest of the project. As one architect noted: *"our role became a facilitator, a kind of front-end for developers"* (Developer, ConsultCorp). In this sense, the architects provided an important new coordination mechanism across the different development teams that increased stability and predictability.

The architect role was also very important in establishing contact and communicating with stakeholders *outside* the project. In this sense, architects became key 'knowledge brokers' [Pawlowski and Robey 2004]. Through their knowledge broker role, architects became essential for scaling and coordinating across complex interdependencies in the project by bridging across different stakeholders, such as user groups, system owners, and external actors. One informant described the complex interdependencies as follows:

> "When one other system under development was down, it almost stopped the entire project. There was a tight coupling between this other system and our project, which we had not accounted for initially. We realized that this system needed to follow the same production schedule as our project — although they were not part of the official project." (Technical Architect, NorTran)

This situation was particularly frustrating as it caused requirements to be unstable in ongoing sprints. Tight coupling between systems implied the need to have a common overall production plan. Hence, again, this situation triggered a more plan-driven practice for coordinating this:

> "We took responsibility for coordinating all the different projects and their production schedules, simply because we were absolutely dependent upon them. At first, the steering committee was only concerned with the current project, so I had to explain to them that we were equally dependent on other projects finishing — if not, we would not be able to put the project into production." (Technical Architect, NorTran)

This shows how both agile and plan-driven practices were essential for scaling the project. Plan-driven practices seemed necessary for preserving the stability of the project while adding new team members, integrating with an increasing number of systems, and involving an increasing number of stakeholders. Agile practices, in terms of the architects relying on more informal practices of facilitating and knowledge brokering across actors, were equally important. Again, this illustrates that large-scale agile projects are not simply scaled in a linear fashion or decomposed in a 'divide and conquer' fashion, but rather that the complexity of large projects must be addressed by a combination of both fine-grained coordination of interdependencies on the one hand, and new types of roles that facilitate knowledge sharing.

5.4.2 Episode 6: Downscaling. As the project progressed towards its final stages during the fourth deliverable, the need for resources to develop new features declined. Hence, down-scaling the number of developers and certain competencies became a crucial issue to reduce cost. Because of the pressure to deliver high quality software on time, balancing this was particularly difficult. This was solved not by simply taking developers off the project, but instead releasing some of them as 'freelancers' who were given a flexible role outside the teams to be able to help with pressing issues across all teams. This allowed management the flexibility to use extra personnel in critical situations where the customer wanted improvements in quality, testing, and small modifications. A project manager explained how these issues were related:

> "The big issue was downscaling. During a test phase you produce a backlog of items that need bug fixing. And it was the production of this backlog which was the bottleneck, not correcting the bugs… So, we had a situation where instead of

Table 6. Old and New Assumptions of Large-Scale Agile Software Development

Old Assumption	**New Assumption**
Agile and plan-driven processes are perceived to be in conflict or mutually exclusive.	Agile and plan-driven practices are mutually enabling.
Hierarchical organizing and self-organizing teams are perceived to be in conflict or mutually exclusive.	Hierarchical organization and self-organization have reciprocal impact on teams.
Scaling of agile practices is seen as linear (e.g., Scrum of Scrums), implying that scaling can be conducted through just adding more participants and increasing the scope without changing practices.	Scaling requires both stability and change simultaneously, and also involves downscaling.

> getting rid of developers, we freed them to do other productive work." (Project Manager Deliverable 3, ConsultCorp)

Again, this illustrates how the arrangement of different practices needed to change. The project could not downscale by just inverting the sequence of the upscaling. Instead, it relied on the new 'freelancer role' to ensure that the backlog with bug fixes was well managed while addressing all bugs. We characterize this as an agile practice: rather than planning which personnel are to be taken off the project, which is very difficult to do, some were now unassigned, allowing them to "roam" and take on work in a fashion that resembles self-organization and self-selection of tasks that is more typical of agile methods.

These last two episodes illustrate how scaling is not simply linear. Both upscaling and downscaling require flexible resources: when upscaling, a simple decomposition approach is not sufficient, but instead, flexible and informal roles are necessary to facilitate and share knowledge. When downscaling, developers are not simply taken off a project, but the resources can again be set free, flexibly and informally, in order to address emergent needs and challenges.

6 DISCUSSION AND THEORY DEVELOPMENT

We now revisit the underlying assumptions identified in our literature review, formulate alternative assumptions based on the case study findings, and then revisit the framework of Farjoun [2010] to develop a set of propositions on how plan-driven practices enable flexibility and agility, and how agile practices mitigate failure and increase predictability. In Section 6.4, we propose a process model for large-scale agile software development (see Figure 5). Finally, we discuss the main limitations of this study and present a set of research directions based on our propositions.

6.1 New Assumptions for Large-Scale Agile

A primary aim of this article was to uncover and problematize assumptions that underpin previous research on large-scale agile software development. These assumptions have far-reaching implications for how software engineering researchers frame their research, what research questions they ask, and subsequently what insights their research offers to practitioners. Our literature review identified three assumptions that are widespread in previous research on large-scale agile (Table 6). As described in the case study, these assumptions were not in accordance with stakeholders' responses to the process challenges in our case study. The case study provides detailed insight into how responses to different process challenges required the project teams to continuously adjust their practices and ways of working. Based on our case study, we suggest an alternative set of assumptions that more readily accommodate the paradox that the introduction of plan-driven elements appeared to make the project more agile.

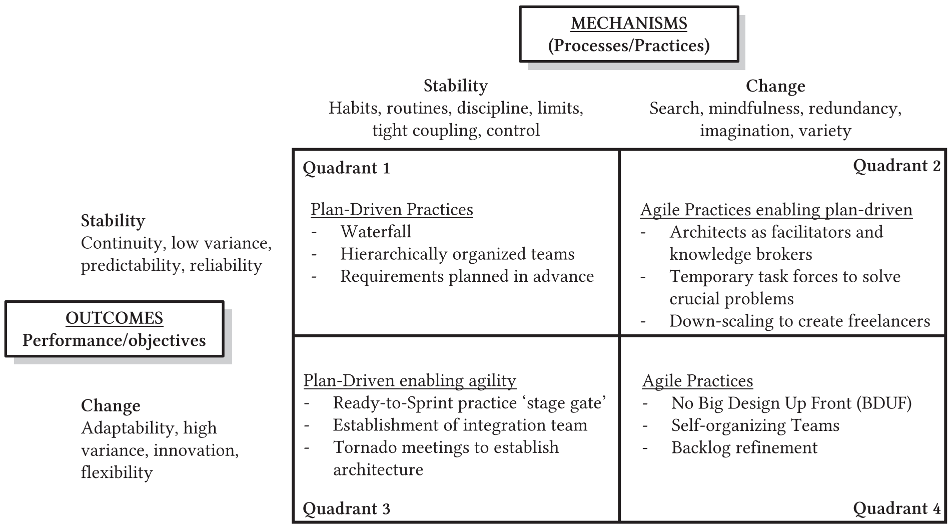
MECHANISMS
(Processes/Practices)
Stability
Habits, routines, discipline, limits, tight coupling, control
Change
Search, mindfulness, redundancy, imagination, variety
Stability
Continuity, low variance, predictability, reliability
OUTCOMES
Performance/objectives
Change
Adaptability, high variance, innovation, flexibility
Quadrant 1
Plan-Driven Practices
- Waterfall
- Hierarchically organized teams
- Requirements planned in advance
Quadrant 2
Agile Practices enabling plan-driven
- Architects as facilitators and knowledge brokers
- Temporary task forces to solve crucial problems
- Down-scaling to create freelancers
Plan-Driven enabling agility
- Ready-to-Sprint practice 'stage gate'
- Establishment of integration team
- Tornado meetings to establish architecture
Quadrant 3
Agile Practices
- No Big Design Up Front (BDUF)
- Self-organizing Teams
- Backlog refinement
Quadrant 4

Fig. 4. Stability and change as mechanisms and outcomes (based on Farjoun [2010]).

We anchor these alternative assumptions on large-scale agile development in Figure 4, which further elaborates the framework of Farjoun [2010] presented in Figure 2 earlier. We suggest that previous research on large-scale agile has operated on the *dualistic* assumption that agile processes and plan-driven processes are in opposition and incompatible; that hierarchical organizing and self-organizing teams are in opposition; and, finally, that scaling an agile project can simply be done in a linear fashion, without having to change practices regarding collaboration and communication. In contrast, we suggest that large-scale agile software development is better understood and explained through the notion of *duality*. Duality is similar to dualism in that it describes the relationship between two entities. However, if the relationship is a duality, then the two entities are interdependent, and both contradictory and complementary at the same time [Farjoun 2010]. This distinction is essential as it conceptualizes stability and change as *mutually enabling* rather than *mutually exclusive*. As the opening vignette of this article illustrates, in a duality, the acrobat's agility is enabled by the pole, but also fundamentally by the safety-net which provides the stability for the entire process.

Farjoun [2010] suggests that stability and change can be complementary in that stability is associated with low variance, predictability, and reliability and can produce change, and that change typically associated with innovation and flexibility is necessary for producing stability. Moreover, they can also be conflicting in the sense that practices that promote change provide less stability, and vice versa. Looking at the relationship as a duality affords more nuanced and balanced theorizations of seemingly paradoxical conditions and mechanisms that produce certain changes and forms of stability in complex organizational settings. Thus, we argue that this lens is highly relevant for understanding large-scale agile, as such processes typically involve a different arrangement of both agile and plan-driven mechanisms and practices to succeed [Rolland et al. 2016].

The notion of duality is not new, but we adopt it here specifically for understanding how large-scale agile software development is different from small-scale agile and that such processes need both plan-driven as well as agile practices in combination to succeed. It also underscores that in order for something to change in the first place, other parts need to be kept stable. Hence, scaling a small project to a large one does not solely rest on either agile or plan-driven practices — but always on a combination of both. We now revisit Figure 2 and populate two further quadrants to reflect this duality perspective in Figure 4. Henceforth, in the subsections below, we elaborate on: (1) how plan-driven practices enable flexibility and agility (Quadrant 3 in Figure 4); and (2) how agile practices mitigate failure and increase predictability (Quadrant 2). At the end of this section, we summarize how new alternative assumptions support a more nuanced understanding of large-scale agile, and how such complex processes can succeed.

6.2 Plan-Driven Practices Enabling Flexibility and Agility

All six episodes presented in Section 5 refer to events resulting from the large-scale nature of the project; the characteristics of large-scale projects led to a need for teams to coordinate, frequently after unexpected events caused problems for teams to solve. It is clear from prior literature that NorTran is not the only project that suffers from such issues. Batra et al. [2010] present a series of challenges that resulted from the characteristics of a large-scale project, including changing requirements, changing schedule, conflicting goals, and communication breakdowns. Previous literature reviews on this topic also reported numerous challenges in relation to the large-scale nature of projects, including communication and interfaces between teams, issues in relation to managers, and the tension between long-term and short-term planning [Edison et al. 2021; Dikert et al. 2016]. While there have been numerous studies of large-scale agile projects, it is perhaps surprising that these studies frequently report challenges as unanticipated outcomes. Thus, we embrace the obvious in our first proposition:

Proposition 1: *Large-scale agile projects have characteristics that lead to tension points.*

While previous studies of large-scale agile projects have suggested that a mix of plan-driven and agile methods is necessary (e.g., Batra et al. [2010]), often invoking the term ‘hybrid approaches’ [Kuhrmann et al. 2018; Kuhrmann et al. 2021], to the best of our knowledge, none of these studies develops theory that explains how and precisely why this is necessary. For example, the survey by Kuhrmann et al. [2018] of European organizations reports several motivations as to why organizations adopt hybrid approaches, including *‘improving stability,’* a need to integrate *‘high-level Waterfall-like and low-level Agile approaches,’* and pragmatism. However, whereas previous work positions “agile transformations” as singular events (e.g., Russo [2021]), which potentially may last considerable time, the various episodes in our study emphasize that large-scale agile processes not only require a mix of plan-driven and agile practices but also that the development process evolves during the project in response to a stream of tension points that emerge over time, rather than “one-off” events. Henceforth, we theorize that:

Proposition 2: *To resolve the tension points in large-scale agile projects, emergent responses are needed to adjust and re-adjust development practices and processes.*

Building on our longitudinal case study of NorTran, we saw that agile and plan-driven practices were used in combination and that this contributed to reducing the coordination and scaling challenges threatening to undermine the entire project. We observed that a combination of upfront planning practices and agile practices was often mutually enabling. For instance, in Episode 2, the ‘ready-to-sprint’ process established a common understanding that made it easier for individual teams to conduct prototyping to improve customer communication and collaboration. In Episode 5, the plan-driven practice of architecture meetings improved coordination across different teams and enabled scaling. Hence, we propose that:

Proposition 3: *In large-scale agile development projects, agile and plan-driven practices are mutually enabling.*

In contrast to underlying assumptions about large-scale agile, we suggest that plan-driven practices are an important mechanism for producing flexibility and agility in the process of large-scale agile software development. In general, plan-driven approaches to software development are seen as a way of establishing a development process that is stable, predictable, and produces low variance (e.g., Humphrey [1989]). In contrast, agile practices and processes are often seen as a way of ensuring flexibility, innovative solutions, and adaptability (e.g., Baham and Hirschheim [2022]; Conboy [2009]). Henceforth, the traditional view is that these practices and processes are in direct conflict, or at least partly conflicting. In problematizing this view, we suggest that a more nuanced position would be to view these as complementary and interdependent. Thus, we consider agile practices and processes as distinctly different from plan-driven practices and processes, but they are complementary in that one requires the other in order to produce both stability and change. More concretely, this position holds that plan-driven practices (in addition to agile practices) are important for providing flexibility and agility in large-scale agile development. This is an important insight, because this directly opposes the view of many agile advocates that plan-driven practices would reduce the agility in a project.

Drawing on the case study of NorTran, we can identify several examples of situations in which plan-driven practices enabled agility during the development process. In Episode 1, the plan-driven practice of "Tornado meetings" reduced some of the main obstacles, with a deliverable early on that helped to establish stability, thereby allowing individual teams to experiment without creating problems for other teams. As illustrated in Episode 2, this was strengthened through the plan-driven practice of "Ready-to-sprint" that was initiated by project management. Moreover, plan-based decisions to reorganize the tasks of the teams so that integration work was done by a single team in Episode 3, also increased the flexibility for all teams. The integration team could experiment to find the optimal ways to integrate with various external systems, and the other teams could spend their efforts on experimenting with all other aspects of the design. Finally, in Episode 6, the plan-driven initiative to establish regular architectural meetings afforded possibilities to scale the project in an evolutionary manner. Hence, a plan-driven practice made agile in the large possible. Based on these insights, we posit the following propositions:

PROPOSITION 4A: *Plan-driven practices can increase internal flexibility of development teams in a context of multiple teams.*

PROPOSITION 4B: *Plan-driven practices can improve coordination across teams and other actors in the project, necessary for scaling agile software development.*

6.3 Agile Practices Mitigating Failure and Increasing Predictability

According to a duality perspective, change can also enable stability. Likewise, we theorize that agile practices, self-organizing teams, and non-linear scaling are all important mechanisms for ensuring reliability and predictability in large-scale agile projects. Increasing reliability and predictability is typically associated with plan-driven practices and processes, but increased experimentation and redundancy is also necessary for such outcomes to materialize. As such, agile practices and self-organizing teams utilize experimentation and redundancy as mechanisms for achieving greater stability.

First, in contrast to the prevalent dualistic perspective found in the literature, we theorize that agile practices and self-organizing teams are fundamental to mitigate failure in large-scale agile. As seen in Episode 4, the situated and bottom-up initiative of establishing 'task forces' solved crucial problems typically related to compliance with nonfunctional requirements such as performance issues. In this regard, the task forces were an agile response to increase the reliability of

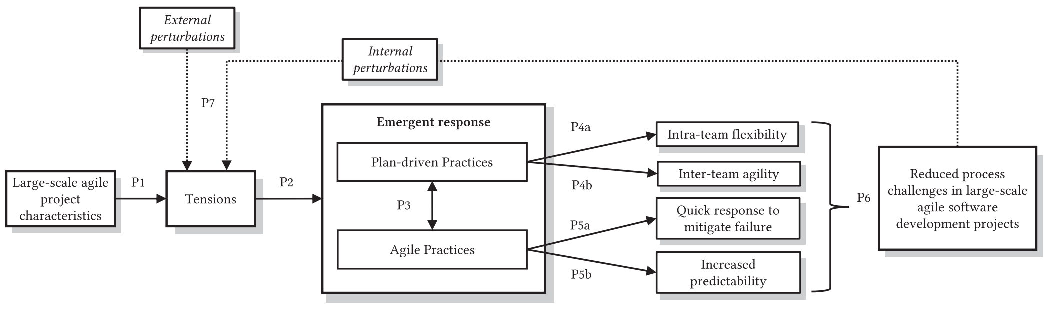

Fig. 5. Process model with propositions based on our study.

deliverables, and especially to mitigate potential failure of the delivered products. Hence, this leads us to the following proposition:

Proposition 5a: *Agile practices allow for quick responses to problems involving complex sociotechnical interdependencies so that failures are mitigated.*

Second, rather than seeing moderately overlapping responsibilities of agile teams and individuals as a threat to reliability and effectiveness in large-scale agile software development processes, we recognize that this can be a resource and a mechanism facilitating it. Again, drawing on Farjoun [2010], this underscores how agile practices and processes contribute to increasing stability in large-scale software development. Empirically, a typical illustration of this was evident in Episode 6, in which individual developers and testers were taken out of their teams to operate like 'freelancers.' This experiment succeeded in producing a controlled downscaling of the project while preserving crucial competence. Hence, we suggest the following:

Proposition 5b: *Agile practices can be a mechanism for increasing predictability in all phases of large-scale agile processes.*

6.4 A Process Model for Large-Scale Agile Software Development

In this article, we problematize large-scale agile, and draw on the concept of duality, leading to a number of contributions which are summarized in the process model presented in Figure 5. As such, large-scale agile has some characteristics that typically imply increasing process challenges due to utilizing the same agile practices in the context of large projects as in small projects [Rolland et al. 2016], difficulties and lack of inter-team coordination [Bick et al. 2018], and difficulties in scaling through simply adding more teams [Paasivaara et al. 2012]. However, new alternative assumptions backed by a duality lens (as opposed to a dualism lens) suggest that plan-driven and agile practices are not necessary in conflict but can be mutually enabling. As seen in Figure 5, we theorize that responses that are based on this tend to reduce the specific process challenges around preserving the flexibility in feature teams, establishing coordination mechanisms across teams and actors, and managing scaling in large-scale agile projects. As such, novel combinations of agile and plan-driven practices, as seen in the NorTran case, can greatly reduce process challenges typically related to coordination across teams and various stakeholders, as well as scaling and downscaling in large-scale agile projects. Thus, we theorize that:

Proposition 6: *Novel combinations of agile and plan-driven practices can collectively reduce process challenges in large-scale agile development projects.*

Finally, we do not claim that all process problems will be solved at once by combining agile and plan-driven practices as exemplified in the six episodes. Rather, we suggest that this as a continuous process in which novel combinations can solve some of the challenges, potentially reducing process challenges. Each of the six episodes described an emergent response to some challenge, which ultimately reduced the process challenge at hand, but new tensions would arise; continuing perturbations would lead to new tensions within a large-scale agile project. These perturbations may originate from within the project, or externally, when the environment in which the project sits changes. Examples of these include changing regulations, a reduction in project budget as decided by upper-level management, or global events such as the COVID-19 pandemic. Any type of event may lead to tensions within a large-scale agile project; these, in turn, will require new emergent responses. We capture this in the final proposition:

PROPOSITION 7: *Internal or external perturbations in large-scale agile projects lead to tension points.*

6.5 Limitations

Given that this was a complex research study with several components, i.e., systematic literature review, problematization, and a longitudinal case study, there was a potential for threats to validity from a methodological perspective. For the problematization, we followed the steps described by Alvesson and Sandberg [2011]. For the systematic literature review, we followed guidelines by Kitchenham and Charters [2007]. For the case study, we drew on recommendations suggested by Runeson and Höst [2009]. In terms of identifying the fundamental assumptions in the literature on large-scale agile development, one could ask whether we really have captured core fundamental assumptions. While there are many thousands of articles published on agile development, we were reassured that our identification of 197 papers on large-scale agile is consistent with the recent systematic review by Edison et al. [2021], which identified 191 studies on large-scale agile development. As part of narrowing down our set of studies to 67 studies, we eliminated those which were experience reports, as we sought to focus on papers that would provide a strong research and conceptual foundation, thereby ensuring that any assumptions were grounded in the research. Could it be that underlying assumptions were expressed in articles as a part of an argument for a contribution or a research gap, and that the assumptions were not truly held by the authors of these papers? This might be the case in some sources, but our analysis in Sections 3.1 to 3.3 suggests that the identified assumptions *are* held by authors of the selected articles. Hence, we argue our set of studies should be sufficient to identify the widely held assumptions in the literature.

One could further question how we evaluated the articulated assumptions to select the three fundamental assumptions presented in Section 3. We held several discussions within the research team to use our knowledge of relevant prior research in order to focus on assumptions which were relevant, common, and fundamental to the further development of the research field. Other researchers could have made other choices, and we acknowledge that this is a matter of judgment. We have sought to explain why the three selected assumptions are fundamental in Section 3.

Given that our overall research approach was qualitative, the traditional validity criteria used for quantitative studies —namely, construct validity, internal validity and external validity— are not the most appropriate. Rather, we draw on Lincoln and Guba [1985], who proposed the following alternative criteria for qualitative research: credibility, dependability, transferability, and confirmability.

Credibility refers to the extent to which the findings make sense and can be believed or recognized by participants or readers. We drew on a number of techniques that can help achieve this, including prolonged engagement, participant observation, member-checking, and peer-debriefing. Firstly, the research was conducted over a period of several years. A member-checking process took place whereby the interviewees in the case study participated in two workshops at which findings were presented and discussed. This meant that there were several opportunities for participants to reflect on our analysis and suggest changes if necessary. Peer review occurred when preliminary findings were presented at a conference [Rolland et al. 2016].

Dependability refers to the extent to which the research process is logical, traceable, clearly documented, and can be repeated. In this case, data-coding followed a coherent and traceable process, and was undertaken by multiple researchers who performed both hand-coding and software-supported coding (HyperResearch). Intermediate coding results were extensively discussed and scrutinized through a dedicated workshop and subsequent virtual discussion. This process produced an 'audit trail' that documented the research process from data collection through to drawing of conclusions.

Transferability is concerned with how the findings could be applicable to other contexts and is similar to the notions of generalizability and external validity in quantitative research. While proof of transferability cannot be shown, the richness of the findings can help indicate to what extent the research can apply in other contexts. Here we sought to get a rich picture of the situation as it applied in one real development context. Indeed, we are heartened by the very apt observation of Mintzberg [1979, p. 583]: *'what, for example, is wrong with samples of one?'* The goal of the case study research was not to draw generalizable conclusions, but rather to understand how processes evolve in a large-scale agile project, which could then help us to develop new theory.

Finally, confirmability is concerned with establishing that findings are clearly derived from the data. Our methodological discussion above provides evidence for this. Lincoln and Guba [1985] suggest that confirmability can be established by successfully achieving credibility, transferability, and dependability. Nowell et al. [2017] recommend providing a clear methodological and theoretical rationale, which they suggest is useful to help ensure confirmability. Here we have sought to provide such a rationale and have gathered our findings into a theoretical model which can be tested and evolved in further research.

7 CONCLUSIONS AND FUTURE WORK

The aim of this article has been to challenge some of the fundamental assumptions in the current literature on large-scale agile to provide new practical and theoretical avenues for researchers and practitioners to follow. The article draws on both a systematic literature review on large-scale agile software development and a longitudinal case study of a successful large-scale agile project. This study provides three distinct contributions. First, we contribute by providing a systematic literature review of large-scale agile software development. While a systematic review of the topic has been published before Edison et al. [2021], this article additionally utilizes Alvesson and Sandberg's method for problematizing fundamental assumptions in large-scale agile. We identified three basic overarching assumptions relating to (1) how agile and plan-driven processes are perceived as in conflict or mutually exclusive; (2) hierarchical organizing and self-organizing teams are perceived as in conflict or mutually exclusive; and (3) how scaling of agile practices is seen as linear (e.g., Scrum of Scrums), implying that scaling can be conducted through just adding more participants and increasing the scope without changing practices. Based on this, we concluded that the literature on large-scale agile software development seems to reinforce the same underlying principles as in small-scale agile software development with a few co-located teams. Against this backdrop, the second main contribution of this article is the presentation of a longitudinal case study on a successful large-scale agile project. A novel aspect of our case study is the use of Newman and Robey's process model as an analytical approach, which to the best of our knowledge has not been used in software engineering research before. This approach proved effective to capture key events ("episodes") that occurred during our study. The empirical findings from our case study of successful large-scale agile development were at odds with the underlying assumptions in the systematic literature review. Consequently, based on our analysis of the case study, we developed

Table 7. Propositions and Suggestions for Future Work

Proposition	Research directions
Proposition 1. Large-scale agile projects have characteristics that lead to tension points.	What is the range of relevant characteristics of large-scale agile software development that may lead to tension points? How do types of characteristics vary by the scale of their impact?
Proposition 2. To resolve the tension points in large-scale agile projects, emergent responses are needed to adjust and readjust development practices and processes.	Given the limited research on large-scale over time, future research should cover longitudinal and evolutionary studies of agile practices. Influence of changes in project context on agile methods, such as contemporary trends including Digital Transformation, AI, cost-cutting programs, staffing-related issues, and technology changes.
Proposition 3. In large-scale agile development projects, agile and plan-driven practices are mutually enabling.	How do agile practices support plan-driven practices, and vice versa? How do agile and plan-driven practices interact? What are attributes of plan-driven practices vs. agile practices, and how do these attributes vary?
Proposition 4a. Plan-driven practices can increase internal flexibility of development teams in a context of multiple teams.	How do plan-driven practices influence cognitive load of a development team? How do plan-driven practices influence teamwork effectiveness? Are plan-driven practices perceived as offering stability or as a barrier to team autonomy and self-organization?
Proposition 4b. Plan-driven practices can improve coordination across teams and other actors in the project, necessary for scaling agile software development.	What role do plan-driven practices have as a mediator of inter-team and intra-team coordination?
Proposition 5a. Agile practices allow for quick responses to problems involving complex socio-technical interdependencies so that failures are mitigated.	How do methods influence decision-making regarding critical issues such as providing business value from the project?
Proposition 5b. Agile practices can be a mechanism for increasing predictability in all phases of large-scale agile processes.	How do agile practices influence predictability in the various phases of a project?
Proposition 6. Novel combinations of agile and plan-driven practices can collectively reduce process challenges in large-scale agile development projects.	What type of plan-driven and agile practices are good combinations?
Proposition 7. Internal or external perturbations in large-scale agile projects lead to tension points.	What is the range of perturbations that can cause tensions in large-scale agile software development? How does the impact of perturbations vary by their source, i.e., whether they are internal or external? How do perturbations vary, i.e., by the size of the impact?

three alternative assumptions for large-scale agile. Thirdly, based on the new underlying assumptions and the empirical evidence from the NorTran case, we derived nine theoretical propositions and a new process model.

Table 7 suggests a number of research directions based on this set of propositions. Rather than reemphasizing and reiterating the distinctions between agile and plan-driven methods, we argue that it is more important to let go of initial characterizations of agile methods, and discover and investigate emergent properties of these methods when applied in large-scale settings.

Our study has uncovered fundamental assumptions in the literature on large-scale agile development, which we have argued are not suited to solve some of the pressing coordination and scaling challenges in large-scale agile development. Given the criticality of large software projects for society today, we hope the perspectives developed in this article will lead to more relevant and theoretically robust studies, which better reflect the characteristics of complex and large-scale development efforts. Evoking our metaphor of acrobats and safety-nets, we suggest that, just like an acrobat, agile approaches in large-scale contexts need some level of stability, akin to a safety net, in order to sustain flexibility over time. On the other hand, there would be no need for a safety net, i.e., plan-driven practices, if the acrobats were not seeking to overcome challenges requiring agility and flexibility.

APPENDICES

A SYSTEMATIC LITERATURE REVIEW PROCEDURE

The first step of our study comprised a systematic literature review (SLR) to identify studies that discuss the use of agile methods for large-scale projects (see Figure A.1). We conducted searches in SCOPUS and the AIS e-library of studies published until and including 2019. Prior to the systematic literature review, we developed a review protocol as recommended by Kitchenham and Charters [2007]. We created the following search string for use in SCOPUS:

> TITLE((“extreme programming” OR scrum OR (agile AND software) OR “agile development” OR “agile project*” OR “agile team*” OR “agile method*” OR “agile approach*” OR “agile practice*”)) AND TITLE (“large-scale” OR “large scale” OR (large* OR big* OR huge OR multi*) AND (organization* OR organisation* OR project* OR team*))

The first step identified 132 papers. We conducted a systematic process to identify all relevant articles in multiple steps, summarized in Figure A.1. The selection was based on a clearly defined

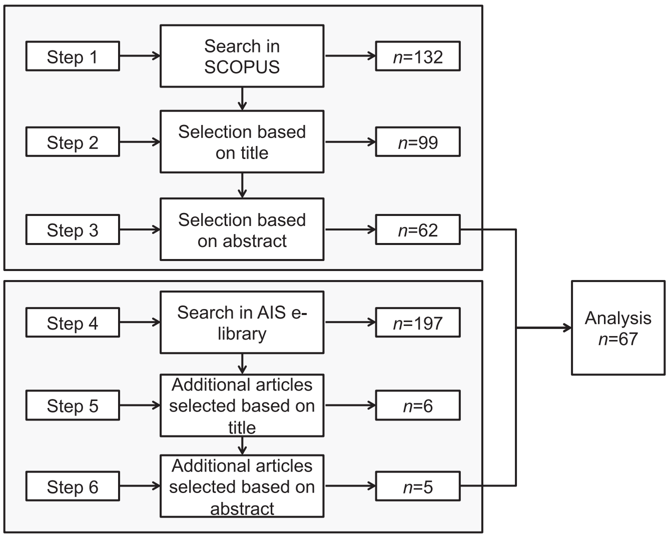

Fig. A.1. The study selection process of the systematic literature review.

Table A.1. Quotes from Selected Papers Illustrating the Three Assumptions

Reference	Assumption 1: Agile and Plan-driven Methods perceived to be Mutually Exclusive	Assumption 2: Self-Organization and Hierarchical Coordination are perceived to be Mutually Exclusive	Assumption 3: Scaling agile methods is seen as a Simple Linear Composition
Arseni [2016]		Decision-making can stall when visibility and transparency are limited. Creating a design authority can help to avoid these problems and maximize opportunities.	Product owner team (POT) multi-users team in charge to manage Product Backlog.
Baca et al. [2015]			Scrum of Scrums meeting an important technique in scaling to large project teams. Security group includes security manager, security architect, security master.
Badampudi et al. [2013]	Misalignment of need for predictability and dependability with agile.	Every scrum group would have their own priorities to finish their tasks.	Dedicated integration team.
Barlow et al. [2011]	Plan-driven assumes that project interdependencies are mostly sequential. Agile assumes the opposite, they de-emphasize formal, upfront planning.	Empowering developers to make important decisions makes development faster. Projects would still include a full up-front design phase while allowing programmers to make decisions during coding and testing phases.	
Bass [2015]			Product owner team identifies new functions that allow teams to scale up. Nine functions within the product owner role are identified.
Bass [2016]			Programme Governance Groups, Product Owner Teams or Scrum of Scrums meetings to overcome challenges to the expansion of agile methods to large-scale development.
Bass [2014]			Scrum of scrums help scale agile methods to large programs. Further, scrum masters can specialize by assigning these activities within a scrum master group.
Bass and Haxby [2019]		Self-organizing teams relinquish some autonomy toward an architecture board or design authority that determines common policies and approaches.	Product-owner (PO) role tailoring in which the role is no longer performed by a single individual but by a product-owner team (POT).

(Continued)

Table A.1. Continued

Reference	Assumption 1: Agile and Plan-driven Methods perceived to be Mutually Exclusive	Assumption 2: Self-Organization and Hierarchical Coordination are perceived to be Mutually Exclusive	Assumption 3: Scaling agile methods is seen as a Simple Linear Composition
Berger and Benyon-Davies [2009]		Development by small joint development teams, in which decision-making is empowered and consequently speedy.	
Batra et al. [2011]	Lightweight agile software development methods have emerged as alternatives to process-heavy plan- based methodologies.		Scrum can be managed as a hierarchy of Scrum of Scrums.
Batra et al. [2010]	A concern that the project would lose discipline if an agile approach were instituted.	Principles that were not supported include the best architectures, requirements, and designs emerge from self-organizing teams. Individuals were empowered, but management had no issues confronting ego-centric individuals.	
Bick et al. [2016]		Top-down planning refers to a mechanistic, centralized approach. Bottom-up adjustment, on the other hand, is largely organic and decentralised.	Scaling via Iterative Proxy Collaboration — CPO, SoS, central architecture team.
Cao et al. [2004]	Agile methods lack of up-front design and documentation.	We didn't have layers and layers of management. We got rid of those. Decentralizing development-oriented decision-making is critical for a successful agile — push decision-making down, empower the people who are actually doing the work.	
Cho et al. [2006]			The daily Scrum of Scrums is a daily meeting for SMs from multiple Scrum teams.
Costa et al. [2014]		In opposition to the previous methodologies, agile development processes are based on self-organized teams resolving their problems.	
Dingsøyr et al. [2014]			Coordination of teams can be achieved in a new forum, such as a Scrum of Scrums forum. Several forums are needed for coordination, such as multiple Scrum of Scrums.

(Continued)

Table A.1. Continued

Reference	Assumption 1: Agile and Plan-driven Methods perceived to be Mutually Exclusive	Assumption 2: Self-Organization and Hierarchical Coordination are perceived to be Mutually Exclusive	Assumption 3: Scaling agile methods is seen as a Simple Linear Composition
Dingsøyr et al. [2017]			Program management met twice a week in a forum — "Metascrum." The Metascrum included managers from the main projects and the central program management.
Dingsøyr et al. [2018]	The fundamental assumption behind traditional methods is that systems are fully specifiable and built through meticulous and extensive planning. Agile methods, on the other hand, assume that systems can be built through continuous design, improvement, and testing based on rapid feedback and change.		
Eckstein [2016]		Sociocracy enables self-organization… scaling agile requires also scaling self-organization.	
Elshamy and Elssamadisy [2006]	We have the non-agile solution to this problem, which is design upfront.		
Elshamy and Elssamadisy [2007]			Sub-teams will have their own stand-ups. To ensure information exchange between teams everyday, a member of each sub-team should attend another team stand-up.
Fægri and Moe [2015]		People need to apply their own judgment in when they need to seek information…Project culture and project management promoted face-to-face communication and rejected written reports.	
Farmer [2004]		Management let us find our own way, rather than forcing process on us from above.	
Fruhling and DeVreede [2006]	To address plan-driven methodology shortcomings, new development models were proposed, such as agile.		

(Continued)

Table A.1. Continued

Reference	Assumption 1: Agile and Plan-driven Methods perceived to be Mutually Exclusive	Assumption 2: Self-Organization and Hierarchical Coordination are perceived to be Mutually Exclusive	Assumption 3: Scaling agile methods is seen as a Simple Linear Composition
Goh et al. [2013]		Formal outcome control and the "freedom" given to the project manager to implement many of the requirements (informal self-control).	
Gunyho and Gutiérrez Plaza [2011]	Being adaptable to change may be seen in contradiction with planning.		
Gupta et al. [2017]		Is your team self-organizing, rather than functioning in command and control… participatory decision-making, rather than bending to authoritarian decision-making … team decisions consensus driven, rather than leader driven.	Chief Scrum Master, Chief PO, Scrum Master-cum-Part Product Owner (SMPO).
Gustavsson [2018]		Fundamental principle in agile is to allow autonomy to the team. This autonomy is a major reason for success in agile development.	
Hannay and Benestad [2010]	Vendors want to work agile, and now we're suddenly supposed to work Waterfall.	The combination of autonomous teams and the necessity for overall organizational control structure may lead to conflicts.	
Heikkila et al. [2013a]	ScrumBut: We use Scrum, but we can't build a piece of functionality in a month, so our Sprints are 6 weeks long (which is like a Waterfall).		Area Product Owner (APO) and Chief Product Owner (CPO).
Heikkilä et al. [2013b]			PO team consisted of a Chief Product Owner (Chief PO) and ten Proxy Product Owners (PPOs).
Hobbs and Petit [2017]	Both the traditional and the agile methodologies co-exist in separate subunits.		
Hoda and Murugesan [2016]		Self-organizing agile teams take ownership of management responsibilities which were hitherto limited to project managers.	

(Continued)

Table A.1. Continued

Reference	Assumption 1: Agile and Plan-driven Methods perceived to be Mutually Exclusive	Assumption 2: Self-Organization and Hierarchical Coordination are perceived to be Mutually Exclusive	Assumption 3: Scaling agile methods is seen as a Simple Linear Composition
Jørgensen [2019]	Some software professionals believe less in working fully agile when projects get large…it is possible to argue in favour of both agile and more plan-driven, non-agile methods.		
Kähkönen [2004]			Communities-of-Practice, Integration Camp.
Kettunen and Laanti [2008]		Project team should first be empowered to gain the full benefits of the agile.	
Ktata and Lévesque [2009]	Agile development has legalized what was forbidden by traditional plan-driven development.		
Laanti [2008]			Program Content Backlog containing Program, Scrum, and Sprint backlogs, Program PO, Team PO.
Laanti [2017]		Best-in-class agile is empowered, self-controlled adaptive organisation.	
Lagerberg et al. [2013]	Project A (Agile) and Project B (Plan-driven) differ sufficiently in their ways of working and are sufficiently similar in other aspects that the impact of using agile practices can be studied by comparing the two projects.		Cross-functional team includes system analysts, designers and testers, as well as a Scrum Master and a Product Owner.
Lindsjørn et al. [2018]		Large projects need stronger mechanisms to control cost and time schedules.	Scrum of Scrums.
Martini et al. [2013]		(Agile) trend of defining small self-sufficient teams. If a team doesn't have all the knowledge, they may have to wait for the expert to be available.	
Moe et al. [2018]		Principles and work structures emerge during the project and are not predetermined… complex agile projects need more flexible forms of management …rather than pure top-down approaches to governance.	Metascrum, SoS.

(Continued)

Table A.1. Continued

Reference	Assumption 1: Agile and Plan-driven Methods perceived to be Mutually Exclusive	Assumption 2: Self-Organization and Hierarchical Coordination are perceived to be Mutually Exclusive	Assumption 3: Scaling agile methods is seen as a Simple Linear Composition
Moe et al. [2014]			Technical Area Responsible (TAR) role on Cross-Functional teams (XFTs), Operative Product Owner (OPO), Area Product Owner (APO), System Owner.
Nyfjord et al. [2014]		The best way to coordinate the teams is to ask them how they want to be managed… A manager should accept a certain amount of chaos in the development process…should not try to control everything.	
Paasivaara et al. [2008]			Weekly Scrum-of-Scrums, Synchronized 4-week sprints.
Paasivaara and Lassenius [2016]	Increases the need for formal documentation and thus reduces agility.		Scrum-of-Scrums, Area PO.
Paasivaara and Lassenius [2014]	Transformation from a traditional plan-driven organization to lean and agile.		Feature Coordination CoPs, Coaching CoPs, Developer CoPs.
Paasivaara and Lassenius [2011]			Area Product Owners (APOs) for scaling the Product Owner role, Global Scrum-of-Scrums.
Paasivaara et al. [2012]			Scrum-of-Scrum-of-Scrums (SoSoS).
Qureshi [2012]	Extended XP to include stable requirements, strong architecture, and risk management plan.		
Read and Briggs [2012]			HyperEpic — a structured collection of closely related HyperStories.
Rolland et al. [2015]	Emergent nature of requirements…notoriously difficult to establish a stable and complete set of requirements early on in the process.		
Rolland et al. [2016]		Task forces were not initiated by management, but grew out of a need recognized by developers.	Champion roles were implemented working across teams.
Šablis and Šmite [2016]		When there are many teams, should they be governed or autonomous?	Forum of forums.

(Continued)

Table A.1. Continued

Reference	Assumption 1: Agile and Plan-driven Methods perceived to be Mutually Exclusive	Assumption 2: Self-Organization and Hierarchical Coordination are perceived to be Mutually Exclusive	Assumption 3: Scaling agile methods is seen as a Simple Linear Composition
Scheerer and Kude [2014]		Organic structure of teams.	Scrum-of-Scrums.
Sekitoleko et al. [2014]	The shift towards agile is difficult for companies that are used to heavyweight sequential processes.		Scrum-of-Scrums (SoS), cross-functional teams (XFT).
Søvik and Forfang M. [2010]	Started with a waterfall-like methodology and then adopted Scrum after less than a year.		
Stettina and Smit [2016]			We designed the Team Portfolio Scrum (TPS) practice and the Team Portfolio Owner (TPO) role to support the implementation of portfolio management.
Sundararajan et al. [2014]		“process-centric, command and control” v. “people-centric, self-organising…make sites self-managing, introduce team empowerment.	
Tessem and Maurer [2007]		This way of working gives the developer significant autonomy in the daily work.	A concept like “Scrum of Scrums” is useful for making larger teams agile.
Uludağ et al. [2019]		Squads are self-organizing and autonomous teams that have all the skills to design, develop, test, and release for production.	All teams are part of an agile release train (ART), a team of teams that delivers a continuous flow of incremental releases.
Wale-Kolade [2015]		“Follow the leader” approach versus the “individualistic” approach to encourage people to think for themselves and not be so rigid in following your leader.	Scrum of Scrums model . . . a technique for scaling Scrum practices, thus enabling inter-team coordination and consensus.
van Waardenburg and van Vliet [2013]	The agile process is often preceded by traditional requirements elicitation and analysis phases.	The hierarchical, centralized decision-making in plan-driven methods versus the empowerment of agile developers to make their own decisions.	Combining product backlogs of teams that depend on one another helps teams plan and align dependent work items.
Vlietland and van Vliet [2015]	Even though Agile principles aim to introduce flexibility, the need for plans and structure remains.		An interdependent chain of Scrum team. Scrum of Scrums; For managing more than seven Scrum teams, an intermediate organizational layer is suggested between the product teams and Scrum teams to cater for the necessary coordination.

(Continued)

Table A.1. Continued

Reference	Assumption 1: Agile and Plan-driven Methods perceived to be Mutually Exclusive	Assumption 2: Self-Organization and Hierarchical Coordination are perceived to be Mutually Exclusive	Assumption 3: Scaling agile methods is seen as a Simple Linear Composition
Weiss and Brune [2017]	The interface between agile development teams and plan-driven release management is critical and challenging.	Teams should receive enough freedom to adapt agile methods to their specific needs.	
Zheng et al. [2011]	Managing carefully the balance between flexibility and rigour.	Tension between the deliberate action of planning and the uncontrolled processes of drifting.	

set of inclusion and exclusion criteria which are described here. In step 2, we examined article titles only. If the decision whether or not to include a paper was not straightforward, it was included for further assessment in step 3. Overall, we selected a total of 99 articles in step 2. During step 3, we examined the abstract. Again, where no definitive decision could be made, articles were retained for the next step. A total of 62 articles were included at this point.

After this search and selection of results from SCOPUS, we conducted a search in the AIS e-library, using a simplified search string as its search capabilities are not as extensive as those of SCOPUS. We used the following simplified string:

> title: (“extreme programming” OR scrum OR (agile AND software) OR “agile development” OR “agile project*” OR “agile team*” OR “agile method*” OR “agile approach*” OR “agile practice*”)

While this simplified string is less specific — and, therefore, likely to include more irrelevant search results —the total number of search results was relatively low (n = 197). We followed the same systematic selection process to identify relevant papers in this initial set of 197, through which we identified 5 papers that were not already included in the selection identified through SCOPUS. The total search process therefore resulted in 67 papers (62 following the initial SCOPUS search and 5 additional papers from the AIS e-library).

We used the following inclusion and exclusion criteria during the paper selection process for the review. Inclusion criteria are explicit statements or guidelines that prompt the inclusion of an article, whereas exclusion criteria are explicit statements or guidelines that lead to the non-selection of an article. As the selection progressed, criteria were further clarified as needed to remove any ambiguities. These criteria were employed during steps 2 and 3 in the process (see Figure A.1).

Inclusion Criteria

- Articles that present studies of the use of any agile methods for large projects.
- Articles that present studies of the use of agile methods in “large organizations.”

Exclusion Criteria

- Any article not written in English.
- Any article that uses the terms “large” or “big” in other contexts than “large projects,” for example, “Big Data.”

- Any article that does not have a focus on the use of agile methods in a large software development project context, but instead on another topic of interest within the setting of large-scale projects, e.g., quality attributes or motivation, or user experience (UX) in large-scale agile projects, or project estimation, or the use of agile for general change management outside the context of software development.
- Any article that refers to "distributed projects" or "distributed organizations" exclusively, without any mention of "large scale projects."
- Any article that discusses "large-scale agile transformations"; the focus of these articles is organizations that are transitioning to using agile methods, rather than the use of agile methods employed in large-scale projects.
- Any article that discusses multi-project scrum teams; these are scrum teams which work on multiple projects at the same time.
- Any document that is not a peer reviewed article, such as books, editorials, presentation summaries, abstracts or briefings.
- Literature reviews of large-scale agile.

In order to identify the assumptions from this body of literature, we read all papers in full. All authors were involved in this process. It became clear during this process that there were a number of common assumptions that appeared in multiple papers. The analysis was conducted over an extensive period of several months and involved numerous meetings, both in-person and online, as well as a number of dedicated workshops. During the analysis, we identified three high-level assumptions which were recurrent across the selection of papers. These three assumptions were as follows: (1) Agile and plan-driven methods are perceived to be mutually exclusive; (2) Self-organization and hierarchical coordination are perceived to be mutually exclusive; and (3) Scaling agile methods is seen as a simple linear composition. Table A.1 lists all 67 papers included in the review and identifies how the assumptions were manifest in each of the papers.

B AUXILIARY EPISODES

Besides the episodes discussed above, we identified several others that challenged the three key assumptions we identified (see Table B.1). Some of the entities we identified, such as Solution Description and Blurred Boundaries, are also featured in a different large-scale project reported by Dingsøyr et al. [2018].

Table B.1. Summary of Additional Episodes

Dominant assumption	Episode description	Challenging the assumption
Agile and Plan-Driven Methods are perceived to be Mutually Exclusive	Introduction of Solution Description (SD) as ‘big design up front’ style process, but one that allowed a flexibility in terms of the amount of detail. The SD represents an iterative process that afforded a pragmatic approach to “just-in-time” design.	The Solution Description was not a plan-driven style design document, but rather a flexible, more agile approach to an evolving design document that allowed an incremental process of detailing, thus combining a plan-driven and agile approach.
	Blurred boundaries: the boundaries between development process phases such as “analysis of needs” and “solution description” phases blurred over time.	The large-scale nature of the project required some type of phased approach to establish milestones, such as “analysis of needs” and the “solution description” phases, the boundaries between different phases blurred over time as the people in the development and architecture teams were close in proximity. A more holistic orientation emerged that allowed people to consider requirements and how these could be satisfied in a solution description.
Self-managing Teams and Hierarchically organized Teams are perceived to be Mutually Exclusive	From the start, roles were imposed on team members, such as team lead, test responsible, technical architect, and functional architect, while teams as a whole remained responsible for their sprint backlogs.	Traditionally, agile teams as a whole have a joint responsibility to deliver software, whereas the traditional plan-driven approach would assign specific roles and responsibilities to individuals. At NorTran, a combination was used whereby the team as a whole remained responsible, yet roles were imposed on individual team members.
Scaling through Simple Linear Composition	Exponential increase in coordination mechanisms as project grew in size.	Scaling up a project cannot be simply done by adding additional layers of coordination, such as the Scrum-of-Scrums activity. Instead, a more dramatic exponential increase, rather than a linear increase, in coordination mechanisms may be necessary.
	Physical co-location of project with an open work area enabled efficient direct one-to-one dialogue as participants had become aware of others’ work tasks, responsibilities, and background knowledge.	Scaling up of projects doesn’t necessarily need more coordination mechanisms: additional teams could also benefit from being co-located, leveraging unplanned interactions.

ACKNOWLEDGMENTS

In loving memory of our friend, colleague, and co-author Knut Rolland, who sadly passed away during this project.

REFERENCES

P. Abrahamsson, M. A. Babar, and P. Kruchten. 2010. Agility and architecture: Can they coexist? Introduction. *IEEE Software* 27, 2 (2010), 16–22.

P. Abrahamsson, K. Conboy, and X. Wang. 2009. 'Lots Done, More to Do': The current state of agile systems development research. *European Journal of Information Systems* 18, 4 (2009), 281–284.

S. Adikari, C. McDonald, and J. Campbell. 2009. Little design up-front: A design science approach to integrating usability into agile requirements engineering. In: *Human-Computer Interaction. New Trends. HCI 2009*, J. A. Jacko (Eds.). Lecture Notes in Computer Science, Vol. 5610. Springer, Berlin, 549–558.

W. Alsaqaf, M. Daneva, and R. Wieringa. 2017. Quality requirements in large-scale distributed agile projects–a systematic literature review. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, Cham. 219–234.

M. Alvesson and J. Sandberg. 2011. Generating research questions through problematization. *Academy of Management Review* 36, 2 (2011), 247–271.

M. Alvesson and J. Sandberg. 2013. *Constructing Research Questions: Doing Interesting Research*. Sage.

S. W. Ambler and M. Lines. 2012. *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*. IBM Press, Upper Saddle River, NJ.

G. Arseni. 2016. Role of the design authority in large scrum of scrum multi-team-based programs. In *Proceedings of 4th International Conference in Software Engineering for Defence Applications, Advances in Intelligent Systems and Computing*, P. Ciancarini (Ed.), 422.

D. Baca, M. Boldt, B. Carlsson, and A. Jacobsson. 2015. A novel security-enhanced agile software development process applied in an industrial setting. *Proceedings of the 10th International Conference on Availability, Reliability and Security*. 11–19.

D. Badampudi, S. A. Fricker, and A. M. Moreno. 2013. Perspectives on productivity and delays in large-scale agile projects. *Proceedings of the International Conference on Agile Software Development*. 180–194.

C. Baham and R. Hirschheim. 2022. Issues, challenges, and a proposed theoretical core of agile software development research. *Information Systems Journal* 32, 1 (2022).

J. B. Barlow, J. S. Giboney, M. J. Keith, D. W. Wilson, R. M. Schuetzler, P. B. Lowry, and A. Vance. 2011. Overview and guidance on agile development in large organizations. *Communications of the Association for Information Systems* 29, 2 (2011), 25–44.

J. M. Bass. 2014. Scrum master activities: Process tailoring in large enterprise projects. *Proceedings of the International Conference on Global Software Engineering (ICGSE).*

J. M. Bass. 2015. How product owner teams scale agile methods to large distributed enterprises. *Empirical Software Engineering* 20, 6 (2015), 1525–1557.

J. M. Bass. 2016. Artefacts and agile method tailoring in large-scale offshore software development programmes. *Information and Software Technology* 75 (2016), 1–16.

J. M. Bass and A. Haxby. 2019. Tailoring product ownership in large-scale agile projects: Managing scale, distance, and governance. *IEEE Software* 36, 2 (2019), 58–63.

G. Bateson. 1972. *Steps to an Ecology of Mind*. Ballantine, New York.

D. Batra, D. Vandermeer, and K. Dutta. 2011. Extending agile principles to larger, dynamic software projects: A theoretical assessment. *Journal of Database Management* 22, 4 (2011), 73–92.

D. Batra, W. Xia, D. Vandermeer, and K. Dutta. 2010. Balancing agile and structured development approaches to successfully manage large distributed software projects: A case study from the cruise line industry. *Communications of the Association for Information Systems* 27, 1 (2010), 379–394.

K. Beck and B. Boehm. 2003. Agility through discipline: A Debate. *IEEE Software* 36, 6 (2003), 44–46.

H. Berger and P. Beynon-Davies. 2009. The utility of rapid application development in large-scale, complex projects. *Information Systems Journal* 19 (2009), 549–570.

M. Berntzen, R. Hoda, N. B. Moe, and V. Stray. 2023. A taxonomy of inter-team coordination mechanisms in large-scale agile. *IEEE Transactions on Software Engineering* 49, 2 (2023), 699–718. https://doi.org/10.1109/TSE.2022.3160873 https://ieeexplore.ieee.org/abstract/document/9739868">

S. Bick, A. Scheerer, and K. Spohrer. 2016. Inter-team coordination in large agile software development settings: Five ways of practicing agile at scale. *Proceedings of the Scientific Workshop Proceedings of XP2016*. ACM, p. 4.

S. Bick, K. Spohrer, R. Hoda, A. Scheerer, and A. Heinzl. 2018. Coordination challenges in large-scale software development: A case study of planning misalignment in hybrid settings. *IEEE Transactions on Software Engineering* 44, 10 (2018), 932–950.

E. Bjarnason, K. Wnuk, and B. Regnell. 2011. A case study on benefits and side effects of agile practices in large-scale requirements engineering. *Proceedings of the 1st Workshop on Agile Requirements Engineering*.

B. Boehm. 2002. Get ready for agile methods, with care. *IEEE Computer* 35, 1 (2002), 64–69. https://ieeexplore.ieee.org/document/976920

B. Boehm and R. Turner. 2003. Using risk to balance agile and plan-driven methods. *IEEE Computer* 36, 6 (2003), 57–66. https://ieeexplore.ieee.org/document/976920

G. Booch. 2015. Keynote at the 37th International Conference on Software Engineering: The Future of Software Engineering. Retrieved August 26, 2023 from https://www.youtube.com/watch?v=h1TGJJ-F-fE

L. Cao, K. Mohan, P. Xu, and B. Ramesh. 2004. How extreme does extreme programming have to Be? Adapting XP Practices to Large-Scale Projects. *Proceedings of the Hawaii International Conference on System Sciences*, R.H. Sprague Jr (Ed.), Big Island, HI. 1335–1344.

J. Cho. 2009. A hybrid software development method for large-scale projects: Rational unified process with scrum. *Issues in Information Systems* 10, 2 (2009), 340–348.

J. Cho, Y. Kim, and D. Olsen. 2006. A case study on the applicability and effectiveness of scrum software development in mission-critical and large-scale projects. *Proceedings of the Americas Conference on Information Systems*. Paper 445.

K. Conboy. 2009. Agility from first principles: Reconstructing the concept of agility in information systems development. *Information Systems Research* 20, 3 (2009), 329–354.

K. Conboy and N. Carroll. 2019. Implementing large-scale agile frameworks: Challenges and recommendations. *IEEE Software* 36, 2 (2019), 44–50.

N. Costa, N. Santos, N. Ferreira, and R. J. Machado. 2014. Delivering user stories for implementing logical software architectures by multiple scrum teams. *Proceedings of the 14th International Conference on Computational Science and Its Applications, Lecture Notes in Computer Science*, Vol. 8581. Springer, Berlin, 747–762.

K. Crowston, K. Chudoba, M. B. Watson-Manheim, and P. Rahmati. 2016. Inter-team coordination in large-scale agile development: A test of organizational discontinuity theory. *Proceedings of the Scientific Workshop Proceedings of XP2016*. 1–5.

M. A. Cusumano and R. W. Selby. 1997. How Microsoft builds software. *Communications of the ACM* 40, 6 (1997), 53–61.

P. Deemer, G. Benefield, C. Larman, and B. Vodde, 2010. *The scrum primer*. Retrieved August 26, 2023 from http://www.goodagile.com/scrumprimer/scrumprimer.pdf

K. Dikert, M. Paasivaara, and C. Lassenius. 2016. Challenges and success factors for large-scale agile transformations: A systematic literature review. *Journal of Systems and Software* 119 (2016), 87–108

T. Dingsøyr, F. O. Bjørnson, J. Schrof, and T. Sporsem. 2023. A longitudinal explanatory case study of coordination in a very large development programme: The impact of transitioning from a first- to a second-generation large-scale agile development method. *Empirical Software Engineering* 28, 1 (2023), 1–49.

T. Dingsøyr, D. Falessi, and K. Power. 2019. Agile development at scale: The next frontier. *IEEE Software* 36, 2 (2019), 30–38.

T. Dingsøyr, T. Fægri, and J. Itkonen. 2014. What is large in large-scale? A taxonomy of scale for agile software development. In: *Product-Focused Software Process Improvement, Lecture Notes in Computer Science,* Vol. 8892, A. Jedlitschka, P. Kuvaja, M. Kuhrmann, T. Männistö, J. Münch, and M. Raatikainen (Eds.). Springer International Publishing, 273–276.

T. Dingsøyr, N. B. Moe, T. E. Fægri, and E. A. Seim. 2018. Exploring software development at the very large-scale: A revelatory case study and research agenda for agile method adaptation. *Empirical Software Engineering* 23 (2018), 490–520.

T. Dingsøyr, K. Rolland, N. Moe, and E. Seim. 2017. Coordination in multi-team programmes: An investigation of the group mode in large-scale agile software development. *Procedia Computer Science* 121 (2017), 123–128.

Y. Dittrich, C. B. Michelsen, P. Tell, P. Lous, and A. Ebdrup. 2020. Exploring the evolution of software practices. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 493–504.

H. Edison, X. Wang, and K. C.. 2021. Comparing methods for large-scale agile software development: A systematic literature review. *IEEE Transactions on Software Engineering* 48, 8 (2021), 2709–2731.

J. Eckstein. 2016. Sociocracy – An organization model for large-scale agile development. *Proceedings of XP2016 Workshops*, Edinburgh, Scotland, UK.

A. Elshamy and A. Elssamadisy. 2006. Divide After You Conquer: An agile software development practice for large projects. *Proceedings of the 7th International Conference on Extreme Programming and Agile Processes in Software Engineering, XP 2006*, Lecture Notes in Computer Science, Vol. 4044. Springer, Oulu, Finland.

A. Elshamy and A. Elssamadisy. 2007. Applying agile to large projects: New agile software development practices for large projects. *Proceedings of the 8th International Conference on Agile Processes in Software Engineering and eXtreme Programming, XP 2007*, Lecture Notes in Computer Science, Vol. 4536. Springer, Como, Italy, 46–53.

N. A. Ernst and G. C. Murphy. 2012. Case studies in just-in-time requirements analysis. *Proceedings of the 2nd IEEE International Workshop on Empirical Requirements Engineering*. 25–32.

T. E. Fægri and N. B. Moe. 2015. Re-conceptualizing requirements engineering: Findings from a large-scale, agile project. *Proceedings of XP 2015 Workshops*. Helsinki, Finland.

M. Farjoun. 2010. Beyond dualism: Stability and change as a duality. *Academy of Management Review* 35, 2 (2010), 202–225.

M. Farmer. 2004. DecisionSpace Infrastructure: Agile development in a large, distributed team. *Proceedings of the Agile Development Conference*. IEEE Computer Society.

B. Fitzgerald. 1996. Formalized systems development methodologies: A critical perspective. *Information Systems Journal* 6, 1 (1996), 3–23.

B. Fitzgerald, K. J. Stol, R. O'Sullivan, and D. O'Brien. 2013. Scaling agile methods to regulated environments: An industry case study. *Proceedings of the International Conference on Software Engineering*. San Francisco, CA, USA.

M. Fowler and J. Highsmith. 2001. The agile manifesto. *Software Development* 9, 8 (2001), 28–35.

A. Fruhling and G. J. De Vreede. 2006. Field experiences with eXtreme programming: Developing an emergency response system. *Journal of Management Information Systems* 22, 4 (2006), 39–68.

A. Giddens. 1979. *Central Problems in Social Theory: Action, Structure and Contradictions in Social Analysis*, University of California Press, Berkeley, CA.

J. C. L. Goh, S. L. Pan, and M. Zuo. 2013. Developing the agile IS development practices in large-scale IT projects: The trust-mediated organizational controls and IT project team capabilities perspectives. *Journal of the Association for Information Systems* 14, 12 (2013), 722–756.

P. Gregory, L. Barroca, H. Sharp, A. Deshpande, and K. Taylor. 2016. The challenges that challenge: Engaging with agile practitioners' concerns. *Information and Software Technology* 77 (2016), 92–104.

G. Gunyho and J. Gutiérrez Plaza. 2011. Evolution of longer-term planning in a large scale agile project — F-Secure's experience. In: *Lecture Notes in Business Information Processing*. Springer, Cham, 306–315.

R Gupta, P. Manikreddy, and K. Arya. 2017. Pragmatic scrum transformation: Challenges, practices & impacts during the journey a case study in a multi-location legacy software product development team. *Proceedings of the 10th Innovations in Software Engineering Conference*. 147–156.

T. Gustavsson. 2017. Assigned roles for Inter-team coordination in Large-Scale Agile Development: A literature review. *Proceedings of the XP2017 Scientific Workshops*. ACM, 15.

T. Gustavsson. 2018. Impacts on team performance in large-scale agile software development. *Proceedings of the 17th Business Informatics Research Short Papers, BIR-WS 2018, CEUR-WS*. 421–431.

J. E. Hannay and H. C. Benestad. 2010. Perceived productivity threats in large agile development projects. *Proceedings of the 4th International Symposium on Empirical Software Engineering and Measurement*. Bolzano-Bozen.

G. Hanssen, G. Wedzinga, and M. Stupid. 2017. An assessment of avionics software development Practice: Justifications for an agile development process. *International Conference on Agile Software Development*. Springer.

V. T. Heikkilä, M. Paasivaara, and C. Lassenius. 2013a. Scrumbut, but Does It Matter? A mixed-method study of the planning process of a multi-team scrum organization. *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2013*. Baltimore, MD, 85–94.

V. T. Heikkilä, M. Paasivaara, C. Lassenius, and C. Engblom. 2013b. Continuous release planning in a large-scale scrum development organization at Ericsson. *Proceedings XP 2013, Lecture Notes in Business Information Processing*, Vol. 149. Springer, Cham, 195–209.

M. J. Hilt, D. Wagner, V. Osterlehner, and A. Kampker. 2016. Agile predevelopment of production technologies for electric energy systems – A case study in the automotive industry. *Procedia CIRP* 50 (2016), 88–93.

B. Hobbs and Y. Petit. 2017. Agile methods on large projects in large organizations. *Project Management Journal* 48, 3 (2017), 3–19.

R. Hoda and L. K. Murugesan. 2016. Multi-level agile project management challenges: A self-organizing team perspective. *Journal of Systems and Software* 117 (2016), 245–257.

R. Hoda, J. Noble, and S. Marshall. 2013. Self-organizing roles on agile software development teams. *IEEE Transactions on Software Engineering* 39, 3 (2013), 422–444.

W. S. Humphrey. 1989. *Managing the Software Process*. Addison-Wesley Longman Publishing Co., Inc.

W. A. Jackson. 1999. Dualism, duality and the complexity of economic institutions. *International Journal of Social Economics* 26, 4 (1999), 545–558.

M. Jørgensen. 2019. Relationships between project size, agile practices, and successful software development: Results and analysis. *IEEE Software* 36, 2 (2019), 39–43.

T. Kähkönen. 2004. Agile methods for large organizations — building communities of practice. *Proceedings of the Agile Development Conference, ADC 2004*. Salt Lake City, UT, 2–10.

P. Kettunen and M. Laanti. 2008. Combining agile software projects and large-scale organizational agility. *Software Process Improvement and Practice* 13, 2 (2008), 183–193.

B. A. Kitchenham and S. Charters. 2007. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*, EBSE Technical Report EBSE-2007-01, Version 2.3.

O. Ktata and G. Lévesque. 2009. Agile development: Issues and avenues requiring a substantial enhancement of the business perspective in large projects. *Proceedings of the 2nd Canadian Conference on Computer Science and Software Engineering*. 59–66

M. Kuhrmann, P. Diebold, J. Munch, P. Tell, K. Trektere, F. McCaffery, V. Garousi, M. Felderer, O. Linssen, E. Hanser, and C. R. Prause. 2018. Hybrid software development approaches in practice: A European perspective. *IEEE Software* 36, 4 (2018), 20–31.

M. Kuhrmann, P. Tell, R. Hebig, J. Klünder, J. Münch, O. Linssen, D. Pfahl, M. Felderer, C. Prause, S. MacDonell, J. Nakatumba-Nabende, D. Raffo, S. Beecham, E. Tüzün, G. López, N. Paez, D. Fontdevila, S. Licorish, S. Küpper, G. Ruhe, E. Knauss, O. Özcan-Top, P. Clarke, F. H. McCaffery, M. Genero, A. Vizcaino, M. Piattini, M. Kalinowski, T. Conte, R. Prikladnicki, S. Krusche, A. Coşkunçay, E. Scott, F. Calefato, S. Pimonova, R.-H. Pfeiffer, U. P. Schultz, R. Heldal, M. Fazal-Baqaie, C. Anslow, M. Nayebi, K. Schneider, S. Sauer, D. Winkler, S. Biffl, M. C. Bastarrica, and I. Richardson. 2021. What makes agile software development agile. *IEEE Transactions on Software Engineering* 48, 9 (2021), 3523–3539. https://ieeexplore.ieee.org/abstract/document/9496156?casa_token=we2ekchAcWIAAAAA:bzvX-FupYfThpIr3scDGl9pxoDvH5ZuBuiawmAGzt76UDwYUIcDsDDZ1mIg_FELVv-7uxMDaDw

M. Laanti. 2008. Implementing program model with agile principles in a large software development organization. *Proceedings of the Annual IEEE International Computer Software and Applications Conference.* 1383–1391.

M. Laanti. 2017. Agile transformation model for large software development organizations. *Proceedings of the XP2017.* ACM, 19.

L. Lagerberg, T. Skude, P. Emanuelsson, K. Sandahl, and D. Stahl. 2013. The impact of agile principles and practices on large-scale software development projects. *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement.* 348–356.

A. Langley. 1999. Strategies for theorizing from process data. *Academy of Management Review* 24, 4 (1999), 691–710.

A. Langley, C. Smallman, H. Tsoukas, and A. H. Van de Ven. 2013. Process studies of change in organization and management: Unveiling temporality, activity, and flow. *Academy of Management Journal* 56, 1 (2013), 1–13.

D. Leffingwell. 2016. *SAFe 4.0 Reference Guide: Scaled Agile Framework for Lean Software and Systems Engineering.* Addison-Wesley Professional.

Y. Lincoln and E. G. Guba. 1985. *Naturalistic Inquiry.* Sage, Newbury Park, CA.

Y. Lindsjørn, G. R. Bergersen, T. Dingsøyr, and D. I. K. Sjøberg. 2018. Teamwork quality and team performance: Exploring differences between small and large agile projects. *Proceedings of XP2018.* Porto, Portugal, 267–274.

A. Martini, L. Pareto, and J. Bosch. 2013. Improving businesses success by managing interactions among agile teams in large organizations. In *Software Business. From Physical Products to Software Services and Solutions: 4th International Conference (ICSOB'13, Potsdam, Germany, June 11-14, 2013. Proceedings 4)*, Springer Berlin Heidelberg, 60–72.

L. M. Maruping, V. Venkatesh, and R. Agarwal. 2009. A control theory perspective on agile methodology use and changing user requirements. *Information Systems Research* 20, 3 (2009), 377–399.

F. McCaffery, M. Lepmets, K. Trektere, O. Özcan-Top, and M. Pikkarainen. 2016. Agile medical device software development: Introducing agile practices into MDevSPICE. *International Journal on Advances in Life Sciences* 8, 1-2 (2016), 133–142.

M. B. Miles and A. M. Huberman. 1994. *Qualitative Data Analysis: An Expanded Sourcebook* (2nd Ed.). Sage.

H. Mintzberg. 1979. *The Structuring of Organisations.* Prentice-Hall, Englewood Cliffs, NJ.

N. B. Moe, T. Dingsøyr, and K. Rolland. 2018. To schedule or not to schedule? An investigation of meetings as an inter-team coordination mechanism in large-scale agile software development. *International Journal of Information Systems and Project Management* 6, 3 (2018), 45–59.

N. B. Moe, D. Šmite, A. Šblis, A. L. Börjesson, and P. Andréasson. 2014. Networking in a large-scale distributed agile project. *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement.*

P. Naur and B. Randell. (Eds.). (1968) *Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee.* Scientific Affairs Division, NATO, Brussels.

S. Nerur, A. Cannon, V. Balijepally, and P. Bond. 2010. Towards an understanding of the conceptual underpinnings of agile development methodologies. In *Agile Software Development*, T. Dingsøyr, T. Dybå, and N. B. Moe, (Eds.). Springer, Berlin Heidelberg, 15–29.

S. Nerur, R. Mahapatra, and G. Mangalaraj. 2005. Challenges of migrating to agile methodologies. *Communications of the ACM* 48, 5 (2005), 72–78.

M. Newman and D. Robey. 1992. A social process model of user-analyst relationships. *MIS Quarterly* 16, 2 (1992), 249–266.

L. S. Nowell, J. M. Norris, D. E. White, and N. J. Moules. 2017. Thematic analysis: Striving to meet the trustworthiness criteria. *International Journal of Qualitative Methods* 16, 1 (2017).

J. Nyfjord, S. Baathallath, and H. Kjellin. 2014. Conventions for coordinating large agile projects. *Proceedings of XP 2014 Workshops, Lecture Notes in Business Information Processing*, Vol. 199. Springer, Cham, 58–72.

M. Paasivaara, S. Durasiewicz, and C. Lassenius. 2008. Distributed agile development: Using scrum in a large project. *Proceedings of the IEEE International Conference on Global Software Engineering.* IEEE Computer Society. 87–95.

M. Paasivaara and C. Lassenius. 2011. Scaling scrum in a large distributed project. *Proceedings of the International Symposium on Empirical Software Engineering and Measurement.*

M. Paasivaara and C. Lassenius. 2014. Communities of practice in a large distributed agile software development organization—Case Ericsson. *Information and Software Technology* 56, 12 (2014), 1556–1577.

M. Paasivaara and C. Lassenius. 2016. Scaling scrum in a large globally distributed organization: A case study. *Proceedings of IEEE 11th International Conference on Global Software Engineering*. IEEE Computer Society. 74–83.

M. Paasivaara and C. Lassenius. 2019. Empower your agile organization: Community-based decision making in large-scale agile development at Ericsson. *IEEE Software* 36, 2 (2019), 64–69.

M. Paasivaara, C. Lassenius, and V. T. Heikkilä. 2012. Inter-team coordination in large-scale globally distributed scrum: Do Scrum-of-Scrums really work?. *Proceedings of the 6th ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2012*. Lund, 235–238.

S. D. Pawlowski and D. Robey. 2004. Bridging user organizations: Knowledge brokering and the work of information technology professionals. *MIS Quarterly* 28, 4 (2004), 645–672.

A. Putta, M. Paasivaara, and C. Lassenius. 2018. Benefits and challenges of adopting the scaled agile framework (SAFe): Preliminary results from a multivocal literature review. *Proceedings of the International Conference on Product-Focused Software Process Improvement*. Springer, Cham, 334–351.

M. R. J. Qureshi. 2012. Agile software development methodology for medium and large projects. *IET Software* 6, 4 (2012), 358–363.

B. Ramesh, K. Mohan, and L. Cao. 2012. Ambidexterity in agile distributed development: An empirical investigation. *Information Systems Research* 23, 2 (2012), 323–339.

A. Read and R. O. Briggs. 2012. The many lives of an agile story: Design processes, design products, and understandings in a large-scale agile development project. *Proceedings of the 45th Hawaii International Conference on System Sciences*. Maui, HI, USA.

K. H. Rolland, B. Fitzgerald, T. Dingsøyr, and K.-J. Stol. 2016. Problematizing agile in the large: Alternative assumptions for large-scale agile development. *Proceedings of the International Conference on Information Systems*. Dublin, Ireland.

K. H. Rolland, G. Ghinea, and T. Gronli. 2015. Ambidextrous enterprise architecting: Betting on the future and hacking path-dependencies. *Proceedings of the European Conference on Information Systems*.

K. H. Rolland, V. Mikkelsen, and A. Næss. 2016. Tailoring agile in the large: Experience and reflections from a large-scale agile software development project. *Proceedings XP 2016, Lecture Notes in Business Information Processing*, Vol. 251, Springer, Cham, 244–251.

P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering* 14 (2009), 131–164.

D. Russo. 2021. The Agile success model: A mixed-methods study of a large-scale agile transformation. *ACM Transactions on Software Engineering and Methodology* 30, 4 (2021), 1–46.

R. Sabherwal, R. Hirschheim, and T. Goles, 2001. The dynamics of alignment: Insights from a punctuated equilibrium model. *Organization Science* 12, 2 (2001), 179–197.

A. Šāblis and D. Šmite. 2016. Agile teams in large-scale distributed context: Isolated or connected?. *Proceedings of the Scientific Workshop Proceedings of XP2016*. ACM, 10.

K. Schwaber. 2013. unSAFe at Any Speed. Ken Schwaber's Blog: Telling it like it is. Retrieved August 26, 2023 from https://kenschwaber.wordpress.com/2013/08/06/unsafe-atany-speed/

A. Scheerer and T. Kude. 2014. Exploring coordination in large-scale agile software development: A multiteam systems perspective. In *35th International Conference on Information Systems*. Auckland, New Zealand.

N. Sekitoleko, F. Evbota, E. Knauss, A. Sandberg, M. Chaudron, and H. H. Olsson. 2014. Technical dependency challenges in large-scale agile software development. In *Proceedings of the International Conference on Agile Software Development: Agile Processes in Software Engineering and Extreme Programming*. Springer, Rome, Italy.

D. Šmite, N. B. Moe, G. Levinta, and M. Floryan. 2019. Spotify Guilds: How to succeed with knowledge sharing in large-scale agile organizations. *IEEE Software* 36, 2 (2019), 51–57.

H. Søvik and M. Forfang. 2010. Tech challenges in a large-scale agile project. *Proceedings XP2010, Lecture Notes in Business Information Processing*, Vol. 48 (2010), 353–361.

State of Agile, State of Agile Report. 2022. 16th Annual State of Agile Report. Retrieved August 26, 2023 from https://stateofagile.com/#

T. Stålhane, T. Myklebust, and G. K. Hanssen. 2012. The application of safe scrum to IEC61508 certifiable software. *Proceedings of the 11th International Probabilistic Safety Assessment and Management Conference and the Annual European Safety and Reliability Conference*. Helsinki, Finland.

C. J. Stettina and M. N. Smit. 2016. Team portfolio scrum: An action research on multitasking in multi-project scrum teams. *Proceedings of the International Conference on Agile Software Development*. Springer, Cham. 79–91.

S. Sundararajan, M. Bhasi, and P. K. Vijayaraghavan. 2014. Case study on risk management practice in large offshore-outsourced agile software projects. *IET Software* 8, 6 (2014), 245–257.

T. Taylor and T. Standish. 1982. Initial thoughts on rapid prototyping techniques. *ACM SIGSOFT Software Engineering Notes* 7, 5 (1982), 160–166.

B. Tessem and F. Maurer. 2007. Job satisfaction and motivation in a large agile team. *Proceedings of the International Conference on Extreme Programming and Agile Processes in Software Engineering*. 54–61.

Ö. Uludağ, M. Kleehaus, S. Ercelik, and F. Matthes. 2019. Using social network analysis to investigate the collaboration between architects and agile teams: A case study of a large-scale agile development program in a German consumer electronics company. *Proceedings of XP2019, Lecture Notes in Business Information Processing*, Vol. 355 (2019), Springer, Cham, 137–153.

G. van Waardenburg and H. van Vliet. 2013. When agile meets the enterprise. *Information and Software Technology* 55 (2013), 2154–2171.

R. Vidgen and X. Wang. 2009. Coevolving systems and the organization of agile software development. *Information Systems Research* 20, 3 (2009).

V. Vinekar, C. W. Slinkman, and S. Nerur. 2006. Can agile and traditional systems development approaches coexist? An ambidextrous view. *Information Systems Management* 23, 3 (2006), 31–42.

J. Vlietland and H. van Vliet. 2015. Towards a governance framework for chains of scrum teams. *Information and Software Technology* 57 (2015), 52–65.

A. Y. Wale-Kolade. 2015. Integrating usability work into a large inter-organisational agile development project: Tactics developed by usability designers. *Journal of Systems and Software* 100 (2015), 54–66.

S. K. Weiss and P. Brune. 2017. Crossing the boundaries–agile methods in large-scale, plan-driven organizations: A case study from the financial services industry. *Proceedings of the International Conference on Advanced Information Systems Engineering*. Springer, Cham. 380–393.

D. West, M. Gilpin, T. Grant, and A. Anderson. 2011. Water-scrum-fall is the reality of agile for most organizations today. *Forrester Research* (2011).

L. Williams and A. Cockburn. 2003. Agile software development: It's about feedback and change. *IEEE Computer* 36, 6 (2003), 39–43.

Y. Zheng, W. Venter, and T. Cornford. 2011. Collective agility, paradox and organizational improvisation: The development of a particle physics grid. *Information Systems Journal* 21, 4 (2001), 303–333.

Received 5 May 2022; revised 27 June 2023; accepted 3 August 2023