

Chapter 2

Theoretical and Conceptual Foundations

The theoretical foundations outline extant literature on coordination as well as on agile software development. To provide a basis for agile development on a large scale, first the underlying core concepts of coordination and agile software development on a small scale are presented before delving into prior work on coordination in multiteam systems and large-scale agile software development.

2.1 Coordination

Coordination is a multi-faceted research area, which takes its inputs from a variety of fields including but not limited to organizational theory and teamwork studies. Before the following chapter gives an overview of the different facets of coordination, different definitions of coordination will be discussed.

Across the different fields, there exist many definitions of coordination. Table 2.1 gives an overview. While covering a wide array of research interests, three common aspects within the definitions become apparent. The first refers to the *actors* who need to work together, the second being the *work*, which is interdependent, and thirdly a *goal* in the form of a piece of work, which is achieved (Okhuysen and Bechky 2009). In the context of software development, coordination can be viewed as the establishment of common understanding on what should be built, how to build it as well as alignment and integrating activities (Kraut and Streeter 1995). In its purest form, coordination is the “achievement of concerted action” (Kotlarsky et al. 2008, p. 96). It is this definition that will be adopted in this work.

Table 2.1 Definitions of coordination

Definition	Authors
Coordination is the achievement of concerted action	Kotlarsky et al. (2008, p. 96) citing Goodhue and Thompson (1995)
Coordination means integrating or linking together different parts of an organization to accomplish a collective set of tasks	Van De Ven et al. (1976, p. 322)
Coordination is managing dependencies between activities	Malone and Crowston (1994, p. 90)
Different people working on a common project agree to a common definition of what they are building, share information, and mesh their activities	Kraut and Streeter (1995, p. 69)
A temporally unfolding and contextualized process of input regulation and interaction articulation to realize a collective performance	Faraj and Xiao (2006, p. 1157)

2.1.1 Coordination in Organizational Theory

The study of coordination in organizational theory has a long and rich history. In the classical work from March and Simon (1958), the two prominent schools of thought of the time are presented: scientific management (Taylor 1911) and the administrative management school (Gulick and Urwick 1937). Scientific management deals with the analysis and management of workflows, especially of routine production tasks, and seeks to create prescribed operating procedures for effectiveness in organizations (Tosi 2008). The departmentalization model of the administrative school seeks to achieve organizational goals by examining how to organize jobs into components within the organization (e.g. grouping by common purpose or by common processes).

March and Simon (1958) criticized both streams because of their lack of appreciation for the human factor. They distinguished two main types of coordination: (1) coordination by plan and (2) coordination by feedback. In standardized and repetitive situations, the tasks become “programmable” through scheduling and are thus best coordinated by plan. If the circumstances require the diffusion of new information, coordination by feedback seems better suited. Feedback, being based on communication between employees, is also seen as a way to deal with deviations from the plan (March and Simon 1958).

Several years later, Thompson (1967) published his seminal work in which he cites March and Simon (1958) and presents three generic forms how coordination can be achieved. Based on the type of interdependence, an appropriate method of coordination is proposed. Three types of interdependencies are illustrated, (1) *pooled* or *general*, (2) *sequential* and (3) *reciprocal*. The pooled dependency, being the least coupled of all, arises when units within an organization accomplish completely separate tasks and do not need to interact. In the end, each business unit performs its part in the overall picture of the organization, which creates an implicit

dependency to the other entities. The second dependency type, sequential, arises when one unit depends on the output of another to continue its work. A typical example for this is the production assembly line, where each consecutive step needs the input from the previous one. Finally, reciprocal dependency is the strongest form of coupling and occurs when input and output flow in both directions simultaneously between the dependent units (Thompson 1967).

Thompson (1967) suggests an appropriate coordination style for each type of task interdependency. As such, a pooled or generalized interdependence should be coordinated by *standardization* with little communication and decision effort. The sequential dependency type is best coordinated by *planning* and requires medium effort for communication and decision-making. The final type of dependency, reciprocal, should be coordinated by *mutual adjustment*, which is based on March and Simon's (1958) coordination by *feedback*. This coordination type is most challenging with regards to communication and decision efforts (Thompson 1967).

Van De Ven et al. (1976) considered three modes of coordination based on the work of March and Simon (1958) and Thompson (1967), namely *impersonal*, *personal* and *group* coordination. They argue that all coordination types in the form of programming, e.g. plans, rules and hierarchies (see Kieser 1993), can be grouped in the impersonal mode. The feedback or mutual adjustment type can be divided into personal and group modes, depending on the number of people involved. Furthermore, the personal mode includes either horizontal or vertical channels, up or down the hierarchy or across hierarchical levels, through which communication occurs. The group mode can be split into scheduled or unscheduled meetings of several people. In their study, Van De Ven et al. (1976) identified which factors influence the choice of coordination mechanisms. Three factors were chosen: *task uncertainty*, *task interdependence* and *size of work unit* (i.e. number of people employed in a work unit). They found that higher task uncertainty leads to higher use of mutual adjustment through horizontal channels and group meetings. With rising task interdependence, the amount of coordination mechanisms across all three modes rises as well. Finally, a higher unit size induces the use of more impersonal modes in the form of plans.

Mintzberg (1980) introduces five coordination mechanisms: (1) *direct supervision*, (2) *standardization of work processes*, (3) *standardization of outputs*, (4) *skills* and (5) *mutual adjustment*. Coordination via direct supervision entails one individual giving specific orders to others and thereby ensuring coordination. Standardization of work processes involves the regulation of how the work is done by rules or regulations, while the standardization of outputs specifies performance measures or work outputs. Through a set of standard expertise and knowledge, the standardization of skills is achieved. Finally, coordination by mutual adjustment includes communication by informal means among people (Mintzberg 1980). Concerning the location of coordination within organizations, Mintzberg (1980) advances the concept of *centralization* or *decentralization* of decision making within the firm boundaries as a key determinant of organizational structure.

This chapter has reviewed key works in organizational theory relating to coordination and the mechanisms used to achieve it. The next chapter will delve deeper into what is known as coordination theory (Malone and Crowston 1990, 1994).

2.1.2 Coordination Theory

Proposed by Malone and Crowston (1990), the main focus of their work lies on the analysis of coordination with respect to actors, interdependent activities and goals (Malone and Crowston 1994). The naming of this stream of literature is misleading, as the work published does not conform to the longstanding discussion on what constitutes a theory (Bacharach 1989; Sutton and Staw 1995). Bacharach (1989) pointed out based on Hempel (1965), that “the vocabulary of science has two basic functions: (a) to adequately describe the objects and events being investigated and (b) to establish theories by which events and objects can be explained and predicted” (Bacharach 1989, p. 496). While the work published on this topic (cf. Malone et al. 2003; Malone and Crowston 1991, 1994) can be seen as fulfilling function (a) since it describes dependency types and in parts how these interdependencies should be coordinated, no predictive power arises as no hypotheses or propositions are stated.

The types of dependencies considered by Malone et al. (1999) exist between resources and activities. Three basic types of dependencies that ensue from resources being associated with multiple activities are defined. The fit dependency occurs when several activities collectively produce a single resource. The manufacturing industry shows several examples of such dependencies, e.g. the production of cars or planes where several components need to be integrated to eventually form a complete product. The flow dependency occurs whenever a resource is produced by one activity, which is then the input to another. According to Malone et al. (1999), the flow dependency is comprised of three different kinds of constraints, a prerequisite, an accessibility and a usability constraint. All of these dependencies must be managed for a flow dependency, e.g. the right thing (usability) needs to be in the right place (accessibility) at the right time (prerequisite). Finally, a sharing dependency arises when several activities all use the same resource, i.e. the use of the same machine in several production processes.

More examples of dependency types that are given by Malone and Crowston (1994) are listed in Table 2.2. Here, the producer-consumer dependency is

Table 2.2 Dependencies and examples of coordination processes

Dependency	Examples of coordination processes
Producer-consumer (flow)	Sequencing and tracking for prerequisite constraints Inventory management for accessibility constraints Standardization for usability constraints
Task-subtask	Goal selection, goal decomposition

Based on Malone and Crowston (1994)

characterized by an activity producing a resource that is consumed by another activity, essentially the same as a flow dependency mentioned earlier. Exemplary coordination processes for managing such a dependency are sequencing and tracking for a prerequisite constraint, an inventory management for the accessibility constraint and lastly standardization for the usability constraint.

Malone and Crowston (1994) propose that the management of task-subtask dependencies can be achieved by the coordination processes goal selection and decomposition. An often found dependency, the task-subtask dependency, represents a situation where an overall goal includes a group of subtasks, which need to be completed in order to achieve the higher goal. A top-down goal decomposition can be utilized to manage task-subtask dependencies. Although this type of dependency is usually managed through a sequential process of goal selection and decomposition, it is entirely possible that a bottom-up identification of goals is achieved when employees recognize that tasks they are doing or new ideas they have could lead to new goals in line with the overall goal (Malone and Crowston 1994).

As mentioned earlier, no predictive or explanatory power arises from this work. In its current state it is deemed more of a pattern model (Crowston et al. 2006). Furthermore, issues of context and time are not taken into consideration. The diverse range of previous work on the topic of coordination (cf. Okhuysen and Bechky 2009) shows the necessity to include the context into any contemplation on the subject. Especially aspects of time are of great significance, as coordination is considered to be a temporally enfolding phenomenon (see Table 2.1).

Within the area of team cognition studies, coordination of individual team members has been a long-standing topic of interest. In the next chapter, aspects of coordination within this research stream will be presented.

2.1.3 Coordination in Team Cognition Studies

Within the research field of team cognition studies, which covers established concepts such as shared mental models or transactive memory systems within teams, the coordination modes have been subdivided into *explicit* and *implicit* coordination styles (Espinosa et al. 2004). The previously presented coordination modes in organizational theory are considered to be explicit in that they are carried out deliberately to coordinate groups in order to achieve a state of coordination. Implicit coordination on the other hand, is a mode of coordination that arises as a consequence of other acts and is used without the intention to coordinate. An example of this type is the often mentioned “water cooler” talk, i.e. informal exchanges of information between colleagues that nevertheless lead to better coordination.

Espinosa et al. (2010) present a taxonomy of coordination types including, mechanistic, organic and cognitive coordination. The first two originate from organizational theory as presented in Sect. 2.1.1, with cognitive coordination

arising from research on team cognition. While *mechanistic coordination* includes coordination by plan or rules with little communication, *organic coordination* refers to coordination by means of mutual adjustment or feedback via interaction. This communication can be formal and planned or informal and spontaneous. *Cognitive coordination*, on the other hand, is based on tacit team knowledge the actors have about each other and is achieved implicitly (Rico et al. 2008).

Shared mental models (Cannon-Bowers et al. 1993) and transactive memory systems (Moreland et al. 1996) are two examples of cognitive or implicit coordination mechanisms (Espinosa et al. 2004). Shared mental models are the “common or overlapping cognitive representations of task requirements, procedures and role responsibilities” (Cannon-Bowers et al. 1993, p. 222). The transactive memory system contains the knowledge embedded in each person individually and a metamemory about the expertise domains of the other participants in this system (Moreland et al. 1996). While shared mental models represent a shared understanding between actors, transactive memory systems conceptualize the aspect of knowing who knows what. The shared mental model construct is central to teamwork, as it acts as a facilitator for the teams’ goal focus and contributes to common understanding and action (Salas et al. 2005).

While this stream of coordination research proposes team cognition constructs as coordination mechanisms, the question remains what tangible coordination activity takes place. As the implicit nature of these types suggests, no overt coordination activity is observable. As such, these constructs may be viewed as traits, which lead to a reduction of mechanistic coordination, or even as conditions necessary for coordinated action.

2.1.4 Outcomes and Conditions of Coordination

After decades of research in the organizational domain (cf. Mintzberg 1983; Thompson 1967; Van De Ven et al. 1976), recent advances on the conditions and outcomes of coordination have been made outside of this field, namely in the areas of team cognition (cf. Cannon-Bowers et al. 1993; Moreland et al. 1996) and information systems (cf. Pikkarainen et al. 2008; Strode et al. 2012). The following section summarizes research on outcomes and conditions of coordination.

From the definitions of coordination in Table 2.1 it can be deduced that coordination can be conceptualized in two ways. Kotlarsky et al. (2008, p. 96) view coordination as “the achievement of concerted action”, which depicts coordination as a state that is achieved, similarly to Lawrence and Lorsch (1967) who view coordination as a state to be attained. By contrast, Faraj and Xiao (2006, p. 1157) or Malone and Crowston (1994, p. 90) describe it as the management of dependencies or the “contextualized process of input regulation and interaction articulation”, which expresses coordination as a process. In the end, it can be understood either as the process necessary to achieve concerted action or as the state that is achieved by this process.

Both coordination as a process and coordinated action as state are regarded as central aspects in achieving an overarching performance measure. The positive influence of coordination on performance measures has been widely recognized throughout literature (cf. Cheng 1984; Faraj and Xiao 2006; Kozlowski and Bell 2003; Nidumolu 1995; Simon 1976). As such, Cheng (1984, p. 830) describes coordination as “a necessary condition for effective organizational performance”, however it is not sufficient by itself. Within the domain of team research, Kozlowski and Bell (2003, p. 353) state that previous “empirical research has shown team coordination to be an important correlate of team performance”. In the domain of software development, Nidumolu (1995) shows that higher levels of coordination lead to higher levels of project performance. In their classical work from 1967, Lawrence and Lorsch reported a positive relationship between coordination and organizational performance. Similarly, Simon (1976) states that organizational objectives are achieved through the coordination of the participants’ behavior. Finally, Faraj and Xiao (2006, p. 1157) prominently included the positive influence of coordination in their definition by describing coordination as a process to “realize a collective performance”.

The outcomes of coordination processes have been characterized in multiple different ways. As previously stated, Okhuysen and Bechky (2009) conceptualize this outcome as coordinated action. Coordination effectiveness by Lee et al. (2013) focuses on the aspects of redundant work and roadblocks in coordination for their construct. Espinosa et al. (2012) introduce the construct coordination problems including items on missed delivery dates, misunderstanding, redundant work etc. This conceptualization is very encompassing concerning coordination problems, but does not depict the underlying forces, which coordination mechanisms seem to influence. Strode et al. (2011) present their conceptualization of coordination effectiveness with the two main dimensions of implicit and explicit coordination outcomes. The implicit domain includes five knowledge related aspects (e.g. know why, know what is going on, know what to do and when, know who is doing what, know who knows what) and three aspects in the explicit dimension (right place, right thing and right time). However, a problem with their view of coordination effectiveness as an outcome construct is the seemingly mediating role the implicit domain plays on the explicit domain. As such, knowing what to do and when, will strongly influence the explicit dimensions *right thing* and *right time*.

To consolidate the widespread knowledge and identify the underlying conditions of coordination, Okhuysen and Bechky (2009) reviewed existing coordination literature in the diverse streams it has been published in and integrated this knowledge back into organizational theory. Previous studies focused on the mechanisms to achieve coordination, their mix within the coordination strategy or the process of coordination to achieve team effectiveness. Therefore, a unified view on the direct conditions, or intermediate states, leading to coordination has remained elusive. Okhuysen and Bechky (2009) propose three integrating conditions that lead to what they refer to as *coordinated action*: (1) *common understanding*, (2) *predictability* and (3) *accountability* (see Table 2.3). Common understanding supports coordinated action “by providing a shared perspective on the whole task and how

Table 2.3 Definitions of integrating conditions for coordinated action

Term	Definition	Source
Common understanding	A shared perspective on the whole task and how individuals' work fits within the whole	Okhuysen and Bechky (2009, p. 488)
Accountability	Addresses the question of who is responsible for specific elements of the task and makes clear where the responsibilities of interdependent parties lie	Okhuysen and Bechky (2009, p. 483)
Predictability	Enables interdependent parties to anticipate subsequent task related activity by knowing what the elements of the task are and when they happen	Okhuysen and Bechky (2009, p. 486)

individuals' work fits within the whole" (Okhuysen and Bechky 2009, p. 488). The aspect of predictability "enables interdependent parties to anticipate subsequent task-related activity by knowing what the elements of the task are and when they happen" (Okhuysen and Bechky 2009, p. 486). Finally, accountability delimits "who is responsible for specific elements of the task" (Okhuysen and Bechky 2009, p. 483). In proposing these three conditions, they stress the importance of the elements that enable coordinated action and "working together and separately, assist in the enactment of coordinated activity" (Okhuysen and Bechky 2009, p. 492). Through the theoretical separation of the coordination mechanism from the coordinated action they support, the intermediate outcomes of these mechanisms can be investigated and explained more deeply (Okhuysen and Bechky 2009).

2.1.5 Summary

The preceding discussion shows the focus areas of the previous literature on coordination. Prior work has concentrated on dependencies between actors in the system to be coordinated and the necessary coordination mechanisms to achieve concerted action, depending on situational factors. This neglects the underlying factors necessary for coordination and posits that one merely needs the correct coordination mechanisms for the situation at hand. Based on Okhuysen and Bechky's (2009) three integrating conditions for coordinated action, common understanding, accountability and predictability, this work tries to fill this gap by empirically investigating inter-team coordination in a field setting.

2.2 Teams and Multiteam Systems

As little is known about the structure of large-scale agile development organizations, a unit of analysis needs to be identified which depicts the organizational structure present in such systems. To benefit from extant knowledge in neighboring

disciplines, the multiteam systems unit rooted in organizational psychology is chosen to conceptualize the team of teams setup.

Over the past years, companies have focused more and more on structuring their organization into smaller units of people and thus implementing a team-based organization. “A team [...] can be defined as a collection of individuals who share a common goal, whose actions and outcomes are interdependent, who are perceived by themselves and others as a social entity, and who are embedded in an organizational context” (Devine 2002, p. 291). Previous definitions of software development teams as project teams (Devine 2002; Kozlowski and Bell 2003) see these teams as a temporary unit, assembled for a specific purpose and as soon as their job is done they disband. However, in software product development this is not the case. Here, the team composition is of a permanent nature as the same teams will be developing the next version of the software product they have been working on previously. In this research the team is seen as the smallest entity within a multiteam system consisting of several teams.

With the introduction of agile development approaches in the software industry, the guideline has been to create teams of around seven plus or minus two people and scale this through a hierarchical *team of teams* setup (Larman and Vodde 2008), where several teams have to work closely together in order to release a single software product.

These sorts of collectives have been of ongoing interest to the stream of multiteam systems research. This organizational setup has been defined as a *multiteam system* by Mathieu et al. (2001), who assert that MTSs are “two or more teams that interface directly and interdependently in response to environmental contingencies toward the accomplishment of collective goals” (Mathieu et al. 2001, p. 290). In contrast to other organizational forms such as subsystems (cf. Katz and Kahn 1978) or matrix organizations (cf. Davis and Lawrence 1977), an MTS is a team-based collective with members requiring collaborative integration of teamwork which leads to the high degree of interdependence within MTSs (Zaccaro et al. 2012).

The core elements of an MTS are a goal hierarchy and functional inter-team dependencies. The collective goal of this system can be broken down into a goal hierarchy and constitutes a key characteristic of any MTS. The goal hierarchy marks the boundary of an MTS in that all teams within the system share at least a distal goal while the individual teams pursue their more proximal goals. This structure of goals leads to teams displaying input, process and outcome interdependencies with at least one other team (Mathieu et al. 2001). Within software product development the distal goal of all involved teams is the completion of a new release of the software that is being implemented. The proximal goals of each team within the MTS usually comprise of responsibility for a software subcomponent of the product or of certain features to be added to the next release of the software.

Over the past decades, research on teams has been widespread, leading to a deep understanding of team processes and work (cf. Guzzo and Dickson 1996; Ilgen et al. 2004; Kozlowski and Bell 2003; Mathieu and Maynard 2008). Our understanding of multiteam systems and especially these systems in a software development environment remains exceptionally limited.

2.3 Agile Software Development

The history of agile software development is often traced back to the Agile Manifesto in the year 2001 (Fowler and Highsmith 2001). The origins of the underlying concepts, namely iterative and incremental development, reach as far back as the 1950s (Larman and Basili 2003). Even early plan-based approaches established iterative feedback as an essential element. The waterfall model (Benington 1956), often described as the foe of serious agilists, showed first beginnings of iterative ideas through Royce's (1970) adaptation, which included iterative feedback.

Over the years, newer software development models have continued to include more and more ideas based on iterations and feedback. Starting with the spiral model, the notion of several iterations was codified into the development process as a core component (Boehm 1988). While the waterfall model is heavily specification driven, the spiral model is driven by risk management. In the 1990s, resulting from the proliferation of object-oriented programming languages and design, the Rational Unified Process (RUP) (Kruchten 1998) was specified to be model- or architecture-based. It distinguished itself by constituting a process framework, which can be adapted to the setting in which it is deployed.

What all of these process models have in common is their rather heavyweight approach to process management, specifications and large design upfront based on documents, and comparatively long iteration cycles. As a countermovement to these heavyweight models, the mid-1990s saw a push towards more lightweight processes.

One of the first was the Dynamic Systems Development Method (DSDM Consortium, n.d.) in 1994. It values eight principles: (1) Focus on the business need, (2) deliver on time, (3) collaborate, (4) never compromise quality, (5) build incrementally from firm foundations, (6) develop iteratively, (7) communicate continuously and clearly and (8) demonstrate control. A year later, Scrum was first publicly presented (Sutherland and Schwaber 1995). It defines a project management framework based on time-boxed iterations with defined roles and a focus on face-to-face communication. Extreme Programming (XP) saw the light of day in 1996 and was later published in book form (Beck 2001). It is based on a set of values, fundamental principles, activities and practices. In contrast to Scrum, XP defines technical practices for the development work of programmers.

In 2001, several prominent advocates of lightweight development methods gathered to create the Agile Manifesto (Fowler and Highsmith 2001). They proposed four values, which constitute the essence of agile development methods:

individuals and interactions over processes and tool

working software over comprehensive documentation

customer collaboration over contract negotiation

responding to change over following a plan

The first value emphasizes the people orientation of the agile movement. The relationships between team members, customers and partners are valued above

prescribed processes and some methods’ heavy reliance on cumbersome tools. Agile methods value close proximity in the form of co-location and intensive team communication. In the end, a piece of working software is more useful than a sizeable documentation of software that is not what the customer wanted or even worse, unusable. At the end of short iterations, the goal is to present an increment of working software to the customer for validation. With the help of automated testing, quality, as part of the ‘working’ software, is ensured. Trust between customer and development team is the essential foundation of agile methods. Instead of stressing precise contracts, the customer is involved very heavily in the development of the software to ensure he is satisfied with the product. As agile focuses on fast value delivery, risk is minimized for the customer as well. The last value points to the core of agile development, the necessity to adapt to a changing environment. All people involved in an agile project, the customers and the developers, both experts in their field, can judge if a project needs to adjust to better satisfy the needs of the customer and not just plainly follow a specified plan.

These values illustrate the considerable mind shift in agile software development, which emphasizes cross-functional teams with time-boxed development phases and continuous management of requirements. Abrahamsson et al. (2002) state that a development method is agile when it is incremental, cooperative, straightforward and adaptive. This differs strongly from traditional development. The core differences between agile development and traditional approaches are summarized by Nerur et al. (2005) in Table 2.4.

Table 2.4 Traditional versus agile software development

	Traditional development	Agile development
Fundamental assumptions	Systems are fully specifiable, predictable, and are built through meticulous and extensive planning	High-quality software is developed by small teams using the principles of continuous design improvement and testing based on rapid feedback and change
Control	Process centric	People centric
Management style	Command and control	Leadership and collaboration
Knowledge management	Explicit	Tacit
Role assignment	Individual—favors specialization	Self-organizing teams—encourages role inter-changeability
Communication	Formal	Informal
Customer’s role	Important	Critical
Development model	Life-cycle model (waterfall, spiral or some variation)	The evolutionary-delivery model
Desired organizational form	Mechanistic (bureaucratic with high formalization), aimed at large organizations	Organic (flexible and participative encouraging cooperative social action), aimed at small and medium-sized organizations

Based on Nerur et al. (2005, p. 75)

While many methods are considered agile, only a select few have become prominent in industry. By far the most widely used method is Scrum (VersionOne Inc. 2013). The following section will give an overview of the Scrum framework.

2.3.1 *The Scrum Framework*

Originally presented at a conference in 1995 (Sutherland and Schwaber 1995), the idea is based on research from Takeuchi and Nonaka (1986) who investigated new product development in Japan and the United States. The investigated companies have taken a new approach to product development, which they named the rugby approach. Companies utilizing this approach show six characteristics of new product development: (1) built-in instability, (2) self-organizing project teams, (3) overlapping development phases, (4) multilearning, (5) subtle control and (6) organizational transfer of learning (Takeuchi and Nonaka 1986).

Built-in instability is created by top management, which tasks a project team with a very challenging goal and gives the team great freedom in achieving this goal. Because of this freedom, the project team can act like a start-up company in taking initiatives and risks. Instead of relying on a sequential order of process phases, these phases overlap to minimize bottlenecks that may lead to a system standstill. Through overlapping project phases, shared responsibility and commitment are increased. For example, the involvement of production experts in early phases can accelerate the development process, as feedback concerning the production is gathered early on and not in one of the later phases. Multilearning involves gathering information from outside in order to respond quickly to a changing market. On the team level, continuous learning is necessary, as the challenging goals need new strategies to solve them. In contrast to the command and control style of management, the rugby approach suggests self-control and control through peer pressure as ways to manage such projects. Starting with the right people in an open work environment, tolerating mistakes and rewarding group performance as opposed to individual performance, leads to more subtle ways of management being able to control such projects. An organizational transfer of learning is achieved by assigning key people to follow-up projects to promote what they have previously learned (Takeuchi and Nonaka 1986).

Based on the presented rugby approach to new product development, the Scrum framework was established for software development projects. While often referred to as a software development method, Scrum is strictly speaking a project management framework for software development projects. Within this framework, no technical aspects of development work are defined. What follows is an in-depth look at the Scrum framework based on its roles, events and artifacts (based on Deemer et al. 2012; Schwaber and Beedle 2002; Schwaber and Sutherland 2013).

The *Product Owner* (PO) is responsible for the business value of the product to be developed. He identifies product features and translates these into an ordered list, with the highest value items on top. This role has the sole responsibility for the

profit or loss of the developed product. In general, there are two scenarios for the role of the Product Owner. In the first case, the customer and the Product Owner is the same person, which is often the case in internal development projects. The other option is the Product Owner as the representative of the customer, of whom there might be many in the market. In this case, he consolidates the customers' needs and requirements to form a backlog (Deemer et al. 2012).

The *development team* implements the backlog items specified by the Product Owner. There should be no fixed specialists, such as tester or architect, only team members. The team is cross-functional and includes all necessary expertise to implement the backlog items and deliver a potentially shippable product at the end of a sprint. Areas of expertise that do not yet exist within the team are expected to be acquired through continuous learning. As a self-organizing and autonomous team, the members decide which set of backlog items to implement from those specified and prioritized by the Product Owner. The team is most effective if all members of the team are allocated one hundred percent to one product and do not switch between projects, which avoids costly context switching. The size of a development team is specified to be seven plus minus two people. Ideally, a Scrum team is co-located (Deemer et al. 2012).

The *Scrum Master* (SM) acts as a facilitator to the development team and the Product Owner. He helps both to learn and apply the Scrum process and is an essential figure in Scrum's fundamental principle 'inspect and adapt'. Contrary to other project management approaches, the Scrum Master is not a project manager as he neither manages the team, nor manages the product under development. In some team setups, he also takes on part-time development duties. The Scrum Master does not have people management responsibilities; instead, he acts as a coach and teacher and ensures that the Scrum process is followed. Furthermore, he makes sure that impediments are taken care of and resolved. As Scrum usually entails a cultural change within the organization, the Scrum Master guides not only the team and the Product Owner through this change, but also helps the larger organization in implementing Scrum (Deemer et al. 2012).

At the center of Scrum lies the *sprint*, a time-boxed period during which a potentially shippable software product increment is built. In practice, the sprint is usually two or four weeks long. During a development effort, one sprint follows the next to form a continuous succession. Within one sprint, no changes are allowed in the product backlog that would jeopardize the current sprint goal as agreed upon between team and Product Owner. If backlog items turn out to be larger or more difficult than initially expected, items can be de-committed and moved to the next sprint in order to still successfully complete the other items (Schwaber and Sutherland 2013).

As the name suggests, the *sprint planning* is a meeting to prepare for the upcoming sprint. It should answer what will be delivered and how this will be done. In the first part of this meeting, the Product Owner and the team examine what has been determined as the sprint goal. Continuing, the backlog items necessary to achieve that goal are examined and discussed to establish a common understanding of the functionality to be developed. Based on the value-ordered list of backlog

items in the product backlog, the team chooses which items to pick for the next sprint. This selection is usually based on the team's capacity in the upcoming sprint and their past performance. The second part of the planning meeting pertains to how the items in the sprint backlog will be implemented. The team starts by designing the work needed to transform the selected backlog items into a working product increment. At the end of the sprint planning meeting it should be clear what the team commits itself to implement and how this will be done (Schwaber and Sutherland 2013).

The *daily Scrum* is a 15-min meeting recurring every day. It is often called *daily standup*, to show that it is intended to be a quick meeting. In this time-boxed event, the development team synchronizes its activities and creates a plan for the next 24 h. During the meeting, each team member answers three questions: (1) 'What did I accomplish yesterday?', (2) 'What will I do before the next daily Scrum meeting?' and (3) 'Are there obstacles in the way?'. The developers give answers to these questions to the other team members. The daily Scrum is not intended to be a status meeting towards the Scrum Master or other roles, it is purely from the team for the team. The Scrum Master is responsible for removing impediments that were brought up in the daily Scrum. No in-depth discussion is supposed to take place in this daily meeting. If further clarification is needed this can be scheduled for interested parties directly after the daily Scrum meeting (Schwaber and Sutherland 2013).

The *product backlog refinement* is an ongoing process, usually done in the form of a workshop. No more than ten percent of the team's capacity for the sprint should be spent on this activity. This activity, also called *backlog grooming*, is meant to split large backlog items, estimate and prioritize them for future sprints. Both the Product Owner and the development team attend this activity. If this refinement is done regularly, the sprint planning meeting should become relatively simple as the items on the top of the backlog will be prepared in detail, meaning their content and business value is clear as well as an effort estimate available (Deemer et al. 2012).

The *sprint review* is a meeting where all involved stakeholders review the work done during the sprint. Based on the principle of 'inspect and adapt', this meeting focuses on the product. It should allow for an in-depth conversation on specific details of the product, giving the Product Owner the chance to see what is going on within the team and the software and giving the team the chance to learn from the Product Owner what is going on in the market. The team presents the latest software product increment in a live demo. Thereby, all people present have the chance to interact and inspect the software and give feedback to the team (Deemer et al. 2012).

In the *sprint retrospective*, the focus is on team-internal processes and their environment. Here, the team discusses what works well and what does not. At the end of the retrospective, major items that went well and improvement areas are identified. Moreover, a plan for implementing actions regarding these areas is agreed upon (Schwaber and Sutherland 2013).

The *product backlog* is a list of customer-focused items that are desired in the product. The individual backlog items are ordered according to the Product

Owner's perception of their business value. The product backlog is a living artifact as it never stops changing. The Product Owner refines backlog items, changes their value and the team estimates the effort needed to implement the items. The higher an item is ranked in the backlog the more detailed and refined it has become. Items further down on the list are more granular and less detailed (Schwaber and Sutherland 2013).

A *product increment* is the sum of the work completed in the current sprint and all previous ones. It is a potentially shippable product state and must be in a usable condition. The *definition of done* is a shared understanding of when a backlog item is done. This definition can include minimum requirements concerning the documentation or functionality tests for the developed increment. The Product Owner and the team need to agree on this definition and it should continually evolve with the maturity of the team (Schwaber and Sutherland 2013).

2.3.2 Agile Software Development on the Team Level

As agile software development originated from small-scale settings, much of the research has focused on the team level. Here, the effects of individual agile techniques such as pair programming or test-driven development (Balijepally et al. 2009; Erdogmus et al. 2005; Mangalaraj et al. 2009) were a natural point of departure for research.

In a literature review by Erickson et al. (2005), research in the fields of Agile Methodology, Agile Modelling and Extreme Programming with a focus on the later was found. Within the XP publications, two main directions are visible. On the one hand, there are case studies with experience reports of practitioners, which cover the XP approach as a whole, and on the other, there is research on individual or small sets of the core XP practices. The mentioned case studies are lacking detail and generally conclude with the positive assessment of XP as a development method. As an individual practice, mostly pair programming was studied, providing mixed results. Erickson et al. conclude with "hard, empirically-based economic evidence is lacking" and "other empirical efforts to study XP, in total or its core practices, are quite limited as well" (Erickson et al. 2005).

Subsequently, a broader avenue of inquiry was followed concentrating on four main categories: introduction and adoption, human and social factors, customer and developer perceptions and comparative studies (Dybå and Dingsøyr 2008). Dybå and Dingsøyr (2008) report a total of 1996 studies published until 2005 with only 36 showing acceptable rigor, credibility and relevance, most dealing with XP. Within the four identified categories following research was observed.

Introduction and adoption. Although easy to adopt and introduce in small organizations, agile processes can prove to be difficult to implement in complex organizations. Especially practices that involve testing (test-first or continuous testing) need to be introduced early as it takes time and effort to properly embrace these concepts. Nevertheless, test-first programming contributed to higher quality in

code as did pair programming, which, in addition, also enhanced learning among team members. However, a few developers mention it to be exhausting, inefficient and a waste of time, suggesting an attitude polarization among users of that practice. It was found that XP worked best with experienced programmers who have a solid domain knowledge. On the management side, training has to occur simultaneously as technical problems appear earlier, which can lead to unfamiliar situations with issues being raised too early for management. Overall, XP teams showed that improved communication with continuous feedback seems to be a key success factor, although many XP teams were perceived as being more isolated by other teams.

Human and social factors. Evidence of agile teams having faith in their own abilities, while showing respect and responsibility, has surfaced. Moreover, trust is pervasive and goes beyond pair-partner trust, as it was also found to be true across pairs and sub teams. The skills of good XP team members are described as analytical, good people-skills and an affinity for learning.

Customer and developer perceptions. Customers appreciated the agile development process, as daily meetings kept them up to date and the higher involvement reduced confusion about development questions. However, this high customer involvement seems to be unsustainable as planning, testing and retrospective activities are demanding and require the customer to acclimatize to each development organization. The developers' perception of XP and Scrum is very good with employees claiming that pair programming sped up the development process and Scrum led to a reduction of overtime.

Comparative studies. In comparison to a more traditional incremental approach, the agile approach allowed the incorporation of changes at later stages with less impact on the overall project. Furthermore, the presentation of working software to the customer combined with his continuous feedback led to a sharp increase in customer satisfaction and an earlier demonstration of business value. In four studies comparing traditional development methods with XP, the productivity differences vary extremely and do not allow for drawing sensible conclusions. This may in part be due to the inherent difficulty of measuring software development productivity and as Dybå and Dingsøyr (2008) mention, the studies did not have an appropriate recruitment strategy to ensure unbiased comparisons. However, many developers themselves indicated that their productivity increased with the introduction of XP. The aspect of product quality can also be characterized as varying, although the quality differences are in a much narrower interval in comparison to productivity. Over all, the improvement of quality ranges from no difference to a 65% improvement in prerelease quality. In a study comparing developers using XP practices with ones that did not, the XP users were more satisfied and comfortable with their jobs (Dybå and Dingsøyr 2008). The studies show a clear bias towards XP and relatively young development teams. They conclude with the finding that both the number and the quality of studies on agile software development needed to be increased (Dybå and Dingsøyr 2008).

Diverging from co-located small-scale settings, Jalali and Wohlin (2012) focused on research on agile practices in distributed settings, in which globally

distributed teams collaborated over a long time on small to medium-sized projects. They agree that the majority of existing literature is in the form of industrial experience reports. The most common practices found were continuous integration, daily standup meetings, pair programming, retrospectives, Scrum of Scrums meetings and test-driven development. Furthermore, many agile practices had been customized to fit their environment (Jalali and Wohlin 2012). They conclude that an insufficient number of studies analyzing the challenges of applying agile in distributed settings has been conducted in order to conclude that agile was efficiently applicable in large distributed projects. The authors demand for further research on modifications of agile methods to support practitioners with guidelines on how to adapt practices to their needs (Jalali and Wohlin 2012).

Recently, Chuang et al. (2014) published a bibliometric analysis of agile literature in the years between 2001 and 2012. Key outlets for agile research, top individual contributors, most cited articles as well as institutions and countries most engaged in agile research were identified. They conclude with the need for more research not only of a practical but of a scholarly nature in particular, since most research on agile development methods are at the infancy stage (Chuang et al. 2014).

The preceding discussion shows that in the domain of agile software development much of the recent literature has focused on individual practices and adoption of agile methods. After having presented the current state of research on agile software development on the team level, the following chapter delves into the topic of large-scale agile development.

2.3.3 Industrial Frameworks for Large-Scale Agile Development

The foundations of large-scale agile development lie in the previously presented agile development approaches. These are extended or modified to better support the differing large-scale environment. Before presenting the practical approaches to scaling agile development, a fundamental question needs to be answered: How big is large-scale?

This question remains unresolved although a few attempts have been made to come to a consistent definition. Participants of the workshop ‘Towards Principles of Large-Scale Agile Development’ as part of the XP2014 conference (Dingsøyr and Moe 2014) have gathered definitions of large-scale agile development, which can be seen in Table 2.5. Work done by Dingsøyr et al. (2014) suggest that a taxonomy of scale in agile development should best be based on the number of teams involved in the development project. They suggest three levels: (1) small-scale (one team), (2) large-scale (two to nine teams) and (3) very large-scale (ten teams or more).

When transferring agile into large settings, the differences become very obvious. As soon as several teams have to cooperate to develop one piece of software, the coordination overhead becomes elevated as now communication and coordination

Table 2.5 Definitions of large-scale agile development

Over 50 developers OR 1/2 million lines of code OR more than 3 sites/time zones
Over 50 persons, over 5 teams, developing together the same product/project using agile method
Agile being applied to more than one team, one project, one product
Multiple teams working together in order to deliver software artefacts
Based on Dingsøyrr and Moe (2014)

lines increase many times over. Not only the number of people involved adds complexity, in large organizations with many locations, the distributed nature of such development setups brings challenges in the form of temporal, geographic and socio-cultural differences (Hossain et al. 2009). Furthermore, many large development efforts are of a product development nature, i.e. the continuous development of one product over many years. This leads to a considerable code history, or legacy code, that needs to be dealt with. To overcome these challenges and give guidance to practitioners, several frameworks have been developed to scale agile methods to large settings. The frameworks, Large-Scale Scrum (LeSS) (Larman and Vodde, n.d.), Scaled Agile Framework (SAFe) (Leffingwell, n.d.) and Scaled Professional Scrum (Nexus) (Schwaber et al., n.d.) will be presented in the following sections.

Large-Scale Scrum. This is an approach that sets out to scale Scrum within the constraints of pure Scrum. Essentially, LeSS is regular Scrum applied to large-scale settings. While it attempts to preserve the strengths of Scrum, it reinforces the need for more process clarity with defined structures. Within the LeSS framework, two variants exist, one for two to eight teams and one for more than eight teams (Larman and Vodde, n.d.).

Variant A keeps all the basic roles of Scrum unchanged but adapts the meeting structure. The previously mentioned sprint planning meeting is held with representatives of each team, so as not to gather too many people in one meeting. At the end of the sprint, a cross-team retrospective is added to advance the overall system improvement. To support inter-team coordination, additional meetings such as Scrum of Scrums can be added. Scrum of Scrums is a meeting to synchronize inter-team activities. Representatives of each team gather regularly, similar to the daily Scrum, to discuss ongoing implementations and current impediments (Larman and Vodde, n.d.).

Variant B, for systems with more than eight teams, introduces the additional role of an Area Product Owner (APO). This role is responsible for requirement areas, which are customer-centric clusters of product backlog items. The leading Product Owner groups each product backlog item into one requirement area and hands the responsibility to the corresponding Area Product Owner. This item is then prioritized by the Area Product Owner who specializes in one part of the product from the customer perspective. Each requirement area has several teams working in it

that pull their items from the area backlog. A requirement area is organized around customer centric requirements, while traditional development areas are organized around the product architecture (Larman and Vodde, n.d.).

Scaled Agile Framework. This framework operates on three levels, portfolio, program and team (Leffingwell, n.d.). Within the portfolio area, individuals are responsible for strategic themes, investment budgets and their allocation to release trains. These release trains essentially constitute individual programs or products in development. The high-level portfolio backlog is broken down into the program backlogs, which are then broken down into the team backlogs. The program level management takes the program epics (high-level backlog items) from the portfolio level and creates a vision and a product roadmap, which guide the development in the next releases. A product increment starts with a planning event, where the program management introduces the vision for the upcoming release and each team plans their individual share of the total development effort. The team level is made up of the cross-functional development teams, which pick their backlog items from the team backlogs (Leffingwell, n.d.).

Scaled Professional Scrum. The Scaled Professional Scrum framework (Nexus) was created by some of the initial developers of Scrum and considers the inter-operation of teams and dependencies between them, the main challenge of scaling Scrum. This framework introduces a team called the ‘nexus’ integration team that is responsible for successful integration of the work done by the individual development teams. A nexus consists of no more than nine teams, which need to integrate their work in order to create a potentially shippable product increment (Schwaber et al., n.d.).

All three frameworks try to scale agile development to multiple teams, however, the direction of their approaches seems very different. While Large-Scale Scrum and Scaled Professional Scrum are at their hearts Scrum scaled to several teams with only slight modifications, the Scaled Agile Framework is much more heavyweight and encompassing. LeSS and Nexus clearly come from the agile side and try to transfer the basic values and concepts to large-scale settings, while SAFe approaches from the more heavyweight side of the spectrum and tries to bring in agile. On the one hand, the lacking involvement of higher management in the LeSS and Nexus approach is covered explicitly in SAFe, but on the other hand, LeSS and Nexus seem much closer to the original principles of the agile development movement.

Furthermore, one could question the applicability of such industrial scale frameworks, as they mostly tend to assume a green field approach. However, taking into account the previously mentioned environmental factors of large organizations that gradually adopt more lightweight development methods, such radical frameworks that require not only a change of the organizational culture, but also an entire restructuring of the development workforce, might not always be applicable in practice. Oftentimes, adapted or contextualized (cf. Hoda et al. 2010) forms of large-scale agile development approaches are more realistic to be implemented. They do not pose those kind of risks to the organization that often come along with a major restructuring.

2.3.4 *Agile Software Development on the Multiteam System Level*

Since the origins of agile development lie in small team contexts, the associated methods have only recently and hesitantly been promoted and studied in large-scale settings. Only 23 papers could be identified, which deal with large-scale agile software development. All papers were reviewed for their coverage of large-scale agile and classified according to their research method, approach and general topic area. The following paragraphs will present the found literature, structured according to type.

The largest group of literature belonged to the *experience reports*. Within this group, several papers illustrate how agile was implemented and what pitfalls were discovered. Many also described the way these impediments were solved within the context of the implementing organization (Benefield 2008; Fry and Greene 2007; Lee 2008; Moore and Spens 2008; Paasivaara and Lassenius 2011; Smits and Pshigoda 2007; Sutherland et al. 2009). Paasivaara et al. (2014) report on the introduction of so called ‘Value Workshops’ to come to a common understanding across the teams about which values the development area has and how to promote them. Three models of architecture support are proposed by Eckstein (2014) to promote emergent architecture within large-scale agile development systems.

Articles in the group of *conceptual literature* try to shed light on fundamental questions within large-scale agile development. As such, Dingsøyr et al. (2014) examine the concept of size based on the number of teams in one development system. They come to the definition that two to nine teams are to be considered large-scale, while ten and more teams should be considered very large-scale. Twenty-one principles of scaled agile are proposed in Laanti (2014) including the notion of controlling processes as opposed to people or to utilize tacit knowledge. Kettunen and Laanti (2008) introduce a framework for understanding the multi-dimensional nature of agility within large organizations. Finally, Power (2014) develops a decision support model for distinguishing what the ‘large’ is referring to in pursuing agile in an organization. He distinguishes between three different large settings: the implementation in one team in a large organization, the use of agile across a large project or if organizational agility is strived for.

The final group consists of two *empirical articles*, which investigate networking and productivity respectively. Moe et al. (2014) investigated through a case study how a newly introduced role of technical area responsible supports knowledge networks between teams. They conclude that this role is central in the knowledge network and acts as a boundary spanner between teams. Furthermore, the size of the knowledge network depended heavily on the company tenure of the team members. Productivity and delays are the core topics in a study by Badampudi et al. (2013). Through a Grounded Theory interview study of five projects, the challenges within those projects were identified. It was revealed that the influencing factors within requirements creation and use, namely collaboration and knowledge management, were predominantly influencing productivity and delays.

2.4 Prior Work on Coordination in Multiteam Systems and Large-Scale Agile Development

Although a considerable amount of research on agile software development has been published (see Sects. 2.3.2 and 2.3.4), the specific topic of coordination in agile multiteam systems remains neglected in previous literature. The following sections present the scarce extant work on coordination in multiteam systems and coordination in large-scale agile development.

2.4.1 *Coordination in Multiteam Systems*

While the concept of MTS has received increasing attention in organizational psychology over the last decade (e.g. Asencio et al. 2012; Davison et al. 2012; DeChurch and Marks 2006; Lanaj et al. 2012), the topic of coordination is underdeveloped within this stream as well. Marks et al. (2001) present a time-based conceptual framework of team processes, including action and transition episodes. Action phases primarily include activities directly related to goal accomplishment, and transition phases include evaluation or planning activities. Within MTSs, the management of these performance episodes is viewed as a central part of coordination (DeChurch and Marks 2006). Marks et al. (2005) found that cross-team processes had the most value in MTSs with a highly interdependent goal hierarchy. Well-managed MTS transition processes influenced MTS performance positively, but did not support team level action processes. Decentralized planning led to enhanced multiteam system performance by fostering proactivity and higher aspiration levels. Nevertheless, strong negative effects were found in excessive risk seeking and coordination failures (Lanaj et al. 2012).

Asencio et al. (2012) propose multiteam charters as a means to develop efficient leadership structures and communication networks. Boundary spanners and communication norms across teams are mentioned as important considerations in MTS collaboration. These differentiated team roles are viewed as a key factor for performance by Davison et al. (2012). Teams that included boundary spanning roles consistently outperformed teams that did not. The reasoning lies in the information processing complexity inherent in large organizations, which leads to the need for formalized boundary spanning (Davison et al. 2012).

In their study of leadership in multiteam systems, DeChurch and Marks (2006) trained teams of leaders in two ways, either by facilitating strategy development or coordination. They found that strategy training was positively related to explicit coordination, with coordination training affecting implicit coordination more heavily.

Beyond these first forays, little is known about coordination in MTSs. However, the notion of cross-team processes (Marks et al. 2005) including communication across teams (Asencio et al. 2012) suggests the importance of directionality of

coordination in MTSs. Furthermore, the aspect of decentralized planning (Lanaj et al. 2012) conveys the influence of coordination locality in MTSs.

2.4.2 *Coordination in Large-Scale Agile Development*

The topic of coordination in large-scale agile development literature is extremely scarce. Overall, only six papers were identified that approach the topic of coordination in large-scale settings. The following sections present these articles according to the areas they focus on.

Challenges of Coordination through Representatives. Paasivaara et al. (2012) report on a case study of two distributed large-scale Scrum projects. Both projects comprised of at least twenty teams and were distributed worldwide. The practice under study was the Scrum of Scrums meeting, whereby representatives of each team meet regularly to discuss current issues and future topics on an inter-team level. The results show that, with the amount of teams present in each project, the individual interests of all participants were too wide to be of interest to everybody else. Furthermore, some representatives did not know what to report and so did not contribute anything to the meeting. A possible solution that was implemented was a feature-specific Scrum of Scrums meeting for the teams working together on one feature. A location specific Scrum of Scrums meeting was reported as not working very well. Similarly, Hole and Moe (2008) found that modularization of the software enables agility and supports Scrum of Scrums meetings. However, they report that co-location was a facilitator of this type of meeting.

Influences on Coordination Mechanisms and Coordination Strategies. Hole and Moe (2008) investigated how three distributed projects applying agile methods coordinated their work. They observed that in order to reduce standardization and direct supervision in global software development projects, trust was an essential trait. Furthermore, the possibility for short exchanges via online messengers supported mutual adjustment. Li and Maedche (2012) present preliminary results of a study to show what factors influence the formation of coordination strategies. In an agile distributed software development setting, a difference in time zones seemed to increase the use of mechanistic coordination. Both mechanistic and organic coordination mechanisms were needed to cope with changing customer requirements. Finally, the introduction of explicit coordination mechanisms improved mutual trust and shared cognition of long-standing colleagues through intensified communication and agile practices (Li and Maedche 2012). Scheerer et al. (2014) present different conceptual coordination strategy types for inter-team coordination. These types are based on the amount of existing mechanistic, organic and cognitive coordination and are illustrated with examples from large-scale agile product development settings.

Challenges and Their Alleviation in Large-Scale Agile Development. Lagerberg et al. (2013) present results of a quantitative study on two projects with 14 and 15 teams respectively. Only one of the two projects had fully implemented

agile development methods. From the comparison between the two, agile methods facilitated knowledge sharing and increased visibility of the status of other teams. Based on the higher awareness of other teams in the agile project, they concluded that a higher inter-team and intra-team coordination effectiveness was present in comparison to the non-agile project. Lagerberg et al. (2013) propose that the use of complete teams with feature responsibility and open space offices contributed to the higher coordination effectiveness. Scheerer et al. (2015) describe the problem of sequential task dependencies in agile backlogs within multiteam development settings. Through a simulative modeling approach, they conclude that the degree of freedom a Product Owner has in choosing a backlog order is severely limited. As such, one dependency already limits his choice by 50%. In order to mitigate this problem they suggest dependency avoidance through different team structures within multiteam systems or the early detection and active management of dependencies through techniques such as user story mapping or dedicated dependency tracking practices.

Overall, the previous literature remains very limited. However, Hole and Moe's (2008) study suggests that different locations influence the chosen coordination types while Paasivaara et al. (2012) findings advocate an influence of development system size on the direction of coordination. Finally, work done by Scheerer et al. (2015) shows the influence of task dependencies on the coordination of software development MTSS. After having illustrated the limited previous work on coordination in MTSS and large-scale agile development, what follows next is the construction of the research framework underlying this study.

2.5 Research Framework

The point of departure for this study is the view that coordinated action is a necessary condition for organizational performance (cf. Cheng 1984; Lawrence and Lorsch 1967; Simon 1976). Therefore, the perspective of Okhuysen and Bechky (2009) and Kotlarsky et al. (2008) is taken that coordination is an outcome state in the form of coordinated action which in turn necessitates the integrating conditions common understanding, predictability and accountability (see Sect. 2.1.4).

Coordination Configuration. The notion of a coordination strategy is expanded upon by the introduction of a *coordination configuration*. While previous research has examined types of coordination mechanisms (cf. Strode et al. 2012) and their direction in conjunction with their location (cf. Nidumolu 1995, 1996), to the best of the author's knowledge no study has incorporated all three aspects of coordination type, locus, and direction into one configuration. The coordination configuration is based on the previously discussed literature (see Sect. 2.1) with three core dimensions. The *coordination type* describes the type of coordination mechanisms in use and ranges from a pure mechanistic approach to a predominantly organic

type (Espinosa et al. 2010; Thompson 1967; Van De Ven et al. 1976). The *coordination locus* resembles what Mintzberg (1980) calls decentralization of decision making (see Sect. 2.1.1) in that it signifies the location in the multiteam system where coordination activities take place. It ranges from a primarily centralized system, e.g. with decisions about coordination situated centrally in the MTS, to a decentralized approach, e.g. where coordination is located in a more dispersed fashion among the individual teams within the MTS. Within the *coordination direction* dimension, the orientation of communication necessary for coordination is depicted. This can occur either in a vertical fashion across hierarchical levels or within one level between different actors of the latter (Davison et al. 2012; Mintzberg 1980; Nidumolu 1995; Van De Ven et al. 1976). The three dimensions can be arranged to form archetypes, of which two are outlined in Fig. 2.1. *Top-down planning* represents a mechanistic, centralized approach with predominantly vertical coordination. On the other hand, *bottom-up adjustment* is portrayed as a largely organic and decentralized strategy with horizontal coordination. These two extreme ends of a continuum do not represent a general, static orientation towards coordination in a multiteam system. Rather, these archetypes together with multiple variations in between, are to be understood as a set of coordination configurations that together form a change process in reaction to a discrete event. The integrating conditions for coordination are established by enacting specific coordination configurations that are a composite of the just mentioned dimensions within the configuration.

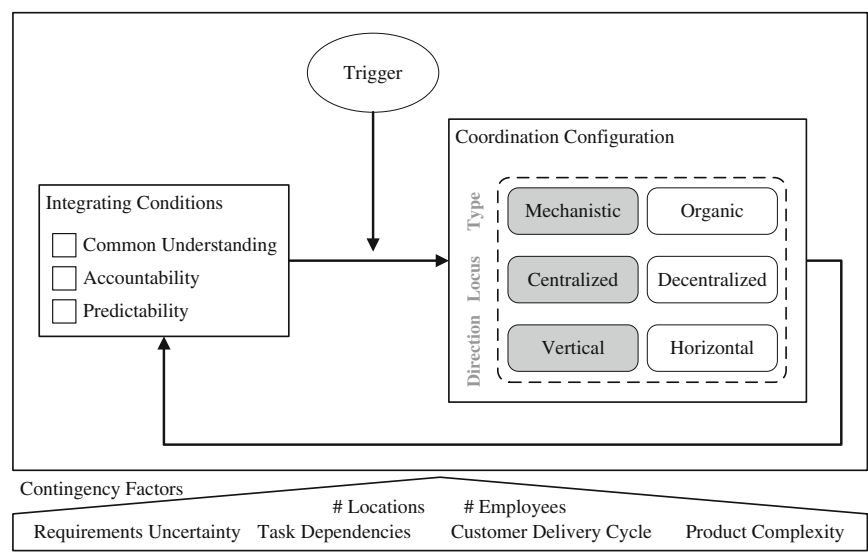


Fig. 2.1 Research Framework. Based on Davison et al. (2012), Espinosa et al. (2010), Mintzberg (1980), Okhuysen and Bechky (2009), Thompson (1967), Van De Ven et al. (1976)

Trigger. Other studies have viewed the link between coordination mechanisms and the situational context in terms of fit. That is, the utilized coordination mechanisms are based on situational factors of the coordination process, i.e. time zone differences require more mechanistic coordination or higher project complexity increases the frequency of synchronization activities (cf. Li and Maedche 2012; Strode et al. 2012). This study views the enacted coordination configuration as a reaction to a trigger within the multiteam system. Therefore, the previously static view of the coordination strategy is altered into a quasi-dynamic one, as this research proposes a time-dependent view of the coordination process. These triggers can manifest in different ways. Poole et al. (2000) characterize two causal forces, ones that operate continuously and others that only come into play at specific points in time. The more apparent trigger is what will be called the discrete exogenous trigger. These types of trigger are of a discrete nature and can be attributed to a specific point in time. The second type is what will be called a latent endogenous trigger. These are more slowly moving causes which are of a latent nature and build up over time until a threshold is reached whence they act to trigger a change (Grzymala-Busse 2010).

Integrating Conditions. Previous studies (see Sect. 2.1) have considered the direct influence of coordination mechanisms, e.g. coordination by plan or rules, or coordination by mutual adjustment, on coordination outcomes. As the underlying factors that coordination, especially task coordination, tries to achieve are not considered, the conditions as proposed by Okhuysen and Bechky (2009) are viewed as the integrating conditions to be achieved for coordinated action to occur (see Sect. 2.1.4). As the focus of this study lies on task coordination and not on execution activities, coordinated action can be regarded as the outcome of the integrating conditions common understanding, predictability and accountability, while the task coordination activities are depicted through the coordination configuration.

Contingency Factors. The coordination within an MTS and thus the enacted coordination configuration may be influenced by contingent factors, which impel the system to react in different ways based on the manifestation of these factors. Organizational aspects such as the number of locations (Hole and Moe 2008) and employees (Paasivaara et al. 2012) of an MTS may influence coordination. Furthermore, procedural traits in the form of customer delivery cycles affect the time available to coordinate tasks and thus may affect coordination. Finally, aspects such as requirements uncertainty, product complexity (Tatikonda and Rosenthal 2000) and task dependencies (Scheerer et al. 2015) have been shown to have an impact on planning and task coordination procedures within MTSs and thus lead to an altered coordination configuration enactment.

In conclusion, the preliminary research framework acts as a frame to examine and illustrate changes in the coordination configuration originating from a triggering event. These changes lead to the establishment of integrating conditions that are viewed as preconditions for coordinated action.

References

- Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile software development methods. *Vtt Publications*, 478(3), 167–168. Retrieved from <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.161.5931&rep=rep1&type=pdf>
- Asencio, R., Carter, D. R., DeChurch, L. A., Zaccaro, S. J., & Fiore, S. M. (2012). Charting a course for collaboration: A multiteam perspective. *Translational Behavioral Medicine*, 2(4), 487–494. Retrieved from <http://link.springer.com/article/10.1007/s13142-012-0170-3>
- Bacharach, S. B. (1989). Organizational theories: Some criteria for evaluation. *Academy of Management Review*, 14(4), 496–515.
- Badampudi, D., Fricker, B., & Moreno, A. M. (2013). Perspectives on productivity and delays in large-scale agile projects. In *Proceedings of the 14th International Conference Agile Processes in Software Engineering and Extreme Programming* (pp. 180–194).
- Balijepally, V., Mahapatra, R., Nerur, S. P., & Price, K. H. (2009). Are two heads better than one for software development? The productivity paradox of pair programming. *MIS Quarterly*, 33(1), 91–118. Retrieved from <http://aisel.aisnet.org/misq/vol33/iss1/7/>
- Beck, K. (2001). *Extreme programming explained: Embrace change*. Addison-Wesley Professional.
- Benefield, G. (2008). Rolling out agile in a large enterprise. In *Proceedings of the Annual Hawaii International Conference on System Sciences* (pp. 1–10).
- Benington, H. D. (1956). Production of large computer programs. In *Proceedings, ONR Symposium on Advanced Programming Methods for Digital Computers, June 1956* (pp. 15–27).
- Boehm, B. (1988). A spiral model of software development and enhancement. *Computer*, 21(5), 61–72.
- Cannon-Bowers, J. A., Salas, E., & Converse, S. (1993). Shared mental models in expert team decision making. In N. John Castellan (Ed.), *Current issues in individual and group decision making* (pp. 221–246).
- Cheng, J. L. C. (1984). Organizational coordination, uncertainty, and performance: An integrative study. *Human Relations*, 37(10), 829–851.
- Chuang, S.-W., Luor, T., & Lu, H.-P. (2014). Assessment of institutions, scholars, and contributions on agile software development (2001–2012). *Journal of Systems and Software*, 93, 84–101. Retrieved from <http://www.sciencedirect.com/science/article/pii/S0164121214000697>
- Crowston, K., Rubleske, J., & Howison, J. (2006). Coordination theory: A ten-year retrospective. In P. Zhang & D. Galletta (Eds.), *Human-computer interaction in management information systems* (pp. 120–138). M. E. Sharpe, Inc.
- Davis, S. M., & Lawrence, P. R. (1977). *Matrix*. Reading: Addison-Wesley.
- Davison, R. B., Hollenbeck, J. R., Barnes, C. M., Sleesman, D. J., & Ilgen, D. R. (2012). Coordinated action in multiteam systems. *Journal of Applied Psychology*, 97(4), 808–824. Retrieved from <http://www.ncbi.nlm.nih.gov/pubmed/22201246>
- DeChurch, L. A., & Marks, M. A. (2006). Leadership in multiteam systems. *Journal of Applied Psychology*, 91(2), 311–329. Retrieved from <http://www.ncbi.nlm.nih.gov/pubmed/16551186>
- Deemer, P., Benefield, G., Larman, C., & Vodde, B. (2012). The Scrum primer version 2.0. Retrieved from http://www.scrumprimer.org/scruprimer20_small.pdf
- Devine, D. J. (2002). A review and integration of classification systems relevant to teams in organizations. *Group Dynamics: Theory, Research, and Practice*, 6(4), 291–310.
- Dingsøyr, T., Fægri, T. E., & Itkonen, J. (2014). What is large in large-scale? A taxonomy of scale for agile software development. In A. Jedlitschka, P. Kuvaja, M. Kuhrmann, T. Männistö, J. Münch, & M. Raatikainen (Eds.), *Product-focused software process improvement* (Vol. 8892, pp. 273–276). Springer International Publishing. Retrieved from <http://link.springer.com/10.1007/978-3-319-13835-0>
- Dingsøyr, T., & Moe, N. B. (2014). Towards principles of large-scale agile development. In T. Dingsøyr, N. Moe, R. Tonelli, S. Counsell, C. Gencel, & K. Petersen (Eds.), *Agile methods*.

- Large-scale development, refactoring, testing, and estimation* (Vol. 199, pp. 1–8). Springer International Publishing. Retrieved from <http://www.springer.com/computer/swe/book/978-3-319-14357-6>
- Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, 50(9–10), 833–859. Retrieved from <http://linkinghub.elsevier.com/retrieve/pii/S0950584908000256>
- Eckstein, J. (2014). Architecture in large scale agile development. In T. Dingsøyr, N. Moe, R. Tonelli, S. Counsell, C. Gencel, & K. Petersen (Eds.), *Agile methods. Large-scale development, refactoring, testing, and estimation* (Vol. 199, pp. 21–29). Springer International Publishing. Retrieved from http://dx.doi.org/10.1007/978-3-319-14358-3_3
- Erdogmus, H., Morisio, M., & Torchiano, M. (2005). On the effectiveness of the test-first approach to programming. *IEEE Transactions on Software Engineering*, 31(3), 226–237.
- Erickson, J., Lyytinen, K., & Siau, K. (2005). Agile modeling, agile software development, and extreme programming: The state of research. *Journal of Database Management*, 16(4), 88–100. Retrieved from <http://www.igi-pub.com/articles/details.asp?ID=5327>
- Espinosa, J. A., Armour, F., & Boh, W. F. (2010). Coordination in enterprise architecting: An interview study. In *System Sciences (HICSS), 2010 43rd Hawaii International Conference on* (pp. 1–10).
- Espinosa, J. A., Cummings, J. N., & Pickering, C. (2012). Time separation, coordination, and performance in technical teams. *IEEE Transactions on Engineering Management*, 59(1), 91–103.
- Espinosa, J. A., Lerch, J. F., Kraut, R. E., Salas, E., & Fiore, S. M. (2004). Explicit vs. implicit coordination mechanisms and task dependencies: One size does not fit all. In *Team cognition: Understanding the factors that drive process and performance* (pp. 107–129). Washington, DC: American Psychological Association.
- Faraj, S., & Xiao, Y. (2006). Coordination in fast-response organizations. *Management Science*, 52(8), 1155–1169. Retrieved from <http://dx.doi.org/10.1287/mnsc.1060.0526>
- Fowler, M., & Highsmith, J. (2001). The agile manifesto. Retrieved from <http://www.pmp-projects.org/Agile-Manifesto.pdf>
- Fry, C., & Greene, S. (2007). Large scale agile transformation in an on-demand world. In *Proceedings of the AGILE Conference 2007* (pp. 136–142). Washington, DC.
- Goodhue, D. L., & Thompson, R. L. (1995). Task-technology fit and individual performance. *MIS Quarterly*, 19(2), pp. 213–236. Retrieved from <http://www.jstor.org/stable/249689>
- Grzymala-Busse, A. (2010). Time will tell? Temporality and the analysis of causal mechanisms and processes. *Comparative Political Studies*, 44(9), 1267–1297. Retrieved from <http://cps.sagepub.com/content/44/9/1267>
- Gulick, L., & Urwick, L. (1937). *Papers on the science of administration*. New York: Institute of Public Administration, Columbia University.
- Guzzo, R. A., & Dickson, M. W. (1996). Teams in organizations: Recent research on performance and effectiveness. *Annual Review of Psychology*, 47(1), 307–338. Retrieved from <http://dx.doi.org/10.1146/annurev.psych.47.1.307>
- Hempel, C. (1965). *Aspects of scientific explanation and other essays in the philosophy of science*. New York: The Free Press.
- Hoda, R., Kruchten, P., Noble, J., & Marshall, S. (2010). Agility in context. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications* (pp. 74–88). New York, NY, USA: ACM. Retrieved from <http://doi.acm.org/10.1145/1869459.1869467>
- Hole, S., & Moe, N. B. (2008). A case study of coordination in distributed agile software development. In R. O'Connor, N. Baddoo, K. Smolander, & R. Messnarz (Eds.), *Software process improvement* (Vol. 16, pp. 189–200). Springer Berlin Heidelberg. Retrieved from http://dx.doi.org/10.1007/978-3-540-85936-9_17
- Hossain, E., Babar, M. A., & Paik, H. P. H. (2009). Using scrum in global software development: A systematic literature review. In *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on* (pp. 175–184).

- Ilgen, D. R., Hollenbeck, J. R., Johnson, M., & Jundt, D. (2004). Teams in organizations: From input-process-output models to IMO models. *Annual Review of Psychology*, 56(1), 517–543. Retrieved from <http://dx.doi.org/10.1146/annurev.psych.56.091103.070250>
- Jalali, S., & Wohlin, C. (2012). Global software engineering and agile practices: A systematic review. *Journal of Software: Evolution and Process*, 24(6), 643–659. Retrieved from <http://dx.doi.org/10.1002/smr.561>
- Katz, D., & Kahn, R. L. (1978). *The social psychology of organizations*. Wiley.
- Kettunen, P., & Laanti, M. (2008). Combining agile software projects and large-scale organizational agility. *Software Process: Improvement and Practice*, 13(2), 183–193. Retrieved from <http://dx.doi.org/10.1002/spip.354>
- Kieser, A. (Ed.). (1993). *Organisationstheorien*. Kohlhammer.
- Kotlarsky, J., van Fenema, P. C., & Willcocks, L. P. (2008). Developing a knowledge-based perspective on coordination: The case of global software projects. *Information & Management*, 45(2), 96–108.
- Kozlowski, S. W. J., & Bell, B. S. (2003). Work groups and teams in organizations work groups and teams in organizations. In W. C. Borman, D. R. Ilgen, & R. J. Klimoski (Eds.), *Handbook of psychology: Industrial and organizational psychology* (Vol. 12, pp. 333–375). New York: Wiley.
- Kraut, R. E., & Streeter, L. A. (1995). Coordination in software development. *Communications of the ACM*, 38(3), 69–81.
- Kruchten, P. (1998). *The rational unified process: An introduction*. Amsterdam: Addison-Wesley Longman.
- Laanti, M. (2014). Characteristics and principles of scaled agile. In T. Dingsøyr, N. Moe, R. Tonelli, S. Counsell, C. Gencel, & K. Petersen (Eds.), *Agile methods. Large-scale development, refactoring, testing, and estimation* (Vol. 199, pp. 9–20). Springer International Publishing. Retrieved from http://dx.doi.org/10.1007/978-3-319-14358-3_2
- Lagerberg, L., Skude, T., Emanuelsson, P., Sandahl, K., & Stahl, D. (2013). The impact of agile principles and practices on large-scale software development projects: A multiple-case study of two projects at Ericsson. In *International Symposium on Empirical Software Engineering and Measurement* (pp. 348–356).
- Lanaj, K., Hollenbeck, J. R., Ilgen, D. R., Barnes, C. M., & Harmon, S. J. (2012). The double-edged sword of decentralized planning in multiteam systems. *Academy of Management Journal*, 56(3), 1–61. Retrieved from <http://amj.aom.org/content/early/2012/07/20/amj.2011.0350.short>
- Larman, C., & Basili, V. R. (2003). Iterative and incremental development: A brief history. *Computer*, 36(6), 47–56.
- Larman, C., & Vodde, B. (2008). *Scaling lean & agile development: Thinking and organizational tools for large-scale Scrum*. Upper Saddle River, N.J: Addison-Wesley Professional.
- Larman, C., & Vodde, B. (n.d.). LeSS Framework. Retrieved August 18, 2015, from <http://less.works/>
- Lawrence, P. R., & Lorsch, J. W. (1967). Differentiation and integration in complex organizations. *Administrative Science Quarterly*, 12(1), 1–47. Retrieved from <http://www.jstor.org/stable/2391211>
- Lee, E. C. (2008). Forming to performing: Transitioning large-scale project into agile. In *Agile 2008 Conference* (pp. 106–111). Toronto, ON.
- Lee, G., Espinosa, J. A., & DeLone, W. (2013). Task environment complexity, global team dispersion, process capabilities, and coordination in software development. *IEEE Transactions on Software Engineering*, 39(12), 1753–1771.
- Leffingwell, D. (n.d.). Scaled agile framework. Retrieved from <http://www.scaledagileframework.com/>
- Li, Y., & Maedche, A. (2012). Formulating effective coordination strategies in agile global software development teams. In *Proceedings of the International Conference on Information Systems (ICIS 2012)* (pp. 1–6).

- Malone, T. W., & Crowston, K. (1990). What is coordination theory and how can it help design cooperative work systems? In *Proceedings of the Conference on Computer Supported Cooperative Work*. Los Angeles.
- Malone, T. W., & Crowston, K. (1991). Toward an interdisciplinary theory of coordination. *ACM Computing Surveys*, 120(120), 1–45. Retrieved from <http://dspace.mit.edu/handle/1721.1/2356>
- Malone, T. W., & Crowston, K. (1994). The interdisciplinary study of coordination. *ACM Computing Surveys*, 26(1), 87–119. Retrieved from <http://portal.acm.org/citation.cfm?doid=174666.174668>
- Malone, T. W., Crowston, K., & Herman, G. A. (Eds.). (2003). *Organizing business knowledge: The MIT process handbook*. Cambridge, MA: MIT press.
- Malone, T. W., Crowston, K., Lee, J., Pentland, B., Dellarocas, C., Wyner, G., ... O'Donnell, E. (1999). Tools for inventing organizations: Toward a handbook of organizational processes. *Management Science*, 45(3), 425–443. Retrieved from <http://mansci.journal.informs.org/cgi/doi/10.1287/mnsc.45.3.425>
- Mangalaraj, G., Mahapatra, R., & Nerur, S. P. (2009). Acceptance of software process innovations —The case of extreme programming. *European Journal of Information Systems*, 18(4), 344–354. Retrieved from <http://www.palgrave-journals.com/ejis/journal/v18/n4/abs/ejis200923a.html>
- March, J. G., & Simon, H. A. (1958). *Organizations*. New York: Wiley.
- Marks, M. A., DeChurch, L. A., Mathieu, J. E., Panzer, F. J., & Alonso, A. (2005). Teamwork in multiteam systems. *Journal of Applied Psychology*, 90(5), 964–971. Retrieved from <http://www.ncbi.nlm.nih.gov/pubmed/16162068>
- Marks, M. A., Mathieu, J. E., & Zaccaro, S. J. (2001). A temporally based framework and taxonomy of team processes. *The Academy of Management Review*, 26(3), pp. 356–376. Retrieved from <http://www.jstor.org/stable/259182>
- Mathieu, J. E., Marks, M. A., & Zaccaro, S. J. (2001). Multiteam systems. In N. Anderson, D. S. Ones, H. K. Sinangil, & C. Viswesvaran (Eds.), *Handbook of industrial, work and organizational psychology, Volume 2 Organizational psychology* (Vol. 2, pp. 289–313). London: Sage Publications Ltd.
- Mathieu, J. E., & Maynard, M. T. (2008). Team effectiveness 1997–2007: A review of recent advancements and a glimpse into the future. *Journal of Management*, 34(3), 410–476. Retrieved from <http://jom.sagepub.com/cgi/doi/10.1177/0149206308316061>
- Mintzberg, H. (1980). Structure in 5's: A synthesis of the research on organization design. *Management Science*, 26(3), 322–341. Retrieved from <http://mansci.journal.informs.org/content/26/3/322.short>
- Mintzberg, H. (1983). *Structure in fives: Designing effective organizations*. Prentice-Hall, Inc.
- Moe, N. B., Šmite, D., Šablis, A., Börjesson, A.-L., & Andréasson, P. (2014). Networking in a large-scale distributed agile project. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 12:1–12:8). New York, NY: ACM. Retrieved from <http://doi.acm.org/10.1145/2652524.2652584>
- Moore, E., & Spens, J. (2008). Scaling agile: Finding your agile tribe. In *Agile 2008 Conference* (pp. 121–124). Toronto, ON.
- Moreland, R., Argote, L., & Krishnan, R. (1996). Socially shared cognition at work: Transactive memory and group performance. In *What's social about social cognition? Research on socially shared cognition in small groups* (pp. 57–84). Sage Publications, Inc. Retrieved from <http://psycnet.apa.org/psycinfo/1996-98278-003>
- Nerur, S. P., Mahapatra, R. K., & Mangalaraj, G. (2005). Challenges of migrating to agile methodologies. *Communications of the ACM*, 48(5), 72–78.
- Nidumolu, S. (1995). The effect of coordination and uncertainty on software project performance: Residual performance risk as an intervening variable. *Information Systems Research*, 6(3), 191.
- Nidumolu, S. (1996). A comparison of the structural contingency and risk-based perspectives on coordination in software-development projects. *Journal of Management Information System*, 13(2), 77–113. Retrieved from <http://dl.acm.org/citation.cfm?id=1189558.1189564>

- Okhuysen, G. A., & Bechky, B. A. (2009). Coordination in organizations: An integrative perspective. *The Academy of Management Annals*, 3(1), 463–502.
- Paasivaara, M., & Lassenius, C. (2011). Scaling scrum in a large distributed project. In *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on* (pp. 363–367).
- Paasivaara, M., Lassenius, C., & Heikkilä, V. T. (2012). Inter-team coordination in large-scale globally distributed scrum: Do Scrum-of-Scrums really work? In *Empirical Software Engineering and Measurement (ESEM), 2012 ACM-IEEE International Symposium on* (pp. 235–238).
- Paasivaara, M., Väättänen, O., Hallikainen, M., & Lassenius, C. (2014). Supporting a large-scale lean and agile transformation by defining common values. In T. Dingsøyr, N. Moe, R. Tonelli, S. Counsell, C. Gencel, & K. Petersen (Eds.), *Agile methods. Large-scale development, refactoring, testing, and estimation* (Vol. 199, pp. 73–82). Springer International Publishing. Retrieved from http://dx.doi.org/10.1007/978-3-319-14358-3_7
- Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., & Still, J. (2008). The impact of agile practices on communication in software development. *Empirical Software Engineering*, 13(3), 303–337.
- Poole, M. S., Van De Ven, A. H., Dooley, K., & Holmes, M. E. (2000). *Organizational change and innovation processes: Theory and methods for research*. Oxford University Press.
- Power, K. (2014). A model for understanding when scaling agile is appropriate in large organizations. In T. Dingsøyr, N. Moe, R. Tonelli, S. Counsell, C. Gencel, & K. Petersen (Eds.), *Agile methods. Large-scale development, refactoring, testing, and estimation* (Vol. 199, pp. 83–92). Springer International Publishing. Retrieved from http://dx.doi.org/10.1007/978-3-319-14358-3_8
- Rico, R., Sánchez-Manzanares, M., Gil, F., & Gibson, C. (2008). Team implicit coordination processes: A team knowledge-based approach. *Academy of Management Review*, 33(1), 163–184.
- Royce, W. W. (1970). Managing the development of large software systems. In *Proceedings of IEEE WESCON* (Vol. 26, pp. 328–388).
- Salas, E., Sims, D. E., & Burke, C. S. (2005). Is there a “big five” in teamwork? *Small Group Research*, 36(5), 555.
- Scheerer, A., Bick, S., Hildenbrand, T., & Heinzl, A. (2015). The effects of team backlog dependencies on agile multiteam systems: A graph theoretical approach. In *System Sciences (HICSS), 2015 48th Hawaii International Conference on* (pp. 5124–5132). Koloa, HI. Retrieved from <http://ieeexplore.ieee.org/document/7070428/>
- Scheerer, A., Hildenbrand, T., & Kude, T. (2014). Coordination in large-scale agile software development: A multiteam systems perspective. In *System Sciences (HICSS), 2014 47th Hawaii International Conference on* (pp. 4780–4788). Waikoloa, HI. Retrieved from <http://ieeexplore.ieee.org/document/6759189/>
- Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum*. Prentice Hall.
- Schwaber, K., Dame, D., Hundhausen, R., Kong, P., Maher, R., Porter, S., ... Verheyen, G. (n.d.). Scaled Professional Scrum (Nexus) Framework. Retrieved August 21, 2015, from https://kenschwaber.files.wordpress.com/2015/06/nexusguide_v1-0.pdf
- Schwaber, K., & Sutherland, J. (2013). The Scrum Guide. Retrieved from <https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide.pdf>
- Simon, H. A. (1976). *Administrative behavior: A study of decision-making processes in administrative organization*. The Free Press.
- Smits, H., & Pshigoda, G. (2007). Implementing scrum in a distributed software development organization. In *Agile Conference (AGILE), 2007* (pp. 371–375).
- Strode, D. E., Hope, B., Huff, S. L., & Link, S. (2011). Coordination effectiveness in an agile software development context. In *PACIS 2011*.
- Strode, D. E., Huff, S. L., Hope, B., & Link, S. (2012). Coordination in co-located agile software development projects. *Journal of Systems and Software*, 85(6), 1222–1238. Retrieved from <http://dx.doi.org/10.1016/j.jss.2012.02.017>

- Sutherland, J., Schoonheim, G., & Rijk, M. (2009). Fully distributed scrum: Replicating local productivity and quality with offshore teams. In *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on* (pp. 1–8).
- Sutherland, J., & Schwaber, K. (1995). Business object design and implementation workshop. In *Addendum to the proceedings of the 10th annual conference on Object-oriented programming systems, languages, and applications (Addendum)* (pp. 170–175). New York, NY, USA: ACM. Retrieved from <http://doi.acm.org/10.1145/260094.260274>
- Sutton, R. I., & Staw, B. M. (1995). What theory is not. *Administrative Science Quarterly*, 40(3), 371–384.
- Takeuchi, H., & Nonaka, I. (1986). The new new product development game. *Harvard Business Review*, 64(1), 137–146. Retrieved from <http://linkinghub.elsevier.com/retrieve/pii/0737678286900536>
- Tatikonda, M. V., & Rosenthal, S. R. (2000). Technology novelty, project complexity, and product development project execution success: A deeper look at task uncertainty in product innovation. *IEEE Transactions on Engineering Management*, 47(1), 74–87.
- Taylor, F. W. (1911). *The principles of scientific management*. New York, London: Harper & Brothers.
- Thompson, J. D. (1967). *Organizations in action: Social science bases of administrative theory* (Vol. 48). New York: McGraw-Hill.
- Tosi, H. L. (2008). James March and Herbert Simon, Organizations. In *Theories of organization* (pp. 93–102). SAGE Publications, Inc.
- Van De Ven, A. H., Delbecq, A. L., & Koenig, R. J. (1976). Determinants of coordination modes within organizations. *American Sociological Review*, 41(2), 322–338. Retrieved from <http://www.jstor.org/stable/2094477>
- VersionOne Inc. (2013). 8th Annual State of Agile Development Survey. Retrieved from www.versionone.com/pdf/2013-state-of-agile-survey.pdf
- Zaccaro, S. J., Marks, M. A., & DeChurch, L. A. (2012). Multiteam systems: An introduction. In S. J. Zaccaro, M. A. Marks, & L. A. DeChurch (Eds.), *Multiteam systems an organization form for dynamic and complex environments* (pp. 3–32). New York, NY, USA: Routledge.

<http://www.springer.com/978-3-319-55326-9>

Coordination in Large-Scale Agile Software
Development
Integrating Conditions and Configurations in Multiteam
Systems
Scheerer, A.
2017, XV, 128 p. 42 illus., 8 illus. in color., Hardcover
ISBN: 978-3-319-55326-9