Inter-Team Coordination in Large Agile Software Development Settings: Five Ways of Practicing Agile at Scale

Saskia Bick
SAP SE*
Dietmar-Hopp-Allee 16
69190, Walldorf, Germany
saskia.bick@sap.com

Alexander Scheerer
SAP SE
Dietmar-Hopp-Allee 16
69190, Walldorf, Germany
alexander.scheerer@sap.com

Kai Spohrer
University of Mannheim
L15, 1-6, 520
68161, Mannheim, Germany
spohrer@uni-mannheim.de

ABSTRACT

Scaling agile software development to settings with multiple interconnected teams requires inter-team coordination. We present a multiple case study at one of the world's largest enterprise software vendors, SAP SE, where we analyzed five ways of practicing agile at scale. We illustrate case by case the coordination mechanisms and practices the individual development programs applied to cope with the challenges of scaled agile software development.

Categories and Subject Descriptors

D.2.9 [**Management**]: Life Cycle, Programming Teams
K.6.3 [**Software Management**]: Software Development Process

Keywords

Large-Scale Agile, Software Development, Inter-Team Coordination, Dependencies, Multiteam Systems.

*Saskia Bick is a PhD student at the University of Mannheim working in a joint research project with SAP SE (bick@uni-mannheim.de).

1. INTRODUCTION

Agile software development principles and methodologies were originally targeted at small development settings [3, 6]. Increasingly, however, larger organizations appreciate the benefits of agile development and make use of well-reported advantages such as the flexibility and adaptability of agile development teams [8]. Particularly the agile development framework "scrum" is being applied in many small as well as large software development organizations today [17]. The more individuals and teams work on a large-scale development project, though, the more its complexity increases. This is due to the fact that the number of inter-team dependencies tends to increase with each additional development team. Hence, the more inter-team dependencies exist, the more coordination effort is needed [4, 16] for individual teams to achieve their proximal goals and for the entire development system to achieve the overall, collective goal [10].

XP '16 Workshops, May 24 2016, Edinburgh, Scotland Uk

DOI: http://dx.doi.org/10.1145/2962695.2962699

Both conceptual and empirical research have examined the nature of team level dependencies [16] and respective coordination mechanisms [4, 5]. However, research on inter-team coordination in larger settings remains extremely scarce. Coordination is the management of dependencies [4]; and inter-team coordination specifically refers to the coordination of activities between two or more teams within a multiteam system (MTS) [10]. Few but very rich empirical studies have been published that focus on the inter-team level and report on benefits, challenges and experiences in large-scale agile development [2, 7, 12–14]. From a more conceptual point of view, Barlow et al. [1] describe a so-called *hybrid approach*, which combines the application of traditional and agile software development, depending on the level of uncertainty and the characteristics of existing interdependencies. Scheerer and Kude introduce two opposing archetypes, namely top-down planning and bottom-up adjustment [15]. Top-down planning refers to a mechanistic, centralized approach with mostly vertical coordination, which can be observed between a team and a superior, for instance. Bottom-up adjustment, on the other hand, is a largely organic and decentralized strategy with mostly horizontal coordination between the teams of an MTS.

In recent years, also several industrial frameworks have emerged that aim at scaling agile development. Large Scale Scrum (LeSS), the Scaled Agile Framework (SAFe), the Nexus framework, and Disciplined Agile Delivery (DAD), however, have a common tendency to assume a greenfield setting. Yet, especially large, established software vendors often do not have the luxury of such legacy-free development structures.

Neither research on inter-team coordination and large-scale agile development, nor industrial solutions have so far managed to explicitly illustrate and analyze coordination in large agile development systems. It is this lack of understanding as to how interdependent teams in large-scale agile development settings coordinate with one another, which we try to address in this work. With our multiple case study, we therefore seek to answer the question *how inter-team coordination is achieved in large-scale agile development systems*. To do so, we conducted 68 interviews with scrum masters (SM), product owners (PO) and other central roles of five large development programs at SAP SE and comparatively analyzed their approaches to managing inter-team dependencies in large-scale agile settings.

Practitioners can directly learn from our real-life cases as they may apply selected modes of inter-team coordination, relate them to contingencies present in their own settings, and learn from the best practices described in the discussion section. We add to research on inter-team coordination and large-scale agile development by providing detailed insights into inter-team

coordination processes based on within and between-case analyses.

2. RESEARCH DESIGN

To gain a detailed understanding of different ways of practicing agile at scale and its relation to inter-team coordination, we conducted a multiple case study in five ongoing, large-scale software development programs at our case organization, SAP SE. Our data collection took place between October 2013 and December 2014. During this time, we conducted 68 semi-structured interviews, which lasted between 30 and 90 minutes each, for a total of approximately 59 hours of interview material. We interviewed POs and SMs as well as other selected key roles such as chief product owners (CPO), (chief) architects and program managers. All interviews were recorded, transcribed, coded and analyzed. Additional sources such as organization charts, wiki pages, product architecture maps and backlog management systems provided more information to complete the picture of the five cases.

We selected the cases based on their differences in size, location and development cycles to ensure heterogeneity. Overall, we collected data from five MTS, consisting of 40 teams, which were located in up to six different locations worldwide and had been applying scrum for at least five years before data collection.

At SAP SE, there are some guiding rules considering agile teams: development teams should be colocated teams-of-ten, with one SM and one PO for each team. On the program level, a CPO owns the responsibility for development scope and quality. Within a "development program", which is a relatively stable MTS for ongoing software product development (as opposed to a temporary "project"), there are monthly integration cycles for all teams. Apart from these guidelines, teams have broad areas of autonomy. It is up to each development team to decide upon sprint length (usually two or four weeks) or development practices (e.g. pair programming, peer-code review, TDD, etc.).

3. RESULTS

3.1 The Traditional Case

The first MTS we investigated was developing a logistics solution with 13 teams across four locations. This solution had been in development for more than ten years with around 140 employees involved. The on-premise software product was characterized as being a tightly integrated system with almost no requirements that could be implemented within a single team alone. The teams were structured along the business process the solution was supporting (e.g. one team being responsible for the component 'billing').

Scaling via Central Team Directives. One way to cope with the increased difficulty of coordinating a large number of teams and employees is to install an organizational unit specifically for that purpose (see Figure 1).

In the traditional case, the central team is an example of such an approach. Seven people, including the CPO, architects and product experts, formed this team, which generated the high-level work items in the form of epics and allocated them to a lead development team. The central team met daily for one hour to discuss current and urgent topics as well as to plan the upcoming sprints. This team was a separate entity to the development teams in the MTS and did not include any team representatives.

Once per sprint, the central team hosted a status and handover meeting with each of the development teams individually. In this

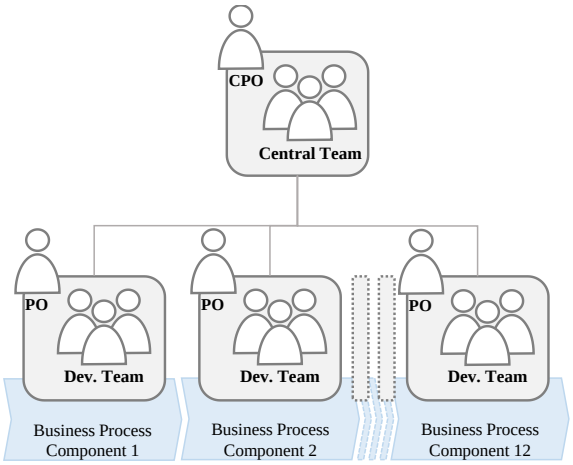

Figure 1 Traditional Case Organizational Setup

meeting, the central team wanted to see what had been achieved by the development team in the previous sprint and communicated the backlog items and their priorities for the next sprint. This scaling approach via a central team aimed to prevent team interlocks resulting from priority conflicts. However, not all members of the MTS were positive about this setup.

> *Interviewee 7: "The POs normally don't like this solution, since we restrict the responsibility of a PO considerably."*

With this approach, however, the central team appeared unable to foresee all dependencies between the teams, which led to recurring escalations in order to resolve resulting issues. As responsibility for inter-team coordination rested with the central team, this also led to a centralized problem solving strategy: the central team was generally called upon to settle any dispute between teams.

The perceived quality of coordination between the teams was quite mixed. Whereas some interviewees saw room for improvement or rated it as acceptable, others considered the coordination within their MTS as highly problematic.

3.2 The Cloud Case

The second case we studied was developing a cloud integration platform to connect on-premise systems with cloud solutions. This MTS consisted of ten teams with around 95 employees spread across two locations. The development of this product had been ongoing for more than five years. In line with the more technical focus of this product, the software architecture was modularized along the technical components needed for the integration services provided by the solution. The teams in this case were nonetheless arranged so that they could implement customer features across the entire software stack.

Scaling via Iterative Proxy Collaboration. The cloud case chose an approach which is close to the Scrum of Scrums method described in literature [9]. Representatives of each team met on the next higher level of the hierarchy and formed a virtual central team. As such, this MTS had a CPO/PO round consisting of the two CPOs and all POs of the development teams (see Figure 2). The CPO/PO round met regularly to discuss the upcoming sprint and to plan which backlog items were to be implemented. This

group also resolved disputes and coordination problems that could not be sorted out between teams directly.

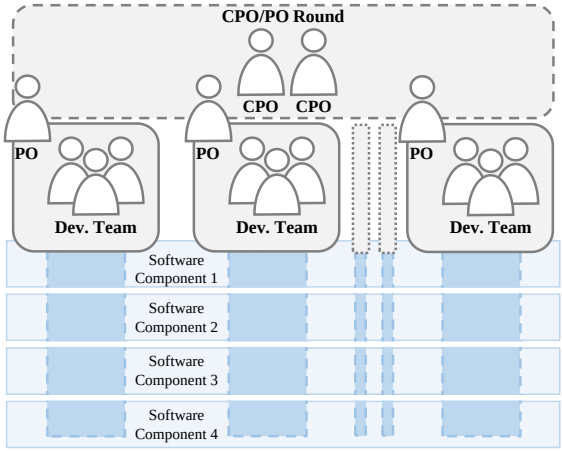

Figure 2 Cloud Case Organizational Setup

In the middle of a current sprint, the CPOs gathered new topics for the next sprint from the POs. These were discussed in detail in the CPO/PO round, explicitly considering dependencies between teams. There was no need for a handover meeting as the team POs were always part of the planning process from the start.

If problems arose during the implementation phase the team POs discussed the issue among themselves and came up with a solution. This solution was then communicated to the CPOs who had to accept the proposal.

Interviewee 36: "If something unexpected happens in the development sprint, we have to file a change request to the CPOs and mail it around to the other POs so that everybody knows that something has changed […]."

Among the interviewed, the overall satisfaction with the coordination was very high.

3.3 The Distributed Case

The distributed case was developing a mobile application development platform with seven teams in six main locations. Overall, this MTS had more than 70 employees, working on an on-premise as well as a cloud version of the solution. The product in its current form had been under development for more than one year, but had existed in different forms before. The solution was structured along largely self-contained software modules that interacted on predefined interfaces. In line with this, the organizational structure defined teams according to these components.

Scaling via Central Team Planning based on Team Inputs. In this case, the scaling of coordination was done through a balance of central planning and decentralized input from the development teams. The initial high-level backlog items were gathered through input from the teams and the product management. The CPO and APOs were responsible for this high-level backlog as well as the prioritization of the backlog items. These were then detailed out by the central architecture team, which discussed technical and functional aspects with the individual development teams (see Figure 3).

Prototypes and proof of concepts were elaborated in close collaboration with the teams. After this validation, the high-level backlog items were detailed out into user stories, which could be implemented by the development teams. All technical dependencies between teams were recorded in a central

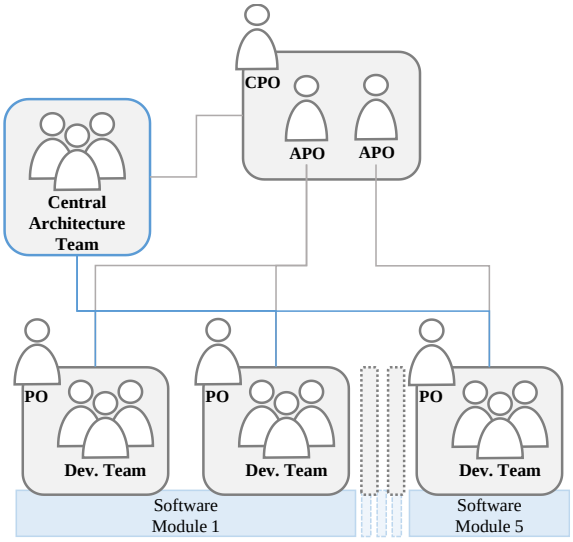

Figure 3 Distributed Case Organizational Setup

document. In the development phase, all SMs came together in a twice-weekly call to discuss the current status and review dependencies based on the mentioned document.

All interviewees regarded the coordination in the MTS as good with some even mentioning it being a well-coordinated project.

3.4 The Colocated Case

This MTS was developing a material management solution with six teams and around 85 employees. The development program was entirely colocated and had been developing this solution for more than three years. The product was developed as an on-premise solution that was characterized as an integrated system, modularized along the business process, which it supported. In accordance with this, the teams in this case were also structured based on the underlying business process, similar to the traditional case.

Scaling via Full Collaboration. This MTS had introduced a joint release planning workshop for all members several releases ago (see Figure 4).

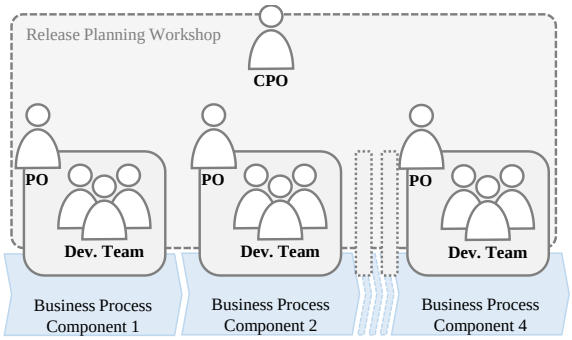

Figure 4 Colocated Case Organizational Setup

In the run-up to this workshop, the team POs prepared the high-level backlog items for the next release, which were to be discussed that day. On the workshop day itself, all the teams were introduced to the business cases of the software product for the next release by the CPO and then started with individual sessions to break down their team backlogs. Dependencies to other teams or topics, which were still unclear, could be discussed right away with the appropriate people.

Contrary to this very inclusive and involving approach for release planning, the problem solving strategy during implementation still relied on escalation to the CPO. Nonetheless, the employees in this MTS were very pleased with this scaling approach.

3.5 The Modular Case

The last MTS presented here was developing a solution to manage safety processes and their compliance. With around 40 employees in four teams at three locations, the modular case was developing both an on-premise as well as a cloud version of their product. This product had been in development for more than ten years. The solution consisted of four separate modules, which had little overlap from a content perspective but shared a common technological foundation. These four modules were also reflected in the organizational structure as each teams was responsible for one of the separate modules.

Scaling via Ad Hoc Communication. Since the teams in this case only shared a foundation layer and had largely independent implementation tasks, there was little need for coordination between the teams (see Figure 5). Therefore, a weekly CPO/PO call served primarily as an update meeting for the CPO about the status of each team. If the MTS had to extend the underlying platform, the involved teams communicated selectively with one another to implement needed functionality.

The quality of coordination in this case was considered to be quite good because an open communication culture allowed the easy access of other team members if a feature required outside input.

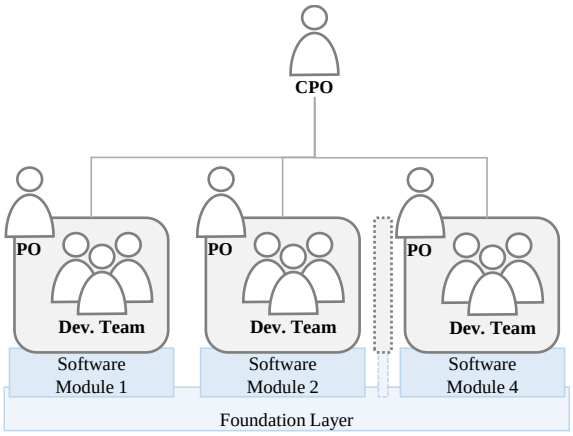

Figure 5 Modular Case Organizational Setup

4. DISCUSSION & IMPLICATIONS

Two aspects stood out during the analysis of these case studies. First, we note that the inter-team coordination approaches vary strongly in terms of their type or nature of coordination [5, 15]. That is, while coordination in the traditional and the distributed case focused more on top-down planning (i.e. via mechanistic, centralized and vertical coordination mechanisms), coordination in the modular and the colocated case tended towards bottom-up adjustment (i.e. organic, decentralized and horizontal coordination mechanisms). The cloud case showed parts of both.

Second, with coordination being the management of inter-team dependencies [4], we analyzed the different inter-team coordination approaches considering how proactively inter-team dependencies were being managed. In two cases, namely the traditional and the modular case, there were little to no practices in place that dealt with active recognition, analysis and management of dependencies. Whereas in the other three cases, i.e. the cloud case, the distributed case and the colocated case, we saw different practices for a proactive management of dependencies. These best practices covered all kinds of recurring meetings to collaboratively plan development activities and gather feedback from all parties involved, e.g. planning workshops and unit-wide retrospectives. In the joint planning workshops, all teams gathered in one room to plan the upcoming release together and thereby identify and address dependencies early on. Other best practices were cross-team testing, cross-topic handling via a (temporally) dedicated lead PO and team enablement in task estimation. The role of a lead product owner took over the temporary responsibility for coordinating issues and resolving dependencies resulting from cross-team topics.

Contrary to our expectations, the strongly differing environmental factors did not seem to play a key role regarding actual inter-team coordination success. However, they appear to influence the decision for or against particular approaches to manage agile at scale. What we found to be more important was a match of practices between the team and inter-team levels. An emphasis on top-down planning apparently works best with detailed and accurate upfront dependency management on an inter-team level, whereas bottom-up adjustment seems to work better when teams do not primarily rely on escalations for dependency resolution but are enabled to work them out on their own account. Overall, a proactive take on inter-team dependency management by both management and team members seems to play together best when there is a focus on collaboration and iteration that helps to account for the different planning timeframes in the two hierarchical levels of ongoing development efforts of large-scale agile development systems.

Our results have important implications for practice. First and foremost, our results indicate that proactive dependency management between development teams is fruitful and equally possible in settings of top-down and bottom-up coordination. The architectural modularization of the software product appears to render different coordination setups more appropriate [11]. Whereas top-down approaches to dependency management appear to work well in settings of teams that work on separate and loosely coupled modules, much more team member involvement seems to be necessary for effective inter-team coordination in settings of teams that work on more integrated and tightly coupled modules. Whether such solutions also lead to sustainable architectural results remains, however, open to further investigations. Managers looking for an appropriate way of scaling agile in their multiteam system may compare their current setups and especially their architecture to the archetypes we

presented to see which successful way of agile at scale may suit their current setup.

Moreover, we are deeply convinced that iterative higher level planning activities should take place well ahead of each sprint and should solicit, aggregate, and incorporate feedback from experienced team members and architects. Putting iterative planning and the required feedback solicitation on the agenda of CPOs and POs may in fact improve inter-team coordination in all setups.

To put these observations from industrial practice onto scientifically firmer grounds, future research should investigate the precise workings of planning in large-scale agile settings. In fact, hybrid approaches between traditional and agile development may be an interesting target for research that incorporates high level planning elements into agile daily routine.

5. REFERENCES

[1] Barlow, J.B. et al. 2011. Overview and Guidance on Agile Development in Large Organizations. *Communications of the Association for Information Systems*. 29, 2 (2011), 25–44.

[2] Batra, D. et al. 2010. Balancing Agile and Structured Development Approaches to Successfully Manage Large Distributed Software Projects: A Case Study from the Cruise Line Industry. *Communications of the Association for Information Systems*. 27, 1 (2010), 379–394.

[3] Boehm, B. and Turner, R. 2003. Using Risk to Balance Agile and Plan-Driven Methods. *Computer*. 36, 6 (2003), 57–66.

[4] Crowston, K. 2003. A Taxonomy of Organizational Dependencies and Coordination Mechanisms. *Organizing Business Knowledge: The MIT Process Handbook*. T.W. Malone et al., eds. MIT Press. 85–108.

[5] Espinosa, J.A. et al. 2010. Coordination in Enterprise Architecting: An Interview Study. *43rd Hawaii International Conference on System Sciences (HICSS)* (Honolulu, HI, USA, 2010), 1–10.

[6] Hoda, R. et al. 2010. Agility in Context. *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)* (Reno, NV, USA, 2010), 74–88.

[7] Kettunen, P. and Laanti, M. 2008. Combining Agile Software Projects and Large-scale Organizational Agility. *Software Process Improvement and Practice*. 13, 2 (2008), 183–193.

[8] Kude, T. et al. 2014. Adaptation Patterns in Agile Information Systems Development Teams. *22nd European Conference on Information Systems (ECIS)* (Tel Aviv, Israel, 2014), 1–15.

[9] Larman, C. and Vodde, B. 2010. *Practices for Scaling Lean & Agile Development Large, Multisite, and Offshore Product Development with Large-Scale Scrum*. Addison Wesley.

[10] Mathieu, J.E. et al. 2001. Multiteam Systems. *Handbook of Industrial, Work and Organizational Psychology*. N. Anderson et al., eds. Routledge. 289–313.

[11] Nord, R. et al. 2014. Agile in distress: Architecture to the rescue. *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation* (2014), 43–57.

[12] Nyfjord, J. et al. 2014. Conventions for Coordinating Large Agile Projects. *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation*. 199, (2014), 58–72.

[13] Paasivaara, M. et al. 2014. Supporting a Large-Scale Lean and Agile Transformation by Defining Common Values. *Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation*. 199, (2014), 73–82.

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

[15] Scheerer, A. and Kude, T. 2014. Exploring Coordination in Large-Scale Agile Software Development: A Multiteam Systems Perspective. *35th International Conference on Information Systems (ICIS)* (Auckland, New Zealand, 2014), 1–11.

[16] Thompson, J.D. 1967. *Organizations in Action: Social Science Bases of Administrative Theory*. McGraw-Hill.

[17] West, D. and Grant, T. 2010. Agile Development: Mainstream Adoption Has Changed Agility. *Forrester Research*. (2010).